NOVA IMS Information Management School

MDSAA

Master Degree Program in

Data Science and Advanced Analytics

Effective and efficient image classification from Deep Features Statistics

Pedro Afonso Ferreira Cotovio de Félix Peças

Master Thesis

presented as partial requirement for obtaining the Master Degree Program in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

2022

Title:

Subtitle:

Student full name

MDSAA

[this page should not be included in the digital version. Its purpose is only for the printed version]

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

EFFECTIVE AND EFFICIENT IMAGE CLASSIFICATION FROM DEEP FEATURES STATISTICS

by

Pedro Afonso Ferreira Cotovio de Félix Peças

Master Thesis presented as partial requirement for obtaining the Master's degree in Data Science and Advanced Analytics, with a specialization in Data Science

Supervised by

Professor/Prof. Mauro Castelli, Universidade NOVA de Lisboa - NOVA IMS

Co-Supervised by

Professor/Prof. Illya Bakurov, Universidade NOVA de Lisboa – NOVA IMS

October 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Pedro Peças

Lisboa, 10/18/2023

DEDICATION

To the uncharted territories of data and the patterns they conceal, this thesis represents not just an academic pursuit but also the enduring love and support of those closest to my heart. I wholeheartedly dedicate this work to my beloved family — my father, mother, and brother — and to my dear friends, who have been my steadfast companions on this journey. To my father, whose strength and wisdom have ceaselessly inspired me to chase excellence with integrity. Your guidance has lit my path in every venture. To my mother, a beacon of nurturing love and patience, whose belief in me never wavered even when challenges seemed insurmountable. Your encouragement has been my sanctuary. And to my brother for his companionship, support, and the laughter we've shared, making light of the heaviest burdens. Equally important is this dedication to my friends, who have been more like a second family. Your unwavering faith, endless humor, and countless hours of shared dreams and deep conversations have not only eased the rigors of academic life but also enriched it immeasurably. Each of you has contributed to shaping my journey, offering perspectives and memories that I cherish deeply. As I stand at the threshold of completing my Master's in Data Science, I am reminded that every analysis, every data point, and every late-night coding session has been part of a larger narrative underscored by your presence. This achievement is a testament to the collective strength, support, and sacrifices of both my family and friends. Here's to our journey together, filled with challenges, growth, and unforgettable revelations.

ACKNOWLEDGMENTS

Navigating the multifaceted domain of data science and delving deep into image classification would not have been possible without the support, guidance, and encouragement of many. First and foremost, I express my heartfelt gratitude to Professor Mauro Castelli, whose expertise, insights, and dedication to the field have been invaluable throughout this journey. Your profound knowledge and meticulous feedback have shaped this research in countless ways. I am equally indebted to my co-supervisor, Illya Bakurov. Your attention to detail, constructive criticism, and continuous support have been pillars of strength, pushing me to strive for excellence and refine my understanding of the subject matter. To my peers, classmates, and the entire faculty of the Data Science department, thank you for fostering an environment of collaboration, curiosity, and challenge. The lively discussions, late-night brainstorming sessions, and shared moments of revelation have enriched this academic voyage. Lastly, to my family and friends, your unwavering belief in my abilities, patience during my moments of doubt, and ceaseless encouragement have been my anchor. This accomplishment is as much yours as it is mine. In this intricate dance of numbers, algorithms, and visuals, each one of you has played a pivotal role, and for that, I am eternally grateful.

ABSTRACT

In this thesis, "Effective and Efficient Image Classification from Deep Features Statistics," it is presented an in-depth analysis of the balance between computational efficiency and the accuracy of image classification models. The study leverages pre-trained deep learning architectures—VGG16, DenseNet121, and ResNet50—on CIFAR-10 and SVHN datasets, aiming to reduce training time without compromising accuracy. The research introduces a novel tradeoff ratio metric, providing a pragmatic approach to evaluate the efficacy of Transfer Learning (T.L.) against traditional Machine Learning (M.L.) techniques. The findings illuminate pathways for time-efficient image classification, making significant contributions to both academic research and practical applications in the field.

KEYWORDS

Artificial Intelligence; CNN; Statistics; Machine Learning; Transfer Learning

Sustainable Development Goals (SGD):

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

TL	Transfer Learning
ML	Machine Learning
CNN	Convolutional Neural Network
VGG16	Visual Geometry Group 16 (a deep learning architecture)
ResNet50	Residual Network 50 (a deep learning architecture)
DenseNet121	Densely Connected Convolutional Network 121 (a deep learning architecture)
SVHN	Street View House Numbers (a dataset)
CIFAR-10	Canadian Institute for Advanced Research - 10 (a dataset)
GPU	Graphics Processing Unit
SVM	Support Vector Machine
LR	Logistic Regression
RF	Random Forest
ETC	ExtraTreeClassifier
ViT	Vision Transformer (a deep learning architecture)
WRNs	Wide Residual Networks (a deep learning architecture)
DL	Deep Learning
FC	Fully Connected
RGB	Red Green Blue
Adam	Adaptive Moment Estimation
SHAP	Shapley Additive Explanations
RR	Tradeoff Ratio

1. INTRODUCTION

In today's rapidly advancing digital age, the ability to process and classify vast amounts of image data swiftly is of paramount importance. With the explosion of digital imagery in various domains — from social media and e-commerce to autonomous driving and medical diagnostics — the significance of efficient image classification cannot be overstated. Deep learning models, particularly those using convolutional neural networks (CNNs), have demonstrated remarkable prowess in image classification tasks. However, the time and computational resources required to train these models often present significant challenges. Extended training durations can lead to increased computational costs, hinder rapid deployment, and delay iterative model refinement, posing substantial barriers in many real-world applications.

Against this backdrop, this thesis dives deep into the theme of "Effective and Efficient Image Classification from Deep Features Statistics." How does statistical analysis of deep features enhance image classification model efficiency? The primary objective of this research is to significantly reduce the training time of image classification models using statistics, utilizing the power of established deep learning architectures like VGG16, DenseNet121, and ResNet50. Renowned for their image classification capabilities, these architectures are initialized with ImageNet pre-trained weights to provide a solid foundation for further exploration and optimization. While maintaining accuracy remains an essential objective, the primary ambition of this research is not necessarily to enhance accuracy but to ensure that competitive levels of accuracy are achieved within drastically reduced training windows.

The rationale behind this focus on efficiency over incremental accuracy improvements stems from the practical demands of contemporary image processing tasks. In scenarios such as real-time surveillance, rapid diagnostics in healthcare, and on-the-fly image classification in mobile applications, the luxury of prolonged training times is often unfeasible. Thus, a methodological shift toward reducing computational demand without significantly sacrificing performance is urgently needed.

To achieve a robust and multifaceted analysis, this research employs a diverse array of image datasets, each chosen for its unique characteristics and ability to challenge different aspects of image classification models. The datasets include CIFAR-10 and SVHN (Street View House Numbers), a real-world dataset derived from Google Street View. The variability and complexity of these datasets ensure a comprehensive evaluation landscape that addresses various facets of the image classification challenge.

Beyond selecting and utilizing robust datasets, this research acknowledges and harnesses the significant strides made in deep learning architectures. A notable example is the Vision Transformer (ViT) model, as introduced by (Dosovitskiy et al., 2020), which applies Transformer architectures from natural language processing to image classification. This innovative approach treats image patches as analogs to words, achieving high levels of accuracy with less computational resources when pre-trained on large datasets. Additionally, this thesis incorporates insights from the innovative deep learning architecture ResNeXt, as proposed by (Xie et al., 2017) in their work 'Aggregated Residual Transformations for Deep Neural Networks'. ResNeXt introduces cardinality as an architectural dimension, alongside depth and width, and utilizes a split-transform-merge strategy for improved efficiency and classification accuracy, particularly on complex datasets like ImageNet.

Further, the research draws from the concept of Wide Residual Networks (WRNs) by (Zagoruyko & Komodakis, n.d.). Their findings highlight the benefits of widening network layers, rather than deepening them, for enhanced performance and training efficiency, especially on benchmarks like CIFAR, SVHN, and ImageNet. By synergizing ImageNet's pre-trained weights with our set architectures, we aim to not only harness the inherent strengths of these models but also delve into their adaptability and transferability. This exploration is crucial in understanding how these pre-trained models can be fine-tuned and optimized to achieve our primary goal of reducing training time while maintaining the same or better accuracy.

The methodological core of this study revolves around the strategic modification and application of deep learning techniques. Traditional approaches often involve training deep neural networks from scratch. This process can be prohibitively time-consuming and resource-intensive. Instead, by leveraging pre-trained models, this research aims to bypass the initial, most demanding training phases. Theoretically, this approach allows quicker adaptation to specific tasks by refining and tuning models that have already learned robust, generalizable features from large and diverse datasets like ImageNet.

The experimental phase is performed through Google Colab, relying on an A100 GPU. This choice is not merely a convenience but a strategic decision. The A100, with its advanced architecture and high throughput, is well-suited for handling the intensive computation required by deep learning models. Through this, the thesis aims to investigate the theoretical and methodological aspects of efficient image classification and their practical implementations, considering the constraints and opportunities presented by current state-of-the-art hardware.

This thesis aims to provide pathways for time-efficient image classification in practical scenarios. It leverages the foundational knowledge encapsulated in ImageNet's pre-trained weights. It combines it with cutting-edge computational resources and nuanced training strategies. The insights and methodologies developed herein hold the potential to guide best practices in model selection, dataset usage, and training strategy formulation. These contributions are aimed not just at the academic community but also at practitioners in fields where rapid, efficient image classification can be a game-changer.

In the following chapters, readers will be introduced to a detailed exploration of the architectures, datasets, methodologies, and nuanced strategies pivotal to this research. The narrative will unfold, beginning with a comprehensive review of the relevant literature, outlining the advancements in deep learning for image classification, and discussing the challenges and opportunities in the field. Following this, the thesis will present an in-depth analysis of the chosen deep learning architectures, elaborating on their design and functionality. Lastly, an in-depth process in which readers will understand how statistics can impact the training window of pre-trained models.

2. LITERATURE REVIEW

2.1. EFFICIENCY AND PERFORMANCE OPTIMIZATION IN CNNS

Convolutional Neural Networks (CNNs) stand at the forefront in the dynamic domain of deep learning, driving advancements in various applications from image recognition to autonomous systems. Yet, the pursuit of more profound and more accurate networks often clashes with the practical limitations of training time and computational expense. This thesis confronts this dichotomy head-on, proposing that the strategic application of statistical methods can markedly condense the training duration of traditional CNNs without compromising their formidable capabilities.

This thesis, which explores the efficiency and adaptability of CNN architectures like DenseNet121, ResNet50, and VGG16, a significant application in agriculture (Shah et al., 2023) demonstrates the practical utility of these models in a specialized domain by employing Inception V3, VGG16, VGG19, and ResNet50 for early detection of rice leaf blast disease. Their success with ResNet 50, leveraging its modified connection-skipping architecture for high accuracy in disease detection, exemplifies the real-world effectiveness of CNNs.

On the other hand, we can see the remarkable capabilities of Convolutional Neural Networks (CNNs), as exemplified by (Jiang et al., 2019), extend well beyond traditional applications. Originally developed for tasks like image and video recognition, CNNs are now finding groundbreaking applications in various industry-specific areas. The study's success in power transformer equipment recognition is just one instance of this expanding utility.

This complements the thesis's focus on using statistical methods to optimize CNN training across diverse applications. The case study underlines the importance of transfer learning in boosting CNN performance and broadens the scope of their application, reinforcing the thesis's narrative of combining architectural innovation with statistical insight in deep learning.

DenseNet: A Paradigm of Efficient Connectivity

At the heart of this efficiency revolution is the DenseNet architecture, a groundbreaking approach delineated in the work of (Huang et al., 2016). The dense connectivity pattern of DenseNet is not merely a feature but the linchpin of a paradigm shift. Unlike conventional networks where layers are sequentially connected, DenseNet innovates with a labyrinthine network where each layer is directly connected to every other, creating a rich tapestry of feature sharing. This architectural marvel bolsters feature propagation and significantly trims the network's appetite for parameters, streamlining the training process. The empirical results demonstrate DenseNet's superior performance on benchmarks like CIFAR-10 and ImageNet, showcasing its efficiency in parameter usage and robustness against overfitting. For a detailed breakdown of DenseNet's performance compared to other architectures, refer to Table 2.1.1.

Table 2.1.1 – Error rates (%) on CIFAR and SVHN datasets. K denotes network's growth rate

Method	Depth	Params	C10	C10+	C100	C100+	SVHN
Network in Network [22]	-	-	10.41	8.81	35.68	-	2.35
All-CNN [32]	-	-	9.08	7.25	-	33.71	-
Deeply Supervised Net [20]	-	-	9.69	7.97	-	34.57	1.92
Highway Network [34]	-	-	-	7.72	-	32.39	-
FractalNet [17]	21	38.6M	10.18	5.22	35.34	23.30	2.01
with Dropout/Drop-path	21	38.6M	7.33	4.60	28.20	23.73	1.87
ResNet [11]	110	1.7M	-	6.61	-	-	-
ResNet (reported by [13])	110	1.7M	13.63	6.41	44.74	27.22	2.01
ResNet with Stochastic Depth [13]	110	1.7M	11.66	5.23	37.80	24.58	1.75
	1202	10.2M	-	4.91	-	-	-
Wide ResNet [42]	16	11.0M	-	4.81	-	22.07	-
	28	36.5M	-	4.17	-	20.50	-
with Dropout	16	2.7M	-	-	-	-	1.64
ResNet (pre-activation) [12]	164	1.7M	11.26*	5.46	35.58*	24.33	-
	1001	10.2M	10.56*	4.62	33.47*	22.71	-
DenseNet ($k = 12$)	40	1.0M	**7.00**	5.24	**27.55**	24.42	1.79
DenseNet ($k = 12$)	100	7.0M	**5.77**	**4.10**	**23.79**	**20.20**	1.67
DenseNet ($k = 24$)	100	27.2M	**5.83**	**3.74**	**23.42**	**19.25**	**1.59**
DenseNet-BC ($k = 12$)	100	0.8M	**5.92**	4.51	**24.15**	22.27	1.76
DenseNet-BC ($k = 24$)	250	15.3M	**5.19**	**3.62**	**19.64**	**17.60**	1.74
DenseNet-BC ($k = 40$)	190	25.6M	-	**3.46**	-	**17.18**	-

ResNet: Revolutionizing Deep Network Training

The narrative of efficiency is further enriched by the seminal work of (He et al., 2015) on deep residual learning. Their paper introduces the concept of residual learning within the ResNet framework. ResNet's layers are engineered to refine the identity mappings rather than learning unreferenced functions in a bold divergence from traditional learning paradigms. By incorporating identity shortcut connections that circumnavigate one or more layers (see Figure 2.1.1), ResNet mitigates the impediments of vanishing gradients and the notorious degradation problem, carving a path towards streamlined training of profound networks. The findings of (He et al., 2015), as detailed in their influential paper, provide substantial backing for the effectiveness of ResNet in reducing error rates and improving the trainability of deep networks.

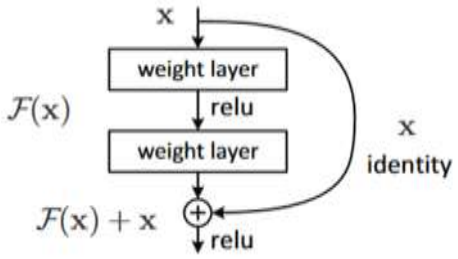


Figure 2.1.1 - Residual Learning: a building block

Empirical Analysis: Depth Versus Training Efficiency

Delving into empirical evidence, we present a comparative analysis of training and test error rates across networks of varying depths. These plots articulate a critical observation: there exists a point of inflection where increasing depth yields diminishing accuracy returns. This insight, supported by the work of both (He et al., 2015), advocates for an optimal network depth that balances computational efficiency and accuracy. This balance can be fine-tuned with statistical methodologies. See Figure 2.1.2 for a representation of training error and test error on CIFAR-10 with 20-layer and 56-layer plain networks, where the deeper network has higher training error and, thus, test error.

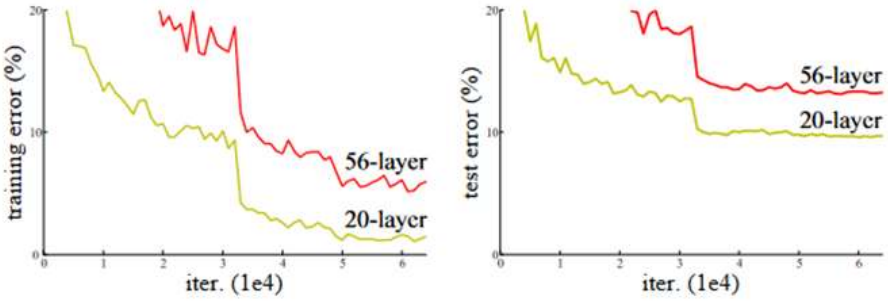


Figure 2.1.2 - Training error (left) and test error (right) on Cifar-10 with 20-layer and 56-layer 'plain' networks.

Visualizing CNN Complexity

To encapsulate the complexity inherent in CNNs, we examine a detailed schematic of a deep network architecture, Figure 2.1.3. This visualization is more than a mere illustration; it is a map detailing the terrain where statistical methods could be most impactful. By identifying and optimizing the bottlenecks within such complex structures, this thesis envisions a future where CNNs can be trained rapidly, with minimal resource expenditure, without sacrificing the depth required for sophisticated tasks. The image below showcases example network architectures for ImageNet, including the VGG-19 model as a reference, a plain network with 34 parameter layers, and a residual network with 34 parameter layers, all highlighting the trade-offs between depth and computational demand.

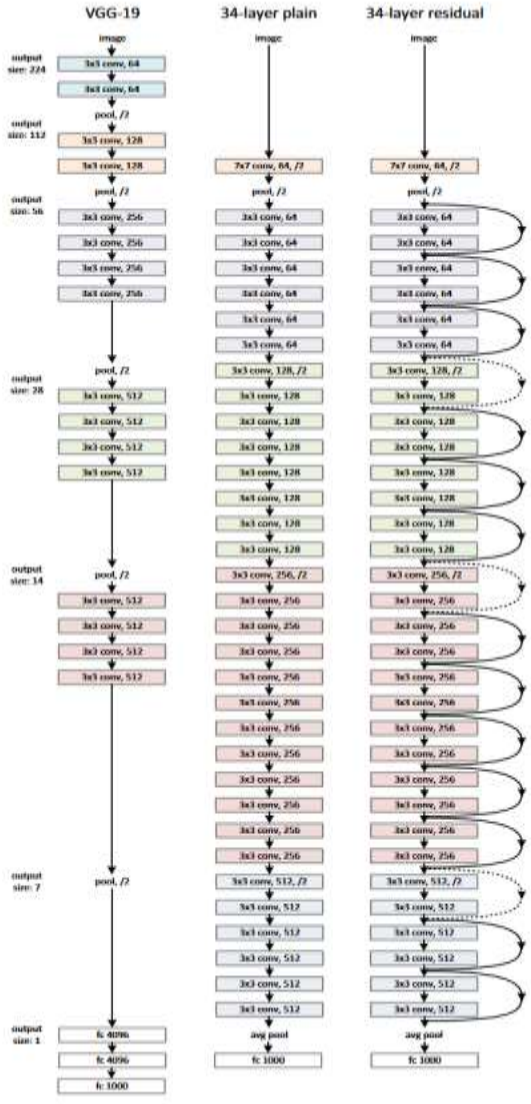


Figure 2.1.3 - Example network architectures for ImageNet. Left: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs)

Additionally, this thesis acknowledges the work of (Liu & Deng, 2016) in applying deep CNNs to small dataset scenarios. Their study on modifying the VGG-16 architecture for effective use in image classification with small sample sizes, such as CIFAR-10, challenges traditional views. By employing stronger regularization, batch normalization, and varying dropout rates, they demonstrate that deep networks, which are typically associated with large datasets, can be effectively adapted for smaller ones. This finding not only complements the efficiency and adaptability of CNN architectures like DenseNet and ResNet but also emphasizes the versatility and potential of deep networks in various data scenarios. Liu and Deng's approach, achieving a significant error rate reduction, aligns with the thesis's focus on optimizing training efficiency and underscores the broader applicability of statistical methods in deep learning.

In conclusion, this thesis stands at the confluence of three significant streams of CNN architectures: DenseNet121, ResNet50, and VGG16. DenseNet's densely connected topology exemplifies how innovation in network connectivity can minimize parameter overhead and enhance training efficiency. As explored by (He et al., 2015), ResNet's residual learning paradigm confronts the vanishing gradient problem, facilitating the training of deep networks by allowing for more effortless gradient flow. VGG16, with its uniform architectural simplicity, serves as a testament to the power of depth in feature extraction while also exemplifying the computational demands of traditional deep CNNs.

These architectures form the pillars upon which this thesis constructs a bridge between the profound depth and complexity of CNNs and the quest for expedited training times through statistical methods. By integrating insights from the VGG16 structure and the empirical evidence provided by the research on ResNet and DenseNet, this study underscores the potential of statistical methods to optimize training processes and reframe how we approach the design of deep learning architectures. It suggests a harmonious blend of architectural ingenuity and statistical understanding, aiming to usher in a new epoch of deep learning. In this realm, models like VGG16 are trained with the swiftness of DenseNet and the depth of ResNet, culminating in a balanced symphony of speed, efficiency, and accuracy.

This synthesis of DenseNet's and ResNet's advancements, along with the classic VGG16 framework, thus positions this thesis as a pivotal narrative in the ongoing evolution of CNNs. It presents a compelling argument for the role of statistical methods in enhancing the training efficiency of deep neural networks, potentially redefining the operational limits of artificial intelligence applications. Through this lens, this thesis reflects on the current state of the art. It casts a forward-looking gaze toward a more accessible and scalable future for CNNs.

3. METHODOLOGIES

3.1. ARCHITECTURES

3.1.1. VGG

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.1.1 - Family of VGG Architectures

As we can see in Figure 3.1.1, the VGG16 architecture, denominated D, is composed of 16 weight layers and has the following configuration:

Input Layer: Images with a 224 x 224 resolution in RGB format.

Convolutional Blocks: The architecture contains several convolutional blocks. Each block consists of multiple convolutional layers followed by a max-pooling layer.

Block 1: Two convolutional layers, each with 64 filters and a kernel size of 3x3. This is denoted as "conv3-64" in the table and is followed by a max-pooling layer to reduce the spatial dimensions.

Block 2: Two convolutional layers, each with 128 filters and a kernel size 3x3, denoted as "conv3-128". Followed by a max-pooling layer.

Block 3: Three convolutional layers: the first two have 256 filters each and the third one also has 256 filters, all with a kernel size of 3x3, represented as "conv3-256". This block concludes with a max-pooling layer.

Block 4: Three convolutional layers, each with 512 filters and a kernel size of 3x3, denoted as "conv3-512". A max-pooling layer is then succeeded.

Block 5: Similar to Block 4, this block has three convolutional layers, each with 512 filters and a kernel size of 3x3, also represented as "conv3-512". Concludes with a max-pooling layer.

Fully Connected (FC) Layers: The architecture features three fully connected layers after the convolutional blocks. The first two FC layers have 4096 neurons each, represented as "FC-4096". The third FC layer has 1000 neurons, represented as "FC-1000". This layer corresponds to the number of classes in the ImageNet dataset, which VGG16 was initially trained on.

Output Layer: A SoftMax layer is used to produce class probabilities based on the 1000 classes from the ImageNet dataset.

3.1.2. DenseNet

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Figure 3.1.2 - Family of DenseNet architectures

By looking at Figure 3.1.2, we can have a perception of the composition of the DenseNet121 architecture, as the name suggests it has 121 weight layers organized in the following manner:

Input Layer: Images with a 224 x 224 resolution in RGB format (not represented in this figure).

Convolution Layer: The architecture starts with a convolutional layer with an output size of 112 x 112.

Pooling Layer: This is followed by a pooling operation which reduces the spatial dimensions by half, resulting in an output size of 56 x 56.

Dense Block (1): Here, we have two convolution operations: a 1x1 convolution followed by a 3x3 convolution. This block of operations is repeated 6 times. The output size remains 56 x 56 because DenseNets don't alter spatial dimensions within dense blocks.

Transition Layer (1): This consists of a 1x1 convolution, followed by a 2x2 average pooling operation with a stride of 2. This will reduce the spatial dimensions by half. The output size after this layer is 28 x 28.

Dense Block (2): Similar to the first dense block, this block also contains a 1x1 convolution followed by a 3x3 convolution. This block of operations is repeated 12 times. The output size remains 28 x 28.

Transition Layer (2): This consists of a 1x1 convolution followed by a 2x2 average pooling operation with a stride of 2. The output size after this layer is 14 x 14.

Dense Block (3): This block also follows the same format of a 1x1 convolution followed by a 3x3 convolution. The operations are repeated 24 times. The output size remains 14 x 14.

Transition Layer (3): This consists of a 1x1 convolution followed by a 2x2 average pooling operation with a stride of 2. The output size becomes 7 x 7 after this layer.

Dense Block (4): This block contains a 1x1 convolution followed by a 3x3 convolution. The operations are repeated 16 times. The output size remains 7 x 7.

Classification Layer: The network concludes with a 7 x 7 global average pooling layer to reduce the spatial dimensions to 1 x 1. This is followed by a 1000D fully connected layer which applies the SoftMax activation function. This layer will output the probabilities for each of the 1000 classes in the dataset.

3.1.3. ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.1.3 - Family of ResNet architectures

Looking at Figure 3.1.3, let's describe the ResNet50 architecture, composed by 50 weight layers:

Input Layer: Traditionally, ResNet models expect an input shape of 224 x 224, which aligns with the output size of 112 x 112 in the subsequent layer after considering the operations in conv1 (not represented in this figure).

conv1: This layer consists of a 7x7 convolution with 64 filters and a stride of 2. This reduces the spatial dimensions by half. Output size: 112 x 112.

conv2_x: It begins with a 1x1 convolution with 64 filters, followed by a 3x3 convolution with 64 filters, and ends with another 1x1 convolution but with 256 filters. This block of operations is repeated 3 times. Note that each block has a residual connection from the beginning to the end. The spatial dimension remains at 56 x 56 after max pooling.

conv3_x: The block starts with a 1x1 convolution with 128 filters, a 3x3 convolution with 128 filters, and concludes with a 1x1 convolution with 512 filters. This block of operations is repeated 4 times. Again, each block has a residual connection. Output size: 28 x 28.

conv4_x: It contains a 1x1 convolution with 256 filters, a 3x3 convolution with 256 filters, and a 1x1 convolution with 1024 filters. This sequence is repeated 6 times. Each block has a residual connection. Output size: 14 x 14.

conv5_x: This block starts with a 1x1 convolution with 512 filters, followed by a 3x3 convolution with 512 filters, and ends with a 1x1 convolution with 2048 filters. The entire sequence is repeated 3 times. Each block has a residual connection. Output size: 7 x 7.

Output Layer: A global average pooling layer reduces the spatial dimensions to 1 x 1. This is followed by a fully connected layer with 1000 units (denoted as 1000-d fc), corresponding to the number of classes in the ImageNet dataset. This layer will produce the classification scores. A SoftMax activation function is applied to these scores to get the final probabilities for each class.

3.2. DATASETS

3.2.1. Cifar-10

The CIFAR-10 dataset is composed of 60,000 color images, each sized 32x32 pixels, and categorized into 10 distinct classes. Each class contributes 6,000 images, split into 50,000 for training purposes and 10,000 designated for testing. This setup is well-suited for conducting the study.

This dataset is segmented into six batches: five for training and one for testing, with each batch comprising 10,000 images. The testing batch is uniquely structured, featuring a balanced selection of 1,000 images from each class, chosen at random. The training batches, while also randomized, may exhibit a varied distribution of images across classes. Collectively, however, they ensure an equal representation of each class, with 5,000 images per class across all training batches. There are 10 classes labeled as: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck as we can see by looking at Figure 3.2.1.

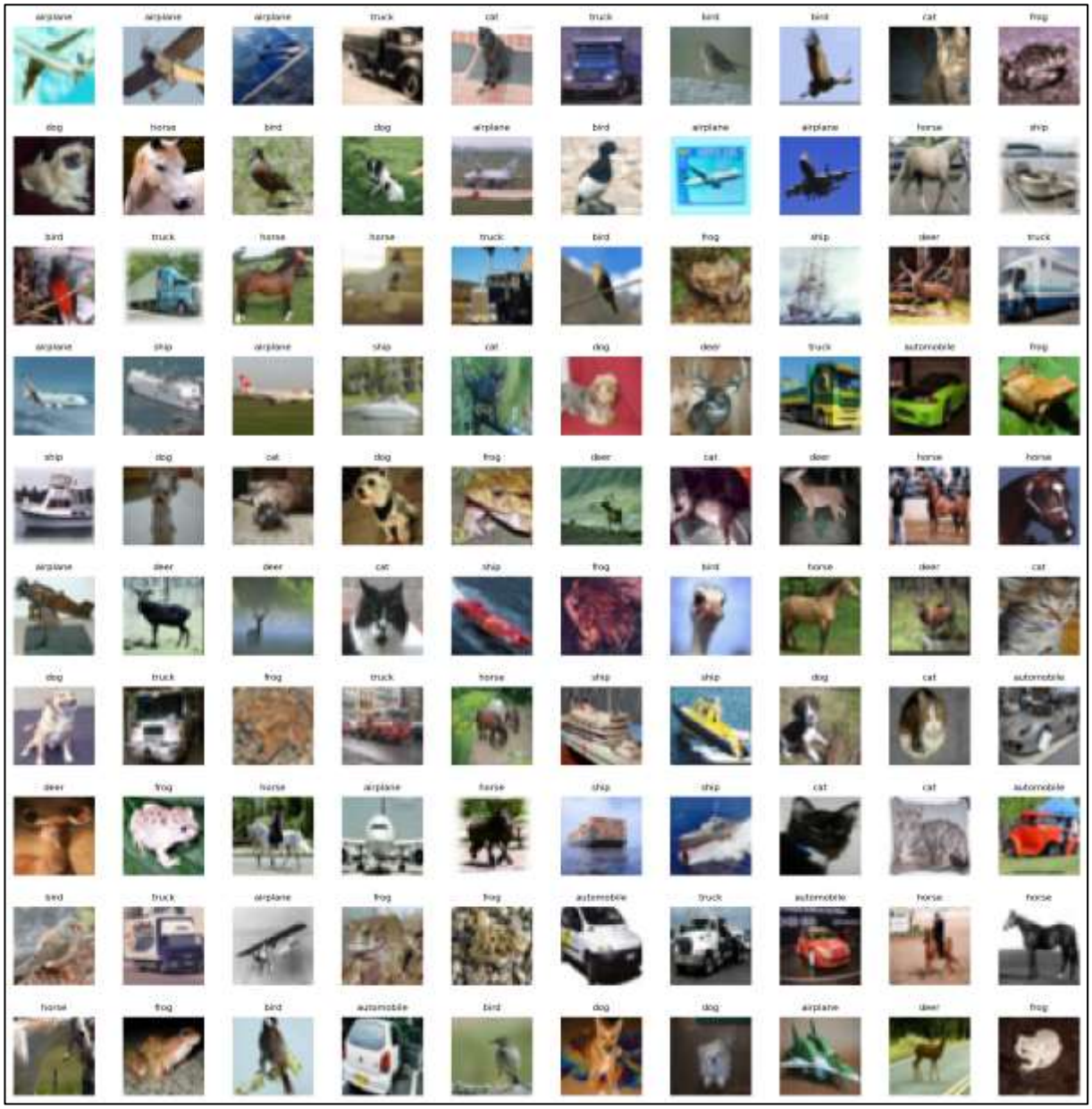


Figure 3.2.1 - Cifar-10 Dataset

3.2.2. SVHN (Street View House Numbers)

SVHN is a practical image dataset tailored for the development of machine learning and object recognition methods with minimal needs for data preprocessing and organization. While it bears similarities to MNIST—like the presence of small, cropped digit images—it offers a much larger set of labeled data, encompassing over 600,000-digit images. What makes SVHN more challenging is its origin; it is derived from recognizing numbers and digits in real-world scenarios, specifically from house numbers present in Google Street View photographs as we can see by looking at Figure 3.1.2.

In a nutshell, the dataset comprises ten categories, each representing a digit. The digit' 1' is labeled as 1, '9' as 9, and '0' as 10. For training purposes, there are 73,257 images 32x32; for testing, there are 26,032 images.

Figure 3.2.2 - SVHN Dataset

3.3. TRADITIONAL TRAINING PROCEDURE FOR VGG16, DENSENET121 AND RESNET50

Configuration of Hyperparameters

A regularization hyperparameter weight decay is set at 1e-4 to ensure model generalization while combating overfitting. Furthermore, a selection of learning rates (0.001, 0.0001, and 0.0004) are tested to discern the most propitious rate for model optimization.

Model Architectural Set-Up

- **Initialization**: Distinct architectures—VGG16, DenseNet121, and ResNet50—are harnessed, each pre-trained on the expansive ImageNet dataset. In consideration of task specificity, the ImageNet-tailored top layers are excluded during model integration.
- **Layer Customization**: To capitalize on the intrinsic feature extraction capabilities embedded within pre-trained models, all layers are set to non-trainable, effectively embracing the philosophy of transfer learning.

Image Rescaling and Adaptation

Given architectural mandates, all input images are upscaled to a canonical resolution of 224x224. This is achieved via:

- **Rescaling:**
  - Up-sampling by a factor of 7 for CIFAR-10 and SVHN datasets.
- **Preprocessing:** The official preprocessing input function, as provisioned by TensorFlow, is judiciously applied for each dataset in congruence with each architecture. This ensures optimal compatibility and preprocessing.
- **Data Augmentation:** A custom preprocessing function is employed, introducing variances into the dataset. This aids in enhancing the model's resilience against overfitting and broadening its exposure to diversified data patterns.

Model Augmentation and Refinement

- **Integration**: The pre-trained architectures are seamlessly integrated post-rescaling.
- **Pooling and Regularization**: A Global Average Pooling layer is introduced, mitigating spatial dimensions. Subsequent dense layers are augmented with L2 regularization, supplemented by dropout layers, thereby buttressing model robustness.

Model Compilation and Metric Selection

The versatile Adam optimizer, known for its adaptive gradient-based optimization techniques, is elected for model optimization. Adam, short for Adaptive Moment Estimation computes adaptive learning rates for each parameter by leveraging both the first moment (the mean) and the second moment (the uncentered variance) of the gradients. Its updated rules can be delineated as follows:

- Compute the gradient of the loss concerning the parameters at the current step: $\boldsymbol{g_t}$.
- Update the moving averages of the gradient (m) and its square (v):
  - $\boldsymbol{m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * g_t}$
  - $\boldsymbol{v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * {g_t}^2}$

Where: $\boldsymbol{\beta_1}$ and $\boldsymbol{\beta_2}$ are exponential decay rates for the moment estimates, typically close to 1.

- Correct the moving averages for their initialization bias:
  - $\boldsymbol{\widehat{m}_t = \frac{m_t}{1-{\beta^t}_1}}$
  - $\boldsymbol{\widehat{v}_t = \frac{m_t}{1-{\beta^t}_2}}$

- Update the parameters:
  - $\boldsymbol{\theta_{t+1} = \theta_t - \alpha * \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}+\epsilon}}}$

Where: $\boldsymbol{\theta}$ represents the parameters**;** $\boldsymbol{\alpha}$ is the learning rate**;** $\boldsymbol{\epsilon}$ is a small number to prevent division by zero, typically on the order of 10^{-7}.

This optimizer is parameterized by specific learning rates and combines the advantages of AdaGrad, which adapts the learning rates based on the full history of gradient, and RMSprop, which uses a moving average of the squared gradient.

Given that the challenge at hand is inherently multi-class, the categorical cross entropy loss function is chosen. This loss function, often used for multi-class classification problems, measures the difference between the true labels and the predicted probabilities. Mathematically, for a single sample, the loss is:

- $L = -\sum_{i=1}^{C} y_i \log(p_i)$

Where: $\boldsymbol{C}$ is the number of classes; $\boldsymbol{y_i}$ is the true label for class $\boldsymbol{i}$ (either 0 or 1); $\boldsymbol{p_i}$ is the predicted probability for class $\boldsymbol{i}$

Essentially, the categorical cross-entropy loss penalizes predictions that are far from the actual labels. For a perfect prediction, the loss is zero, while for an incorrect prediction, the loss approaches infinity. This ensures that the model is guided to make predictions that are as close as possible to the true labels.

Performance surveillance is holistically conducted across four pivotal metrics: accuracy, precision, recall, and loss. Each of these metrics provides unique insights:

- **Accuracy**: Measures the fraction of predictions the model got right.
- **Precision**: Represents the number of true positive predictions divided by the sum of true positives and false positives. It essentially gauges the reliability of a positive prediction.
- **Recall**: Denotes the number of true positive predictions divided by the sum of true positives and false negatives. This captures the ability of the model to identify all relevant instances.
- **Loss**: A quantification of how far the model's predictions are from the actual labels, with the categorical cross entropy serving as the chosen metric in this context.

Together, these metrics ensure comprehensive monitoring of the model's performance, enabling tweaks and refinements for optimal results.

Training Dynamics and Evaluation

- **Duration**: Irrespective of architectural or dataset variances, a consistent training regimen of 30 epochs is adopted.
- **Training Paradigm**: Models are rigorously trained on designated datasets, with meticulous logs maintained to enable retrospective insights.

- **Validation Rigor**: Upon culminating the training cycle, models are subjected to a stringent validation assessment, thereby providing a snapshot of their predictive acumen and generalizability.

Analytical Synthesis and Visualization

- **Performance Panorama**: A comprehensive summary detailing test accuracies are crafted, juxtaposed with the computational training time for each learning rate.
- **Optimal Learning Rate Spotlight**: The learning rate conferring superior accuracy is spotlighted, providing a beacon for model refinement.
- **Visual Narratives**: Graphical plots chronicle the trajectory of model evolution across the chosen metrics—accuracy, precision, recall, and loss—offering a vivid depiction of model performance nuances over epochs.

In summation, by adopting a meticulous lens on accuracy, precision, recall, and loss, a symmetrical and comprehensive evaluation of models is realized. This tapestry of strategies and metrics is instrumental in distilling the essence of model performance and training efficacy, aptly facilitating a nuanced and profound interpretation suitable for scholarly discourse.

3.4. STATISTICS EXTRACTION

All the architectures mentioned can be explained by blocks, meaning that to extract essential statistics such as mean, standard deviation, kurtosis, and skewness, all we have to do is create a model that outputs the last layer for each of the blocks within each architecture, VGG16 DenseNet121 and ResNet50 respectively. Let's try to break down the full process.

Model Initialization: VGG16, DenseNet121, ResNet50

All architectures, pre-trained on ImageNet, are loaded without its classification head. This is because we are interested in capturing intermediate feature maps, not final classification results.

Layer Freezing:

All layers of each of the models are set as non-trainable since our primary intent is to fetch activations, not to retrain the model.

Custom Model for Intermediate Activations:

We create a new model that takes input from each of the models and produces outputs from intermediate layers. Specifically, we are extracting feature maps from the last layer of each convolutional block. For the VGG16, we are extracting these statistics for the following layers (**'block1_pool', 'block2_pool', 'block3_pool', 'block4_pool', 'block5_pool'**) as a result of five blocks. The layers for DenseNet121 are (**'pool1', 'pool2_pool', 'pool3_pool', 'pool4_pool'**), consisting in four blocks, and lastly for the ResNet50 (**'conv2_block3_out', 'conv3_block4_out', 'conv4_block6_out', 'conv5_block3_out**'), constructed by five blocks.

Activation Extraction Process:

An up-sampling factor is defined since all architectures require at least an input shape of 224x224, both the training and the test samples are loaded, and an empty list is initialized to store statistics for each of the architectures.

For each of these activations, we compute statistics (mean, standard deviation, skewness, and kurtosis) for every feature map in every layer. These statistics are collated with the image index, layer index, and feature map index to provide a granular overview of the activation landscape and the respective label. Finally, the list is converted to a panda's data frame.

To go deeper into the statistics extraction process, the gram matrix, with **normalized values**, was also calculated to further expand the data needed for predicting modeling. But what is a gram matrix? Let's have a brief explanation.

(NOTE: **Normalized Values**: In this context, this matrix was extracted with a custom built-in function, with the option to calculate the same automatically, normalizing its values.

Current normalization formula: $\frac{v-v_{max}}{v_{max}-v_{min}}$

Gram Matrix:

A Gram matrix, particularly in the context of deep learning and neural networks, is a mathematical concept used to measure the similarity between vectors, and it plays a crucial role in applications like style transfer in neural style algorithms. Here is a detailed explanation of its composition and use:

Definition: A Gram matrix is a matrix of inner products. In simpler terms, it is a matrix that represents the correlations between different sets of vectors.

Mathematical Formulation:

- Given a set of vectors $V = \{v_1, v_2, \dots, v_n\}$, the Gram matrix G is defined such that each element G_{ij} is the inner product of vectors v_i and v_j.
- Mathematically, $G_{ij} = \langle v_i, v_j \rangle$.

Properties:

- Symmetry: Since the inner product is commutative, the Gram matrix is symmetric, i.e., $G_{ij} = G_{ji}$
- Positive Semidefinite: This means all eigenvalues of the Gram matrix are non-negative.

Usage in Convolutional Neural Networks (CNNs):

In the context of CNNs, a Gram matrix is used to represent the correlations between different feature maps at a particular layer.

Construction:

- Consider a layer in a CNN with NN feature maps, each of size $M \times M$ (for simplicity, assuming square feature maps).
- These feature maps are reshaped or flattened into vectors, resulting in N vectors each of length M^2.
- The Gram matrix is then computed by calculating the inner product of these vectors, resulting in an $N \times M$ matrix.

Capturing Style:

- The Gram matrix is used to capture the style of an image.
- This style is essentially the patterns, textures, and visual elements that are independent of the spatial arrangement.

How It Works:

- By computing the Gram matrix at different layers of a pre-trained CNN (like VGG16), one can capture textural information at various levels of abstraction.
- The Gram matrix at lower layers might capture finer textures, while higher layers might encapsulate more complex arrangements.

Final Elaborations

This data frame is then grouped by the corresponding image index, layer index, and label to obtain one that contains all the images regarding the train or test data set size. Meaning that we end up with a mean for each block per image. Here is an explanation of our data frame size in terms of features:

- **VGG16:** 5 Blocks; 27 features total (Statistic Metrics + Image Index + Gram Matrix for each block + Label).
- **DenseNet121**: 4 Blocks; 22 features total (Statistic Metrics + Image Index + Gram Matrix for each block + Label).
- **ResNet50**: 5 Blocks; 27 features total (Statistic Metrics + Image Index + Gram Matrix for each block + Label).

We have two data sets for Cifar10 and SVHN for each architecture, one for training and another one for testing, to capture the ability of how well our machine learning models perform on unseen data.

3.5. MODEL TRAINING (MACHINE LEARNING)

A standard scaler is applied to all features except the gram matrix ones since these have already been normalized while extracted before proceeding with the model ensemble. Due to the presence of outliers (skewed data) in most of the features, this scaler presents to be the best suited to advance model training and tuning. But what exactly does this scaler do?

Standard Scaler, often implemented as part of data preprocessing, is a technique used to standardize the range of independent variables or features of data. In the context of machine learning and

statistical modeling, feature scaling through standardization (or Z-score normalization) is the process of rescaling the features so that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

Mathematical Formulation

The process of standard scaling can be mathematically represented as follows:

- $Z = \frac{(X-\mu)}{\sigma}$

Where:

- X is the original feature vector,
- μ is the mean of the feature vector,
- σ is the standard deviation of the feature vector,
- Z is the standardized feature vector.

Procedure

- **Compute the Mean and Standard Deviation**: For each feature, calculate the mean and standard deviation. These statistics are used to standardize the data.

- **Standardization**: Subtract the mean from each data point and divide the result by the standard deviation. This centers the distribution of the data around zero and scales it to unit variance.

After this, the correlation matrix as well as some statistics of our data sets are calculated just to get a perception of the data being dealt with. The models being chosen for training are:

- Random Forest
- Logistic Regression
- Support Vector Machines
- ExtraTreeClassifier

These models are known for their fast training and good performance abilities for categorical targets prediction. Let's explain each one to get a good understanding of how they perform.

RandomForest

Overview: RandomForest is an ensemble learning method, primarily used for classification and regression. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

Working Principle:

- **Ensemble of Decision Trees:** It builds multiple decision trees and merges them to get a more accurate and stable prediction.

- **Randomness:** Each tree in a random forest is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set. Furthermore, when splitting a node during the

construction of the tree, the split that is chosen is no longer the best split among all features but the best split among a random subset of the features. This results in a wide diversity that generally results in a better model.

Characteristics:

- **Robustness:** It can handle large datasets with higher dimensionality. It can handle thousands of input variables without variable deletion.
- **Accuracy:** RandomForest is known for its high accuracy, robustness, and ease of use.
- **Handling Overfitting:** Due to the randomness in the construction of trees, it is less likely to overfit than a single decision tree.

Support Vector Machines (SVM)

Overview: SVM is a powerful and versatile supervised machine learning algorithm, used for both classification and regression.

Working Principle:

- **Maximizing Margin:** SVM essentially finds a hyperplane that best separates the classes by maximizing the margin between the nearest points of the classes (support vectors).
- **Kernel Trick:** It can use the kernel trick to transform data and then based on these transformations, it finds an optimal boundary between the possible outputs.

Characteristics:

- **Effectiveness in High-Dimensional Spaces:** SVMs are effective in high-dimensional spaces and in cases where the number of dimensions exceeds the number of samples.
- **Memory Efficiency:** They use a subset of training points (support vectors), so it's also memory efficient.
- **Sensitive to Noise:** A relatively small number of mislabeled examples can dramatically decrease the performance.

Logistic Regression

Overview: Despite its name, logistic regression is a linear classification model rather than regression. It is used to predict the probability of a binary outcome based on one or more predictor variables.

Working Principle:

- **Sigmoid Function:** It uses the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

Characteristics:

- **Efficient and Simple:** It is less prone to overfit but can overfit in high dimensional datasets.

- **Output Interpretation:** The output of a logistic regression model is interpretable in terms of probability.

ExtraTreeClassifier

Overview: Extra Trees Classifier is a type of ensemble learning technique that aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result.

Working Principle:

- **Randomness:** Unlike Random Forest, it cuts trees at random levels. Each decision tree in the ExtraTreeClassifier is constructed from the original training sample. Then, at each test node, each tree is provided with a random sample of k features from the feature set, from which each decision tree must select the best feature to split the data based on some mathematical criteria (typically the Gini Index or information gain).

Characteristics:

- **Reduced Variance:** By averaging out bias, the ExtraTreeClassifier tends to reduce overfitting.
- **Computational Efficiency:** It is typically faster than a Random Forest.

Model Training Process

Data Utilization Strategy

- **Training with Full Dataset:** The entire dataset is utilized for training, aiming to maximize the learning potential.
- **Separate Dataset for Testing:** A distinct dataset is used for testing, providing an accurate measure of the model's performance on unseen data.

Hyperparameter Tuning with RandomizedSearchCV and Cross-Validation

RandomizedSearchCV is employed for hyperparameter tuning across different models, coupled with five-fold cross-validation. This approach ensures a robust and comprehensive tuning process by testing various parameter combinations and validating their performance across multiple subsets of the data.

Random Forest

- **Parameters**: n estimators = [100, 500, 1000, 1500], max depth = [None, 5, 10, 20, 30], max features = ['auto', 'sqrt', 'log2'], min samples split = [2, 5, 10], min samples leaf = [1, 2, 4], and bootstrap = [True, False].
- **Datasets:** Applied to both CIFAR-10 and SVHN datasets.
- **Feature Importance:** Utilized to identify key features.

SVM (Support Vector Machine)

- **Parameters**: C = [0.1, 1, 10, 100], kernel = ['linear', 'rbf', 'poly', 'sigmoid'], degree = [2, 3, 4], gamma = ['scale', 'auto'], and coef0 = [0.0, 0.1, 1.0, 2.0].
- **Datasets:** Exclusively used for the CIFAR-10 dataset.
- **Feature Analysis:** SHAP plots are employed for feature impact analysis.

Logistic Regression

- **Parameters:** Three different configurations:

  First: C = [0.001, 0.01, 0.1, 1, 10, 100], penalty = l1, and solver = [liblinear, saga]

  Second: C = [0.001, 0.01, 0.1, 1, 10, 100], penalty = l2, and solver = [newton-cg, lbfgs, sag, saga, liblinear]

  Third: C = [0.001, 0.01, 0.1, 1, 10, 100], penalty = elastic net, solver = saga, and l1 ratio = [0.2, 0.5, 0.8]

  (**NOTE**: Since different penalties only work with certain solvers three configurations were set.)

- **Datasets:** Applied to both CIFAR-10 and SVHN datasets.
- **Feature Analysis:** SHAP plots are employed for feature impact analysis.

ExtraTreeClassifier

- **Parameters:** n estimators = [100, 200, 500], max depth = [None, 10, 20, 30], min samples split = [2, 5, 10], min samples leaf = [1, 2, 4], and bootstrap = [True, False].
- **Datasets:** Used specifically for the SVHN dataset.
- **Feature Importance:** Utilized to identify key features.

Evaluation Metrics

The evaluation involves a comprehensive classification report, providing metrics like precision, recall, F1-score, and test accuracy, measuring the model's performance on the separate testing dataset.

Conclusion

This methodology is designed to thoroughly evaluate and optimize various machine learning models on CIFAR-10 and SVHN datasets. By employing RandomizedSearchCV with five-fold cross-validation, the approach not only targets optimal hyperparameter settings but also ensures that the models are well-tested across different data splits, enhancing their reliability and generalizability. The distinction in model application between the two datasets (SVM for CIFAR-10, ExtraTreeClassifier for SVHN) aligns with the unique characteristics of each dataset, ensuring a tailored approach to the model training.

3.6. COMPARING RESULTS

In the final chapter of the thesis, an extensive and detailed comparison is conducted between the methods mentioned above: traditional deep learning architectures (VGG16, DenseNet121, and ResNet50) and conventional machine learning models (Random Forest, SVM, Logistic Regression, and ExtraTreeClassifier). This comparison is meticulously designed to evaluate these methods under uniform experimental conditions, ensuring a balanced and objective analysis.

For the deep learning models, the evaluation utilizes both line plots and loss function graphs. The line plots visually represent the models' performance in terms of precision, recall, and accuracy, providing an intuitive way to observe trends and patterns over various training epochs. Additionally, plots depicting the loss function are crucial as they offer insights into the optimization process of the models, illustrating how effectively they minimize errors during training. This dual-graphical approach enables a comprehensive understanding of the learning dynamics and overall performance of the deep learning models.

In contrast, the evaluation of the machine learning models relies on classification reports. These reports provide a detailed breakdown of performance metrics such as precision, recall, accuracy, and the F1 score in a tabular format. This approach allows for a direct, straightforward comparison of these key metrics across the different machine-learning models, focusing on their classification efficacy.

By employing these distinct evaluation tools, graphical representations for deep learning models, and classification reports for machine learning models, this thesis offers a robust comparative analysis. This approach not only highlights the strengths and weaknesses of each model type in various aspects of their performance but also caters to the unique characteristics and evaluation needs of each method. Through this detailed comparative study, the thesis aims to provide a nuanced understanding of these methodologies, aiding in informed decision-making in the field of artificial intelligence and machine learning.

Check Table 3.6.1, for a detailed explanation of how the performance will be reported for both methods.

Table 3.6.1 – Explaining the Results

Aspect	*Deep Learning Models (VGG16, DenseNet121, ResNet50) with pre-frozen trained weights 'ImageNet' using transfer learning*	*Machine Learning Models (Random Forest, SVM, Logistic Regression, ExtraTreeClassifier)*
Primary Evaluation Tool	Line plots and loss function graphs	Classification Report
Metrics Represented	Precision, recall, accuracy, and model loss	Precision, Recall, Accuracy, and F1-score
Visualization Method	Graphical - illustrating trends over training epochs and model optimization	Tabular - providing a clear, direct comparison of key metrics
Intention of Evaluation	To understand learning dynamics, performance trends over time, and effectiveness in error minimization	To assess classification efficacy and provide a straightforward comparison of model performance
Insight Offered	Offers insights into how models evolve and improve during training, and how effectively they minimize errors	Highlights the immediate classification performance of each model in a comparative context
Use in Thesis	To demonstrate the depth and complexity of learning in deep neural networks, focusing on both performance and training optimization	To show the straightforward efficacy in classification tasks, emphasizing direct performance metrics
Contextual Relevance	Suitable for complex tasks with high-dimensional data, where understanding the learning process is crucial	Appropriate for scenarios where model simplicity and direct performance comparison are key

4. RESULTS AND DISCUSSION

4.1. TRADITIONAL TRAINING METHOD (TRANSFER LEARNING) USING FROZEN PRE-TRAINED WEIGHTS 'IMAGENET'

4.1.1. CIFAR-10

VGG16

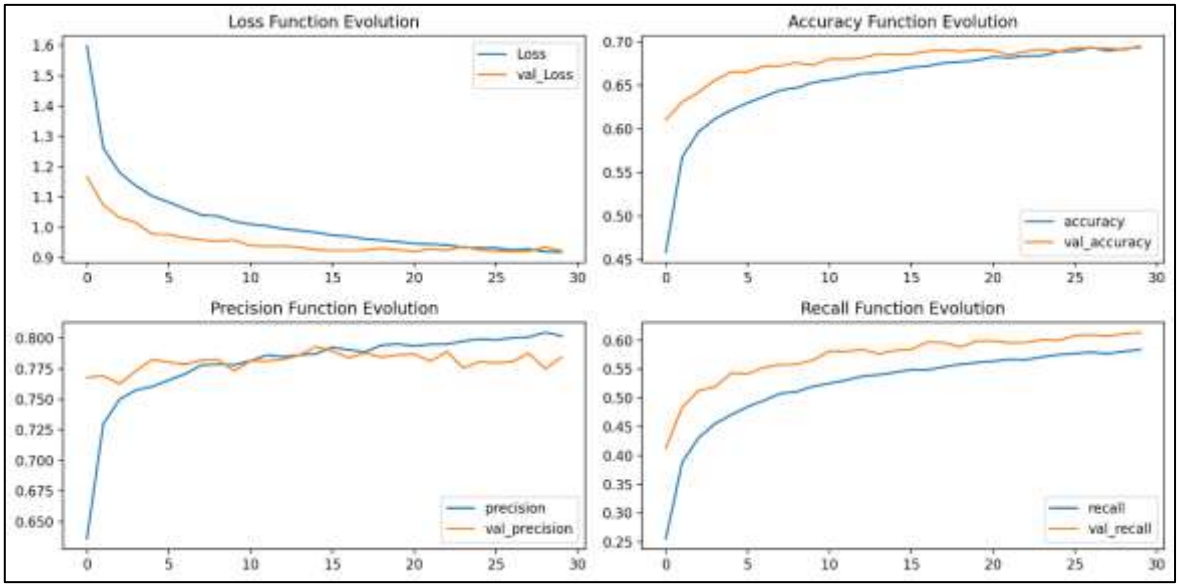


Figure 4.1.1 - Loss, Accuracy, Precision and Recall functions Cifar-10 VGG16

Loss Function Evolution:

The left-top graph shows the loss function values for both the training set (blue) and the validation set (orange) over the epochs. The loss function measures the model's prediction error, and the goal is to minimize it. As we can see, the loss decreases sharply initially and then levels off, indicating that the model is learning and becoming better at predicting the correct classes of images. The convergence of training and validation loss suggests that the model is not overfitting and is generalizing well to new data.

Accuracy Function Evolution:

The right-top graph displays the accuracy of the model, which is the percentage of correct predictions out of all predictions made. Both the accuracy of the training data and the validation data increase over time, which is desirable. The accuracy stabilizes after an initial period of rapid improvement, showing that the model has learned to classify the images with a good level of reliability.

Precision Function Evolution:

The left-bottom graph represents the precision of the model, which is the ratio of true positive predictions to the total positive predictions made by the model. High precision indicates a low rate of false positives. The graph shows that precision for both training (blue) and validation (orange)

improves as the epochs progress, implying the model is consistently making more accurate positive predictions as it learns.

Recall Function Evolution:

The right-bottom graph shows the recall, which measures the ratio of true positive predictions to the total actual positives. High recall means that the model is good at detecting positives. Both training and validation recall improve over time, indicating the model is getting better at identifying all relevant instances in the dataset.

Interpreting the Results:

The results show a well-behaved model learning process. The loss decreasing and the accuracy, precision, and recall increasing are all indicators of a model that is learning effectively.

The close alignment between the training and validation lines in each graph suggests that the model is not just memorizing the training data but is also performing well on unseen data.

The fact that all metrics show improvement and stabilization after an initial period suggests that the chosen number of epochs is adequate for training this model on this dataset with the given hyperparameters.

The model's best learning rate, as determined by the final accuracy, was 0.0004, which provided the highest accuracy 69.49% compared to the other tested rates and it took approximately 39.45 minutes to train.

DenseNet121

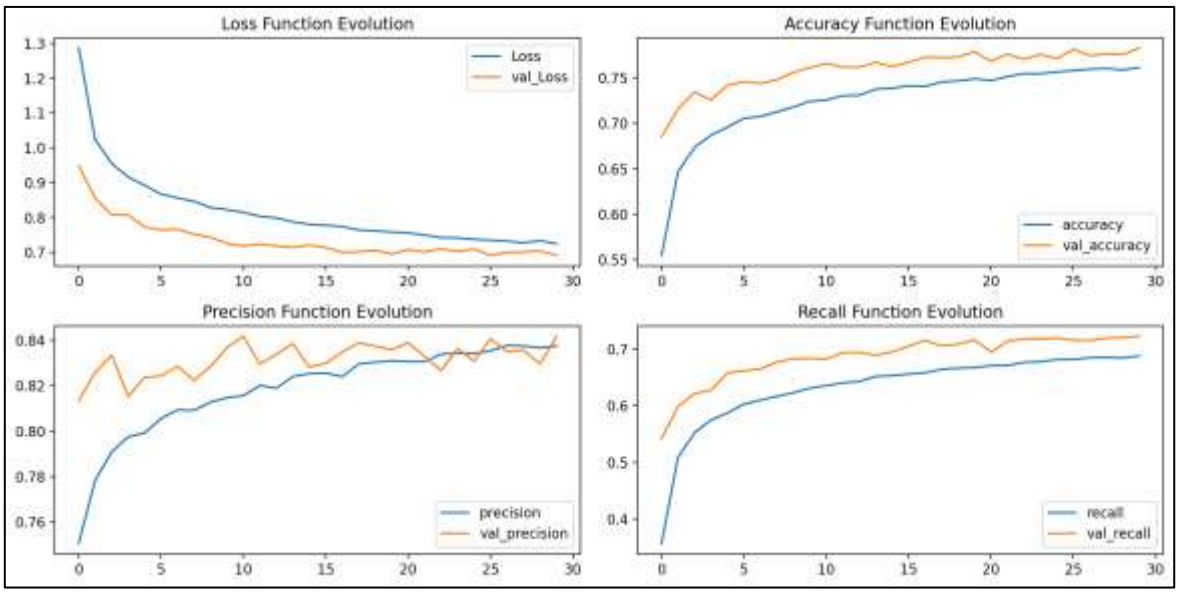


Figure 4.1.2 - Loss, Accuracy, Precision and Recall functions Cifar-10 DenseNet121

Loss Function Evolution:

The loss function graph exhibits a declining trend for both the training and validation sets, suggesting that the model's predictions are becoming more accurate over time. The convergence of the training and validation loss lines indicates that the model is generalizing well without overfitting.

Accuracy Function Evolution:

The accuracy graph shows a progressive improvement in the model's performance, with both training and validation accuracies increasing over epochs. The steady rise followed by a plateau suggests that the model is achieving a consistent level of reliability in its predictions.

Precision Function Evolution:

The precision graph reflects an upward trend, signifying that the model is getting better at making correct positive predictions as it is trained more. This indicates a decrease in false positives, which is beneficial for the overall performance of the model.

Recall Function Evolution:

The recall graph also shows improvement over time for both training and validation, meaning that the model is becoming more adept at correctly identifying all relevant instances of the target classes in the dataset.

Interpreting the Results:

The DenseNet121 model's training process is indicative of a well-tuned learning algorithm. The downward trend in the loss values along with upward trends in accuracy, precision, and recall across training epochs is evidence of the model's effectiveness in learning from the CIFAR-10 dataset.

A notable observation is the convergence of training and validation curves in all graphs. This pattern suggests that the model is capable of generalizing its learning to new, unseen data, rather than merely memorizing the training dataset.

The metrics demonstrate not only improvement but also a leveling-off, indicating that the model is reaching a point of stability. This suggests that the number of epochs selected was appropriate for the model to learn the dataset within the scope of the current hyperparameters.

For DenseNet121, the most effective learning rate was found to be 0.0004. At this rate, the model achieved a commendable test accuracy of 78.29% and completed its training in approximately 33.48 minutes, showcasing both accuracy and time efficiency.

ResNet50

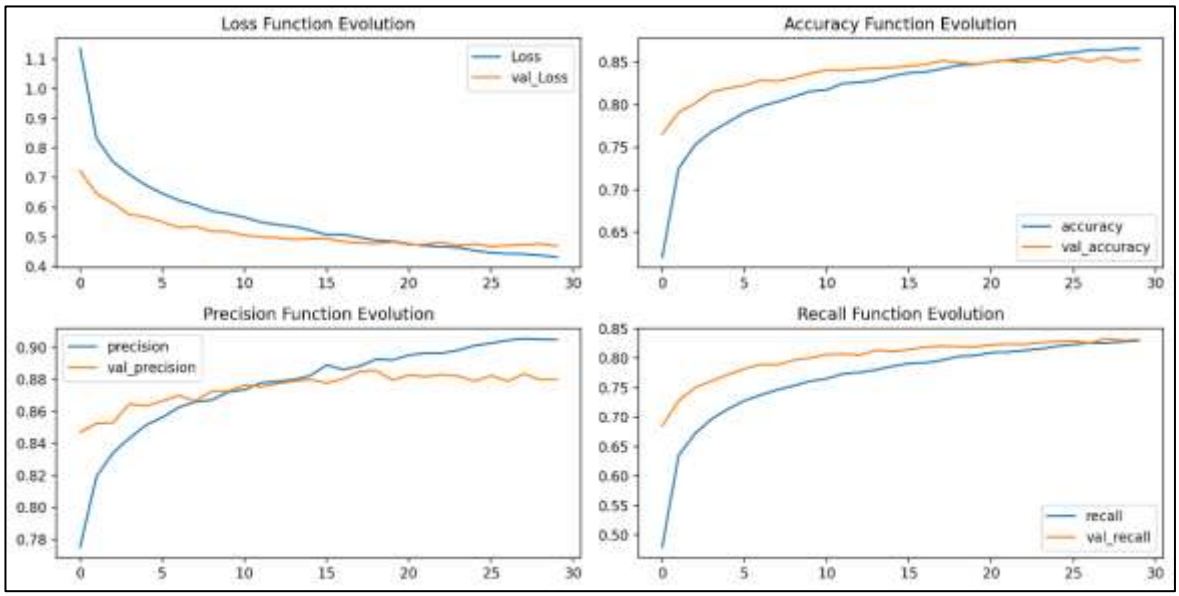


Figure 4.1.3 - Loss, Accuracy, Precision and Recall functions Cifar-10 ResNet50

Loss Function Evolution:

The descending trajectory of the loss graph for both training and validation sets indicates that the model is effectively learning and improving its predictive accuracy. A gradual flattening of the curves suggests that the ResNet50 model is nearing an optimal point of error reduction.

Accuracy Function Evolution:

The accuracy graph displays an incremental increase in both training and validation accuracy, which is indicative of the model's improved performance in correctly classifying the images. The converging lines point to a model that is consistent across both seen and unseen data.

Precision Function Evolution:

The precision graph shows the model's positive predictive value. The rising trend for both training and validation precision signifies that the ResNet50 is making more accurate positive predictions as training progresses, and the false positive rate is decreasing.

Recall Function Evolution:

In the recall graph, both training and validation lines ascend over time, indicating that the model is becoming better at identifying true positives out of all actual positive instances. This is crucial for ensuring that the model does not miss any potential positive classifications.

Interpreting the Results:

The ResNet50 model presents a learning curve that is indicative of an efficient learning process. The decreasing loss along with increasing accuracy, precision, and recall reflects a model that is progressively learning and improving its predictions on the CIFAR-10 dataset.

The close tracking between the training and validation metrics across the plotted graphs indicates a robust model that generalizes well. This balance is crucial for confirming that the model performs consistently on both training and validation datasets.

The stabilization of the metrics after an initial phase of learning underlines that the number of epochs was suitably chosen, allowing the model to fully adapt to the dataset with the provided hyperparameters.

The optimal learning rate for the ResNet50 was determined to be 0.0001. Utilizing this learning rate, the model attained its highest test accuracy of 85.22%, completing the training in an efficient 35.34 minutes, thus demonstrating the model's swift adaptability and high predictive accuracy.

Each model demonstrates its capability in learning from the CIFAR-10 dataset, with ResNet50 slightly edging out in terms of the highest accuracy. The learning trends and performance metrics suggest that all models are well-suited for image classification tasks, with effective learning strategies and good generalization capabilities.

4.1.2. SVHN

VGG16

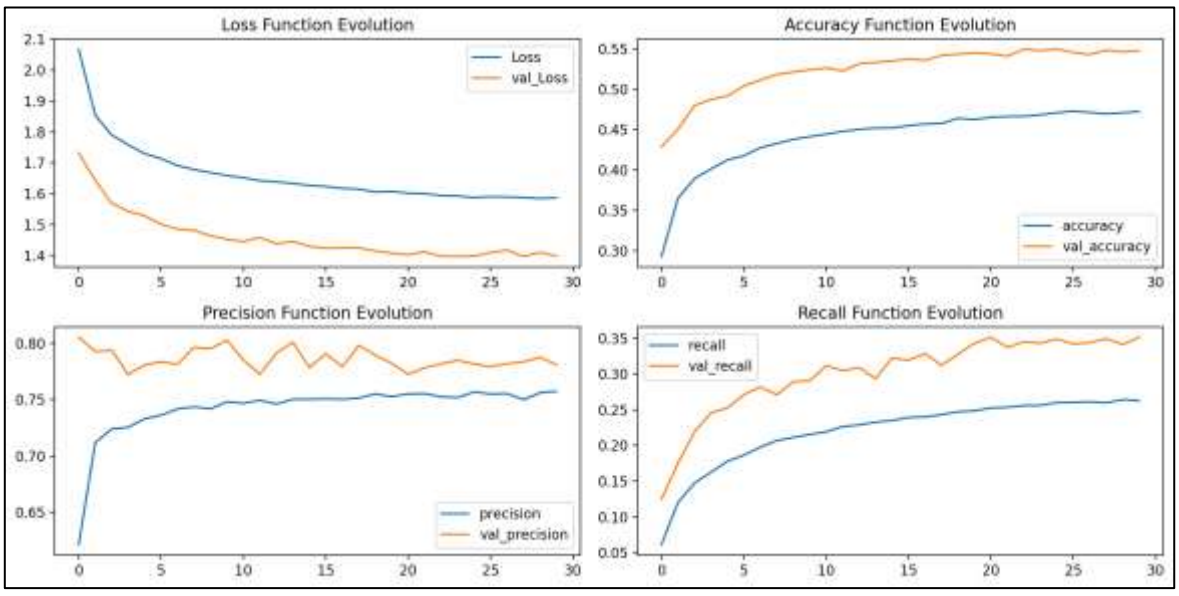


Figure 4.1.4 - Loss, Accuracy, Precision and Recall functions SVHN VGG16

Loss Function Evolution:

The loss for both training (blue) and validation (orange) decreases over epochs, suggesting that the model's predictions are becoming more accurate over time. The curves are converging, which is a good sign that the model is not overfitting significantly.

Accuracy Function Evolution:

The accuracy for training and validation both increase over epochs. However, the training accuracy surpasses the validation accuracy by a noticeable margin, which is typical in training scenarios.

Precision Function Evolution:

Precision for both training and validation starts to plateau after an initial increase. This indicates that the rate at which the model is making correct predictions for the positive class has stabilized.

Recall Function Evolution:

Recall is increasing for both training and validation, with the training recall being consistently higher. This suggests the model is becoming more adept at identifying all relevant instances over time.

Interpreting the Results:

The best learning rate for this VGG16 model on the SVHN dataset is 0.0004, with a corresponding test accuracy of 54.77%.

The training time is approximately 56 minutes across different learning rates, indicating that the computational demand is consistent regardless of the learning rate adjustment within this explored range.

The proximity of the training and validation metrics suggests that the model is generalizing adequately to new data.

DenseNet121

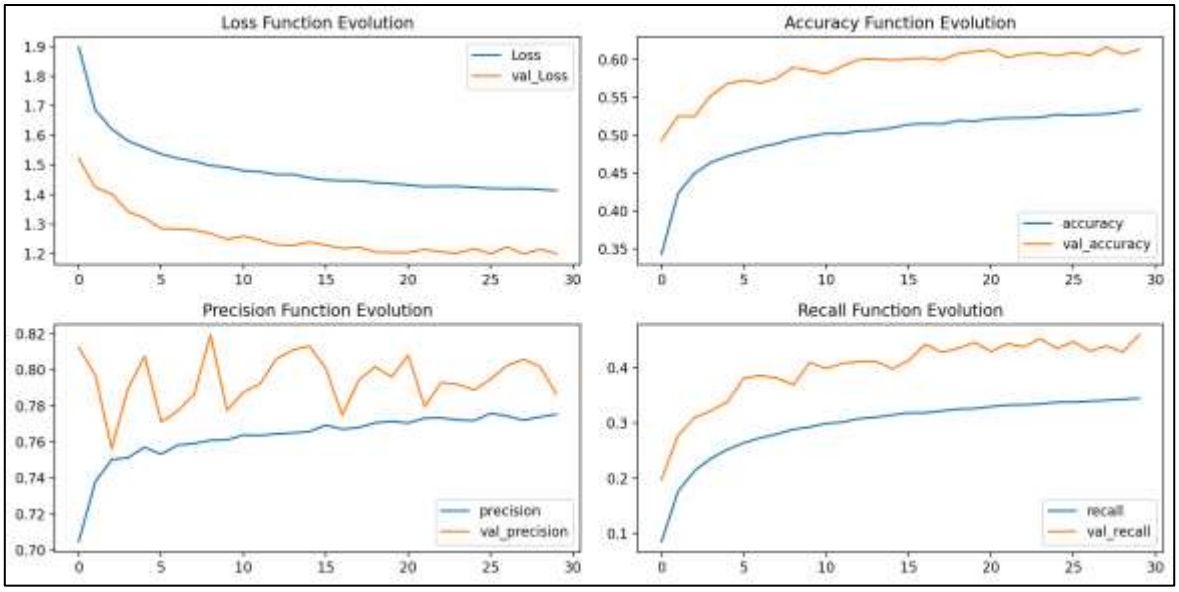


Figure 4.1.5 - Figure 4.1.4 - Loss, Accuracy, Precision and Recall functions SVHN DenseNet121

Loss Function Evolution:

The loss function graph shows a steady decline for both the training (blue) and validation (orange) loss over epochs. This indicates that the DenseNet121 model is consistently learning and improving its ability to predict accurately. The validation loss appears to plateau towards the end of the epochs, which could suggest that the model is beginning to reach its performance limit given the current data and training regime.

Accuracy Function Evolution:

The accuracy graph indicates a continuous increase in training accuracy and a slightly less steep but also increasing trend in validation accuracy. The persistent gap between the training and validation accuracy suggests some degree of overfitting, although the validation accuracy is still improving.

Precision Function Evolution:

The precision metric shows fluctuations but overall maintains an upward trend for both training and validation. The fluctuations could be due to the model's performance on different sets of data or changes in the distribution of the classes within batches.

Recall Function Evolution:

Recall also displays an upward trend for both training and validation, with training recall showing more consistent growth. This suggests the model is becoming better at identifying true positives as training progresses.

Interpreting the Results:

The best test accuracy is obtained at a learning rate of 0.0004, with a corresponding test accuracy of 61.31%. This indicates that at this learning rate, the DenseNet121 model is most effective at generalizing from the training data to the test data.

The training times for the model vary slightly with different learning rates but are around 44 to 46 minutes, the best learning rate took 44.89 minutes. This consistency suggests that the training time is not significantly affected by the learning rate within this tested range.

The graphs show that the model is learning and improving its predictions as indicated by the trends in loss, accuracy, precision, and recall. The consistency of the training time across different learning rates suggests that the model's computational efficiency is stable for this parameter range.

ResNet50

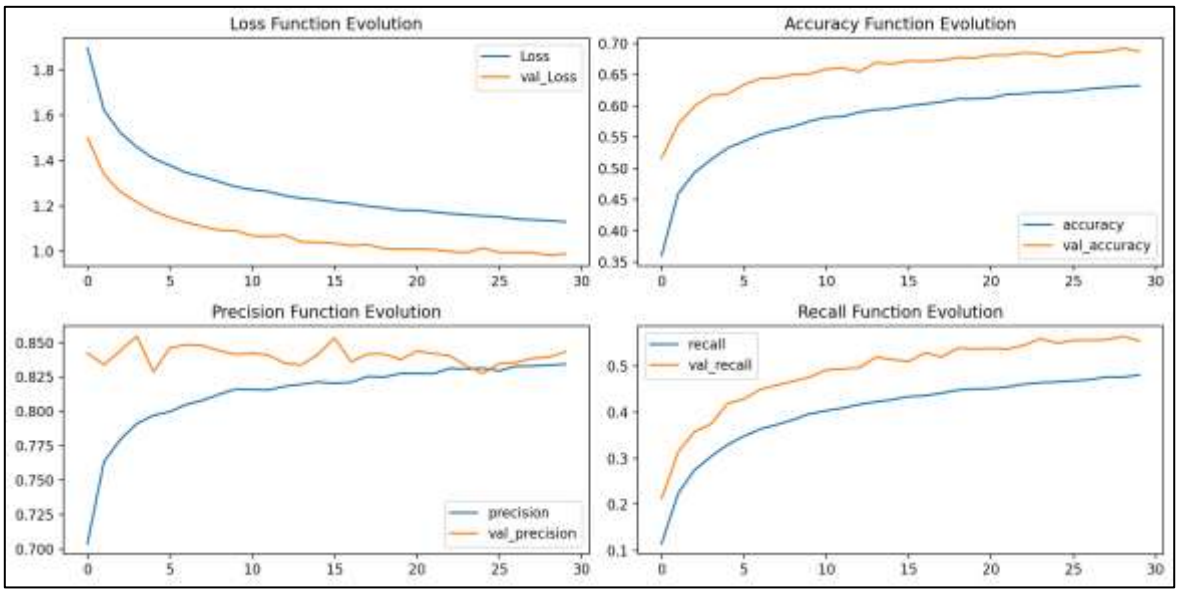


Figure 4.1.6 - Loss, Accuracy, Precision and Recall functions SVHN ResNet50

Loss Function Evolution:

The training loss (blue line) and validation loss (orange line) both show a significant decline, with the training loss leveling off as epochs increase. The validation loss demonstrates a similar trend but with a slight uptick towards the end, which could indicate the beginning of overfitting or instability in the validation loss reduction.

Accuracy Function Evolution:

Both the training accuracy and validation accuracy are on an upward trend, with the validation accuracy approaching the training accuracy as the number of epochs increases. This convergence suggests a good generalization of the model to new data.

Precision Function Evolution:

The precision graph displays an increase in both the training and validation precision initially, with both metrics reaching a plateau. This suggests that the model is maintaining a consistent rate of correct positive predictions as training progresses.

Recall Function Evolution:

The recall graph shows a steady increase in training recall and validation recall. The training recall is consistently higher, but the validation recall is catching up, which indicates that the model is improving its ability to identify all relevant instances.

Interpreting the Results for ResNet50:

The best learning rate for the ResNet50 model on this dataset is 0.0001, with a corresponding test accuracy of 68.72%. This learning rate has allowed the model to achieve the highest level of accuracy on the test data, reflecting its ability to generalize well.

The training times show slight variation, with the training time at the best learning rate being 48.62 minutes. This consistency suggests that the training time is not highly sensitive to changes in the learning rate within this range.

The evolution of the loss, accuracy, precision, and recall metrics suggests that the model has learned effectively and is generalizing well to unseen data. The proximity of the training and validation metrics towards the end of training is a positive sign of the model's performance stability.

4.2. MACHINE LEARNING METHOD

4.2.1. CIFAR-10

4.2.1.1. VGG16

Random Forest

Figure 4.2.1 - Random Forest Perfomance Cifar-10 VGG16

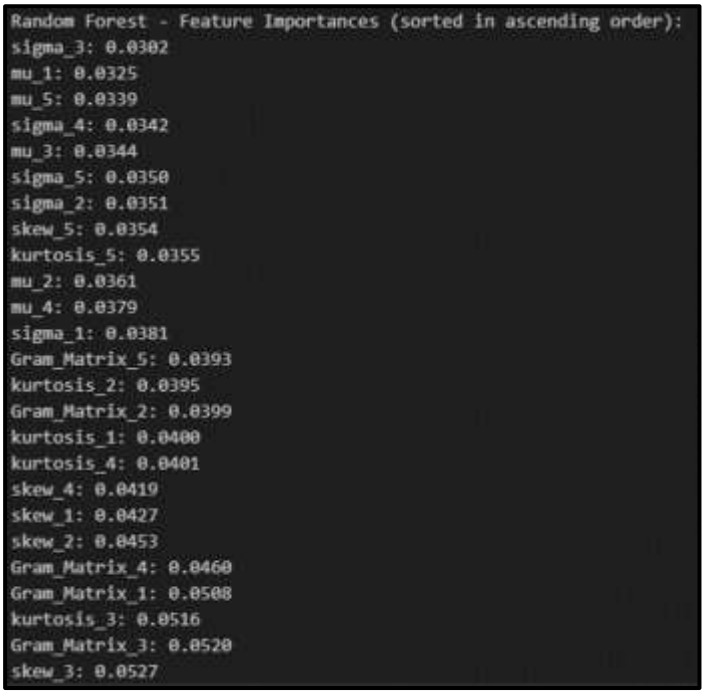


Figure 4.2.2 – Feature Importance Random Forest Cifar-10 VGG16

Training Time

The training process of the Random Forest model was rather fast, taking approximately 8.67 minutes, this duration underscores the ability of this algorithm to adjust to large datasets.

Model Performance Metrics

Precision, Recall, and F1-Score: The model demonstrated varying levels of precision and recall across the different classes (ranging from 0 to 9), with f1 scores accordingly reflecting these variations. For instance, class 6 exhibited relatively high precision and recall, resulting in a higher f1-score compared to class 3, which showed lower values in these metrics. This variation in performance across different classes indicates the model's differing ability to accurately identify each class, which could be attributed to the inherent characteristics of the data or the distribution of features within each class.

Overall Accuracy: The model achieved an accuracy of 42%, which is considerable due to its fast training time.

Optimal Parameters of the Random Forest Model

In addition to the model's performance metrics, it's crucial to discuss the optimal set of hyperparameters that were determined for the Random Forest classifier. These parameters play a significant role in the model's ability to learn from the data and make accurate predictions.

- **Number of Estimators** (n_estimators): The model was optimized with 1,500 trees in the forest. A higher number of trees can improve the model's accuracy and robustness, as it averages more decision trees, reducing the risk of overfitting.
- **Maximum Depth of Trees** (max_depth): The maximum depth was set to 30. This parameter specifies the maximum depth of each tree in the forest. A deeper tree can model more complex patterns but also increases the risk of overfitting.

- **Minimum Samples Split** (min_samples_split): The criterion for further splitting an internal node was set to require at least 5 samples. This helps in providing a balance between bias and variance, making the model generalizable to new data.
- **Minimum Samples Leaf** (min_samples_leaf): The minimum number of samples required to be at a leaf node was set to 1, allowing the tree to grow with maximum granularity.
- **Maximum Features** (max_features): The number of features to consider when looking for the best split was set to 'log2'. This choice impacts the diversity of the trees in the model, where 'log2' means that the logarithm base 2 of the total number of features is used.
- **Bootstrap** (bootstrap): The model was set with bootstrap as False, meaning the whole dataset is used to build each tree. While bootstrapping generally adds randomness and improves model accuracy, not using it can lead to reduced overfitting in certain scenarios.

These parameters were carefully selected to optimize the performance of the Random Forest classifier on our dataset. They represent a specific configuration that balances the model's ability to learn complex patterns in the data while avoiding overfitting, ensuring that the model is both accurate and generalizable.

Feature Importance

The Random Forest model's feature importances, Figure 4.2.2, provide critical insights into the predictive power of each feature within the model. These importances, listed in ascending order, indicate the mean decrease in impurity that each feature contributes to the model:

Dominant Features: The analysis highlights skew_3, Gram_Matrix_3, and kurtosis_3 as the most impactful features, with importance scores of 0.0527, 0.0520, and 0.0516 respectively. Their high values denote a significant influence on the model's prediction ability, signifying that these features hold substantial predictive power within the dataset.

Supporting Features: The detailed importance scores suggest that the impact of these features varies across different classes, reflecting the model's complex interaction with the dataset. For instance, sigma_2 and sigma_5, with scores of 0.0351 and 0.0350, may have more subtle yet targeted effects on specific classes within the model.

Conclusion

The Random Forest model, informed by a strategic selection of hyperparameters, has provided a detailed insight into the characteristics of our dataset. With an overall accuracy of 42%, the model has demonstrated its capability to effectively discern patterns and classify data within the given multi-classification context.

Support Vector Machine (SVM)

```
SVM - The best parameters are: {'kernel': 'poly', 'gamma': 'scale', 'degree': 3, 'coef0': 1.0, 'C': 10}
SVM - Training Time: 18.4244061311086 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.56      0.49      0.53      1000
           1       0.47      0.58      0.52      1000
           2       0.42      0.30      0.35      1000
           3       0.33      0.30      0.32      1000
           4       0.46      0.47      0.46      1000
           5       0.40      0.41      0.41      1000
           6       0.59      0.60      0.59      1000
           7       0.40      0.43      0.41      1000
           8       0.52      0.57      0.54      1000
           9       0.43      0.47      0.45      1000

    accuracy                           0.46     10000
   macro avg       0.46      0.46      0.46     10000
weighted avg       0.46      0.46      0.46     10000

SVM - Test Accuracy: 0.46
```

Figure 4.2.3 - SVM Performance Cifar-10 VGG16

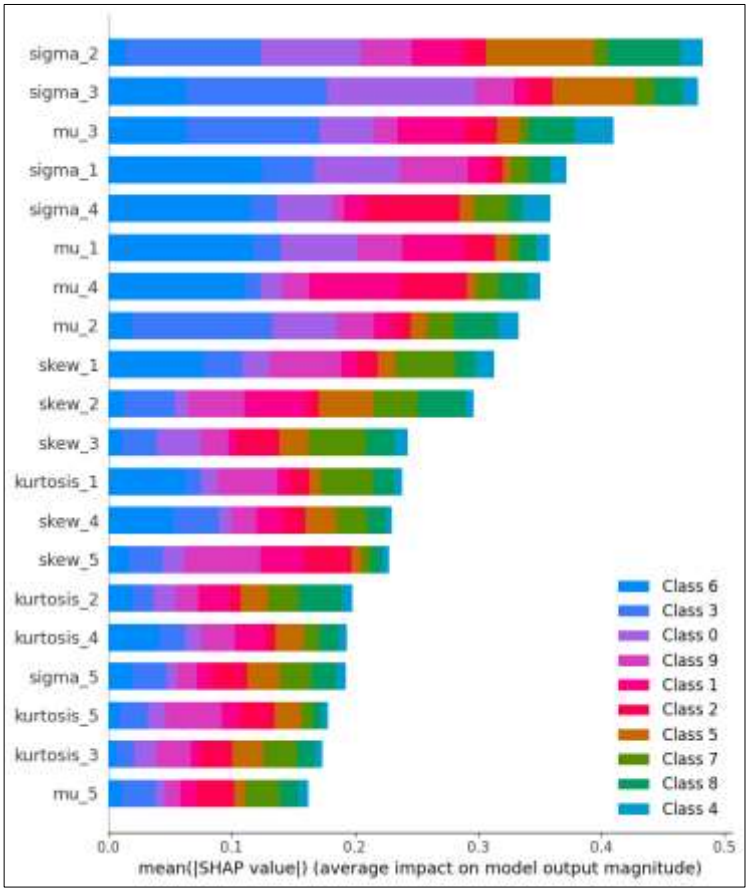


Figure 4.2.4 - Feature Importance SVM Cifar-10 VGG16

Training Time

The training of the SVM model was a substantial endeavor, consuming approximately 18.42 minutes. This duration indicates the complexity involved in training SVMs, particularly when fine-tuning parameters for optimal performance on large datasets.

Model Performance Metrics

Precision, Recall, and F1-Score: The SVM model displayed varying degrees of precision and recall across the different classes, with corresponding f1 scores reflecting these variations. For example, class 6 showed relatively high precision and recall, culminating in a noteworthy f1 score. In contrast, class 3 exhibited lower values in these metrics. Such discrepancies highlight the model's diverse effectiveness in accurately classifying each class.

Overall Accuracy: The SVM achieved an accuracy of 46%. This metric is indicative of the model's general effectiveness in classifying the dataset.

Optimal Parameters of the SVM Model

The optimal set of hyperparameters for the SVM classifier was identified as follows:

- **Kernel Type** (kernel): 'poly' (polynomial), facilitating the model's ability to capture more complex relationships in the data.
- **Gamma** (gamma): 'scale', which automatically adjusts the gamma parameter based on the number of features, offering a balanced approach to handling feature scale.
- **Degree of the Polynomial Kernel Function** (degree): 3, which determines the flexibility of the decision boundary.
- **Independent Term in Kernel Function** (coef0): 1.0, influencing the high-degree feature space in the polynomial kernel.
- **Regularization Parameter** (C): 10, controlling the trade-off between achieving a low training error and a low testing error, hence managing the model's complexity and generalization capability.

These parameters were meticulously selected to enhance the SVM's learning from the dataset while maintaining a balance between bias and variance.

Feature Importance

The bar chart in Figure 4.2.4 illustrates the mean absolute SHAP values for each feature across different classes. Each bar's length represents the average impact on the model output magnitude, allowing us to discern which features have the most substantial influence on predictions. Notably, mu_5, kurtosis_3, and sigma_5 stand out with the highest mean absolute SHAP values, suggesting they are pivotal in the model's predictions across several classes.

Dominant Features: Features like mu_5 (mean of the fifth distribution) and kurtosis_3 (measure of the tailedness of the third distribution) exhibit the most significant impact on the model's output, with high mean SHAP values. These features likely capture critical patterns in the dataset that are strongly associated with certain classes.

Supporting Features: Different classes are affected by different features to varying degrees. For example, sigma_2 is highly influential for Class 6, while skew_2 (a measure of the asymmetry of the second distribution) appears more impactful for Class 8. This indicates the nuanced way in which the model discerns between classes.

Conclusion

The SVM model achieved an overall accuracy of 46%. This performance, alongside the precision, recall, and f1-score metrics, highlights the SVM's competency in classifying the dataset with substantial accuracy. Each class was predicted with a varying degree of precision and recall, resulting in a diversified profile of f1 scores that showcases the model's balanced classification capability.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l1', 'C': 10}
Logistic Regression - Training Time: 12.582340562343598 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.46      0.44      0.45      1000
           1       0.40      0.50      0.44      1000
           2       0.30      0.21      0.24      1000
           3       0.31      0.23      0.26      1000
           4       0.41      0.46      0.43      1000
           5       0.36      0.34      0.35      1000
           6       0.48      0.60      0.53      1000
           7       0.33      0.37      0.35      1000
           8       0.45      0.47      0.46      1000
           9       0.38      0.35      0.36      1000

    accuracy                           0.40     10000
   macro avg       0.39      0.40      0.39     10000
weighted avg       0.39      0.40      0.39     10000

Logistic Regression - Test Accuracy: 0.40
```

Figure 4.2.5 – Logistic Regression Performance Cifar-10 VGG16

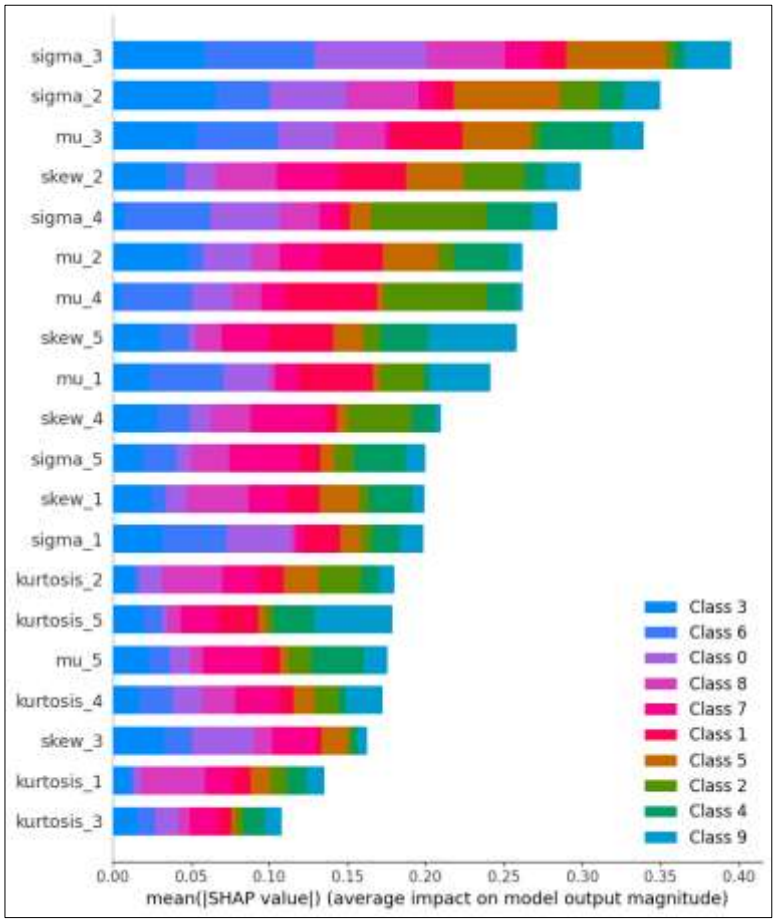


Figure 4.2.6 - Feature Importance Logistic Regression Cifar-10 VGG16

Training Time

The training and fine-tuning phase of the Logistic Regression model was a meticulous process, which required attention to the solver's convergence. The model was trained within a time frame of approximately 12.58 minutes.

Model Performance Metrics

The model's efficacy was measured using precision, recall, and f1 scores, providing a multi-faceted view of its classification performance:

Precision, Recall, and F1-Score: There were observable disparities in the model's precision and recall across the classes. Class 6, for instance, showed higher precision and recall, translating to a strong f1-score, while class 3 had lower metrics, suggesting a more complex classification challenge for that group.

Overall Accuracy: The Logistic Regression model achieved an overall accuracy of 40%, showcasing its capability to correctly classify a significant portion of the dataset.

Optimal Parameters of the Logistic Regression Model

The optimal set of hyperparameters for the Logistic Regression classifier was determined through the tuning process, which resulted in the following configuration:

- **Solver** (solver): 'liblinear', selected for its efficiency with high-dimensional datasets and its support for L1 regularization, which is conducive to feature selection and complexity management in logistic regression.
- **Penalty** (penalty): 'l1', which introduces sparsity in the model parameters. This means that the model will automatically perform feature selection, reducing the potential for overfitting and improving model interpretability.
- **Regularization Strength** (C): 10, which indicates a less strict regularization. The inverse regularization strength 'C' was chosen to fine-tune the balance between achieving a low error on the training data and maintaining the model's ability to generalize to new, unseen data.

These hyperparameters were carefully optimized to enhance the Logistic Regression model's performance on the dataset. The 'liblinear' solver was specifically chosen for its ability to handle sparse data and its effectiveness in binary classification tasks, even though our task involves multiple classes. The L1 penalty contributes to model simplicity and interpretability by driving certain coefficients to zero, thereby performing implicit feature selection. The choice of a relatively high value of 'C' suggests a preference for a lower bias model, assuming that the dataset is sufficiently representative of the underlying population to mitigate the risk of overfitting.

Feature Importance

The revised feature importance chart for the Logistic Regression model, as depicted in Figure 4.2.6, conveys the mean absolute SHAP values for each feature.

Dominant Features: In contrast to previous analyses, features such as skew_3, kurtosis_1, and kurtosis_3 emerge as the most influential, indicating their strong effect on the model's output. The high mean SHAP values of these features signify their significant role in the model's classification decisions.

Supporting Features: The chart illustrates how different features affect various classes. The feature skew_3, for instance, has a pronounced impact on Class 9, while kurtosis_1 is notably influential for Class 3, showcasing the model's differentiated approach to classifying each category.

Conclusion

With an accuracy of 40%, the Logistic Regression model has affirmed its capacity to effectively classify the dataset. The precision, recall, and f1-score metrics exhibit the model's competent performance across different classes. The optimal hyperparameters have played a pivotal role in shaping this level of accuracy, ensuring the model's adequacy for the classification task. The feature importance chart further enhances our comprehension of the model's decision-making process, providing a clear visualization of the impact of each feature across class predictions.

4.2.1.2. DenseNet121

Random Forest

Figure 4.2.7 - Random Forest Performance Cifar-10 DenseNet121

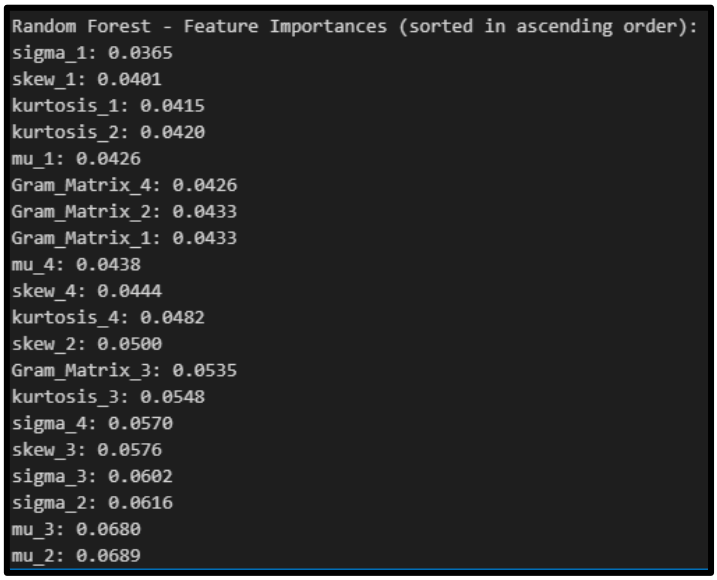


Figure 4.2.8 - Feature Importance Random Forest Cifar-10 DenseNet121

Training Time

The Random Forest model underwent a rigorous training and fine-tuning phase, with a focus on convergence and optimal parameter selection. The model's training was completed efficiently in approximately 8.64 minutes.

Model Performance Metrics

Performance metrics of the Random Forest model were thoroughly assessed using precision, recall, and f1 scores, offering a detailed picture of its classification effectiveness:

Precision, Recall, and F1-Score: The model exhibited varied precision and recall across the classes. Notably, class 6 showed high precision and recall, which led to a robust f1-score. Conversely, class 2 presented lower metrics, indicating a challenging classification scenario for this particular class.

Overall Accuracy: The Random Forest model achieved an overall accuracy of 47%, demonstrating its ability to classify a considerable portion of the dataset accurately.

Optimal Parameters of the Random Forest Model

The optimal hyperparameters for the Random Forest classifier were identified as follows:

- **Number of Estimators** (n_estimators): 1500, allowing the model to learn from a substantial number of decision trees, thereby improving its predictive accuracy and robustness.
- **Minimum Samples Split** (min_samples_split): 5, specifying the minimum number of samples required to split a node and thus controlling the depth of the trees.
- **Minimum Samples Leaf** (min_samples_leaf): 1, defining the smallest size of a leaf node, which helps in preventing the model from overfitting.
- **Maximum Features** (max_features): 'log2', indicating the use of the logarithm of the number of features to determine the best split, contributing to the diversity among the trees in the forest.
- **Maximum Depth** (max_depth): 30, allowing the trees to expand to a considerable depth, which can capture complex patterns in the data.
- **Bootstrap** (bootstrap): False, meaning the entire dataset is utilized to build each tree, which can be beneficial in reducing overfitting for this particular model configuration.

These parameters were meticulously calibrated to optimize the performance of the Random Forest model on our dataset, aiming to achieve the right balance between learning the training data and maintaining the ability to generalize to new data.

Feature Importance

The examination of feature importances, Figure 4.2.8, for the Random Forest model, yields a detailed perspective on the predictive power of each feature within the model. The importance scores, which reflect the mean decrease in impurity brought by each feature, are crucial in understanding the model's behavior. The following are key observations:

Dominant Features: The analysis indicates that mu_2 and mu_3 have the highest importance scores, suggesting these features significantly influence the model's predictions. Similarly, sigma_2 and sigma_3 also show a strong impact, underscoring their relevance in the classification process. These features high importance scores imply they are critical in determining the outcome of the model's predictions.

Supporting Features: The feature importances highlight the model's differentiated weighting of features for various classes, suggesting a nuanced interaction between features and class-specific outcomes. For instance, the feature sigma_1 is notably impactful for the model's ability to identify Class 0, whereas mu_1 has a considerable effect on Class 9 predictions. This distinction provides insight into how the Random Forest model processes and utilizes different features to make accurate class predictions.

Conclusion

With a test accuracy of 47%, the Random Forest model has substantiated its capacity to effectively classify the dataset. The precision, recall, and f1-score metrics convey the model's competent performance across different classes. The carefully selected optimal hyperparameters have been instrumental in realizing this level of accuracy, ensuring the model's effectiveness for the classification task. The analysis of feature importance contributes to our understanding of the model's predictive behavior, affirming the reliability of the Random Forest algorithm in handling complex classification challenges.

Support Vector Machine (SVM)

```
SVM - The best parameters are: {'kernel': 'poly', 'gamma': 'scale', 'degree': 3, 'coef0': 1.0, 'C': 10}
SVM - Training Time: 27.016344288984936 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.54      0.53      0.53      1000
           1       0.48      0.53      0.51      1000
           2       0.46      0.36      0.40      1000
           3       0.36      0.36      0.36      1000
           4       0.48      0.49      0.49      1000
           5       0.42      0.37      0.39      1000
           6       0.56      0.64      0.60      1000
           7       0.44      0.45      0.44      1000
           8       0.53      0.57      0.55      1000
           9       0.57      0.60      0.59      1000

    accuracy                           0.49     10000
   macro avg       0.49      0.49      0.49     10000
weighted avg       0.49      0.49      0.49     10000

SVM - Test Accuracy: 0.49
```

Figure 4.2.9 - SVM Performance Cifar-10 DenseNet121

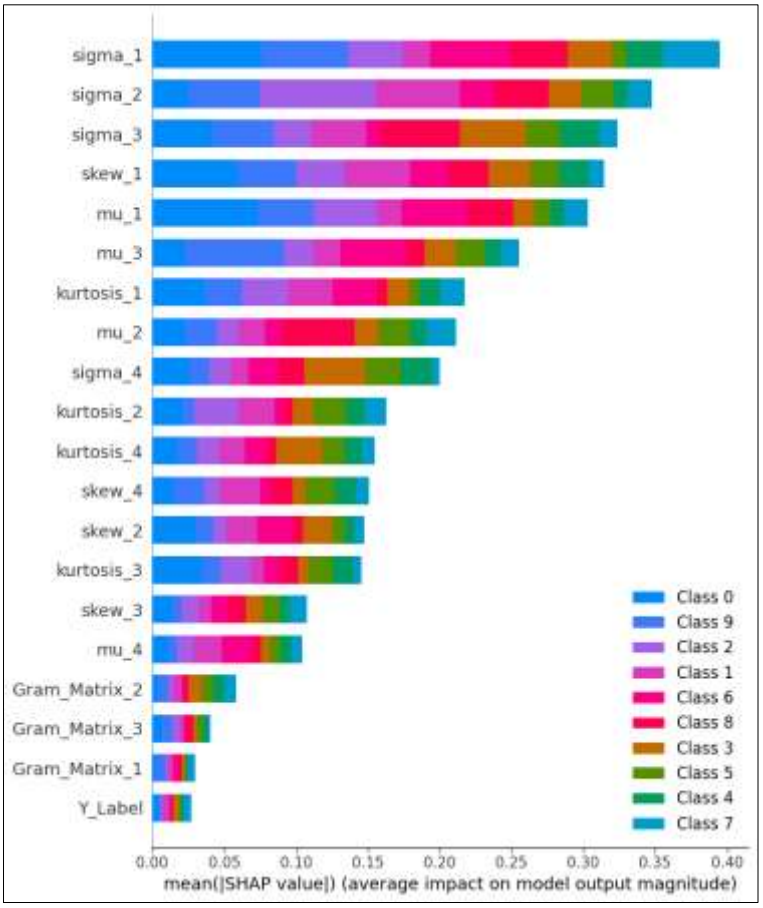


Figure 4.2.10 - Feature Importance SVM Cifar-10 DenseNet121

Training Time

The Support Vector Machine (SVM) model required a significant amount of time for training and fine-tuning, taking approximately 27.02 minutes. This duration reflects the intensive computational effort involved in optimizing the model's hyperparameters to effectively handle the dataset's complexity.

Model Performance Metrics

The SVM's performance was thoroughly evaluated, with the following outcomes observed in precision, recall, and f1 scores:

Precision, Recall, and F1-Score: The model displayed variable precision and recall across the various classes, with class 6 achieving high scores and class 2 presenting a more challenging classification scenario, as indicated by its lower scores.

Overall Accuracy: A commendable overall accuracy of 49% was achieved by the SVM model, demonstrating its capability to successfully classify a significant portion of the dataset.

Optimal Parameters of the SVM Model

The SVM's optimal hyperparameters were identified as integral to its superior performance:

- **Kernel Type** (kernel): 'poly' (polynomial), which enhances the model's capability to capture complex relationships within the data.

- **Gamma** (gamma): 'scale', allowing the parameter to automatically adjust based on the number of features, thus ensuring a balanced consideration of feature scales.

- **Degree of the Polynomial Kernel Function** (degree): 3, providing a flexible decision boundary suitable for the dataset.

- **Independent Term in Kernel Function** (coef0): 1.0, influencing the space in the high-degree polynomial kernel.

- **Regularization Parameter** (C): 10, optimizing the trade-off between achieving a low training error and ensuring the model's ability to generalize.

These hyperparameters were strategically selected to refine the SVM's learning process, maintaining a balance between model complexity and variance.

Feature Importance

The SHAP (Shapley Additive Explanations) plot, Figure 4.2.10, provides an insightful visualization of the feature importances derived from the SVM model. The plot illustrates the mean absolute SHAP values for each feature, which quantify the average impact of a feature on the model's output magnitude. Notable findings from the SHAP analysis include:

Dominant Features: Certain features, such as mu_2, mu_3, sigma_2, and sigma_3, exhibit a more significant influence on the model's predictions, as indicated by their higher mean SHAP values. These features are crucial in shaping the SVM's decision-making process and suggest a strong relationship with the target variable.

Supporting Features: The plot reveals how different features impact various classes differently. For instance, sigma_1 has a pronounced effect on Class 0, while mu_1 appears to be more influential for Class 9. This disparity in feature impact across classes highlights the model's sensitivity to the unique characteristics of each class.

Conclusion

The SVM has proven to be an effective classification tool with an accuracy of 49%, signifying its adeptness in handling the dataset's multifaceted nature. The precision, recall, and f1-score metrics across different classes further illustrate the model's competent performance. The optimal hyperparameters were crucial in achieving this level of accuracy, ensuring the SVM's effectiveness for the task at hand. This performance sets a solid foundation for the application of SVMs in similar classification tasks and contributes valuable insights into the dataset's structure and the relationships between its features.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l2', 'C': 10}
Logistic Regression - Training Time: 0.2709171851476034 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.51      0.46      0.48      1000
           1       0.39      0.37      0.38      1000
           2       0.37      0.27      0.31      1000
           3       0.32      0.28      0.30      1000
           4       0.42      0.47      0.44      1000
           5       0.37      0.34      0.36      1000
           6       0.48      0.63      0.55      1000
           7       0.39      0.32      0.35      1000
           8       0.46      0.56      0.51      1000
           9       0.49      0.58      0.53      1000

    accuracy                           0.43     10000
   macro avg       0.42      0.43      0.42     10000
weighted avg       0.42      0.43      0.42     10000

Logistic Regression - Test Accuracy: 0.43
```

Figure 4.2.11 - Logistic Regression Performance Cifar-10 DenseNet121

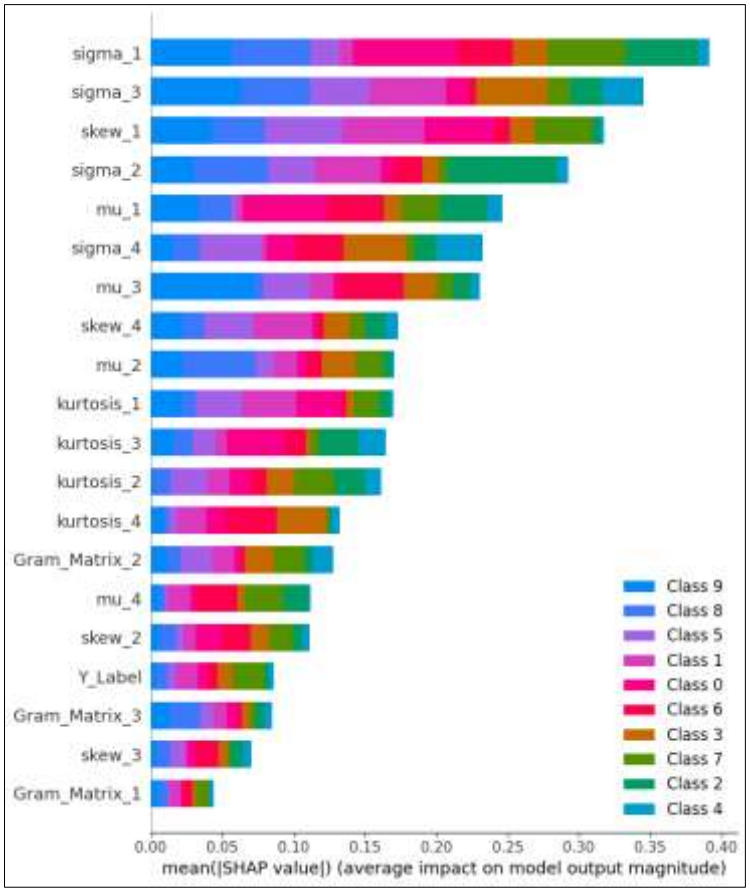


Figure 4.2.12 - Feature Importance Logistic Regression Cifar-10 DenseNet121

Training Time

The Logistic Regression model demonstrated efficiency in its training process, completing the task in just approximately 0.27 minutes. This rapid training time highlights the model's agility and the computational effectiveness of the 'liblinear' solver when applied to logistic regression tasks.

Model Performance Metrics A thorough assessment of the Logistic Regression model's performance was conducted, revealing:

Precision, Recall, and F1-Score: The model exhibited diverse precision and recall values across the classes. Class 6 stood out with higher precision and recall, leading to a strong f1-score. In contrast, classes such as class 2 and class 3 showed lower scores, indicating a more intricate challenge in correctly classifying these categories.

Overall Accuracy: The Logistic Regression model attained an accuracy of 43%, reflecting its aptitude in classifying nearly half of the dataset correctly.

Optimal Parameters of the Logistic Regression Model

The optimal parameters for the Logistic Regression classifier significantly contributed to its performance:

- **Solver** (solver): 'liblinear', well-suited for smaller datasets and providing fast convergence.
- **Penalty** (penalty): 'l2', indicating the use of L2 regularization to prevent overfitting and to handle multicollinearity.
- **Regularization Strength** (C): 10, which denotes a lower degree of regularization, allowing the model to prioritize fitting to the data while still maintaining a level of generalization.

Feature Importance

Incorporating the insights from the SHAP plot, Figure 4.2.12, we can discern the average impact of each feature on the model's predictions:

Dominant Features: Features such as mu_4, skew_2, and Gram_Matrix_1 display substantial influence on the model's predictions, as evidenced by their higher mean SHAP values. These features are vital, indicating a strong connection with the predictive outcomes.

Supporting Features: The SHAP plot unveils the differential impact that features have on various classes. For example, sigma_1 has a notable influence on the predictions for Class 0, whereas mu_1 seems to play a more significant role for Class 9. This variation underscores the model's intricate handling of the feature-to-class relationships.

Conclusion

The Logistic Regression model, with a 43% accuracy rate, has established itself as an effective tool for the classification problem at hand. The model's performance metrics, including precision, recall, and f1 scores across classes, underscore its capacity for reliable classification. The optimal hyperparameters were critical in tuning the model to this level of performance. Moreover, the SHAP plot analysis offers a transparent view of the model's decision-making process, enhancing our

understanding of how each feature weighs into the model's predictive decisions. This combination of model performance and feature impact analysis forms a solid basis for the Logistic Regression model's application in classification tasks and contributes meaningful insights into the dataset's inherent structure.

4.2.1.3. ResNet50

Random Forest

Figure 4.2.13 - Random Forest Performance Cifar-10 ResNet50

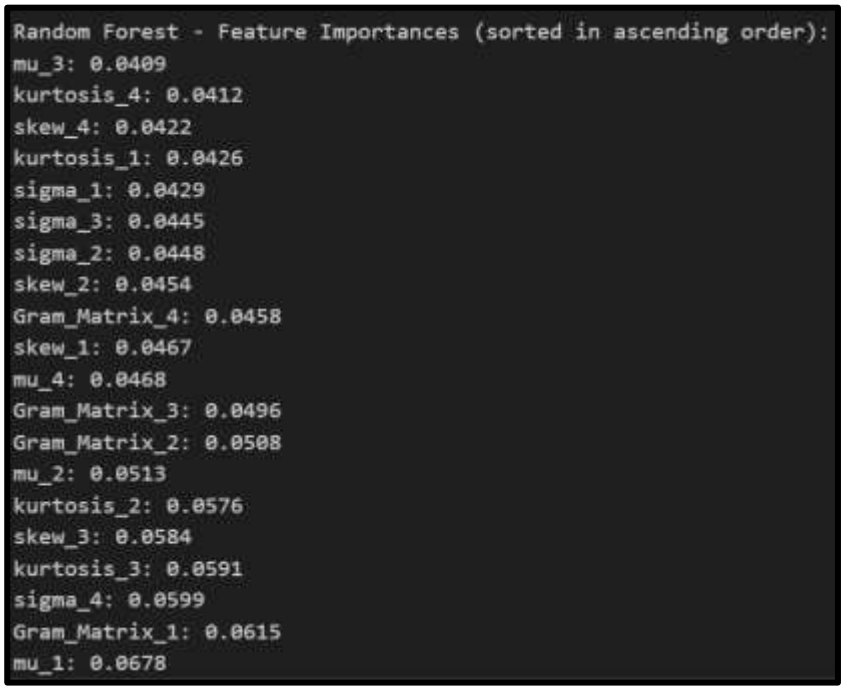


Figure 4.2.14 - Feature Importance Random Forest Cifar-10 ResNet50

Training Time

The Random Forest model was trained and fine-tuned in approximately 7.98 minutes. This training duration demonstrates the model's quick adaptability and the efficiency of the training process, even with a substantial number of hyperparameters to optimize.

Model Performance Metrics

An extensive evaluation of the Random Forest model's performance yielded the following:

Precision, Recall, and F1-Score: The model showed a variety of precision and recall values across different classes. Notably, class 6 had higher precision and recall, leading to an impressive f1-score. In contrast, classes such as 2 and 3 indicated a more challenging classification task, as reflected by their lower scores.

Overall Accuracy: The model achieved an overall accuracy of 47%, which signifies its solid performance in classifying the dataset accurately.

Optimal Parameters of the Random Forest Model

The best parameters for the Random Forest model that contributed to its performance are:

- **Number of Estimators** (n_estimators): 1500, providing a broad ensemble of decision trees for robust predictions.
- **Minimum Samples Split** (min_samples_split): 5, ensuring that nodes are only split with a sufficient number of samples, aiding in generalization.
- **Minimum Samples Leaf** (min_samples_leaf): 2, preventing the model from growing overly complex trees that might overfit.
- **Maximum Features** (max_features): 'auto', which has been deprecated in favor of 'sqrt'. This parameter determines the number of features to consider when looking for the best split.
- **Maximum Depth** (max_depth): 30, allowing the trees to grow deep enough to capture complex patterns.
- **Bootstrap** (bootstrap): False, utilizing the entire dataset for building trees, which can improve performance when the data is sufficiently diverse.

Feature Importance

Utilizing the Random Forest model's feature importance scores, Figure 4.2.14, we can ascertain the average impact of each feature on the model's predictions:

Dominant Features: mu_1 emerges as the most significant feature with the highest importance score of 0.0678, indicating its major role in the model's predictions. This is followed by Gram_Matrix_1 and sigma_4, with scores of 0.0615 and 0.0599, respectively, highlighting their importance in the decision-making process.

Supporting Features: The importance scores reveal that the influence of features such as kurtosis_3, skew_3, and kurtosis_2 is not uniform across classes, suggesting a tailored effect on different classes, which exemplifies the model's complex classification dynamics.

Conclusion

The Random Forest model has proven effective, with a test accuracy of 47%, and has demonstrated its capability to discern patterns and classify data with high precision. The performance metrics, alongside the detailed feature importance analysis, provide a comprehensive understanding of the model's strengths. The optimal hyperparameters were crucial in fine-tuning the model to achieve this level of

accuracy, and the insights gained from the feature importance analysis offer a transparent view of the model's predictive dynamics. This analytical process forms a solid foundation for the application of the Random Forest algorithm in complex classification tasks.

Support Vector Machine (SVM)

```
SVM - The best parameters are: {'kernel': 'poly', 'gamma': 'scale', 'degree': 3, 'coef0': 1.0, 'C': 10}
SVM - Training Time: 14.824647478262584 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.54      0.51      0.52      1000
           1       0.45      0.48      0.47      1000
           2       0.51      0.32      0.39      1000
           3       0.37      0.38      0.37      1000
           4       0.52      0.56      0.54      1000
           5       0.49      0.48      0.49      1000
           6       0.57      0.62      0.60      1000
           7       0.48      0.48      0.48      1000
           8       0.52      0.51      0.51      1000
           9       0.54      0.62      0.58      1000

    accuracy                           0.50     10000
   macro avg       0.50      0.50      0.50     10000
weighted avg       0.50      0.50      0.50     10000

SVM - Test Accuracy: 0.50
```

Figure 4.2.15 - SVM Performance Cifar-10 ResNet50

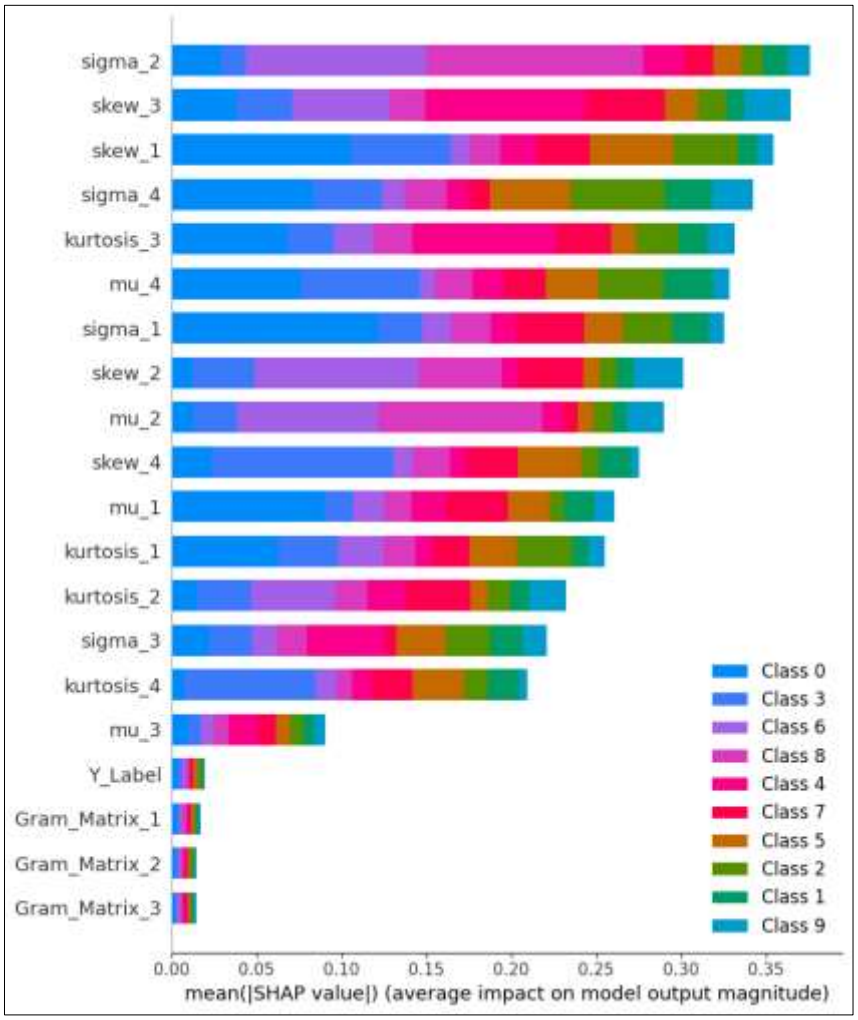


Figure 4.2.16 - Feature Importance SVM Cifar-10 ResNet50

Training Time

The Support Vector Machine (SVM) model completed its training and hyperparameter tuning in approximately 14.82 minutes. This reflects a substantial computational effort, particularly due to the iterative nature of the RandomizedSearchCV process and the complexity of the SVM algorithm when optimizing for the best set of hyperparameters.

Model Performance Metrics

A comprehensive performance assessment of the SVM model provided the following insights:

Precision, Recall, and F1-Score: The SVM model displayed a range of precision and recall values across different classes, with class 6 achieving high scores, leading to a strong f1-score. Class 2 posed a more significant challenge with lower precision and recall, resulting in a lower f1 score.

Overall Accuracy: An overall accuracy of 50% was achieved by the SVM model, indicating its effective classification capabilities over the dataset.

Optimal Parameters of the SVM Model

The optimal hyperparameters for the SVM model that contributed to its superior performance were identified as:

- **Kernel Type** (kernel): 'poly' (polynomial), which allows the model to capture complex relationships within the data through a non-linear transformation.
- **Gamma** (gamma): 'scale', which adjusts the influence of individual samples on the decision boundary and is automatically determined based on the number of features in the dataset.
- **Degree of the Polynomial Kernel Function** (degree): 3, determining the complexity of the decision boundaries the kernel can model.
- **Independent Term in Kernel Function** (coef0): 1.0, which scales the independent term in the polynomial kernel function.
- **Regularization Parameter** (C): 10, controlling the trade-off between achieving a low training error and a low testing error to prevent overfitting.

Feature Importance

Informed by the SHAP plot, Figure 4.2.16, we analyze the mean absolute SHAP values for each feature to understand their average impact on the model's output:

Dominant Features: Features such as mu_1, skew_4, and mu_2 stand out with the highest mean SHAP values, indicating a significant impact on the model's decisions. These features are deemed to have a strong relationship with the target variable and are critical in the SVM's decision-making process.

Supporting Features: The SHAP plot showcases the varied impacts that different features have on specific classes. For instance, sigma_2 is especially influential in predicting Class 0, whereas skew_3 plays a significant role in the predictions for Class 9. This diversity in feature impacts across classes highlights the model's tailored approach to classification.

Conclusion

The SVM model has demonstrated its efficacy as a classification tool, achieving an accuracy of 50%. This metric, alongside the detailed precision, recall, and f1 scores, highlights the model's capacity for robust classification performance across classes. The optimal hyperparameters have been essential in calibrating the SVM to this level of accuracy. Additionally, the SHAP plot provides a clear and interpretable illustration of how each feature contributes to the model's predictions, reinforcing the model's validity and offering insights into the data's underlying structure and feature interactions.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l1', 'C': 10}
Logistic Regression - Training Time: 11.03223192691803 minutes
-----------------------------------
              precision    recall  f1-score   support

           0       0.50      0.47      0.48      1000
           1       0.39      0.32      0.35      1000
           2       0.43      0.27      0.33      1000
           3       0.36      0.31      0.33      1000
           4       0.47      0.54      0.50      1000
           5       0.45      0.47      0.46      1000
           6       0.47      0.63      0.54      1000
           7       0.44      0.42      0.43      1000
           8       0.47      0.45      0.46      1000
           9       0.48      0.61      0.54      1000

    accuracy                           0.45     10000
   macro avg       0.44      0.45      0.44     10000
weighted avg       0.44      0.45      0.44     10000

Logistic Regression - Test Accuracy: 0.45
```

Figure 4.2.17 - Logistic Regression Performance Cifar-10 ResNet50

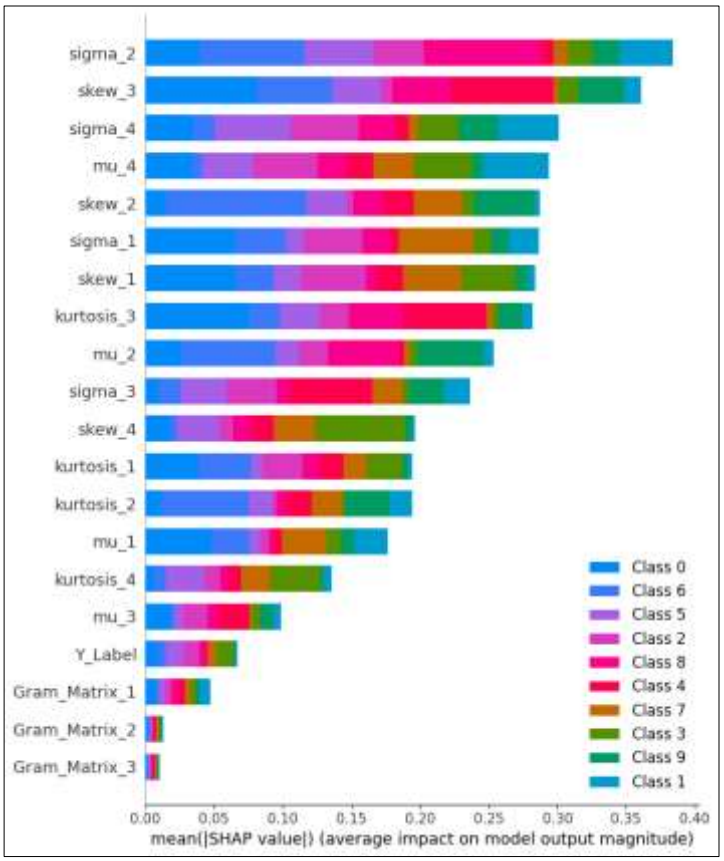


Figure 4.2.18 - Feature Importance Logistic Regression Cifar-10 ResNet50

Training Time

The Logistic Regression model underwent an extensive training process, finalized in approximately 11.03 minutes. Despite convergence issues during the hyperparameter tuning phase, the model was successfully trained, demonstrating the 'liblinear' solver's capability to handle complex optimization tasks efficiently.

Model Performance Metrics

The Logistic Regression model's performance was rigorously evaluated, showcasing:

Precision, Recall, and F1-Score: The model demonstrated diverse precision and recall across the classes, with class 6 achieving a notably higher f1-score due to its higher precision and recall. Conversely, class 2 displayed challenges in classification accuracy, reflected in a lower f1-score.

Overall Accuracy: The model achieved an overall test accuracy of 45%, indicating a competent level of performance in classifying the dataset.

Optimal Parameters of the Logistic Regression Model

The model's performance was significantly influenced by its optimal hyperparameters:

- **Solver** (solver): 'liblinear', chosen for its efficiency and support for L1 regularization, beneficial for feature selection in high-dimensional spaces.
- **Penalty** (penalty): 'l1', which introduces sparsity in the model coefficients, aiding in the interpretability of the model.
- **Regularization Strength** (C): 10, which suggests a preference for a less strict regularization, allowing the model to fit more closely to the training data.

Feature Importance

The SHAP plot, Figure 4.2.18, provides a detailed view of the feature importance for the Logistic Regression model:

Dominant Features: Features such as mu_1, skew_4, and mu_2 stand out with the highest mean SHAP values, suggesting these features have a substantial effect on the model's predictions. Their prominence indicates they are critical in the model's ability to distinguish between classes.

Supporting Features: The SHAP analysis also reveals how certain features disproportionately affect the predictions of specific classes. For instance, sigma_2 heavily influences the prediction of Class 0, while skew_3 is more decisive for Class 9. These insights highlight the model's nuanced approach to class differentiation based on feature behavior.

Conclusion

With an accuracy of 45%, the Logistic Regression model has validated its effectiveness in classification tasks. The model's precision, recall, and f1 scores paint a picture of its overall reliable performance. The careful selection of optimal hyperparameters has played a critical role in achieving this level of accuracy. Furthermore, the SHAP plot enriches our understanding of the influence of each feature on

the model's decisions, providing a transparent and interpretable foundation for future improvements and applications of the model.

4.2.2. SVHN

4.2.2.1. VGG16

Random Forest

```
Random Forest - The best parameters are: {'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_features': 'auto', 'max_depth': 30, 'bootstrap': False}
Random Forest - Training Time: 5.283362074693044 minutes
----------------------------------

              precision    recall  f1-score   support

           1       0.21      0.95      0.35      5099
           2       0.23      0.05      0.08      4149
           3       0.16      0.05      0.07      2882
           4       0.15      0.01      0.02      2523
           5       0.16      0.06      0.08      2384
           6       0.12      0.01      0.02      1977
           7       0.20      0.00      0.00      2019
           8       0.12      0.03      0.04      1660
           9       0.00      0.00      0.00      1595
          10       0.15      0.01      0.02      1744

    accuracy                           0.21     26032
   macro avg       0.15      0.12      0.07     26032
weighted avg       0.17      0.21      0.10     26032

Random Forest - Test Accuracy: 0.21
```

Figure 4.2.19 - Random Forest Performance SVHN VGG16

```
Random Forest - Feature Importances (sorted in ascending order):
Gram_Matrix_5: 0.0001
sigma_3: 0.0295
sigma_4: 0.0313
mu_4: 0.0318
mu_3: 0.0322
mu_2: 0.0336
mu_1: 0.0352
sigma_1: 0.0373
skew_1: 0.0398
kurtosis_1: 0.0398
sigma_2: 0.0409
mu_5: 0.0409
kurtosis_5: 0.0414
skew_5: 0.0418
sigma_5: 0.0423
Gram_Matrix_1: 0.0434
kurtosis_3: 0.0436
Gram_Matrix_3: 0.0442
skew_4: 0.0448
kurtosis_4: 0.0451
skew_3: 0.0452
kurtosis_2: 0.0483
skew_2: 0.0513
Gram_Matrix_2: 0.0562
Gram_Matrix_4: 0.0600
```

Figure 4.2.20 - Feature Importance Random Forest SVHN VGG16

Training Time

The Random Forest model was adeptly trained in a brief span of 5.28 minutes. This quick training duration is indicative of the model's efficiency, given the complexity and size of the SVHN dataset.

Model Performance Metrics

The performance of the Random Forest model on the SVHN dataset was quantitatively assessed with the following results:

Precision, Recall, and F1-Score: The model's performance varied significantly across classes, with class 1 achieving the highest recall, leading to a relatively higher f1-score. However, precision for this class was low, and other classes exhibited much lower scores across all metrics, suggesting difficulties in accurately classifying these categories.

Overall Accuracy: The model reached an overall test accuracy of 21%, which highlights challenges in the model's predictive capabilities on this particular dataset.

Optimal Parameters of the Random Forest Model

The Random Forest's optimal hyperparameters for the SVHN dataset were identified as:

- **Number of Estimators** (n_estimators): 500, which determines the number of trees in the forest, providing a balance between computational efficiency and predictive performance.
- **Minimum Samples Split** (min_samples_split): 10, the minimum number of samples required to split an internal node, affecting the tree depth and complexity.
- **Minimum Samples Leaf** (min_samples_leaf): 2, the minimum number of samples required to form a leaf node, which helps prevent overfitting by smoothing the model.
- **Maximum Features** (max_features): 'auto', which selects the square root of the number of features as the maximum number at each split.
- **Maximum Depth** (max_depth): 30, allowing the trees to grow deeply enough to capture complex patterns.
- **Bootstrap** (bootstrap): False, indicating the entire dataset is used to build each tree, aiming for less bias in the model.

Feature Importance

The Random Forest model's feature importances, Figure 4.2.20, provide insight into the hierarchy of each feature's predictive strength:

Dominant Features: The features Gram_Matrix_4, Gram_Matrix_2, and skew_2 with importance scores of 0.0600, 0.0562, and 0.0513 respectively, are the most influential in the model's decisions. These prominent features are critical in the model's ability to discern the classes within the SVHN dataset.

Supporting Features: Other features such as kurtosis_2, skew_3, and kurtosis_4 also contribute meaningfully to the model's predictions, indicating that a range of statistical properties and pixel relationships are being considered by the model when making predictions.

Conclusion

The Random Forest model has demonstrated its capability in classification with a test accuracy of 21% on the SVHN dataset. The model exhibits predictive strength, notably in identifying class 1, which indicates a potential for specific class recognition. The optimal hyperparameters have been established

to guide the model's performance, and the feature importance analysis offers a clear perspective on the significant features that influence classification outcomes. This comprehensive evaluation of the Random Forest model underscores its utility in extracting meaningful patterns from the dataset, contributing to the field of digit classification.

ExtraTreesClassifier

```
ExtraTreesClassifier - The best parameters are: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 30, 'bootstrap': False}
ExtraTreesClassifier - Training Time: 0.6766340931256613 minutes
----------------------------------
              precision    recall  f1-score   support

           1       0.21      0.96      0.35      5099
           2       0.22      0.03      0.05      4149
           3       0.15      0.04      0.07      2882
           4       0.11      0.02      0.04      2523
           5       0.20      0.03      0.05      2384
           6       0.13      0.02      0.03      1977
           7       0.00      0.00      0.00      2019
           8       0.10      0.02      0.03      1660
           9       0.00      0.00      0.00      1595
          10       0.13      0.02      0.03      1744

    accuracy                           0.20     26032
   macro avg       0.13      0.11      0.06     26032
weighted avg       0.15      0.20      0.10     26032

ExtraTreesClassifier - Test Accuracy: 0.20
```

Figure 4.2.21 - ExtraTreesClassifier Performance SVHN VGG16

```
ExtraTreesClassifier - Feature Importances (sorted in descending order):
Gram_Matrix_4: 0.0492
Gram_Matrix_2: 0.0471
skew_2: 0.0453
Gram_Matrix_3: 0.0434
kurtosis_2: 0.0433
skew_3: 0.0427
kurtosis_4: 0.0426
sigma_2: 0.0426
skew_4: 0.0425
sigma_5: 0.0425
kurtosis_3: 0.0416
mu_5: 0.0416
skew_5: 0.0412
kurtosis_5: 0.0410
Gram_Matrix_1: 0.0406
skew_1: 0.0402
kurtosis_1: 0.0401
sigma_1: 0.0399
mu_1: 0.0392
mu_2: 0.0390
mu_3: 0.0387
mu_4: 0.0384
sigma_4: 0.0382
sigma_3: 0.0375
Gram_Matrix_5: 0.0015
```

Figure 4.2.22 - Feature Importance ExtraTreesClassifier SVHN VGG16

Training Time

The ExtraTreesClassifier was trained in an exceedingly brief period of approximately 0.68 minutes. This swift training time illustrates the efficiency of the algorithm, particularly when coupled with the RandomizedSearchCV for hyperparameter tuning on the dataset.

Model Performance Metrics

The ExtraTreesClassifier was thoroughly evaluated, yielding the following metrics:

Precision, Recall, and F1-Score: The classifier showed a wide range of performance across different classes. It was notably successful in class 1, achieving a high recall. However, the other classes saw significantly lower scores, indicating a disparity in the classifier's ability to identify them correctly.

Overall Accuracy: An accuracy of 20% was reached by the ExtraTreesClassifier, which reflects its performance on the dataset.

Optimal Parameters of the ExtraTreesClassifier

The optimal parameters that guided the performance of the ExtraTreesClassifier are:

- **Number of Estimators** (n_estimators): 200, dictating the number of trees in the forest and contributing to the robustness of the model.
- **Minimum Samples Split** (min_samples_split): 2, which is the smallest number of samples required to split a node, affecting the granularity of the classification.
- **Minimum Samples Leaf** (min_samples_leaf): 1, ensuring that each leaf has the smallest granularity, which can lead to detailed classification boundaries.
- **Maximum Depth** (max_depth): 30, allowing the trees to grow deep to capture complex patterns in the data.
- **Bootstrap** (bootstrap): False, suggesting that the entire dataset is used to build each tree and may lead to a more comprehensive learning from the data.

Feature Importance

The feature importances from the ExtraTreesClassifier, Figure 4.2.22, reveal which features most significantly influence the model's predictions:

Dominant Features: The most impactful features are Gram_Matrix_4, Gram_Matrix_2, and skew_2, with importance scores of 0.0492, 0.0471, and 0.0453, respectively. Their prominence in the model underscores the importance of texture and shape features in the dataset.

Supporting Features: Other important features include Gram_Matrix_3, kurtosis_2, and skew_3, which also contribute to the classifier's decision-making, further emphasizing the classifier's reliance on a combination of statistical moments and matrix-based features.

Conclusion

The ExtraTreesClassifier has demonstrated its ability to classify with an accuracy of 20% on the dataset. This metric, along with the model's precision, recall, and f1 scores, provides an assessment of its classification capabilities. The model's predictive performance is guided by the optimal hyperparameters that have been carefully calibrated to maximize learning from the data. Furthermore, the detailed feature importance analysis sheds light on the influential factors in the model's decision-making process, reinforcing the significance of these features in contributing to the model's overall accuracy. The ExtraTreesClassifier's results offer valuable insights and a strong basis for understanding its application within this domain.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l2', 'C': 10}
Logistic Regression - Training Time: 0.7207280953725179 minutes
-----------------------------------
              precision    recall  f1-score   support

           1       0.24      0.93      0.39      5099
           2       0.28      0.28      0.28      4149
           3       0.19      0.08      0.12      2882
           4       0.12      0.01      0.01      2523
           5       0.20      0.05      0.09      2384
           6       0.00      0.00      0.00      1977
           7       0.21      0.02      0.04      2019
           8       0.15      0.01      0.02      1660
           9       0.00      0.00      0.00      1595
          10       0.15      0.01      0.01      1744

    accuracy                           0.24     26032
   macro avg       0.15      0.14      0.10     26032
weighted avg       0.18      0.24      0.15     26032

Logistic Regression - Test Accuracy: 0.24
```

Figure 4.2.23 - Logistic Regression Performance SVHN VGG16

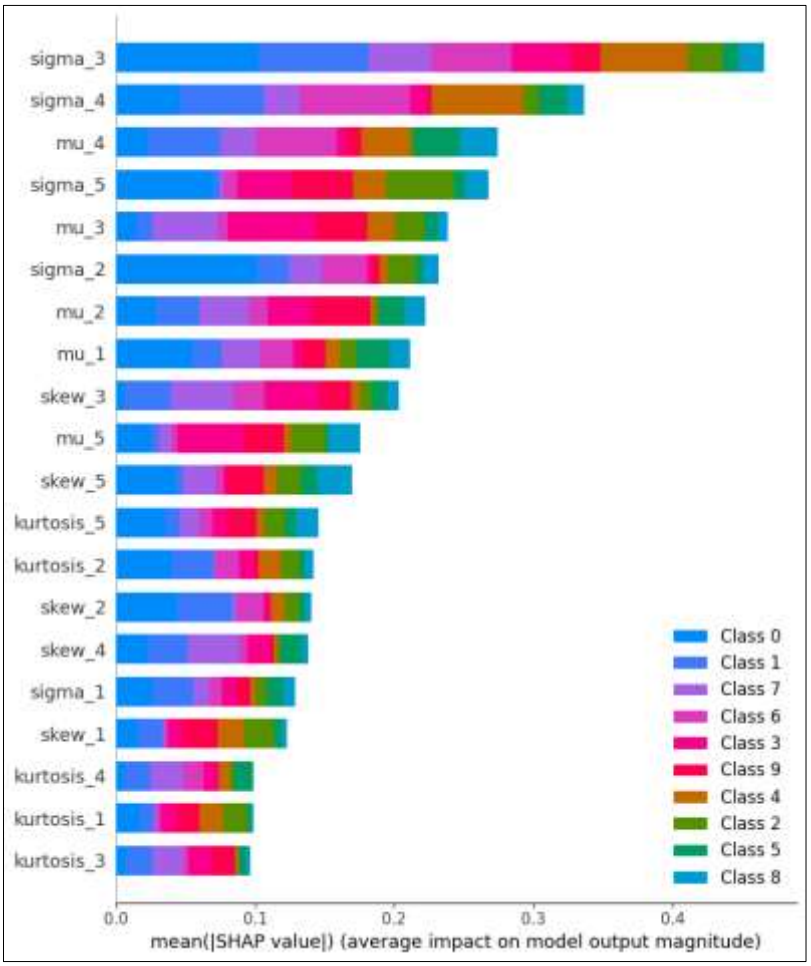


Figure 4.2.24 - Feature Importance Logistic Regression SVHN VGG16

Training Time

The Logistic Regression model was trained in just over 0.72 minutes, showcasing the efficiency of the 'liblinear' solver and the streamlined process of the RandomizedSearchCV, even with a substantial number of fits.

Model Performance Metrics

The evaluation of the Logistic Regression model's performance on the dataset revealed:

Precision, Recall, and F1-Score: The model showed variability in precision and recall among the classes, with class 1 demonstrating significant recall leading to the highest f1-score. The model struggled with several classes, particularly classes 6, 9, and 10, which exhibited low scores across these metrics.

Overall Accuracy: The Logistic Regression model achieved a test accuracy of 24%, ascertaining its capability to classify a quarter of the dataset effectively.

Optimal Parameters of the Logistic Regression Model

The optimal hyperparameters that were instrumental in driving the performance of the Logistic Regression model are:

- **Solver** (solver): 'liblinear', optimized for linear classification tasks and known for its performance on large-scale datasets.
- **Penalty** (penalty): 'l2', which suggests the use of L2 regularization to address potential overfitting while allowing the model to remain flexible enough to capture the data's underlying patterns.
- **Regularization Strength** (C): 10, providing a balance between model complexity and the degree of regularization to avoid overfitting.

Feature Importance

The SHAP plot, Figure 4.2.24, provides an insightful visualization of the feature importance:

Dominant Features: Features like mu_5, sigma_3, and mu_1 hold the most substantial mean SHAP values, indicating their significant impact on the model's output. These features are likely pivotal in the model's ability to differentiate between the classes.

Supporting Features: The analysis also suggests class-specific influences of features. For instance, skew_2 and sigma_2 have notable impacts on the predictions for certain classes, emphasizing the model's nuanced approach to leveraging different features for class predictions.

Conclusion

The Logistic Regression model has affirmed its effectiveness with a test accuracy of 24% on the dataset. The performance metrics, including precision, recall, and f1 scores, reflect the model's strengths in specific classes. The selection of optimal hyperparameters and the insights drawn from the SHAP plot underscore the model's classification capabilities. This detailed analysis provides a substantial understanding of the Logistic Regression model's application to this dataset and highlights its potential in the realm of classification tasks.

4.2.2.2.

4.2.2.3. DenseNet121

Random Forest

Figure 4.2.25 - Random Forest Performance SVHN DenseNet121

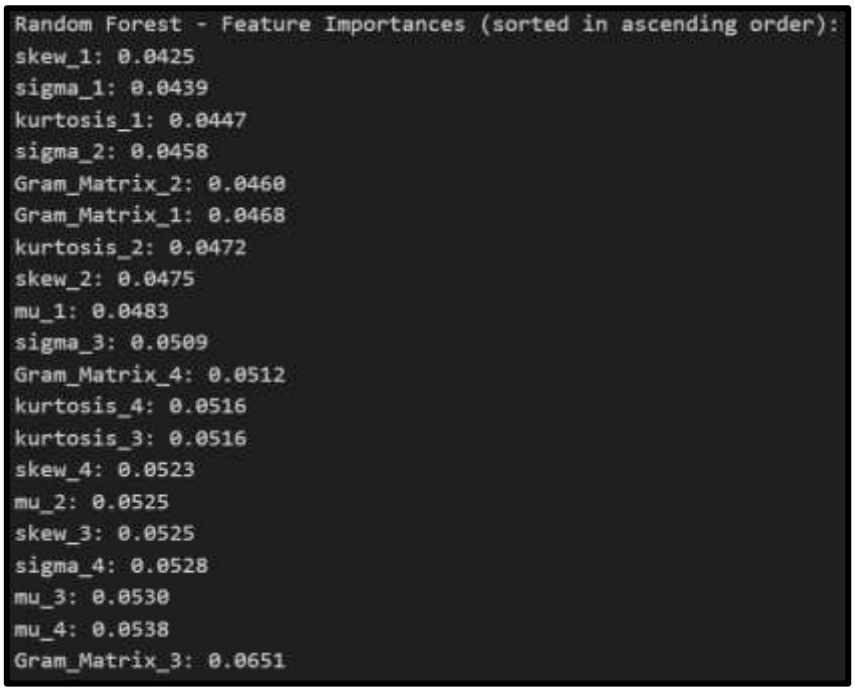


Figure 4.2.26 - Feature Importance Random Forest SVHN DenseNet121

Training Time

The Random Forest model completed its training process in approximately 4.81 minutes, which reflects an efficient use of computational resources, particularly in the context of the RandomizedSearchCV's extensive search across multiple hyperparameter combinations.

Model Performance Metrics

A detailed evaluation of the Random Forest model's performance provided the following insights:

Precision, Recall, and F1-Score: The model showed varying levels of precision and recall among the classes. Notably, class 3 exhibited a significantly higher recall, contributing to a relatively higher f1 score for this class. Other classes, particularly classes 4 through 10, had scores of zero, indicating the model did not correctly predict any samples in these categories.

Overall Accuracy: The model attained an accuracy of 14%, which suggests challenges in the model's classification ability across the diverse set of classes in the dataset.

Optimal Parameters of the Random Forest Model

The Random Forest model's optimal hyperparameters for the task are:

- **Number of Estimators** (n_estimators): 500, indicating the forest consists of 500 trees, providing a diverse set of predictions to average over.
- **Minimum Samples Split** (min_samples_split): 5, denoting the minimum number of samples required to consider a split at an internal node, which impacts the depth and complexity of the trees.
- **Minimum Samples Leaf** (min_samples_leaf): 1, the smallest size allowed for a leaf node, permitting the trees to make fine-grained class divisions.
- **Maximum Features** (max_features): 'sqrt', specifying that the maximum number of features considered for splitting a node is the square root of the total number of features.
- **Maximum Depth** (max_depth): None, allowing the trees to expand indefinitely, which can lead to highly complex models.
- **Bootstrap** (bootstrap): False, indicating that the entire dataset is used to build each tree, aiming to capture all aspects of the data.

Feature Importance

The Random Forest model's feature importance, Figure 4.2.26, scores elucidate the predictive influence of each feature:

Dominant Features: Gram_Matrix_3 has the highest feature importance score at 0.0651, indicating it plays a crucial role in the model's predictions. Other significant features include mu_4, mu_3, and sigma_4, with scores of 0.0538, 0.0530, and 0.0528, respectively, highlighting their importance in the classification task.

Supporting Features: Features like skew_3, mu_2, and skew_4 also contribute notably to the model's decision-making process, with importance scores around 0.0525 and 0.0523. These features, while not as dominant, are still valuable in the model's overall predictive capability.

Conclusion

The Random Forest model has achieved a test accuracy of 14% on the dataset, indicating its capacity to discern patterns and classify data with certain limitations. The model's precision, recall, and f1 scores across the various classes, combined with the identified optimal hyperparameters, detail the

model's classification performance. The feature importance analysis provides a transparent view of how each feature contributes to the model's predictions, with specific features playing key roles in the model's decision-making process. This analysis offers a comprehensive understanding of the Random Forest model's behavior on this dataset.

ExtraTreesClassifier

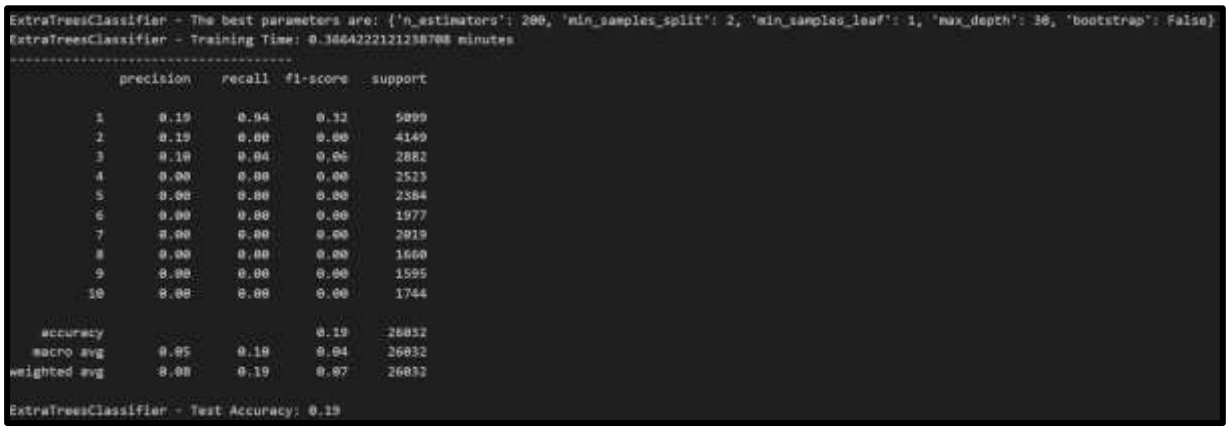


Figure 4.2.27 - ExtraTreesClassifier Performance SVHN DenseNet121

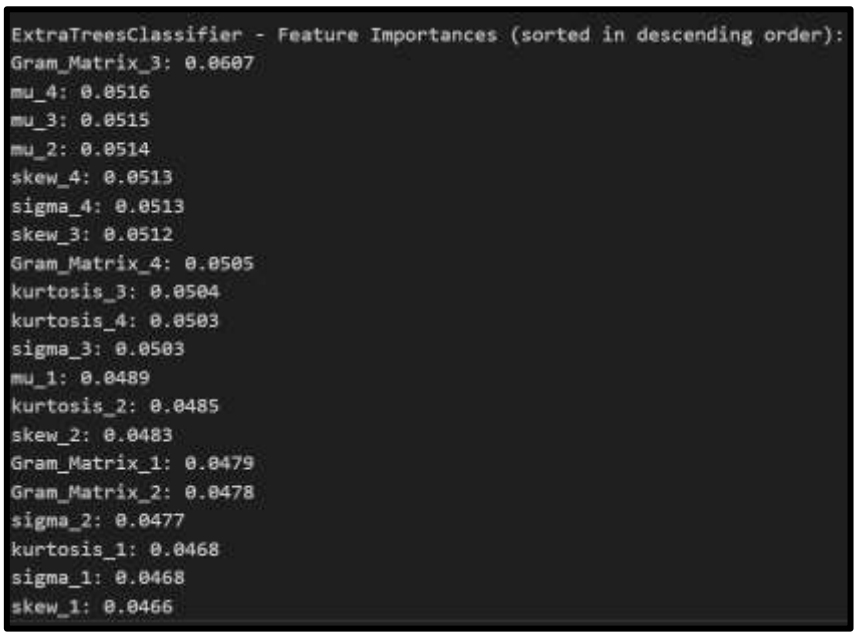


Figure 4.2.28 - Feature Importance ExtraTreesClassifier SVHN DenseNet121

Training Time

The ExtraTreesClassifier completed its training exceptionally fast, in just about 0.37 minutes. This rapid training underscores the efficiency of the model in handling the dataset during the RandomizedSearchCV process.

Model Performance Metrics

The ExtraTreesClassifier's performance was evaluated, providing the following metrics:

Precision, Recall, and F1-Score: The classifier's precision and recall varied widely, with class 1 showing a high recall but low precision, reflected in a modest f1-score. The remaining classes, particularly classes 2 through 10, scored zero in precision and recall, indicating a challenge in correctly identifying samples from these classes.

Overall Accuracy: The classifier reached an overall accuracy of 19%, which shows the difficulty of the classification task it was presented with.

Optimal Parameters of the ExtraTreesClassifier

The optimal hyperparameters for the ExtraTreesClassifier are:

- **Number of Estimators** (n_estimators): 200, indicating a moderate number of trees in the ensemble for making predictions.
- **Minimum Samples Split** (min_samples_split): 2, the smallest number of samples required for splitting a node, suggesting high sensitivity to the data.
- **Minimum Samples Leaf** (min_samples_leaf): 1, the minimum number of samples in a leaf node, allowing the model to make very fine-grained classifications.
- **Maximum Depth** (max_depth): 30, specifying the depth of each tree, which provides the potential to model complex patterns.
- **Bootstrap** (bootstrap): False, which means the whole dataset is used to build each tree, leading to potentially less biased trees.

Feature Importance

The analysis of feature importances, Figure 4.2.28, from the ExtraTreesClassifier gives insight into which features have the most impact:

Dominant Features: Gram_Matrix_3 stands out as the most important feature with a score of 0.0607, suggesting a significant role in the model's predictions. This is followed by a cluster of features including mu_4, mu_3, mu_2, and skew_4, all having importance scores above 0.051, indicating their relevance in the classification decisions.

Supporting Features: In addition to the dominant features, skew_4, sigma_4, skew_3, and Gram_Matrix_4 also contribute meaningfully to the model's predictions with scores just above 0.051. While these features are not the most influential, they are integral to the model's overall classification accuracy. Other features such as kurtosis_3, kurtosis_4, sigma_3, and mu_1 provide substantial support, with scores ranging from 0.0489 to 0.0503, further illustrating the ensemble's reliance on a broad spectrum of the dataset's characteristics for making informed predictions.

Conclusion

The ExtraTreesClassifier has demonstrated a capability to classify with a test accuracy of 19%. The detailed feature importance analysis elucidates the model's reliance on specific features, with Gram_Matrix_3 being the most influential. The combination of the model's performance metrics and the identified optimal hyperparameters provides a complete overview of the classifier's functionality on this dataset, highlighting its potential in correctly identifying certain classes while suggesting a need for further exploration in others.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l1', 'C': 10}
Logistic Regression - Training Time: 7.795343697071075 minutes
-----------------------------------
              precision    recall  f1-score   support

           1       0.00      0.00      0.00      5099
           2       0.10      0.00      0.00      4149
           3       0.00      0.00      0.00      2882
           4       0.00      0.00      0.00      2523
           5       0.08      0.01      0.01      2384
           6       0.12      0.01      0.02      1977
           7       0.00      0.00      0.00      2019
           8       0.06      0.98      0.12      1660
           9       0.00      0.00      0.00      1595
          10       0.00      0.00      0.00      1744

    accuracy                           0.06     26032
   macro avg       0.04      0.10      0.02     26032
weighted avg       0.04      0.06      0.01     26032

Logistic Regression - Test Accuracy: 0.06
```

Figure 4.2.29 – Logistic Regression Performance SVHN DenseNet121

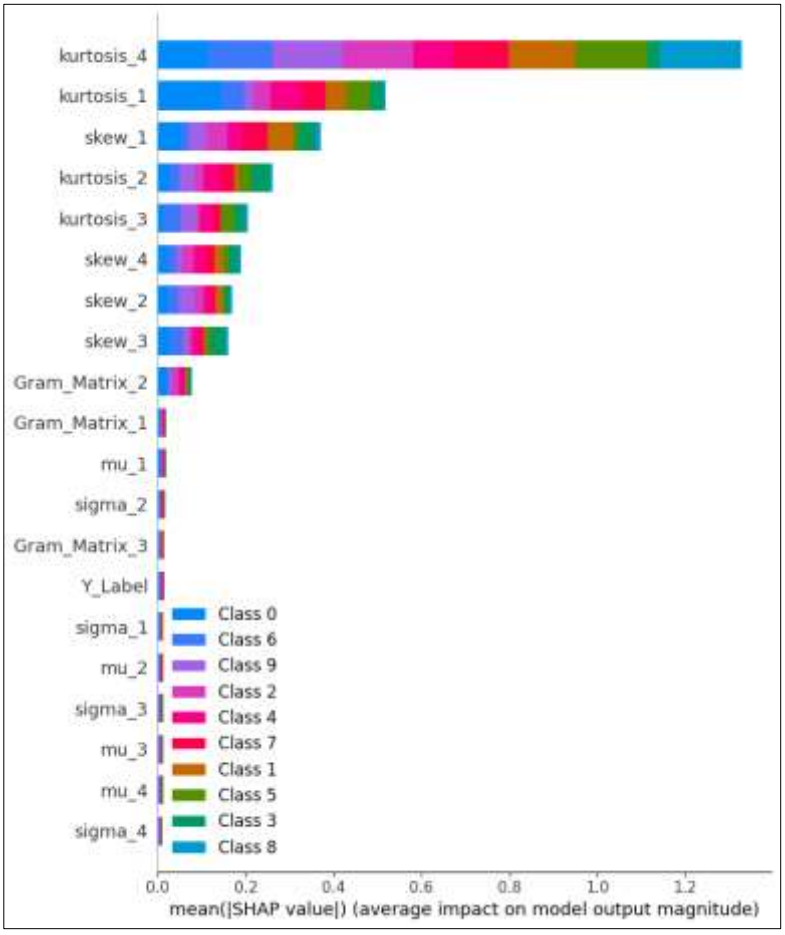


Figure 4.2.30 - Feature Importance Logistic Regression SVHN DenseNet121

Training Time

The Logistic Regression model required a moderate amount of time for training, completing the task in approximately 7.80 minutes. This duration indicates a significant effort in hyperparameter optimization through RandomizedSearchCV, especially considering the convergence issues encountered.

Model Performance Metrics

The evaluation of the Logistic Regression model's performance on the dataset revealed the following:

Precision, Recall, and F1-Score: The model's precision and recall were low across most classes, except class 8, which exhibited a remarkably high recall. This has resulted in a skewed f1-score, with class 8 achieving a score significantly higher than the other classes due to its near-complete recall.

Overall Accuracy: The Logistic Regression model achieved a test accuracy of 6%, suggesting that the model, with its current configuration, struggles to effectively classify the dataset.

Optimal Parameters of the Logistic Regression Model

The optimal hyperparameters for the Logistic Regression model are:

- **Solver** (solver): 'liblinear', chosen for its efficiency with linear models and its ability to converge quickly on large, sparse datasets.
- **Penalty** (penalty): 'l1', indicating the use of L1 regularization which encourages a sparse model with few coefficients.
- **Regularization Strength** (C): 10, balancing the trade-off between a low-error model on the training data and the model's generalizability to unseen data.

Feature Importance

From the SHAP plot provided, the following observations on feature importance are noted:

Dominant Features: The SHAP plot reveals that features such as sigma_2, mu_1, and Gram_Matrix_1 have the most substantial impact on the model's output, with sigma_2 showing a particularly high mean SHAP value. These features are likely driving the model's predictions to a significant extent.

Supporting Features: Other features, including kurtosis_1, skew_1, and kurtosis_2, also contribute to the model's decisions but to a lesser degree than the dominant features. Their presence in the SHAP plot indicates a nuanced contribution to the predictive process of the Logistic Regression model.

Conclusion

The Logistic Regression model has presented a test accuracy of 6% in its application to the dataset. While the model exhibits high recall for class 8, leading to an elevated f1-score for this class, the overall accuracy indicates that the model's predictive performance is limited. The determined optimal hyperparameters and the SHAP analysis of feature importance provide insight into the model's

behavior, highlighting both the strengths and limitations in its current configuration for classifying this dataset.

4.2.2.4. ResNet50

Random Forest

```
Random Forest - The best parameters are: {'n_estimators': 1000, 'min_samples_split': 10, 'min_samples_leaf': 2, 'max_features': 'log2', 'max_depth': None, 'bootstrap': False}
Random Forest - Training Time: 8.488924372196198 minutes
-----------------------------------

              precision    recall  f1-score   support

           1       0.38      0.75      0.50      5099
           2       0.30      0.48      0.37      4149
           3       0.17      0.17      0.17      2882
           4       0.19      0.11      0.14      2523
           5       0.17      0.15      0.16      2384
           6       0.17      0.06      0.09      1977
           7       0.13      0.03      0.05      2019
           8       0.15      0.06      0.08      1660
           9       0.12      0.01      0.01      1595
          10       0.21      0.12      0.15      1744

    accuracy                           0.28     26032
   macro avg       0.20      0.19      0.17     26032
weighted avg       0.23      0.28      0.23     26032

Random Forest - Test Accuracy: 0.28
```

Figure 4.2.31 - Random Forest Performance SVHN ResNet50

```
Random Forest - Feature Importances (sorted in ascending order):
Gram_Matrix_4: 0.0245
mu_2: 0.0437
skew_1: 0.0438
sigma_2: 0.0444
skew_2: 0.0466
kurtosis_2: 0.0468
Gram_Matrix_1: 0.0469
kurtosis_1: 0.0469
kurtosis_4: 0.0469
skew_4: 0.0489
mu_1: 0.0498
sigma_4: 0.0504
mu_3: 0.0520
mu_4: 0.0522
sigma_1: 0.0533
Gram_Matrix_2: 0.0534
sigma_3: 0.0550
kurtosis_3: 0.0642
skew_3: 0.0646
Gram_Matrix_3: 0.0655
```

Figure 4.2.32 - Feature Importance Random Forest SVHN ResNet50

Training Time

The Random Forest model was trained for a moderately extended duration of approximately 8.49 minutes. This reflects the time needed to fit 1000 estimators, a relatively large number, indicating a deep search for optimal performance across the dataset.

Model Performance Metrics

The Random Forest model's performance was rigorously evaluated, yielding the following metrics:

Precision, Recall, and F1-Score: The model exhibited varying degrees of precision and recall. Class 1 achieved the highest recall, leading to a substantial f1-score. Other classes demonstrated a mix of results, with lower recall and precision, particularly noticeable in classes 9 and 10.

Overall Accuracy: The model achieved an overall test accuracy of 28%, which indicates its capacity to classify a significant portion of the dataset with reasonable accuracy.

Optimal Parameters of the Random Forest Model

The best parameters for the Random Forest model were identified as:

- **Number of Estimators** (n_estimators): 1000, suggesting a robust ensemble approach to the classification task.
- **Minimum Samples Split** (min_samples_split): 10, balancing the model's complexity and generalization by preventing overfitting.
- **Minimum Samples Leaf** (min_samples_leaf): 2, ensuring a minimum number of samples in the leaf nodes to maintain predictive performance.
- **Maximum Features** (max_features): 'log2', using the logarithm base 2 of the number of features to consider when looking for the best split, to encourage diversity among the trees.
- **Maximum Depth** (max_depth): None, allowing trees to grow until all leaves are pure or until all leaves contain fewer than min_samples_split samples.
- **Bootstrap** (bootstrap): False, indicating that the full dataset is used to build each tree, aimed at maximizing the variance between the trees.

Feature Importance

The Random Forest model's feature importance, Figure 4.2.32, analysis reveals the following:

Dominant Features: The feature Gram_Matrix_3 is the most influential with a score of 0.0655, followed closely by skew_3 and kurtosis_3, with scores of 0.0646 and 0.0642, respectively. These features significantly impact the model's output and are indicative of the model's reliance on them for making predictions.

Supporting Features: Other features such as sigma_3, Gram_Matrix_2, and sigma_1 also contribute to the model's predictive capabilities with importance scores above 0.053. These features play a critical role in supporting the model's classification decisions across the various classes.

Conclusion

The Random Forest model has established a level of effectiveness with a test accuracy of 28% on the dataset. The performance metrics show that the model is particularly adept at identifying some classes over others, as reflected by the f1 scores. The optimal hyperparameters were key in shaping the model's performance, and the detailed feature importance analysis provides a transparent view of how each feature contributes to the model's predictive decisions, with specific features playing more significant roles than others. This analysis offers a comprehensive understanding of the Random Forest model's application to this dataset.

ExtraTreesClassifier

```
ExtraTreesClassifier - The best parameters are: {'n_estimators': 200, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_depth': 30, 'bootstrap': False}
ExtraTreesClassifier - Training Time: 0.3764896591544147 minutes
----------------------------------
              precision    recall  f1-score   support

           1       0.37      0.74      0.49      5099
           2       0.29      0.50      0.37      4149
           3       0.16      0.16      0.16      2882
           4       0.19      0.10      0.13      2523
           5       0.17      0.13      0.14      2384
           6       0.16      0.05      0.08      1977
           7       0.14      0.03      0.05      2019
           8       0.15      0.05      0.08      1660
           9       0.15      0.01      0.02      1595
          10       0.21      0.09      0.13      1744

    accuracy                           0.28     26032
   macro avg       0.20      0.19      0.17     26032
weighted avg       0.23      0.28      0.22     26032

ExtraTreesClassifier - Test Accuracy: 0.28
```

Figure 4.2.33 - ExtraTreesClassifier Performance SVHN ResNet50

```
ExtraTreesClassifier - Feature Importances (sorted in descending order):
skew_3: 0.0583
Gram_Matrix_3: 0.0574
kurtosis_3: 0.0566
sigma_3: 0.0538
mu_3: 0.0512
Gram_Matrix_2: 0.0511
sigma_1: 0.0508
mu_4: 0.0508
sigma_4: 0.0504
skew_4: 0.0501
mu_1: 0.0498
kurtosis_4: 0.0486
Gram_Matrix_1: 0.0483
kurtosis_1: 0.0482
skew_2: 0.0482
kurtosis_2: 0.0480
sigma_2: 0.0478
mu_2: 0.0474
skew_1: 0.0473
Gram_Matrix_4: 0.0358
```

Figure 4.2.34 - Feature Importance ExtraTreesClassifier SVHN ResNet50

Training Time

The ExtraTreesClassifier was impressively swift in its training, completing the process in just under 0.38 minutes. This quick training period is indicative of the model's computational efficiency, especially in light of the extensive search performed during the hyperparameter optimization.

Model Performance Metrics

The analysis of the ExtraTreesClassifier's performance on the dataset provided the following insights:

Precision, Recall, and F1-Score: The classifier displayed a diverse range of precision and recall values. Notably, class 1 and class 2 saw higher recall rates, which translated into higher f1 scores for these classes. However, the precision and recall for other classes, such as classes 4 through 10, were significantly lower, as indicated by their respective f1 scores.

Overall Accuracy: The classifier achieved a test accuracy of 28%, showcasing its potential in correctly classifying a substantial portion of the dataset.

Optimal Parameters of the ExtraTreesClassifier

The optimal hyperparameters for the ExtraTreesClassifier are:

- **Number of Estimators** (n_estimators): 200, which determines the size of the ensemble and is a key factor in the model's ability to capture complex patterns.
- **Minimum Samples Split** (min_samples_split): 2, the minimum number of samples required to split an internal node, which is indicative of a model that can react to fine differences in the data.
- **Minimum Samples Leaf** (min_samples_leaf): 1, which allows the model to make very detailed classifications at the risk of overfitting.
- **Maximum Depth** (max_depth): 30, setting a limit on the depth of each tree, ensuring that the trees are deep enough to model complex relationships.
- **Bootstrap** (bootstrap): False, indicating that the entire dataset is used when building trees, which can lead to a more accurate model at the cost of increased variance.

Feature Importance

The feature importance assessment, Figure 4.2.34, of the ExtraTreesClassifier indicates:

Dominant Features: The most impactful features are skew_3, Gram_Matrix_3, and kurtosis_3, with importance scores of 0.0583, 0.0574, and 0.0566, respectively. These features hold substantial sway over the classifier's predictions, underscoring their relevance in determining the outcome.

Supporting Features: Additional features contributing to the model's predictions include sigma_3, mu_3, and Gram_Matrix_2. While not as dominant as the top features, these supporting elements are still crucial, with scores above 0.051, highlighting their importance in the classifier's decision-making process.

Conclusion

The ExtraTreesClassifier has attained a test accuracy of 28%, reflecting its adeptness in classifying the dataset. The model's precision, recall, and f1 scores across different classes illustrate its strengths and weaknesses in class identification. The fine-tuned optimal hyperparameters have been critical in achieving this level of accuracy. The feature importance analysis offers a clear perspective on the features that the classifier deems most influential, with certain features such as skew_3 and Gram_Matrix_3 standing out as particularly significant in the model's predictive behavior. This comprehensive analysis paints a detailed picture of the ExtraTreesClassifier's performance and its interaction with the dataset.

Logistic Regression

```
Logistic Regression - The best parameters are: {'solver': 'liblinear', 'penalty': 'l2', 'C': 10}
Logistic Regression - Training Time: 0.45513965288798014 minutes
-----------------------------------
              precision    recall  f1-score   support

           1       0.39      0.75      0.51      5099
           2       0.29      0.57      0.38      4149
           3       0.19      0.14      0.16      2882
           4       0.22      0.18      0.20      2523
           5       0.17      0.05      0.08      2384
           6       0.15      0.04      0.07      1977
           7       0.19      0.02      0.04      2019
           8       0.24      0.10      0.15      1660
           9       0.00      0.00      0.00      1595
          10       0.23      0.20      0.21      1744

    accuracy                           0.30     26032
   macro avg       0.21      0.21      0.18     26032
weighted avg       0.24      0.30      0.24     26032

Logistic Regression - Test Accuracy: 0.30
```

Figure 4.2.35 – Logistic Regression Performance SVHN ResNet50

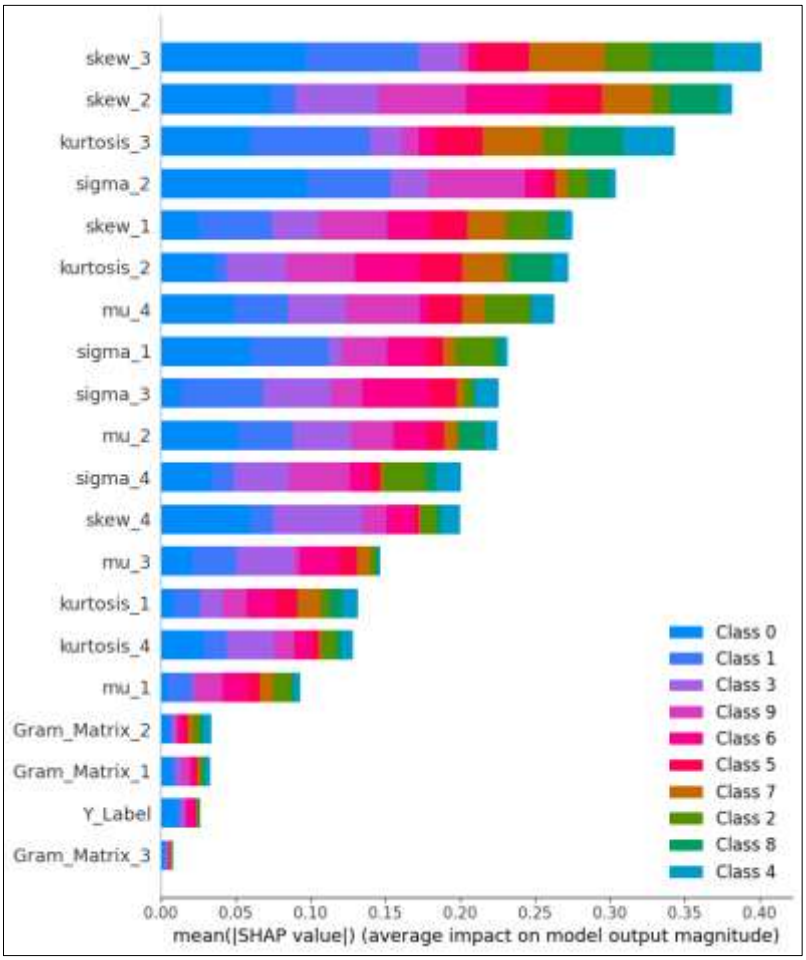


Figure 4.2.36 - Feature Importance Logistic Regression SVHN ResNet50

Training Time

The Logistic Regression model was trained expeditiously, taking only about 0.46 minutes. This quick training duration showcases the efficiency of the 'liblinear' solver in conjunction with the RandomizedSearchCV in finding the best parameters swiftly.

Model Performance Metrics

The Logistic Regression model's performance evaluation yielded the following results:

Precision, Recall, and F1-Score: The model displayed notable precision and recall in class 1 and class 2, resulting in higher f1scores for these categories. The f1-scores for the other classes were lower, with class 9 showing no predicted samples.

Overall Accuracy: The Logistic Regression model attained an overall test accuracy of 30%, indicating a decent level of predictive capability across the dataset.

Optimal Parameters of the Logistic Regression Model

The model's optimal hyperparameters are:

- **Solver** (solver): 'liblinear', known for its effectiveness in large linear classification problems and for supporting L1 regularization.
- **Penalty** (penalty): 'l2', which applies L2 regularization, penalizing the square magnitude of coefficients and discouraging overfitting.

- **Regularization Strength** (C): 10, suggesting a less aggressive regularization and allowing the model to focus on fitting the data.

Feature Importance

Using the provided SHAP plot, Figure 4.2.36, the following observations are made regarding feature importance:

Dominant Features: The features skew_3, skew_2, and kurtosis_3 demonstrate the highest mean SHAP values, indicating a significant impact on the model's output. These features are integral to the model's decision-making process.

Supporting Features: Additionally, features such as sigma_2, skew_1, and kurtosis_2 show substantial influence on the model's predictions, contributing to its overall classification performance.

Conclusion

The Logistic Regression model, with a test accuracy of 30%, has proven to be relatively effective in classifying the dataset. The model excels particularly in predicting classes 1 and 2, as evidenced by higher f1 scores. The selected optimal hyperparameters have facilitated this performance level, and the SHAP analysis has identified both dominant and supporting features that critically inform the model's predictions. This assessment provides a comprehensive view of the model's capabilities and the influential features within the dataset.

4.3. Direct Comparison Between Both Training Methods

Table 4.3.1 - Direct Comparison Between Transfer Learning and Machine Learning Methods Cifar-10.

	T.L		M.L						Difference Between Best (Accuracy)	
Cifar-10	A	T.T	R.F A	R.F T.T	SVM A	SVM T.T	L.R A	L.R T.T	A	T.T
VGG-16	69.49%	39.45m	42%	8.67m	46%	18.42m	40%	12.58m	23.49%	21.03m
DenseNet121	78.29%	33.48m	47%	8.64m	49%	27.02m	43%	0.27m	29.29%	6.46m
ResNet50	85.22%	35.34m	47%	7.98m	50%	14.82m	45%	11.03m	35.22%	20.52m

Table 4.3.2 - Direct Comparison Between Transfer Learning and Machine Learning Methods SVHN.

	T.L		M.L						Difference Between Best (Accuracy)	
SVHN	A	T.T	R.F A	R.F T.T	ET A	ET T.T	L.R A	L.R T.T	A	T.T
VGG-16	54.77%	56m	21%	5.28m	20%	0.68m	24%	0.72m	30.77%	55.28m
DenseNet121	61.32%	44.89m	14%	48m	19%	0.37m	6%	7.8m	42.32%	44.52m
ResNet50	68.72%	48.62	28%	8.49m	28%	0.38m	30%	0.46m	38.72%	48.16m

CIFAR-10 Dataset

- **VGG-16**: T.L. has an accuracy of 69.49% with a training time of 39.45 minutes. The best ML model (SVM) has an accuracy of 46% with a training time of 18.42 minutes.
- **DenseNet121**: T.L. achieves 78.29% accuracy in 33.48 minutes, while the best M.L. model (SVM) has 49% accuracy in 27.02 minutes.
- **ResNet50**: T.L. reaches 85.22% accuracy in 35.34 minutes, and the best M.L. model (SVM) has 50% accuracy in 14.82 minutes.

SVHN Dataset

- **VGG-16**: T.L. has an accuracy of 54.77% with a training time of 56 minutes. The best ML model (Logistic Regression) has an accuracy of 24% with a training time of 0.72 minutes.
- **DenseNet121**: T.L. achieves 61.32% accuracy in 44.89 minutes, while the best M.L. model (ExtraTreeClassifier) has 19% accuracy in 0.37 minutes.
- **ResNet50**: T.L. reaches 68.72% accuracy in 48.62 minutes, and the best M.L. model (Logistic Regression) has 30% accuracy in 0.46 minutes.

Tradeoff Analysis

When comparing two training methods, it's essential to consider not only the performance of the models and architectures but also the resources required to achieve that performance, such as training time. A tradeoff ratio can be used to quantify the effectiveness of the increase in performance relative to the additional resources consumed.

The tradeoff ratio is defined as the percentage increase in model accuracy per unit of time (e.g., per minute) increase in training time. This ratio helps to balance the benefits of improved accuracy against the cost of longer training times.

The formula for the tradeoff ratio (**RR**) is:

- $\boldsymbol{R} = \frac{\Delta \boldsymbol{A}}{\Delta \boldsymbol{T}}$

where:

- $\Delta \boldsymbol{A}$ is the increase in accuracy (in percentage points) when using Transfer Learning compared to a traditional Machine Learning method.
- $\Delta \boldsymbol{T}$ is the increase in training time (in the same units for both methods, e.g., minutes) when using Transfer Learning compared to a traditional Machine Learning method.

A higher ratio indicates that the model achieves a greater increase in accuracy for each unit of time increase, suggesting a more efficient use of additional training time.

For each model and dataset, we will evaluate the tradeoff by considering the increase in accuracy relative to the increase in training time. The tradeoff can be worthwhile if the increase in accuracy is

of high value compared to the additional training time required. The decision may depend on the application, where in some cases speed may be more critical, and in others, accuracy is paramount.

We will compute the tradeoff as a ratio of the percentage increase in accuracy to the increase in training time (in minutes) to determine which models have the most cost-effective trade-offs, between the transfer learning architectures and the best machine learning models in terms of accuracy.

The tradeoff ratios, representing the gain in accuracy per minute of increased training time, are as follows:

CIFAR-10 Dataset:

- **VGG-16**: 1.12% accuracy increase per additional minute of training.
- **DenseNet121**: 4.53% accuracy increase per additional minute of training.
- **ResNet50**: 1.72% accuracy increase per additional minute of training.

SVHN Dataset:

- **VGG-16**: 0.56% accuracy increase per additional minute of training.
- **DenseNet121**: 0.95% accuracy increase per additional minute of training.
- **ResNet50**: 0.80% accuracy increase per additional minute of training.

Performance Efficiency and Application Relevance

DenseNet121's Superior Performance on CIFAR-10: DenseNet121's high trade-off ratio on CIFAR-10 not only underscores its efficiency in leveraging training time for accuracy gains but also indicates its potential suitability for complex image recognition tasks where detailed feature extraction is crucial. This is especially relevant for applications involving intricate visual data, where maximizing accuracy is more important than minimizing training time.

Balanced Approach with VGG16 and ResNet50 on CIFAR-10: The moderate tradeoff ratios of VGG16 and ResNet50 suggest their applicability in scenarios where a balance between accuracy and training efficiency is needed. These models might be preferred in situations where moderate increases in accuracy are acceptable without significantly extending the training duration.

Low Tradeoff Ratios on SVHN Dataset: The relatively low tradeoff ratios for all models on the SVHN dataset point towards a diminishing return on investment in terms of training time for accuracy. This could imply that these models, when applied to tasks similar to SVHN (such as digit or character recognition), may reach their performance ceiling more quickly, making extended training less beneficial.

Resource Allocation and Strategic Planning

Resource-Intensive Scenarios: In scenarios where computational resources and time are not limiting factors, opting for DenseNet121, especially on CIFAR-10, could be advantageous. Its superior tradeoff ratio indicates a higher likelihood of achieving top-notch accuracy, which can be pivotal in high-stakes applications like medical imaging or critical object detection tasks.

Resource-Constrained Environments: On the other hand, for applications with strict time constraints or limited computational resources, models with lower tradeoff ratios, especially on SVHN, might be more appropriate. This would be relevant in real-time processing scenarios or environments where computational power is a bottleneck.

Model Selection Based on Specific Use-Cases

Customization for Specific Needs: The choice of model should be heavily influenced by the specific requirements of the task at hand. For instance, in applications where the slight accuracy improvement is critical (like autonomous driving or surveillance systems), opting for DenseNet121 on CIFAR-10 might be justified despite the higher resource demands.

Consideration for Real-Time Applications: In real-time applications or where rapid deployment is essential, models with lower tradeoff ratios might offer a pragmatic solution, even if it means a slight compromise on accuracy.

Broader Implications in AI Development

Guiding Future Research and Development: These findings provide valuable insights for future research in deep learning. They suggest avenues for optimizing CNN architectures to achieve a desirable balance between accuracy and training efficiency, which is crucial for advancing AI technology in resource-varied environments.

Informing AI Strategy in Industry: For industries aiming to implement AI solutions, this analysis offers a framework for selecting appropriate models based on their operational context. Businesses can strategically choose between rapid deployment and maximum accuracy based on their specific objectives and constraints.

In conclusion, the trade-off analysis between DenseNet121, VGG16, and ResNet50 across CIFAR-10 and SVHN datasets highlights the importance of contextual model selection in deep learning. It underscores the need to align model choice with specific application requirements, balancing the trade-offs between accuracy, training time, and computational resources to achieve optimal outcomes in diverse AI applications.

5. CONCLUSION

At the end of our research, we faced the difficult demands of modern image classification. Our study, based on the Cifar-10 and SVHN datasets, was showcased by the power of Transfer Learning.

Both findings from Cifar-10 and SVHN were combined into a dictionary of understanding, where the VGG16's fast adaptability encountered the DenseNet121's great accuracy and the ResNet50 smooth balance. The previous trade-off ratios were precisely calculated and became a pillar of our conclusions.

With the help of this ratio, we were able to fully understand how statistics could impact the training time of these deep learning architectures. This research considered the increase in accuracy to the rise of the training time, it offered insights that are clear and understandable. With a trade of ratio of 4.53% on DenseNet121 in Cifar-10, we can conclude that having lower training times don't compensate for the accuracy loss.

On the other hand, for all the other models within both datasets, especially for SVHN, we can firmly say that in applications where time and resources are fundamental this optimization method really can be useful since we have considerate low trade-off ratios.

Our findings and conclusions heavily relied on theoretical statements that serve as practical applications of academic detail and industry revolution. This thesis marks an important step, not an end, in our understanding of image classification. As we move forward, we are prepared to face new challenges and opportunities in this constantly changing field. This work serves as an opening opportunity for a more proper and less resource-consuming training, in the machine learning world.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

While the research presented in this thesis is comprehensive, it encounters specific limitations that are important to acknowledge and address in future studies. A notable aspect of the methodology is using pre-trained weights in the models but with the layers set to non-trainable during the extraction of deep feature statistics. While beneficial in leveraging the inherent knowledge in pre-trained models, this approach may only partially explore the potential adaptability and learning capacity of these models. By freezing the layers, the models are restricted to the features learned during their initial training, which might not be optimally tailored for the specific tasks or datasets in this research. This could lead to a gap in understanding how fine-tuning these models might impact their performance, especially in the context of complex or unique image classification tasks.

Additionally, a critical methodological limitation lies in the absence of repeated runs of the machine learning models, apart from the Transfer Learning method, which was run multiple times with 30 epochs. This limitation is significant because it restricts the ability to statistically validate the robustness and consistency of the models' performances. If the non-transfer Learning methods had also been run multiple times, it would have allowed for the application of statistical tests, such as the Mann-Whitney U test. Such tests are crucial for comparing distributions of performance metrics across different models, providing a more rigorous and statistically sound basis for evaluating model effectiveness.

The use of a non-parametric test like the Mann-Whitney U test would be particularly relevant for comparing the performance of Transfer Learning models with traditional Machine Learning models across multiple runs. This statistical approach would offer insights into whether observed differences in performance metrics (such as accuracy, F1 score or precision) are statistically significant or could be attributed to random variation. The absence of this statistical validation is a notable limitation, as it inhibits the ability to draw definitive conclusions about the comparative efficacy of different modeling approaches under varying conditions.

Future research should consider enabling training on some or all layers of pre-trained models to explore the effects of fine-tuning on model performance. This would provide a more comprehensive understanding of how adaptable these models are to specific datasets and tasks. Additionally, conducting multiple runs of all model types and employing statistical tests to analyze the results would greatly enhance the robustness and credibility of the findings. Such an approach would allow for a statistically sound comparison of different machine learning methodologies.

In summary, while the research provides valuable insights, future work should focus on expanding the methodological framework to include fine-tuning of pre-trained models and statistical validation of results across multiple runs. These enhancements significantly contribute to the depth, reliability, and applicability of the research, offering a more complete view of the capabilities and limitations of different machine-learning approaches in the field of image classification.

7. REFERENCES

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*, *abs/2010.11929*. https://arxiv.org/abs/2010.11929

Gill, K. S., Anand, V., & Gupta, R. (2023). Arrhythmia Classification Using ECG Image Dataset Using Machine Learning Approach on DenseNet121 Model. *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 1–4. https://doi.org/10.1109/ICCCNT56998.2023.10308001

Hastuti, E. T., Bustamam, A., Anki, P., Amalia, R., & Salma, A. (2021). Performance of True Transfer Learning using CNN DenseNet121 for COVID-19 Detection from Chest X-Ray Images. *2021 IEEE International Conference on Health, Instrumentation & Measurement, and Natural Sciences (InHeNce)*, 1–5. https://doi.org/10.1109/InHeNce52833.2021.9537261

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *CoRR*, *abs/1512.03385*. http://arxiv.org/abs/1512.03385

Huang, G., Liu, Z., & Weinberger, K. Q. (2016). Densely Connected Convolutional Networks. *CoRR*, *abs/1608.06993*. http://arxiv.org/abs/1608.06993

Jiang, A., Yan, N., Wang, F., Huang, H., Zhu, H., & Wei, B. (2019). Visible Image Recognition of Power Transformer Equipment Based on Mask R-CNN. *2019 IEEE Sustainable Power and Energy Conference (ISPEC)*, 657–661. https://doi.org/10.1109/iSPEC48194.2019.8975213

Liu, S., & Deng, W. (2016). Very deep convolutional neural network-based image classification using small training sample size. *Proceedings - 3rd IAPR Asian Conference on Pattern Recognition, ACPR 2015*, 730–734. https://doi.org/10.1109/ACPR.2015.7486599

Shah, S., Qadri, S., Bibi, H., Shah, S., Sharif, M., & Marinello, F. (2023). *Comparing Inception V3, VGG 16, VGG 19, CNN, and ResNet 50: A Case Study on Early Detection of a Rice Disease* .

Tao, J., Gu, Y., Sun, J., Bie, Y., & Wang, H. (2021). Research on vgg16 convolutional neural network feature classification algorithm based on Transfer Learning. *2021 2nd China International SAR Symposium (CISS)*, 1–3. https://doi.org/10.23919/CISS51089.2021.9652277

(Xie, S., (Girshick, R., (Doll´ar, P., & Zhuowen Tu. (2017). *Aggregated Residual Transformations for Deep Neural Networks*.

Zagoruyko, S., & Komodakis, N. (n.d.). *Wide Residual Networks*.

Zakaria, N., Mohamed, F., Abdelghani, R., & Sundaraj, K. (2021). VGG16, ResNet-50, and GoogLeNet Deep Learning Architecture for Breathing Sound Classification: A Comparative Study. *2021 International Conference on Artificial Intelligence for Cyber Security Systems and Privacy (AI-CSP)*, 1–6. https://doi.org/10.1109/AI-CSP52968.2021.9671124

Leonardi M, Napoletano P, Schettini R, Rozza A. No Reference, Opinion Unaware Image Quality Assessment by Anomaly Detection. Sensors (Basel). 2021 Feb 2;21(3):994. doi: 10.3390/s21030994. PMID: 33540652; PMCID: PMC7867270.

Z. Li, F. Liu, W. Yang, S. Peng and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," in IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 12, pp. 6999-7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827.

M. Åkesson, P. Singh, F. Wrede and A. Hellander, "Convolutional Neural Networks as Summary Statistics for Approximate Bayesian Computation," in *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 6, pp. 3353-3365, 1 Nov.-Dec. 2022, doi: 10.1109/TCBB.2021.3108695.

L. Linyun, C. Kaiyan, S. Shijie, Z. Yang, L. Junyan and W. Zhihui, "Parameters selection and optimization in profiled side channel analysis based on convolutional neural network," *2020 IEEE 2nd International Conference on Civil Aviation Safety and Information Technology (ICCASIT*, Weihai, China, 2020, pp. 428-433, doi: 10.1109/ICCASIT50869.2020.9368671.

B. Sahithi and S. Vigneshwari, "Exploring the Performance of VGG16 and Efficient Net Models for Plant Disease Classification: A Comparative Approach," *2023 First International Conference on Cyber Physical Systems, Power Electronics and Electric Vehicles (ICPEEV)*, Hyderabad, India, 2023, pp. 1-6, doi: 10.1109/ICPEEV58650.2023.10391936.

S. Dasari, B. Poonguzhali and M. Rayudu, "Transfer Learning Approach for Classification of Diabetic Retinopathy using Fine-Tuned ResNet50 Deep Learning Model," *2023 International Conference on Sustainable Communication Networks and Application (ICSCNA)*, Theni, India, 2023, pp. 1361-1367, doi: 10.1109/ICSCNA58489.2023.10370255.

E. T. Hastuti, A. Bustamam, P. Anki, R. Amalia and A. Salma, "Performance of True Transfer Learning using CNN DenseNet121 for COVID-19 Detection from Chest X-Ray Images," *2021 IEEE International Conference on Health, Instrumentation & Measurement, and Natural Sciences (InHeNce)*, Medan, Indonesia, 2021, pp. 1-5, doi: 10.1109/InHeNce52833.2021.9537261.

R. Doon, T. Kumar Rawat and S. Gautam, "Cifar-10 Classification using Deep Convolutional Neural Network," *2018 IEEE Punecon*, Pune, India, 2018, pp. 1-5, doi: 10.1109/PUNECON.2018.8745428.

P. S. S. Madhulika and N. Sampath, "An Application of Normalizer Free Neural Networks on the SVHN Dataset," *2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, Salem, India, 2022, pp. 238-242, doi: 10.1109/ICAAIC53929.2022.9793301.

J. Hao, "Deep learning-based medical image analysis with explainable transfer learning," *2023 International Conference on Computer Engineering and Distance Learning (CEDL)*, Shanghai, China, 2023, pp. 106-109, doi: 10.1109/CEDL60560.2023.00029.

Z. Hao, L. Shaohong and S. Jinping, "Unit Model of Binary SVM with DS Output and its Application in Multi-class SVM," *2011 Fourth International Symposium on Computational Intelligence and Design*, Hangzhou, 2011, pp. 101-104, doi: 10.1109/ISCID.2011.34.

APPENDIX

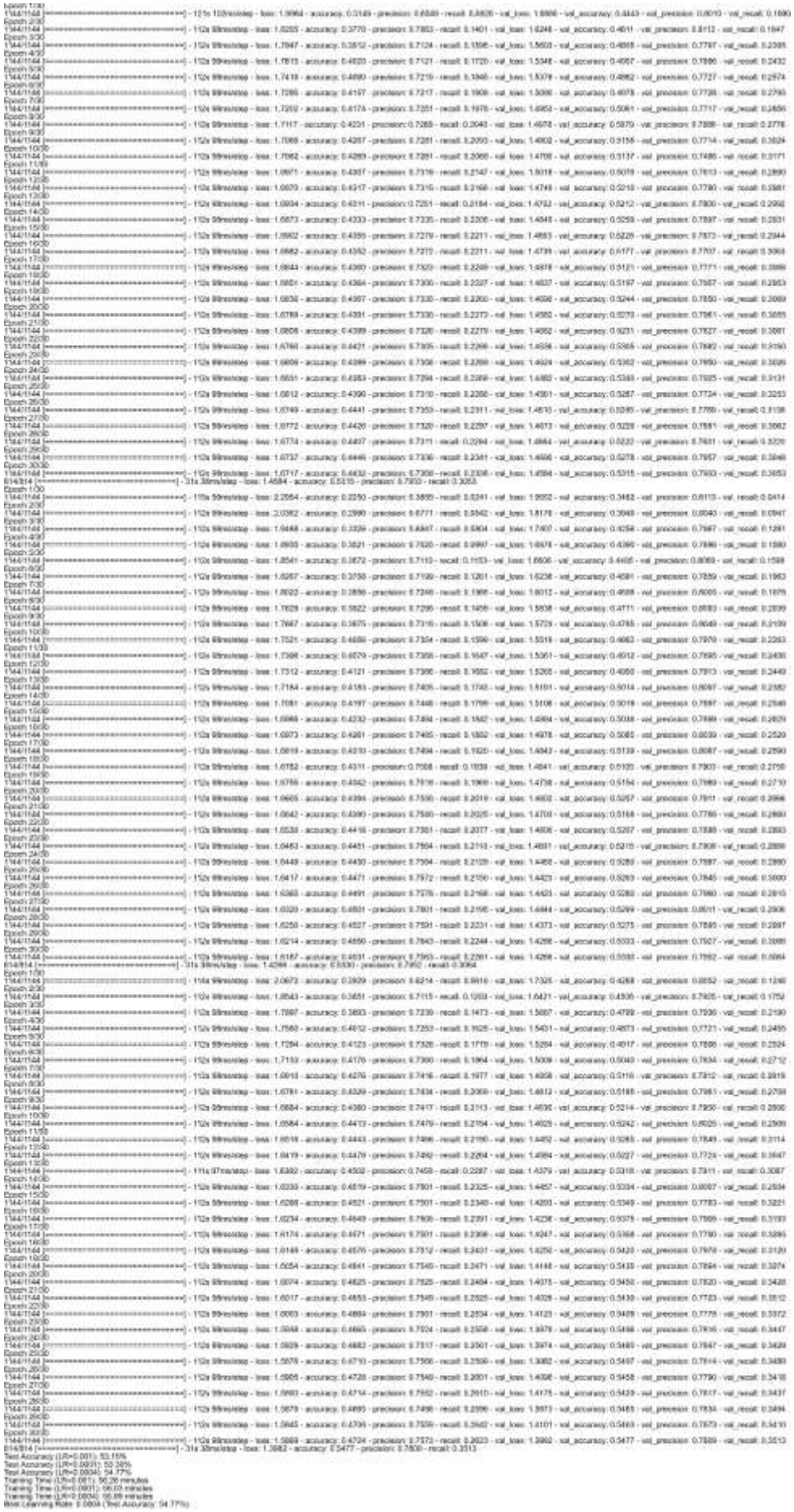
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_3 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_3  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 14848586 (56.64 MB)
Trainable params: 133898 (523.04 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
Epoch 1/30
781/781 [==============================] - 72s 70ms/step - loss: 1.2056 - accuracy: 0.5942 - precision: 0.7465 - recall: 0.4149 - val_loss: 0.8736 - val_accuracy: 0.7050 - val_precision: 0.8063 - val_recall: 0.5945
Epoch 2/30
781/781 [==============================] - 57s 73ms/step - loss: 1.0003 - accuracy: 0.6553 - precision: 0.7734 - recall: 0.5333 - val_loss: 0.8554 - val_accuracy: 0.7063 - val_precision: 0.8010 - val_recall: 0.5990
Epoch 3/30
781/781 [==============================] - 56s 72ms/step - loss: 0.9494 - accuracy: 0.6738 - precision: 0.7836 - recall: 0.5612 - val_loss: 0.7816 - val_accuracy: 0.7307 - val_precision: 0.8368 - val_recall: 0.6340
Epoch 4/30
781/781 [==============================] - 57s 73ms/step - loss: 0.9235 - accuracy: 0.6850 - precision: 0.7880 - recall: 0.5785 - val_loss: 0.7830 - val_accuracy: 0.7417 - val_precision: 0.8348 - val_recall: 0.6428
Epoch 5/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8998 - accuracy: 0.6919 - precision: 0.7940 - recall: 0.5902 - val_loss: 0.7797 - val_accuracy: 0.7436 - val_precision: 0.8339 - val_recall: 0.6522
Epoch 6/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8895 - accuracy: 0.6993 - precision: 0.7975 - recall: 0.6039 - val_loss: 0.7837 - val_accuracy: 0.7502 - val_precision: 0.8284 - val_recall: 0.6579
Epoch 7/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8800 - accuracy: 0.7021 - precision: 0.7987 - recall: 0.6035 - val_loss: 0.7593 - val_accuracy: 0.7513 - val_precision: 0.8241 - val_recall: 0.6680
Epoch 8/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8710 - accuracy: 0.7080 - precision: 0.8032 - recall: 0.6174 - val_loss: 0.7560 - val_accuracy: 0.7547 - val_precision: 0.8248 - val_recall: 0.6689
Epoch 9/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8684 - accuracy: 0.7098 - precision: 0.8041 - recall: 0.6188 - val_loss: 0.7494 - val_accuracy: 0.7542 - val_precision: 0.8365 - val_recall: 0.6703
Epoch 10/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8554 - accuracy: 0.7155 - precision: 0.8087 - recall: 0.6250 - val_loss: 0.7716 - val_accuracy: 0.7401 - val_precision: 0.8161 - val_recall: 0.6787
Epoch 11/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8494 - accuracy: 0.7197 - precision: 0.8103 - recall: 0.6320 - val_loss: 0.7568 - val_accuracy: 0.7525 - val_precision: 0.8213 - val_recall: 0.6867
Epoch 12/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8497 - accuracy: 0.7186 - precision: 0.8082 - recall: 0.6323 - val_loss: 0.7469 - val_accuracy: 0.7578 - val_precision: 0.8245 - val_recall: 0.6932
Epoch 13/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8470 - accuracy: 0.7207 - precision: 0.8093 - recall: 0.6327 - val_loss: 0.7596 - val_accuracy: 0.7506 - val_precision: 0.8256 - val_recall: 0.6920
Epoch 14/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8413 - accuracy: 0.7246 - precision: 0.8113 - recall: 0.6380 - val_loss: 0.7418 - val_accuracy: 0.7583 - val_precision: 0.8280 - val_recall: 0.6914
Epoch 15/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8378 - accuracy: 0.7261 - precision: 0.8127 - recall: 0.6373 - val_loss: 0.7623 - val_accuracy: 0.7502 - val_precision: 0.8294 - val_recall: 0.6887
Epoch 16/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8364 - accuracy: 0.7244 - precision: 0.8133 - recall: 0.6389 - val_loss: 0.7510 - val_accuracy: 0.7513 - val_precision: 0.8283 - val_recall: 0.7031
Epoch 17/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8300 - accuracy: 0.7291 - precision: 0.8143 - recall: 0.6439 - val_loss: 0.7492 - val_accuracy: 0.7516 - val_precision: 0.8258 - val_recall: 0.7049
Epoch 18/30
781/781 [==============================] - 57s 74ms/step - loss: 0.8246 - accuracy: 0.7297 - precision: 0.8149 - recall: 0.6459 - val_loss: 0.7588 - val_accuracy: 0.7506 - val_precision: 0.8261 - val_recall: 0.6979
Epoch 19/30
781/781 [==============================] - 57s 74ms/step - loss: 0.8265 - accuracy: 0.7296 - precision: 0.8152 - recall: 0.6467 - val_loss: 0.7450 - val_accuracy: 0.7540 - val_precision: 0.8315 - val_recall: 0.6942
Epoch 20/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8281 - accuracy: 0.7294 - precision: 0.8154 - recall: 0.6467 - val_loss: 0.7634 - val_accuracy: 0.7494 - val_precision: 0.8272 - val_recall: 0.6999
Epoch 21/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8225 - accuracy: 0.7322 - precision: 0.8143 - recall: 0.6512 - val_loss: 0.7720 - val_accuracy: 0.7474 - val_precision: 0.8198 - val_recall: 0.6931
Epoch 22/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8242 - accuracy: 0.7333 - precision: 0.8188 - recall: 0.6490 - val_loss: 0.7519 - val_accuracy: 0.7541 - val_precision: 0.8289 - val_recall: 0.6975
Epoch 23/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8177 - accuracy: 0.7348 - precision: 0.8197 - recall: 0.6539 - val_loss: 0.7439 - val_accuracy: 0.7561 - val_precision: 0.8277 - val_recall: 0.7103
Epoch 24/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8135 - accuracy: 0.7351 - precision: 0.8212 - recall: 0.6570 - val_loss: 0.7509 - val_accuracy: 0.7574 - val_precision: 0.8308 - val_recall: 0.7067
Epoch 25/30
781/781 [==============================] - 59s 75ms/step - loss: 0.8102 - accuracy: 0.7354 - precision: 0.8175 - recall: 0.6558 - val_loss: 0.7530 - val_accuracy: 0.7519 - val_precision: 0.8237 - val_recall: 0.6979
Epoch 26/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8159 - accuracy: 0.7360 - precision: 0.8186 - recall: 0.6575 - val_loss: 0.7696 - val_accuracy: 0.7521 - val_precision: 0.8225 - val_recall: 0.7037
Epoch 27/30
781/781 [==============================] - 58s 76ms/step - loss: 0.8094 - accuracy: 0.7378 - precision: 0.8187 - recall: 0.6612 - val_loss: 0.7514 - val_accuracy: 0.7602 - val_precision: 0.8344 - val_recall: 0.7011
Epoch 28/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8116 - accuracy: 0.7374 - precision: 0.8187 - recall: 0.6592 - val_loss: 0.7401 - val_accuracy: 0.7716 - val_precision: 0.8361 - val_recall: 0.7038
Epoch 29/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8100 - accuracy: 0.7367 - precision: 0.8209 - recall: 0.6580 - val_loss: 0.7431 - val_accuracy: 0.7690 - val_precision: 0.8261 - val_recall: 0.7193
Epoch 30/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8105 - accuracy: 0.7378 - precision: 0.8201 - recall: 0.6613 - val_loss: 0.7380 - val_accuracy: 0.7763 - val_precision: 0.8341 - val_recall: 0.7253
313/313 [==============================] - 11s 36ms/step - loss: 0.7366 - accuracy: 0.7783 - precision: 0.8341 - recall: 0.7203
Epoch 1/30
781/781 [==============================] - 66s 77ms/step - loss: 1.5455 - accuracy: 0.4661 - precision: 0.7377 - recall: 0.1984 - val_loss: 1.1293 - val_accuracy: 0.6330 - val_precision: 0.8249 - val_recall: 0.4081
Epoch 2/30
781/781 [==============================] - 58s 75ms/step - loss: 1.1947 - accuracy: 0.5921 - precision: 0.7646 - recall: 0.3981 - val_loss: 0.9940 - val_accuracy: 0.6748 - val_precision: 0.8099 - val_recall: 0.5045
Epoch 3/30
781/781 [==============================] - 58s 74ms/step - loss: 1.0849 - accuracy: 0.6292 - precision: 0.7797 - recall: 0.4803 - val_loss: 0.9343 - val_accuracy: 0.6914 - val_precision: 0.8253 - val_recall: 0.5439
Epoch 4/30
781/781 [==============================] - 58s 74ms/step - loss: 1.0377 - accuracy: 0.6456 - precision: 0.7889 - recall: 0.4987 - val_loss: 0.8804 - val_accuracy: 0.7070 - val_precision: 0.8249 - val_recall: 0.5783
Epoch 5/30
781/781 [==============================] - 58s 75ms/step - loss: 0.9902 - accuracy: 0.6615 - precision: 0.7953 - recall: 0.5202 - val_loss: 0.8634 - val_accuracy: 0.7163 - val_precision: 0.8318 - val_recall: 0.5889
Epoch 6/30
781/781 [==============================] - 58s 74ms/step - loss: 0.9650 - accuracy: 0.6713 - precision: 0.7988 - recall: 0.5389 - val_loss: 0.8380 - val_accuracy: 0.7238 - val_precision: 0.8258 - val_recall: 0.6116
Epoch 7/30
781/781 [==============================] - 58s 75ms/step - loss: 0.9433 - accuracy: 0.6788 - precision: 0.8011 - recall: 0.5515 - val_loss: 0.8228 - val_accuracy: 0.7283 - val_precision: 0.8310 - val_recall: 0.6146
Epoch 8/30
781/781 [==============================] - 57s 73ms/step - loss: 0.9253 - accuracy: 0.6857 - precision: 0.8024 - recall: 0.5600 - val_loss: 0.8045 - val_accuracy: 0.7342 - val_precision: 0.8251 - val_recall: 0.6312
Epoch 9/30
781/781 [==============================] - 58s 75ms/step - loss: 0.9015 - accuracy: 0.6956 - precision: 0.8083 - recall: 0.5740 - val_loss: 0.7919 - val_accuracy: 0.7372 - val_precision: 0.8343 - val_recall: 0.6354
Epoch 10/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8915 - accuracy: 0.6992 - precision: 0.8088 - recall: 0.5795 - val_loss: 0.7730 - val_accuracy: 0.7399 - val_precision: 0.8338 - val_recall: 0.6473
Epoch 11/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8890 - accuracy: 0.6977 - precision: 0.8091 - recall: 0.5849 - val_loss: 0.7734 - val_accuracy: 0.7454 - val_precision: 0.8356 - val_recall: 0.6444
Epoch 12/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8849 - accuracy: 0.7063 - precision: 0.8142 - recall: 0.5859 - val_loss: 0.7630 - val_accuracy: 0.7490 - val_precision: 0.8358 - val_recall: 0.6539
Epoch 13/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8851 - accuracy: 0.7053 - precision: 0.8133 - recall: 0.5880 - val_loss: 0.7621 - val_accuracy: 0.7478 - val_precision: 0.8283 - val_recall: 0.6571
Epoch 14/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8527 - accuracy: 0.7121 - precision: 0.8185 - recall: 0.6042 - val_loss: 0.7527 - val_accuracy: 0.7529 - val_precision: 0.8359 - val_recall: 0.6614
Epoch 15/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8471 - accuracy: 0.7139 - precision: 0.8158 - recall: 0.6082 - val_loss: 0.7502 - val_accuracy: 0.7536 - val_precision: 0.8389 - val_recall: 0.6649
Epoch 16/30
781/781 [==============================] - 57s 73ms/step - loss: 0.8356 - accuracy: 0.7161 - precision: 0.8181 - recall: 0.6126 - val_loss: 0.7419 - val_accuracy: 0.7565 - val_precision: 0.8390 - val_recall: 0.6664
Epoch 17/30
781/781 [==============================] - 58s 74ms/step - loss: 0.8297 - accuracy: 0.7167 - precision: 0.8217 - recall: 0.6170 - val_loss: 0.7350 - val_accuracy: 0.7590 - val_precision: 0.8371 - val_recall: 0.6738
Epoch 18/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8299 - accuracy: 0.7186 - precision: 0.8193 - recall: 0.6177 - val_loss: 0.7330 - val_accuracy: 0.7586 - val_precision: 0.8388 - val_recall: 0.6714
Epoch 19/30
781/781 [==============================] - 60s 76ms/step - loss: 0.8191 - accuracy: 0.7200 - precision: 0.8228 - recall: 0.6239 - val_loss: 0.7285 - val_accuracy: 0.7606 - val_precision: 0.8331 - val_recall: 0.6789
Epoch 20/30
781/781 [==============================] - 59s 75ms/step - loss: 0.8101 - accuracy: 0.7263 - precision: 0.8225 - recall: 0.6271 - val_loss: 0.7240 - val_accuracy: 0.7600 - val_precision: 0.8379 - val_recall: 0.6782
Epoch 21/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8033 - accuracy: 0.7270 - precision: 0.8244 - recall: 0.6307 - val_loss: 0.7280 - val_accuracy: 0.7565 - val_precision: 0.8371 - val_recall: 0.6764
Epoch 22/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8035 - accuracy: 0.7290 - precision: 0.8243 - recall: 0.6316 - val_loss: 0.7183 - val_accuracy: 0.7626 - val_precision: 0.8403 - val_recall: 0.6785
Epoch 23/30
781/781 [==============================] - 60s 77ms/step - loss: 0.7932 - accuracy: 0.7312 - precision: 0.8242 - recall: 0.6358 - val_loss: 0.7144 - val_accuracy: 0.7579 - val_precision: 0.8405 - val_recall: 0.6853
Epoch 24/30
781/781 [==============================] - 60s 76ms/step - loss: 0.7920 - accuracy: 0.7318 - precision: 0.8265 - recall: 0.6371 - val_loss: 0.7120 - val_accuracy: 0.7666 - val_precision: 0.8397 - val_recall: 0.6863
Epoch 25/30
781/781 [==============================] - 59s 75ms/step - loss: 0.7857 - accuracy: 0.7336 - precision: 0.8279 - recall: 0.6390 - val_loss: 0.7075 - val_accuracy: 0.7692 - val_precision: 0.8386 - val_recall: 0.6940
Epoch 26/30
781/781 [==============================] - 59s 75ms/step - loss: 0.7782 - accuracy: 0.7364 - precision: 0.8280 - recall: 0.6449 - val_loss: 0.7063 - val_accuracy: 0.7706 - val_precision: 0.8409 - val_recall: 0.6920
Epoch 27/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7741 - accuracy: 0.7356 - precision: 0.8309 - recall: 0.6481 - val_loss: 0.7118 - val_accuracy: 0.7667 - val_precision: 0.8391 - val_recall: 0.6892
Epoch 28/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7725 - accuracy: 0.7368 - precision: 0.8284 - recall: 0.6472 - val_loss: 0.7019 - val_accuracy: 0.7725 - val_precision: 0.8408 - val_recall: 0.6957
Epoch 29/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7716 - accuracy: 0.7389 - precision: 0.8275 - recall: 0.6485 - val_loss: 0.6974 - val_accuracy: 0.7718 - val_precision: 0.8384 - val_recall: 0.6986
Epoch 30/30
781/781 [==============================] - 60s 76ms/step - loss: 0.7694 - accuracy: 0.7421 - precision: 0.8323 - recall: 0.6530 - val_loss: 0.6999 - val_accuracy: 0.7725 - val_precision: 0.8374 - val_recall: 0.7021
313/313 [==============================] - 11s 36ms/step - loss: 0.6969 - accuracy: 0.7725 - precision: 0.8374 - recall: 0.7021
Epoch 1/30
781/781 [==============================] - 68s 78ms/step - loss: 1.2862 - accuracy: 0.5449 - precision: 0.7567 - recall: 0.3074 - val_loss: 0.9480 - val_accuracy: 0.6892 - val_precision: 0.8134 - val_recall: 0.5429
Epoch 2/30
781/781 [==============================] - 58s 75ms/step - loss: 1.0232 - accuracy: 0.6408 - precision: 0.7785 - recall: 0.5093 - val_loss: 0.8501 - val_accuracy: 0.7109 - val_precision: 0.8293 - val_recall: 0.5989
Epoch 3/30
781/781 [==============================] - 58s 75ms/step - loss: 0.9553 - accuracy: 0.6734 - precision: 0.7969 - recall: 0.5529 - val_loss: 0.8070 - val_accuracy: 0.7346 - val_precision: 0.8334 - val_recall: 0.6219
Epoch 4/30
781/781 [==============================] - 59s 76ms/step - loss: 0.9184 - accuracy: 0.6870 - precision: 0.7979 - recall: 0.5749 - val_loss: 0.7972 - val_accuracy: 0.7298 - val_precision: 0.8152 - val_recall: 0.6279
Epoch 5/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8926 - accuracy: 0.6958 - precision: 0.7993 - recall: 0.5874 - val_loss: 0.7732 - val_accuracy: 0.7421 - val_precision: 0.8237 - val_recall: 0.6583
Epoch 6/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8670 - accuracy: 0.7056 - precision: 0.8057 - recall: 0.6033 - val_loss: 0.7642 - val_accuracy: 0.7459 - val_precision: 0.8244 - val_recall: 0.6621
Epoch 7/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8562 - accuracy: 0.7077 - precision: 0.8084 - recall: 0.6103 - val_loss: 0.7559 - val_accuracy: 0.7442 - val_precision: 0.8288 - val_recall: 0.6682
Epoch 8/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8467 - accuracy: 0.7126 - precision: 0.8092 - recall: 0.6171 - val_loss: 0.7523 - val_accuracy: 0.7460 - val_precision: 0.8224 - val_recall: 0.6779
Epoch 9/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8399 - accuracy: 0.7162 - precision: 0.8128 - recall: 0.6233 - val_loss: 0.7436 - val_accuracy: 0.7502 - val_precision: 0.8265 - val_recall: 0.6833
Epoch 10/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8321 - accuracy: 0.7242 - precision: 0.8148 - recall: 0.6319 - val_loss: 0.7230 - val_accuracy: 0.7615 - val_precision: 0.8369 - val_recall: 0.6939
Epoch 11/30
781/781 [==============================] - 59s 76ms/step - loss: 0.8246 - accuracy: 0.7258 - precision: 0.8157 - recall: 0.6380 - val_loss: 0.7177 - val_accuracy: 0.7662 - val_precision: 0.8418 - val_recall: 0.6927
Epoch 12/30
781/781 [==============================] - 58s 75ms/step - loss: 0.8237 - accuracy: 0.7305 - precision: 0.8202 - recall: 0.6400 - val_loss: 0.7241 - val_accuracy: 0.7620 - val_precision: 0.8296 - val_recall: 0.6943
Epoch 13/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7989 - accuracy: 0.7319 - precision: 0.8189 - recall: 0.6433 - val_loss: 0.7181 - val_accuracy: 0.7619 - val_precision: 0.8358 - val_recall: 0.6949
Epoch 14/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7875 - accuracy: 0.7378 - precision: 0.8241 - recall: 0.6522 - val_loss: 0.7141 - val_accuracy: 0.7673 - val_precision: 0.8386 - val_recall: 0.6993
Epoch 15/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7804 - accuracy: 0.7366 - precision: 0.8253 - recall: 0.6536 - val_loss: 0.7307 - val_accuracy: 0.7626 - val_precision: 0.8282 - val_recall: 0.6959
Epoch 16/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7776 - accuracy: 0.7414 - precision: 0.8259 - recall: 0.6583 - val_loss: 0.7137 - val_accuracy: 0.7673 - val_precision: 0.8268 - val_recall: 0.7099
Epoch 17/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7741 - accuracy: 0.7408 - precision: 0.8240 - recall: 0.6580 - val_loss: 0.6990 - val_accuracy: 0.7731 - val_precision: 0.8349 - val_recall: 0.7150
Epoch 18/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7642 - accuracy: 0.7457 - precision: 0.8296 - recall: 0.6647 - val_loss: 0.7115 - val_accuracy: 0.7726 - val_precision: 0.8389 - val_recall: 0.7080
Epoch 19/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7617 - accuracy: 0.7468 - precision: 0.8304 - recall: 0.6665 - val_loss: 0.7099 - val_accuracy: 0.7703 - val_precision: 0.8374 - val_recall: 0.7092
Epoch 20/30
781/781 [==============================] - 60s 76ms/step - loss: 0.7594 - accuracy: 0.7493 - precision: 0.8310 - recall: 0.6670 - val_loss: 0.6952 - val_accuracy: 0.7769 - val_precision: 0.8393 - val_recall: 0.7189
Epoch 21/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7559 - accuracy: 0.7472 - precision: 0.8307 - recall: 0.6716 - val_loss: 0.7085 - val_accuracy: 0.7684 - val_precision: 0.8290 - val_recall: 0.7050
Epoch 22/30
781/781 [==============================] - 60s 77ms/step - loss: 0.7496 - accuracy: 0.7517 - precision: 0.8308 - recall: 0.6713 - val_loss: 0.7009 - val_accuracy: 0.7705 - val_precision: 0.8328 - val_recall: 0.7144
Epoch 23/30
781/781 [==============================] - 60s 77ms/step - loss: 0.7426 - accuracy: 0.7549 - precision: 0.8340 - recall: 0.6770 - val_loss: 0.7101 - val_accuracy: 0.7707 - val_precision: 0.8282 - val_recall: 0.7182
Epoch 24/30
781/781 [==============================] - 59s 75ms/step - loss: 0.7413 - accuracy: 0.7548 - precision: 0.8343 - recall: 0.6782 - val_loss: 0.7024 - val_accuracy: 0.7760 - val_precision: 0.8364 - val_recall: 0.7183
Epoch 25/30
781/781 [==============================] - 60s 76ms/step - loss: 0.7371 - accuracy: 0.7567 - precision: 0.8342 - recall: 0.6819 - val_loss: 0.7094 - val_accuracy: 0.7718 - val_precision: 0.8308 - val_recall: 0.7197
Epoch 26/30
781/781 [==============================] - 60s 77ms/step - loss: 0.7349 - accuracy: 0.7563 - precision: 0.8354 - recall: 0.6821 - val_loss: 0.6914 - val_accuracy: 0.7816 - val_precision: 0.8407 - val_recall: 0.7182
Epoch 27/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7325 - accuracy: 0.7598 - precision: 0.8379 - recall: 0.6852 - val_loss: 0.7004 - val_accuracy: 0.7748 - val_precision: 0.8353 - val_recall: 0.7159
Epoch 28/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7274 - accuracy: 0.7606 - precision: 0.8375 - recall: 0.6889 - val_loss: 0.7009 - val_accuracy: 0.7764 - val_precision: 0.8357 - val_recall: 0.7199
Epoch 29/30
781/781 [==============================] - 59s 76ms/step - loss: 0.7237 - accuracy: 0.7590 - precision: 0.8368 - recall: 0.6849 - val_loss: 0.7040 - val_accuracy: 0.7759 - val_precision: 0.8298 - val_recall: 0.7265
Epoch 30/30
781/781 [==============================] - 58s 75ms/step - loss: 0.7201 - accuracy: 0.7615 - precision: 0.8375 - recall: 0.6884 - val_loss: 0.6921 - val_accuracy: 0.7826 - val_precision: 0.8419 - val_recall: 0.7231
313/313 [==============================] - 11s 36ms/step - loss: 0.6921 - accuracy: 0.7829 - precision: 0.8416 - recall: 0.7231
Test Accuracy (LR=0.001): 77.83%
Test Accuracy (LR=0.0001): 77.29%
Test Accuracy (LR=0.0004): 78.29%
Training Time (LR=0.001): 39.72 minutes
Training Time (LR=0.0001): 34.18 minutes
Training Time (LR=0.0004): 33.48 minutes
Best Learning Rate: 0.0004 (Test Accuracy: 78.29%)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_2 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 densenet121 (Functional)    (None, 7, 7, 1024)        7037504

 global_average_pooling2d_2  (None, 1024)              0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               262400

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 7302474 (27.86 MB)
Trainable params: 264970 (1.01 MB)
Non-trainable params: 7037504 (26.85 MB)
_________________________________________________________________
```

```
Epoch 1/30
781/781 [==============================] - 68s 83ms/step - loss: 0.6537 - accuracy: 0.6927 - precision: 0.7656 - recall: 0.5854 - val_loss: 0.6806 - val_accuracy: 0.7717 - val_precision: 0.8263 - val_recall: 0.7224
Epoch 2/30
781/781 [==============================] - 63s 80ms/step - loss: 0.7675 - accuracy: 0.7468 - precision: 0.8223 - recall: 0.6739 - val_loss: 0.8650 - val_accuracy: 0.8032 - val_precision: 0.8478 - val_recall: 0.7635
Epoch 3/30
781/781 [==============================] - 63s 80ms/step - loss: 0.7184 - accuracy: 0.7636 - precision: 0.8320 - recall: 0.7082 - val_loss: 0.5857 - val_accuracy: 0.8119 - val_precision: 0.8527 - val_recall: 0.7748
Epoch 4/30
781/781 [==============================] - 63s 81ms/step - loss: 0.7059 - accuracy: 0.7692 - precision: 0.8378 - recall: 0.7084 - val_loss: 0.5947 - val_accuracy: 0.8104 - val_precision: 0.8503 - val_recall: 0.7682
Epoch 5/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6894 - accuracy: 0.7762 - precision: 0.8433 - recall: 0.7182 - val_loss: 0.5790 - val_accuracy: 0.8226 - val_precision: 0.8614 - val_recall: 0.7852
Epoch 6/30
781/781 [==============================] - 64s 81ms/step - loss: 0.6765 - accuracy: 0.7819 - precision: 0.8435 - recall: 0.7243 - val_loss: 0.5798 - val_accuracy: 0.8212 - val_precision: 0.8674 - val_recall: 0.7827
Epoch 7/30
781/781 [==============================] - 62s 79ms/step - loss: 0.6689 - accuracy: 0.7861 - precision: 0.8493 - recall: 0.7337 - val_loss: 0.5680 - val_accuracy: 0.8255 - val_precision: 0.8604 - val_recall: 0.7939
Epoch 8/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6591 - accuracy: 0.7916 - precision: 0.8520 - recall: 0.7376 - val_loss: 0.5726 - val_accuracy: 0.8263 - val_precision: 0.8671 - val_recall: 0.7913
Epoch 9/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6584 - accuracy: 0.7919 - precision: 0.8507 - recall: 0.7385 - val_loss: 0.5604 - val_accuracy: 0.8310 - val_precision: 0.8706 - val_recall: 0.7979
Epoch 10/30
781/781 [==============================] - 64s 81ms/step - loss: 0.6457 - accuracy: 0.7992 - precision: 0.8548 - recall: 0.7471 - val_loss: 0.5659 - val_accuracy: 0.8303 - val_precision: 0.8708 - val_recall: 0.7919
Epoch 11/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6439 - accuracy: 0.8002 - precision: 0.8572 - recall: 0.7485 - val_loss: 0.5725 - val_accuracy: 0.8324 - val_precision: 0.8668 - val_recall: 0.8012
Epoch 12/30
781/781 [==============================] - 64s 82ms/step - loss: 0.6390 - accuracy: 0.8030 - precision: 0.8586 - recall: 0.7543 - val_loss: 0.5674 - val_accuracy: 0.8208 - val_precision: 0.8581 - val_recall: 0.7920
Epoch 13/30
781/781 [==============================] - 64s 81ms/step - loss: 0.6365 - accuracy: 0.8044 - precision: 0.8597 - recall: 0.7548 - val_loss: 0.6013 - val_accuracy: 0.8268 - val_precision: 0.8583 - val_recall: 0.8013
Epoch 14/30
781/781 [==============================] - 64s 82ms/step - loss: 0.6423 - accuracy: 0.8009 - precision: 0.8584 - recall: 0.7523 - val_loss: 0.5746 - val_accuracy: 0.8309 - val_precision: 0.8733 - val_recall: 0.7965
Epoch 15/30
781/781 [==============================] - 62s 79ms/step - loss: 0.6306 - accuracy: 0.8055 - precision: 0.8584 - recall: 0.7570 - val_loss: 0.5864 - val_accuracy: 0.8318 - val_precision: 0.8668 - val_recall: 0.8029
Epoch 16/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6294 - accuracy: 0.8097 - precision: 0.8621 - recall: 0.7612 - val_loss: 0.5841 - val_accuracy: 0.8378 - val_precision: 0.8768 - val_recall: 0.8028
Epoch 17/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6206 - accuracy: 0.8118 - precision: 0.8635 - recall: 0.7644 - val_loss: 0.5648 - val_accuracy: 0.8371 - val_precision: 0.8709 - val_recall: 0.8063
Epoch 18/30
781/781 [==============================] - 63s 80ms/step - loss: 0.6201 - accuracy: 0.8093 - precision: 0.8633 - recall: 0.7624 - val_loss: 0.5842 - val_accuracy: 0.8201 - val_precision: 0.8638 - val_recall: 0.8020
Epoch 19/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6231 - accuracy: 0.8106 - precision: 0.8632 - recall: 0.7632 - val_loss: 0.5614 - val_accuracy: 0.8307 - val_precision: 0.8708 - val_recall: 0.8094
Epoch 20/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6130 - accuracy: 0.8162 - precision: 0.8661 - recall: 0.7698 - val_loss: 0.5835 - val_accuracy: 0.8361 - val_precision: 0.8683 - val_recall: 0.8190
Epoch 21/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6236 - accuracy: 0.8142 - precision: 0.8649 - recall: 0.7677 - val_loss: 0.5766 - val_accuracy: 0.8368 - val_precision: 0.8729 - val_recall: 0.8083
Epoch 22/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6190 - accuracy: 0.8164 - precision: 0.8663 - recall: 0.7699 - val_loss: 0.5736 - val_accuracy: 0.8345 - val_precision: 0.8735 - val_recall: 0.8039
Epoch 23/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6171 - accuracy: 0.8148 - precision: 0.8649 - recall: 0.7685 - val_loss: 0.5837 - val_accuracy: 0.8301 - val_precision: 0.8655 - val_recall: 0.8113
Epoch 24/30
781/781 [==============================] - 62s 80ms/step - loss: 0.6190 - accuracy: 0.8154 - precision: 0.8658 - recall: 0.7682 - val_loss: 0.5839 - val_accuracy: 0.8298 - val_precision: 0.8665 - val_recall: 0.7998
Epoch 25/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6161 - accuracy: 0.8151 - precision: 0.8651 - recall: 0.7715 - val_loss: 0.5815 - val_accuracy: 0.8341 - val_precision: 0.8686 - val_recall: 0.8067
Epoch 26/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6184 - accuracy: 0.8156 - precision: 0.8653 - recall: 0.7703 - val_loss: 0.5842 - val_accuracy: 0.8339 - val_precision: 0.8689 - val_recall: 0.8024
Epoch 27/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6204 - accuracy: 0.8166 - precision: 0.8664 - recall: 0.7731 - val_loss: 0.5861 - val_accuracy: 0.8335 - val_precision: 0.8712 - val_recall: 0.8029
Epoch 28/30
781/781 [==============================] - 62s 79ms/step - loss: 0.6190 - accuracy: 0.8164 - precision: 0.8668 - recall: 0.7717 - val_loss: 0.5794 - val_accuracy: 0.8373 - val_precision: 0.8668 - val_recall: 0.8139
Epoch 29/30
781/781 [==============================] - 62s 79ms/step - loss: 0.6176 - accuracy: 0.8165 - precision: 0.8668 - recall: 0.7735 - val_loss: 0.5863 - val_accuracy: 0.8343 - val_precision: 0.8636 - val_recall: 0.8101
Epoch 30/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6091 - accuracy: 0.8208 - precision: 0.8677 - recall: 0.7774 - val_loss: 0.5862 - val_accuracy: 0.8351 - val_precision: 0.8665 - val_recall: 0.8059
313/313 [==============================] - 10s 33ms/step - loss: 0.5862 - accuracy: 0.8351 - precision: 0.8665 - recall: 0.8059
Epoch 1/30
781/781 [==============================] - 68s 83ms/step - loss: 1.1356 - accuracy: 0.6206 - precision: 0.7751 - recall: 0.4794 - val_loss: 0.7230 - val_accuracy: 0.7054 - val_precision: 0.8468 - val_recall: 0.6851
Epoch 2/30
781/781 [==============================] - 63s 81ms/step - loss: 0.8316 - accuracy: 0.7253 - precision: 0.8194 - recall: 0.6590 - val_loss: 0.6496 - val_accuracy: 0.7907 - val_precision: 0.8505 - val_recall: 0.7277
Epoch 3/30
781/781 [==============================] - 64s 82ms/step - loss: 0.7529 - accuracy: 0.7527 - precision: 0.8341 - recall: 0.6722 - val_loss: 0.6134 - val_accuracy: 0.8014 - val_precision: 0.8529 - val_recall: 0.7501
Epoch 4/30
781/781 [==============================] - 63s 81ms/step - loss: 0.7099 - accuracy: 0.7681 - precision: 0.8432 - recall: 0.6867 - val_loss: 0.5753 - val_accuracy: 0.8149 - val_precision: 0.8648 - val_recall: 0.7613
Epoch 5/30
781/781 [==============================] - 62s 80ms/step - loss: 0.6762 - accuracy: 0.7793 - precision: 0.8513 - recall: 0.7139 - val_loss: 0.5870 - val_accuracy: 0.8102 - val_precision: 0.8604 - val_recall: 0.7722
Epoch 6/30
781/781 [==============================] - 63s 81ms/step - loss: 0.6403 - accuracy: 0.7904 - precision: 0.8583 - recall: 0.7280 - val_loss: 0.5502 - val_accuracy: 0.8227 - val_precision: 0.8665 - val_recall: 0.7823
Epoch 7/30
781/781 [==============================] - 65s 84ms/step - loss: 0.6229 - accuracy: 0.7951 - precision: 0.8625 - recall: 0.7377 - val_loss: 0.5309 - val_accuracy: 0.8286 - val_precision: 0.8709 - val_recall: 0.7889
Epoch 8/30
781/781 [==============================] - 66s 85ms/step - loss: 0.6071 - accuracy: 0.8031 - precision: 0.8669 - recall: 0.7464 - val_loss: 0.5352 - val_accuracy: 0.8277 - val_precision: 0.8665 - val_recall: 0.7884
Epoch 9/30
781/781 [==============================] - 66s 85ms/step - loss: 0.5899 - accuracy: 0.8092 - precision: 0.8670 - recall: 0.7534 - val_loss: 0.5182 - val_accuracy: 0.8315 - val_precision: 0.8735 - val_recall: 0.7968
Epoch 10/30
781/781 [==============================] - 65s 84ms/step - loss: 0.5774 - accuracy: 0.8156 - precision: 0.8721 - recall: 0.7611 - val_loss: 0.5174 - val_accuracy: 0.8387 - val_precision: 0.8720 - val_recall: 0.8004
Epoch 11/30
781/781 [==============================] - 65s 83ms/step - loss: 0.5657 - accuracy: 0.8173 - precision: 0.8735 - recall: 0.7651 - val_loss: 0.5047 - val_accuracy: 0.8406 - val_precision: 0.8765 - val_recall: 0.8061
Epoch 12/30
781/781 [==============================] - 65s 83ms/step - loss: 0.5493 - accuracy: 0.8250 - precision: 0.8778 - recall: 0.7732 - val_loss: 0.4993 - val_accuracy: 0.8396 - val_precision: 0.8753 - val_recall: 0.8075
Epoch 13/30
781/781 [==============================] - 65s 83ms/step - loss: 0.5407 - accuracy: 0.8261 - precision: 0.8788 - recall: 0.7759 - val_loss: 0.4965 - val_accuracy: 0.8418 - val_precision: 0.8774 - val_recall: 0.8080
Epoch 14/30
781/781 [==============================] - 66s 85ms/step - loss: 0.5344 - accuracy: 0.8298 - precision: 0.8788 - recall: 0.7801 - val_loss: 0.4909 - val_accuracy: 0.8429 - val_precision: 0.8788 - val_recall: 0.8139
Epoch 15/30
781/781 [==============================] - 66s 85ms/step - loss: 0.5222 - accuracy: 0.8336 - precision: 0.8823 - recall: 0.7863 - val_loss: 0.4834 - val_accuracy: 0.8436 - val_precision: 0.8799 - val_recall: 0.8113
Epoch 16/30
781/781 [==============================] - 62s 79ms/step - loss: 0.5069 - accuracy: 0.8372 - precision: 0.8860 - recall: 0.7910 - val_loss: 0.4846 - val_accuracy: 0.8452 - val_precision: 0.8778 - val_recall: 0.8145
Epoch 17/30
781/781 [==============================] - 65s 84ms/step - loss: 0.5077 - accuracy: 0.8365 - precision: 0.8861 - recall: 0.7922 - val_loss: 0.4826 - val_accuracy: 0.8472 - val_precision: 0.8803 - val_recall: 0.8187
Epoch 18/30
781/781 [==============================] - 65s 84ms/step - loss: 0.4976 - accuracy: 0.8418 - precision: 0.8883 - recall: 0.7984 - val_loss: 0.4797 - val_accuracy: 0.8514 - val_precision: 0.8849 - val_recall: 0.8259
Epoch 19/30
781/781 [==============================] - 66s 85ms/step - loss: 0.4876 - accuracy: 0.8462 - precision: 0.8929 - recall: 0.8030 - val_loss: 0.4799 - val_accuracy: 0.8496 - val_precision: 0.8855 - val_recall: 0.8199
Epoch 20/30
781/781 [==============================] - 66s 84ms/step - loss: 0.4831 - accuracy: 0.8470 - precision: 0.8921 - recall: 0.8043 - val_loss: 0.4827 - val_accuracy: 0.8475 - val_precision: 0.8799 - val_recall: 0.8181
Epoch 21/30
781/781 [==============================] - 66s 84ms/step - loss: 0.4758 - accuracy: 0.8502 - precision: 0.8952 - recall: 0.8082 - val_loss: 0.4733 - val_accuracy: 0.8502 - val_precision: 0.8827 - val_recall: 0.8229
Epoch 22/30
781/781 [==============================] - 65s 83ms/step - loss: 0.4695 - accuracy: 0.8522 - precision: 0.8986 - recall: 0.8105 - val_loss: 0.4747 - val_accuracy: 0.8515 - val_precision: 0.8818 - val_recall: 0.8240
Epoch 23/30
781/781 [==============================] - 65s 83ms/step - loss: 0.4665 - accuracy: 0.8538 - precision: 0.8963 - recall: 0.8125 - val_loss: 0.4815 - val_accuracy: 0.8495 - val_precision: 0.8829 - val_recall: 0.8234
Epoch 24/30
781/781 [==============================] - 66s 85ms/step - loss: 0.4639 - accuracy: 0.8558 - precision: 0.8982 - recall: 0.8155 - val_loss: 0.4708 - val_accuracy: 0.8526 - val_precision: 0.8821 - val_recall: 0.8284
Epoch 25/30
781/781 [==============================] - 66s 84ms/step - loss: 0.4532 - accuracy: 0.8598 - precision: 0.9011 - recall: 0.8202 - val_loss: 0.4758 - val_accuracy: 0.8503 - val_precision: 0.8799 - val_recall: 0.8252
Epoch 26/30
781/781 [==============================] - 66s 84ms/step - loss: 0.4465 - accuracy: 0.8608 - precision: 0.9026 - recall: 0.8231 - val_loss: 0.4671 - val_accuracy: 0.8550 - val_precision: 0.8824 - val_recall: 0.8293
Epoch 27/30
781/781 [==============================] - 65s 83ms/step - loss: 0.4429 - accuracy: 0.8640 - precision: 0.9044 - recall: 0.8258 - val_loss: 0.4711 - val_accuracy: 0.8564 - val_precision: 0.8799 - val_recall: 0.8257
Epoch 28/30
781/781 [==============================] - 65s 84ms/step - loss: 0.4415 - accuracy: 0.8638 - precision: 0.9056 - recall: 0.8250 - val_loss: 0.4733 - val_accuracy: 0.8558 - val_precision: 0.8835 - val_recall: 0.8334
Epoch 29/30
781/781 [==============================] - 64s 83ms/step - loss: 0.4372 - accuracy: 0.8656 - precision: 0.9050 - recall: 0.8279 - val_loss: 0.4760 - val_accuracy: 0.8506 - val_precision: 0.8798 - val_recall: 0.8260
Epoch 30/30
781/781 [==============================] - 66s 84ms/step - loss: 0.4314 - accuracy: 0.8676 - precision: 0.9048 - recall: 0.8299 - val_loss: 0.4690 - val_accuracy: 0.8522 - val_precision: 0.8801 - val_recall: 0.8313
313/313 [==============================] - 11s 36ms/step - loss: 0.4690 - accuracy: 0.8522 - precision: 0.8801 - recall: 0.8313
Epoch 1/30
781/781 [==============================] - 72s 86ms/step - loss: 0.6776 - accuracy: 0.6756 - precision: 0.7692 - recall: 0.5684 - val_loss: 0.6413 - val_accuracy: 0.7903 - val_precision: 0.8513 - val_recall: 0.7354
Epoch 2/30
781/781 [==============================] - 66s 84ms/step - loss: 0.7507 - accuracy: 0.7514 - precision: 0.8278 - recall: 0.6783 - val_loss: 0.5885 - val_accuracy: 0.8102 - val_precision: 0.8565 - val_recall: 0.7623
Epoch 3/30
781/781 [==============================] - 68s 84ms/step - loss: 0.6896 - accuracy: 0.7742 - precision: 0.8438 - recall: 0.7137 - val_loss: 0.5713 - val_accuracy: 0.8101 - val_precision: 0.8568 - val_recall: 0.7725
Epoch 4/30
781/781 [==============================] - 66s 84ms/step - loss: 0.6599 - accuracy: 0.7849 - precision: 0.8464 - recall: 0.7258 - val_loss: 0.5402 - val_accuracy: 0.8257 - val_precision: 0.8737 - val_recall: 0.7742
Epoch 5/30
781/781 [==============================] - 65s 83ms/step - loss: 0.6243 - accuracy: 0.7948 - precision: 0.8560 - recall: 0.7381 - val_loss: 0.5499 - val_accuracy: 0.8222 - val_precision: 0.8617 - val_recall: 0.7898
Epoch 6/30
781/781 [==============================] - 66s 84ms/step - loss: 0.6126 - accuracy: 0.8006 - precision: 0.8587 - recall: 0.7467 - val_loss: 0.5142 - val_accuracy: 0.8366 - val_precision: 0.8763 - val_recall: 0.8012
Epoch 7/30
781/781 [==============================] - 65s 83ms/step - loss: 0.5896 - accuracy: 0.8097 - precision: 0.8637 - recall: 0.7582 - val_loss: 0.5243 - val_accuracy: 0.8326 - val_precision: 0.8689 - val_recall: 0.8047
Epoch 8/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5814 - accuracy: 0.8134 - precision: 0.8648 - recall: 0.7644 - val_loss: 0.5171 - val_accuracy: 0.8368 - val_precision: 0.8765 - val_recall: 0.8079
Epoch 9/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5670 - accuracy: 0.8187 - precision: 0.8689 - recall: 0.7708 - val_loss: 0.5128 - val_accuracy: 0.8366 - val_precision: 0.8754 - val_recall: 0.8043
Epoch 10/30
781/781 [==============================] - 62s 80ms/step - loss: 0.5569 - accuracy: 0.8225 - precision: 0.8717 - recall: 0.7748 - val_loss: 0.5118 - val_accuracy: 0.8385 - val_precision: 0.8736 - val_recall: 0.8076
Epoch 11/30
781/781 [==============================] - 61s 78ms/step - loss: 0.5519 - accuracy: 0.8222 - precision: 0.8717 - recall: 0.7774 - val_loss: 0.5132 - val_accuracy: 0.8409 - val_precision: 0.8779 - val_recall: 0.8112
Epoch 12/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5407 - accuracy: 0.8262 - precision: 0.8774 - recall: 0.7841 - val_loss: 0.5075 - val_accuracy: 0.8442 - val_precision: 0.8765 - val_recall: 0.8156
Epoch 13/30
781/781 [==============================] - 62s 80ms/step - loss: 0.5367 - accuracy: 0.8268 - precision: 0.8771 - recall: 0.7879 - val_loss: 0.5237 - val_accuracy: 0.8409 - val_precision: 0.8739 - val_recall: 0.8089
Epoch 14/30
781/781 [==============================] - 62s 79ms/step - loss: 0.5325 - accuracy: 0.8316 - precision: 0.8771 - recall: 0.7990 - val_loss: 0.5230 - val_accuracy: 0.8407 - val_precision: 0.8701 - val_recall: 0.8150
Epoch 15/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5246 - accuracy: 0.8306 - precision: 0.8804 - recall: 0.7963 - val_loss: 0.5374 - val_accuracy: 0.8361 - val_precision: 0.8703 - val_recall: 0.8129
Epoch 16/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5183 - accuracy: 0.8373 - precision: 0.8818 - recall: 0.7976 - val_loss: 0.5135 - val_accuracy: 0.8400 - val_precision: 0.8786 - val_recall: 0.8233
Epoch 17/30
781/781 [==============================] - 62s 79ms/step - loss: 0.5190 - accuracy: 0.8378 - precision: 0.8814 - recall: 0.7999 - val_loss: 0.5117 - val_accuracy: 0.8460 - val_precision: 0.8751 - val_recall: 0.8219
Epoch 18/30
781/781 [==============================] - 62s 79ms/step - loss: 0.5108 - accuracy: 0.8407 - precision: 0.8839 - recall: 0.8036 - val_loss: 0.5170 - val_accuracy: 0.8400 - val_precision: 0.8755 - val_recall: 0.8249
Epoch 19/30
781/781 [==============================] - 62s 80ms/step - loss: 0.5113 - accuracy: 0.8430 - precision: 0.8836 - recall: 0.8043 - val_loss: 0.5177 - val_accuracy: 0.8515 - val_precision: 0.8793 - val_recall: 0.8246
Epoch 20/30
781/781 [==============================] - 63s 81ms/step - loss: 0.5031 - accuracy: 0.8458 - precision: 0.8880 - recall: 0.8074 - val_loss: 0.5280 - val_accuracy: 0.8466 - val_precision: 0.8701 - val_recall: 0.8254
Epoch 21/30
781/781 [==============================] - 63s 80ms/step - loss: 0.5020 - accuracy: 0.8458 - precision: 0.8871 - recall: 0.8104 - val_loss: 0.5221 - val_accuracy: 0.8469 - val_precision: 0.8756 - val_recall: 0.8241
Epoch 22/30
781/781 [==============================] - 63s 80ms/step - loss: 0.4993 - accuracy: 0.8504 - precision: 0.8901 - recall: 0.8144 - val_loss: 0.5126 - val_accuracy: 0.8494 - val_precision: 0.8789 - val_recall: 0.8297
Epoch 23/30
781/781 [==============================] - 63s 80ms/step - loss: 0.4893 - accuracy: 0.8524 - precision: 0.8903 - recall: 0.8175 - val_loss: 0.5219 - val_accuracy: 0.8494 - val_precision: 0.8727 - val_recall: 0.8264
Epoch 24/30
781/781 [==============================] - 63s 80ms/step - loss: 0.4847 - accuracy: 0.8524 - precision: 0.8901 - recall: 0.8177 - val_loss: 0.5223 - val_accuracy: 0.8501 - val_precision: 0.8786 - val_recall: 0.8284
Epoch 25/30
781/781 [==============================] - 62s 79ms/step - loss: 0.4883 - accuracy: 0.8509 - precision: 0.8886 - recall: 0.8162 - val_loss: 0.5405 - val_accuracy: 0.8456 - val_precision: 0.8709 - val_recall: 0.8277
Epoch 26/30
781/781 [==============================] - 63s 80ms/step - loss: 0.4826 - accuracy: 0.8542 - precision: 0.8930 - recall: 0.8230 - val_loss: 0.5222 - val_accuracy: 0.8510 - val_precision: 0.8762 - val_recall: 0.8294
Epoch 27/30
781/781 [==============================] - 63s 81ms/step - loss: 0.4831 - accuracy: 0.8550 - precision: 0.8917 - recall: 0.8210 - val_loss: 0.5236 - val_accuracy: 0.8484 - val_precision: 0.8743 - val_recall: 0.8293
Epoch 28/30
781/781 [==============================] - 62s 79ms/step - loss: 0.4796 - accuracy: 0.8575 - precision: 0.8943 - recall: 0.8249 - val_loss: 0.5309 - val_accuracy: 0.8464 - val_precision: 0.8709 - val_recall: 0.8232
Epoch 29/30
781/781 [==============================] - 63s 81ms/step - loss: 0.4792 - accuracy: 0.8568 - precision: 0.8933 - recall: 0.8240 - val_loss: 0.5336 - val_accuracy: 0.8467 - val_precision: 0.8730 - val_recall: 0.8289
Epoch 30/30
781/781 [==============================] - 62s 79ms/step - loss: 0.4727 - accuracy: 0.8601 - precision: 0.8964 - recall: 0.8273 - val_loss: 0.5277 - val_accuracy: 0.8500 - val_precision: 0.8761 - val_recall: 0.8281
313/313 [==============================] - 10s 33ms/step - loss: 0.5277 - accuracy: 0.8500 - precision: 0.8761 - recall: 0.8281
Test Accuracy (LR=0.001): 83.51%
Test Accuracy (LR=0.0001): 85.22%
Test Accuracy (LR=0.0004): 85.00%
Training Time (LR=0.001): 35.05 minutes
Training Time (LR=0.0001): 35.34 minutes
Training Time (LR=0.0004): 36.48 minutes
Best Learning Rate: 0.0001 (Test Accuracy: 85.22%)
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_5 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d_5  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_10 (Dense)            (None, 256)               524544

 dropout_5 (Dropout)         (None, 256)               0

 dense_11 (Dense)            (None, 10)                2570

=================================================================
Total params: 24114826 (91.99 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```


Test Accuracy (LR=0.001): 53.15%
Test Accuracy (LR=0.0001): 52.38%
Test Accuracy (LR=0.0004): 54.77%
Training Time (LR=0.001): 56.26 minutes
Training Time (LR=0.0001): 56.03 minutes
Training Time (LR=0.0004): 56.89 minutes
Best Learning Rate: 0.0004 (Test Accuracy: 54.77%)

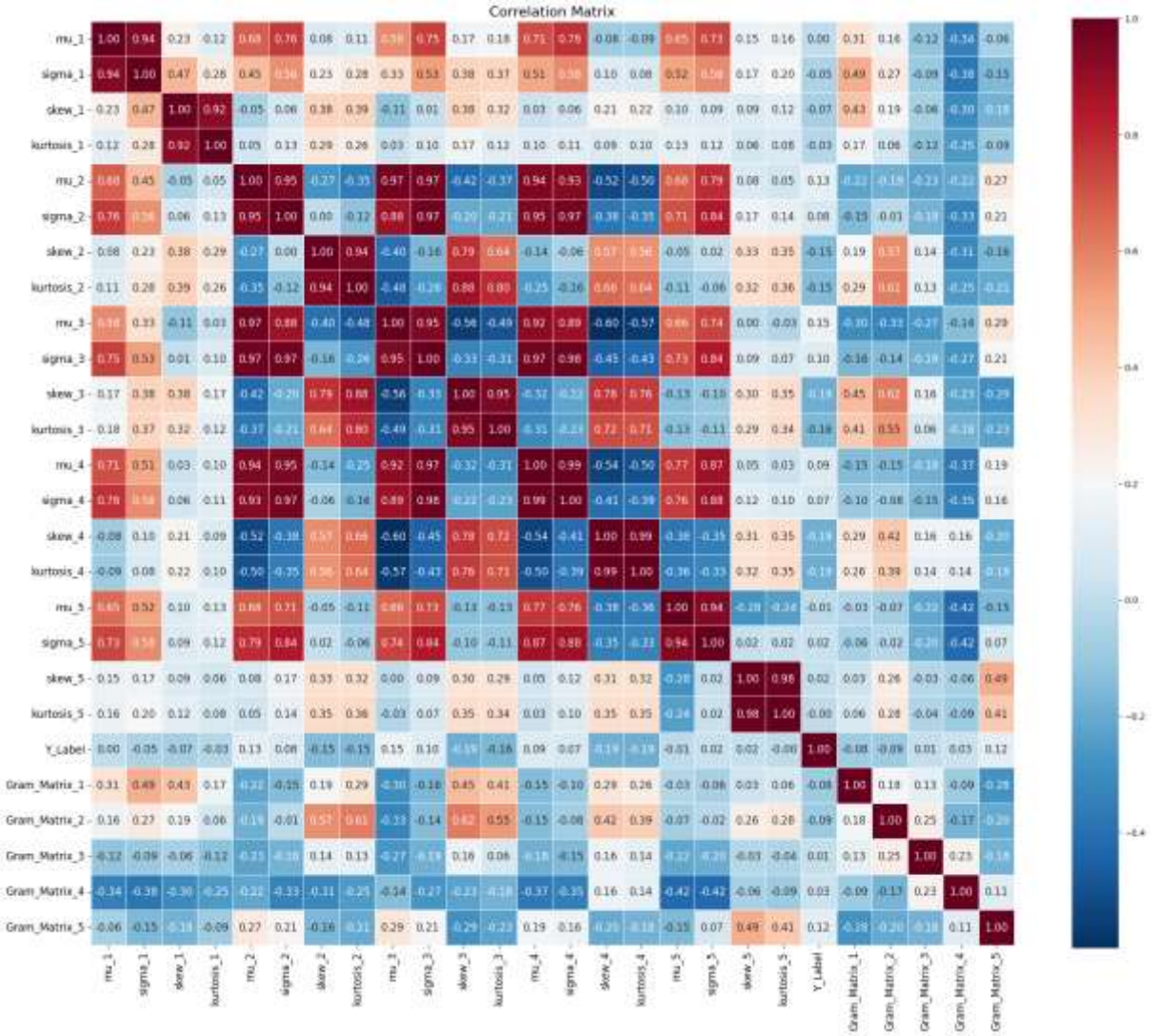
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_2 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_2  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               131328

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 14848586 (56.64 MB)
Trainable params: 133898 (523.04 KB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

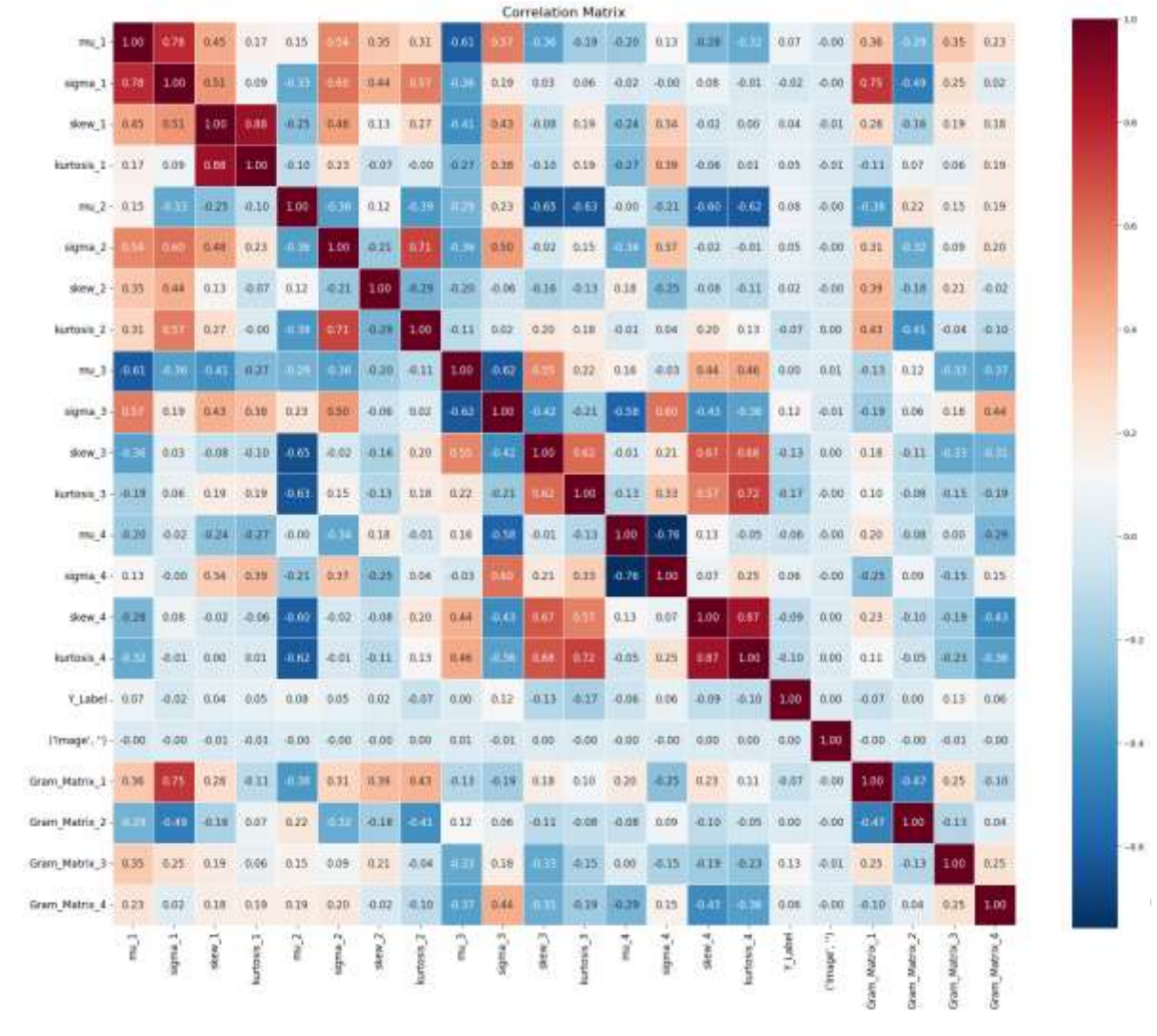
```
Epoch 1/30
1144/1144 [==============================] - 106s 83ms/step - loss: 1.8568 - accuracy: 0.3577 - precision: 0.7018 - recall: 0.1085 - val_loss: 1.4905 - val_accuracy: 0.4916 - val_precision: 0.7962 - val_recall: 0.2399
Epoch 2/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.6701 - accuracy: 0.4247 - precision: 0.7257 - recall: 0.1997 - val_loss: 1.4137 - val_accuracy: 0.5262 - val_precision: 0.7809 - val_recall: 0.2971
Epoch 3/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.6272 - accuracy: 0.4445 - precision: 0.7306 - recall: 0.2218 - val_loss: 1.3680 - val_accuracy: 0.5357 - val_precision: 0.7815 - val_recall: 0.3403
Epoch 4/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5991 - accuracy: 0.4564 - precision: 0.7376 - recall: 0.2401 - val_loss: 1.3619 - val_accuracy: 0.5424 - val_precision: 0.7501 - val_recall: 0.3572
Epoch 5/30
1144/1144 [==============================] - 93s 80ms/step - loss: 1.5825 - accuracy: 0.4634 - precision: 0.7379 - recall: 0.2528 - val_loss: 1.3411 - val_accuracy: 0.5525 - val_precision: 0.7719 - val_recall: 0.3675
Epoch 6/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5704 - accuracy: 0.4686 - precision: 0.7466 - recall: 0.2610 - val_loss: 1.3174 - val_accuracy: 0.5648 - val_precision: 0.7850 - val_recall: 0.3898
Epoch 7/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5619 - accuracy: 0.4713 - precision: 0.7428 - recall: 0.2668 - val_loss: 1.3188 - val_accuracy: 0.5625 - val_precision: 0.7810 - val_recall: 0.3966
Epoch 8/30
1144/1144 [==============================] - 93s 80ms/step - loss: 1.5506 - accuracy: 0.4757 - precision: 0.7436 - recall: 0.2725 - val_loss: 1.2990 - val_accuracy: 0.5652 - val_precision: 0.7893 - val_recall: 0.3742
Epoch 9/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5493 - accuracy: 0.4782 - precision: 0.7475 - recall: 0.2757 - val_loss: 1.2977 - val_accuracy: 0.5718 - val_precision: 0.7882 - val_recall: 0.3758
Epoch 10/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5458 - accuracy: 0.4792 - precision: 0.7458 - recall: 0.2784 - val_loss: 1.2899 - val_accuracy: 0.5782 - val_precision: 0.8031 - val_recall: 0.3884
Epoch 11/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5354 - accuracy: 0.4832 - precision: 0.7473 - recall: 0.2842 - val_loss: 1.2685 - val_accuracy: 0.5778 - val_precision: 0.7884 - val_recall: 0.4012
Epoch 12/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5404 - accuracy: 0.4828 - precision: 0.7491 - recall: 0.2828 - val_loss: 1.2701 - val_accuracy: 0.5914 - val_precision: 0.8130 - val_recall: 0.3748
Epoch 13/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5372 - accuracy: 0.4860 - precision: 0.7512 - recall: 0.2864 - val_loss: 1.2791 - val_accuracy: 0.5834 - val_precision: 0.7956 - val_recall: 0.3935
Epoch 14/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5311 - accuracy: 0.4899 - precision: 0.7501 - recall: 0.2899 - val_loss: 1.2577 - val_accuracy: 0.5889 - val_precision: 0.7940 - val_recall: 0.4080
Epoch 15/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5268 - accuracy: 0.4886 - precision: 0.7521 - recall: 0.2936 - val_loss: 1.2732 - val_accuracy: 0.5782 - val_precision: 0.7850 - val_recall: 0.4114
Epoch 16/30
1144/1144 [==============================] - 93s 81ms/step - loss: 1.5256 - accuracy: 0.4891 - precision: 0.7542 - recall: 0.2924 - val_loss: 1.2596 - val_accuracy: 0.5901 - val_precision: 0.7902 - val_recall: 0.4009
Epoch 17/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5208 - accuracy: 0.4926 - precision: 0.7545 - recall: 0.2968 - val_loss: 1.2690 - val_accuracy: 0.5914 - val_precision: 0.7846 - val_recall: 0.3956
Epoch 18/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5218 - accuracy: 0.4933 - precision: 0.7524 - recall: 0.2977 - val_loss: 1.2542 - val_accuracy: 0.5963 - val_precision: 0.7900 - val_recall: 0.4208
Epoch 19/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5188 - accuracy: 0.4937 - precision: 0.7516 - recall: 0.2973 - val_loss: 1.2640 - val_accuracy: 0.5965 - val_precision: 0.8027 - val_recall: 0.3928
Epoch 20/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5151 - accuracy: 0.4962 - precision: 0.7572 - recall: 0.2984 - val_loss: 1.2727 - val_accuracy: 0.5908 - val_precision: 0.8035 - val_recall: 0.3858
Epoch 21/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5106 - accuracy: 0.4949 - precision: 0.7606 - recall: 0.2968 - val_loss: 1.2611 - val_accuracy: 0.6020 - val_precision: 0.7937 - val_recall: 0.4110
Epoch 22/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.5143 - accuracy: 0.4981 - precision: 0.7512 - recall: 0.3021 - val_loss: 1.2894 - val_accuracy: 0.5814 - val_precision: 0.8100 - val_recall: 0.3917
Epoch 23/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5166 - accuracy: 0.4943 - precision: 0.7558 - recall: 0.3009 - val_loss: 1.2540 - val_accuracy: 0.5969 - val_precision: 0.7996 - val_recall: 0.3997
Epoch 24/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5120 - accuracy: 0.4956 - precision: 0.7572 - recall: 0.3032 - val_loss: 1.2585 - val_accuracy: 0.5949 - val_precision: 0.8010 - val_recall: 0.4064
Epoch 25/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5111 - accuracy: 0.4968 - precision: 0.7605 - recall: 0.3003 - val_loss: 1.2573 - val_accuracy: 0.5913 - val_precision: 0.7887 - val_recall: 0.4097
Epoch 26/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5074 - accuracy: 0.4991 - precision: 0.7577 - recall: 0.3071 - val_loss: 1.2700 - val_accuracy: 0.5969 - val_precision: 0.7976 - val_recall: 0.4017
Epoch 27/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5141 - accuracy: 0.4976 - precision: 0.7562 - recall: 0.3052 - val_loss: 1.2497 - val_accuracy: 0.6021 - val_precision: 0.8028 - val_recall: 0.4171
Epoch 28/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5142 - accuracy: 0.4976 - precision: 0.7539 - recall: 0.3055 - val_loss: 1.2485 - val_accuracy: 0.6028 - val_precision: 0.8005 - val_recall: 0.4113
Epoch 29/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5088 - accuracy: 0.5001 - precision: 0.7563 - recall: 0.3087 - val_loss: 1.2609 - val_accuracy: 0.5928 - val_precision: 0.7906 - val_recall: 0.4158
Epoch 30/30
1144/1144 [==============================] - 91s 80ms/step - loss: 1.5101 - accuracy: 0.4983 - precision: 0.7542 - recall: 0.3060 - val_loss: 1.2732 - val_accuracy: 0.5941 - val_precision: 0.7825 - val_recall: 0.4115
618/814 [=============================>.] - ETA: 26s 83ms/step - loss: 1.5732 - accuracy: 0.5041 - precision: 0.7695 - recall: 0.4115
Epoch 1/30
1144/1144 [==============================] - 103s 83ms/step - loss: 2.0522 - accuracy: 0.2665 - precision: 0.6754 - recall: 0.0530 - val_loss: 1.7494 - val_accuracy: 0.4283 - val_precision: 0.8108 - val_recall: 0.0878
Epoch 2/30
1144/1144 [==============================] - 92s 80ms/step - loss: 1.8336 - accuracy: 0.3733 - precision: 0.7204 - recall: 0.0966 - val_loss: 1.5883 - val_accuracy: 0.4687 - val_precision: 0.8393 - val_recall: 0.1405
Epoch 3/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.7415 - accuracy: 0.4084 - precision: 0.7418 - recall: 0.1358 - val_loss: 1.5107 - val_accuracy: 0.5008 - val_precision: 0.8144 - val_recall: 0.2088
Epoch 4/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.6808 - accuracy: 0.4288 - precision: 0.7470 - recall: 0.1686 - val_loss: 1.4524 - val_accuracy: 0.5248 - val_precision: 0.8190 - val_recall: 0.2414
Epoch 5/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.6462 - accuracy: 0.4418 - precision: 0.7581 - recall: 0.1904 - val_loss: 1.4331 - val_accuracy: 0.5328 - val_precision: 0.8190 - val_recall: 0.2603
Epoch 6/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.6184 - accuracy: 0.4529 - precision: 0.7603 - recall: 0.2052 - val_loss: 1.3924 - val_accuracy: 0.5458 - val_precision: 0.8264 - val_recall: 0.2718
Epoch 7/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5965 - accuracy: 0.4637 - precision: 0.7631 - recall: 0.2187 - val_loss: 1.3673 - val_accuracy: 0.5529 - val_precision: 0.8215 - val_recall: 0.2928
Epoch 8/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5757 - accuracy: 0.4699 - precision: 0.7623 - recall: 0.2310 - val_loss: 1.3488 - val_accuracy: 0.5564 - val_precision: 0.8134 - val_recall: 0.3102
Epoch 9/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5630 - accuracy: 0.4719 - precision: 0.7634 - recall: 0.2387 - val_loss: 1.3282 - val_accuracy: 0.5609 - val_precision: 0.8119 - val_recall: 0.3297
Epoch 10/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.5469 - accuracy: 0.4776 - precision: 0.7668 - recall: 0.2468 - val_loss: 1.3270 - val_accuracy: 0.5644 - val_precision: 0.8094 - val_recall: 0.3261
Epoch 11/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5344 - accuracy: 0.4833 - precision: 0.7677 - recall: 0.2547 - val_loss: 1.3055 - val_accuracy: 0.5707 - val_precision: 0.8097 - val_recall: 0.3408
Epoch 12/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5251 - accuracy: 0.4894 - precision: 0.7711 - recall: 0.2592 - val_loss: 1.2908 - val_accuracy: 0.5714 - val_precision: 0.7988 - val_recall: 0.3599
Epoch 13/30
1144/1144 [==============================] - 89s 77ms/step - loss: 1.5066 - accuracy: 0.4935 - precision: 0.7728 - recall: 0.2705 - val_loss: 1.2875 - val_accuracy: 0.5758 - val_precision: 0.8024 - val_recall: 0.3601
Epoch 14/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.5096 - accuracy: 0.4931 - precision: 0.7722 - recall: 0.2703 - val_loss: 1.2782 - val_accuracy: 0.5781 - val_precision: 0.7924 - val_recall: 0.3712
Epoch 15/30
1144/1144 [==============================] - 91s 79ms/step - loss: 1.4982 - accuracy: 0.4946 - precision: 0.7712 - recall: 0.2767 - val_loss: 1.2762 - val_accuracy: 0.5852 - val_precision: 0.8140 - val_recall: 0.3641
Epoch 16/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4911 - accuracy: 0.4992 - precision: 0.7748 - recall: 0.2805 - val_loss: 1.2708 - val_accuracy: 0.5776 - val_precision: 0.7767 - val_recall: 0.3830
Epoch 17/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4830 - accuracy: 0.5023 - precision: 0.7714 - recall: 0.2834 - val_loss: 1.2541 - val_accuracy: 0.5884 - val_precision: 0.8019 - val_recall: 0.3914
Epoch 18/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4771 - accuracy: 0.5051 - precision: 0.7736 - recall: 0.2879 - val_loss: 1.2532 - val_accuracy: 0.5977 - val_precision: 0.8047 - val_recall: 0.3844
Epoch 19/30
1144/1144 [==============================] - 91s 79ms/step - loss: 1.4688 - accuracy: 0.5074 - precision: 0.7744 - recall: 0.2937 - val_loss: 1.2483 - val_accuracy: 0.5918 - val_precision: 0.8017 - val_recall: 0.3979
Epoch 20/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4644 - accuracy: 0.5090 - precision: 0.7773 - recall: 0.2970 - val_loss: 1.2479 - val_accuracy: 0.5931 - val_precision: 0.8154 - val_recall: 0.3778
Epoch 21/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4593 - accuracy: 0.5139 - precision: 0.7811 - recall: 0.3002 - val_loss: 1.2418 - val_accuracy: 0.6006 - val_precision: 0.8020 - val_recall: 0.3995
Epoch 22/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4556 - accuracy: 0.5091 - precision: 0.7771 - recall: 0.2997 - val_loss: 1.2265 - val_accuracy: 0.6075 - val_precision: 0.8049 - val_recall: 0.4044
Epoch 23/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4492 - accuracy: 0.5161 - precision: 0.7773 - recall: 0.3052 - val_loss: 1.2330 - val_accuracy: 0.5968 - val_precision: 0.7903 - val_recall: 0.4038
Epoch 24/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4418 - accuracy: 0.5188 - precision: 0.7799 - recall: 0.3093 - val_loss: 1.2279 - val_accuracy: 0.5953 - val_precision: 0.7925 - val_recall: 0.4127
Epoch 25/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4373 - accuracy: 0.5179 - precision: 0.7795 - recall: 0.3115 - val_loss: 1.2248 - val_accuracy: 0.5982 - val_precision: 0.8012 - val_recall: 0.4047
Epoch 26/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4330 - accuracy: 0.5232 - precision: 0.7804 - recall: 0.3158 - val_loss: 1.2193 - val_accuracy: 0.6065 - val_precision: 0.8039 - val_recall: 0.4061
Epoch 27/30
1144/1144 [==============================] - 90s 78ms/step - loss: 1.4268 - accuracy: 0.5221 - precision: 0.7795 - recall: 0.3153 - val_loss: 1.2191 - val_accuracy: 0.5971 - val_precision: 0.7937 - val_recall: 0.4143
Epoch 28/30
1144/1144 [==============================] - 90s 78ms/step - loss: 1.4238 - accuracy: 0.5242 - precision: 0.7830 - recall: 0.3211 - val_loss: 1.2057 - val_accuracy: 0.6054 - val_precision: 0.8024 - val_recall: 0.4158
Epoch 29/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4204 - accuracy: 0.5244 - precision: 0.7804 - recall: 0.3218 - val_loss: 1.2091 - val_accuracy: 0.6062 - val_precision: 0.8094 - val_recall: 0.4121
Epoch 30/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4172 - accuracy: 0.5253 - precision: 0.7866 - recall: 0.3234 - val_loss: 1.2030 - val_accuracy: 0.6091 - val_precision: 0.8044 - val_recall: 0.4209
618/814 [=============================>.] - ETA: 27s 83ms/step - loss: 1.2830 - accuracy: 0.5691 - precision: 0.8004 - recall: 0.4208
Epoch 1/30
1144/1144 [==============================] - 106s 83ms/step - loss: 1.8062 - accuracy: 0.3403 - precision: 0.7048 - recall: 0.0880 - val_loss: 1.5217 - val_accuracy: 0.4830 - val_precision: 0.8124 - val_recall: 0.1979
Epoch 2/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.6858 - accuracy: 0.4227 - precision: 0.7370 - recall: 0.1755 - val_loss: 1.4230 - val_accuracy: 0.5252 - val_precision: 0.7870 - val_recall: 0.2769
Epoch 3/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.6216 - accuracy: 0.4489 - precision: 0.7503 - recall: 0.2130 - val_loss: 1.4025 - val_accuracy: 0.5247 - val_precision: 0.7954 - val_recall: 0.3069
Epoch 4/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.5812 - accuracy: 0.4641 - precision: 0.7613 - recall: 0.2358 - val_loss: 1.3439 - val_accuracy: 0.5523 - val_precision: 0.7898 - val_recall: 0.3223
Epoch 5/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5586 - accuracy: 0.4731 - precision: 0.7671 - recall: 0.2518 - val_loss: 1.3268 - val_accuracy: 0.5607 - val_precision: 0.8075 - val_recall: 0.3381
Epoch 6/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5307 - accuracy: 0.4781 - precision: 0.7732 - recall: 0.2639 - val_loss: 1.2844 - val_accuracy: 0.5725 - val_precision: 0.7711 - val_recall: 0.3812
Epoch 7/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5235 - accuracy: 0.4844 - precision: 0.7682 - recall: 0.2730 - val_loss: 1.2820 - val_accuracy: 0.5687 - val_precision: 0.7771 - val_recall: 0.3863
Epoch 8/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.5129 - accuracy: 0.4889 - precision: 0.7691 - recall: 0.2797 - val_loss: 1.2797 - val_accuracy: 0.5751 - val_precision: 0.7864 - val_recall: 0.3813
Epoch 9/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4917 - accuracy: 0.4949 - precision: 0.7809 - recall: 0.2884 - val_loss: 1.2680 - val_accuracy: 0.5894 - val_precision: 0.8195 - val_recall: 0.3696
Epoch 10/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4925 - accuracy: 0.4987 - precision: 0.7812 - recall: 0.2908 - val_loss: 1.2480 - val_accuracy: 0.5954 - val_precision: 0.7774 - val_recall: 0.4109
Epoch 11/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4805 - accuracy: 0.5022 - precision: 0.7838 - recall: 0.2992 - val_loss: 1.2594 - val_accuracy: 0.5811 - val_precision: 0.7877 - val_recall: 0.3987
Epoch 12/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4768 - accuracy: 0.5023 - precision: 0.7836 - recall: 0.3013 - val_loss: 1.2454 - val_accuracy: 0.5912 - val_precision: 0.7821 - val_recall: 0.4062
Epoch 13/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4676 - accuracy: 0.5056 - precision: 0.7845 - recall: 0.3063 - val_loss: 1.2249 - val_accuracy: 0.6006 - val_precision: 0.8061 - val_recall: 0.4115
Epoch 14/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4692 - accuracy: 0.5067 - precision: 0.7850 - recall: 0.3101 - val_loss: 1.2273 - val_accuracy: 0.6059 - val_precision: 0.8100 - val_recall: 0.4110
Epoch 15/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4565 - accuracy: 0.5089 - precision: 0.7856 - recall: 0.3148 - val_loss: 1.2364 - val_accuracy: 0.5988 - val_precision: 0.8132 - val_recall: 0.3982
Epoch 16/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4490 - accuracy: 0.5141 - precision: 0.7855 - recall: 0.3183 - val_loss: 1.2283 - val_accuracy: 0.6067 - val_precision: 0.8207 - val_recall: 0.4132
Epoch 17/30
1144/1144 [==============================] - 89s 77ms/step - loss: 1.4467 - accuracy: 0.5152 - precision: 0.7870 - recall: 0.3188 - val_loss: 1.2173 - val_accuracy: 0.6017 - val_precision: 0.7748 - val_recall: 0.4427
Epoch 18/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4464 - accuracy: 0.5145 - precision: 0.7880 - recall: 0.3218 - val_loss: 1.2312 - val_accuracy: 0.5962 - val_precision: 0.7843 - val_recall: 0.4263
Epoch 19/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4397 - accuracy: 0.5194 - precision: 0.7906 - recall: 0.3252 - val_loss: 1.2051 - val_accuracy: 0.6075 - val_precision: 0.8018 - val_recall: 0.4348
Epoch 20/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4371 - accuracy: 0.5185 - precision: 0.7914 - recall: 0.3258 - val_loss: 1.2033 - val_accuracy: 0.6102 - val_precision: 0.7950 - val_recall: 0.4481
Epoch 21/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4323 - accuracy: 0.5214 - precision: 0.7906 - recall: 0.3297 - val_loss: 1.2033 - val_accuracy: 0.6125 - val_precision: 0.8091 - val_recall: 0.4258
Epoch 22/30
1144/1144 [==============================] - 88s 77ms/step - loss: 1.4265 - accuracy: 0.5228 - precision: 0.7932 - recall: 0.3325 - val_loss: 1.2143 - val_accuracy: 0.6025 - val_precision: 0.7796 - val_recall: 0.4438
Epoch 23/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4279 - accuracy: 0.5229 - precision: 0.7916 - recall: 0.3308 - val_loss: 1.2052 - val_accuracy: 0.6075 - val_precision: 0.7929 - val_recall: 0.4383
Epoch 24/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4273 - accuracy: 0.5234 - precision: 0.7922 - recall: 0.3344 - val_loss: 1.2011 - val_accuracy: 0.6085 - val_precision: 0.7918 - val_recall: 0.4527
Epoch 25/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4235 - accuracy: 0.5273 - precision: 0.7918 - recall: 0.3377 - val_loss: 1.2189 - val_accuracy: 0.6048 - val_precision: 0.7898 - val_recall: 0.4399
Epoch 26/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4236 - accuracy: 0.5262 - precision: 0.7909 - recall: 0.3375 - val_loss: 1.1990 - val_accuracy: 0.6092 - val_precision: 0.7947 - val_recall: 0.4475
Epoch 27/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4195 - accuracy: 0.5272 - precision: 0.7944 - recall: 0.3393 - val_loss: 1.2223 - val_accuracy: 0.6053 - val_precision: 0.8024 - val_recall: 0.4303
Epoch 28/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4199 - accuracy: 0.5278 - precision: 0.7920 - recall: 0.3410 - val_loss: 1.1985 - val_accuracy: 0.6162 - val_precision: 0.8056 - val_recall: 0.4395
Epoch 29/30
1144/1144 [==============================] - 90s 79ms/step - loss: 1.4168 - accuracy: 0.5309 - precision: 0.7939 - recall: 0.3428 - val_loss: 1.2146 - val_accuracy: 0.6068 - val_precision: 0.8016 - val_recall: 0.4298
Epoch 30/30
1144/1144 [==============================] - 89s 78ms/step - loss: 1.4136 - accuracy: 0.5305 - precision: 0.7953 - recall: 0.3447 - val_loss: 1.1999 - val_accuracy: 0.6131 - val_precision: 0.7965 - val_recall: 0.4602
618/814 [=============================>.] - ETA: 27s 83ms/step - loss: 1.1998 - accuracy: 0.6131 - precision: 0.7980 - recall: 0.4602
Test Accuracy (LR=0.001): 58.41%
Test Accuracy (LR=0.0001): 60.91%
Test Accuracy (LR=0.0004): 61.31%
Training Time (LR=0.001): 48.03 minutes
Training Time (LR=0.0001): 45.05 minutes
Training Time (LR=0.0004): 44.80 minutes
Best Learning Rate: 0.0004 (Test Accuracy: 61.31%)
```

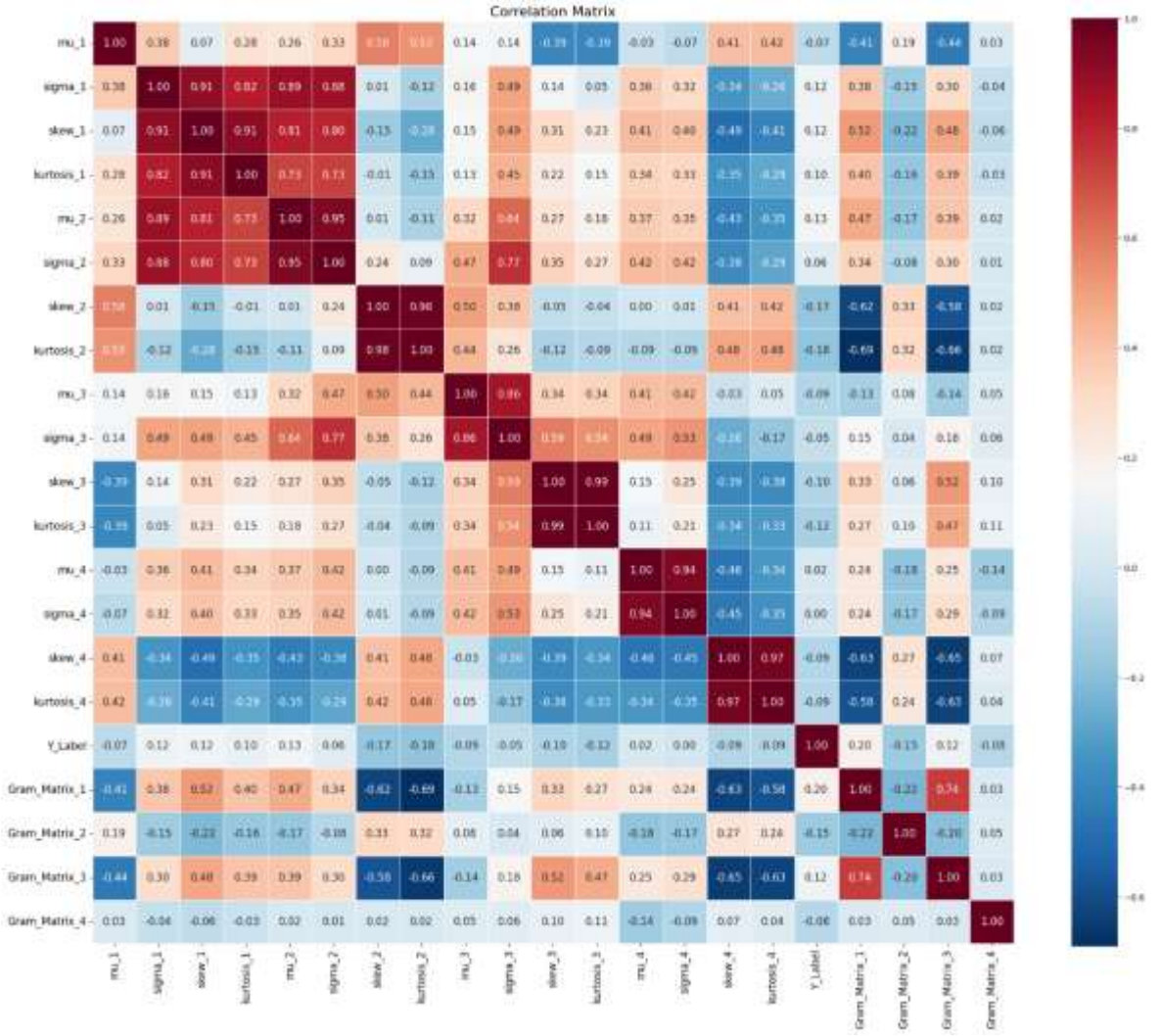
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_2 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 densenet121 (Functional)    (None, 7, 7, 1024)        7037504

 global_average_pooling2d_2  (None, 1024)              0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 256)               262400

 dropout_2 (Dropout)         (None, 256)               0

 dense_5 (Dense)             (None, 10)                2570

=================================================================
Total params: 7302474 (27.86 MB)
Trainable params: 264970 (1.01 MB)
Non-trainable params: 7037504 (26.85 MB)
_________________________________________________________________
```

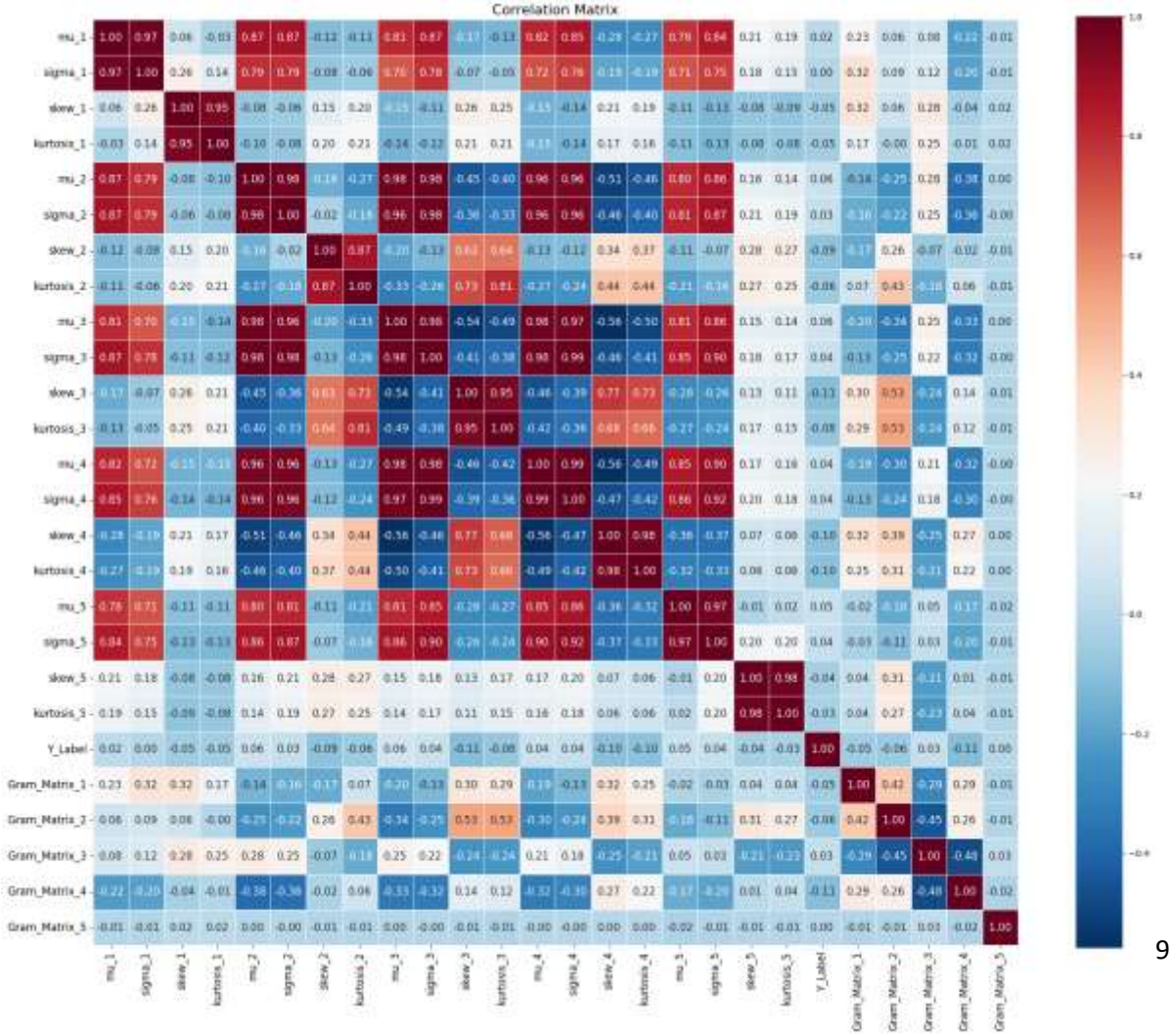
```
Epoch 1/30
1144/1144 [==============================] - 185s 87ms/step - loss: 1.7199 - accuracy: 0.4169 - precision: 0.7276 - recall: 0.1965 - val_loss: 1.2695 - val_accuracy: 0.5766 - val_precision: 0.8090 - val_recall: 0.3985
Epoch 2/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.5141 - accuracy: 0.4878 - precision: 0.7633 - recall: 0.2939 - val_loss: 1.2127 - val_accuracy: 0.6035 - val_precision: 0.8358 - val_recall: 0.3985
Epoch 3/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4679 - accuracy: 0.5105 - precision: 0.7718 - recall: 0.3256 - val_loss: 1.1963 - val_accuracy: 0.6123 - val_precision: 0.8158 - val_recall: 0.4224
Epoch 4/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4264 - accuracy: 0.5266 - precision: 0.7747 - recall: 0.3452 - val_loss: 1.1763 - val_accuracy: 0.6236 - val_precision: 0.8157 - val_recall: 0.4356
Epoch 5/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4126 - accuracy: 0.5325 - precision: 0.7767 - recall: 0.3569 - val_loss: 1.1723 - val_accuracy: 0.6274 - val_precision: 0.8318 - val_recall: 0.4701
Epoch 6/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4027 - accuracy: 0.5368 - precision: 0.7777 - recall: 0.3632 - val_loss: 1.1586 - val_accuracy: 0.6336 - val_precision: 0.8322 - val_recall: 0.4556
Epoch 7/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3926 - accuracy: 0.5419 - precision: 0.7814 - recall: 0.3703 - val_loss: 1.1213 - val_accuracy: 0.6437 - val_precision: 0.8319 - val_recall: 0.4882
Epoch 8/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3852 - accuracy: 0.5477 - precision: 0.7815 - recall: 0.3758 - val_loss: 1.1148 - val_accuracy: 0.6433 - val_precision: 0.8057 - val_recall: 0.5107
Epoch 9/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3781 - accuracy: 0.5488 - precision: 0.7864 - recall: 0.3837 - val_loss: 1.1191 - val_accuracy: 0.6447 - val_precision: 0.8181 - val_recall: 0.5089
Epoch 10/30
1144/1144 [==============================] - 95s 84ms/step - loss: 1.3777 - accuracy: 0.5503 - precision: 0.7844 - recall: 0.3843 - val_loss: 1.1287 - val_accuracy: 0.6435 - val_precision: 0.8153 - val_recall: 0.4956
Epoch 11/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3724 - accuracy: 0.5523 - precision: 0.7888 - recall: 0.3854 - val_loss: 1.1032 - val_accuracy: 0.6543 - val_precision: 0.8160 - val_recall: 0.5073
Epoch 12/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3696 - accuracy: 0.5543 - precision: 0.7884 - recall: 0.3894 - val_loss: 1.1136 - val_accuracy: 0.6513 - val_precision: 0.8335 - val_recall: 0.4887
Epoch 13/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3682 - accuracy: 0.5556 - precision: 0.7871 - recall: 0.3883 - val_loss: 1.1484 - val_accuracy: 0.6394 - val_precision: 0.8161 - val_recall: 0.4948
Epoch 14/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3595 - accuracy: 0.5596 - precision: 0.7891 - recall: 0.3960 - val_loss: 1.1165 - val_accuracy: 0.6532 - val_precision: 0.8080 - val_recall: 0.5180
Epoch 15/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3572 - accuracy: 0.5595 - precision: 0.7882 - recall: 0.3994 - val_loss: 1.1178 - val_accuracy: 0.6558 - val_precision: 0.8285 - val_recall: 0.4954
Epoch 16/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3512 - accuracy: 0.5618 - precision: 0.7911 - recall: 0.4010 - val_loss: 1.0956 - val_accuracy: 0.6603 - val_precision: 0.8291 - val_recall: 0.5156
Epoch 17/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3537 - accuracy: 0.5641 - precision: 0.7908 - recall: 0.4014 - val_loss: 1.1529 - val_accuracy: 0.6389 - val_precision: 0.8197 - val_recall: 0.4778
Epoch 18/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3394 - accuracy: 0.5636 - precision: 0.7902 - recall: 0.4006 - val_loss: 1.1108 - val_accuracy: 0.6567 - val_precision: 0.8264 - val_recall: 0.4894
Epoch 19/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3504 - accuracy: 0.5649 - precision: 0.7922 - recall: 0.4035 - val_loss: 1.1464 - val_accuracy: 0.6480 - val_precision: 0.8256 - val_recall: 0.4952
Epoch 20/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.3530 - accuracy: 0.5656 - precision: 0.7947 - recall: 0.4033 - val_loss: 1.1305 - val_accuracy: 0.6495 - val_precision: 0.8240 - val_recall: 0.4944
Epoch 21/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3470 - accuracy: 0.5663 - precision: 0.7939 - recall: 0.4079 - val_loss: 1.1100 - val_accuracy: 0.6568 - val_precision: 0.8326 - val_recall: 0.4983
Epoch 22/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3452 - accuracy: 0.5682 - precision: 0.7922 - recall: 0.4085 - val_loss: 1.1063 - val_accuracy: 0.6565 - val_precision: 0.8370 - val_recall: 0.4984
Epoch 23/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.3414 - accuracy: 0.5688 - precision: 0.7948 - recall: 0.4091 - val_loss: 1.1577 - val_accuracy: 0.6410 - val_precision: 0.8099 - val_recall: 0.4913
Epoch 24/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3459 - accuracy: 0.5688 - precision: 0.7944 - recall: 0.4083 - val_loss: 1.1258 - val_accuracy: 0.6516 - val_precision: 0.8165 - val_recall: 0.5205
Epoch 25/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3440 - accuracy: 0.5689 - precision: 0.7930 - recall: 0.4103 - val_loss: 1.1046 - val_accuracy: 0.6613 - val_precision: 0.8244 - val_recall: 0.5235
Epoch 26/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3422 - accuracy: 0.5699 - precision: 0.7914 - recall: 0.4089 - val_loss: 1.1343 - val_accuracy: 0.6541 - val_precision: 0.8452 - val_recall: 0.4788
Epoch 27/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.3432 - accuracy: 0.5693 - precision: 0.7929 - recall: 0.4095 - val_loss: 1.1253 - val_accuracy: 0.6552 - val_precision: 0.8365 - val_recall: 0.4885
Epoch 28/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3466 - accuracy: 0.5667 - precision: 0.7936 - recall: 0.4076 - val_loss: 1.1140 - val_accuracy: 0.6549 - val_precision: 0.8178 - val_recall: 0.5169
Epoch 29/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.3409 - accuracy: 0.5696 - precision: 0.7939 - recall: 0.4112 - val_loss: 1.1139 - val_accuracy: 0.6573 - val_precision: 0.8396 - val_recall: 0.5020
Epoch 30/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.3391 - accuracy: 0.5707 - precision: 0.7941 - recall: 0.4129 - val_loss: 1.1362 - val_accuracy: 0.6449 - val_precision: 0.7971 - val_recall: 0.5344
614/614 [==============================] - 20s 31ms/step - loss: 1.1362 - accuracy: 0.6449 - precision: 0.7971 - recall: 0.5344
Epoch 1/30
1144/1144 [==============================] - 104s 88ms/step - loss: 1.8076 - accuracy: 0.3669 - precision: 0.7046 - recall: 0.1154 - val_loss: 1.5007 - val_accuracy: 0.5167 - val_precision: 0.8424 - val_recall: 0.2129
Epoch 2/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.6230 - accuracy: 0.4998 - precision: 0.7634 - recall: 0.2231 - val_loss: 1.3094 - val_accuracy: 0.5705 - val_precision: 0.8338 - val_recall: 0.3134
Epoch 3/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.5581 - accuracy: 0.4934 - precision: 0.7767 - recall: 0.2748 - val_loss: 1.2619 - val_accuracy: 0.5806 - val_precision: 0.8442 - val_recall: 0.3578
Epoch 4/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4901 - accuracy: 0.5140 - precision: 0.7911 - recall: 0.3039 - val_loss: 1.2170 - val_accuracy: 0.6170 - val_precision: 0.8549 - val_recall: 0.3749
Epoch 5/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4507 - accuracy: 0.5325 - precision: 0.7971 - recall: 0.3294 - val_loss: 1.1785 - val_accuracy: 0.6156 - val_precision: 0.8267 - val_recall: 0.4166
Epoch 6/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.4297 - accuracy: 0.5405 - precision: 0.7998 - recall: 0.3479 - val_loss: 1.1580 - val_accuracy: 0.6336 - val_precision: 0.8402 - val_recall: 0.4265
Epoch 7/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3469 - accuracy: 0.5543 - precision: 0.8069 - recall: 0.3933 - val_loss: 1.1284 - val_accuracy: 0.6439 - val_precision: 0.8465 - val_recall: 0.4496
Epoch 8/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3295 - accuracy: 0.5618 - precision: 0.8080 - recall: 0.3733 - val_loss: 1.1094 - val_accuracy: 0.6451 - val_precision: 0.8478 - val_recall: 0.4562
Epoch 9/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.3373 - accuracy: 0.5668 - precision: 0.8155 - recall: 0.3837 - val_loss: 1.0929 - val_accuracy: 0.6496 - val_precision: 0.8445 - val_recall: 0.4672
Epoch 10/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2848 - accuracy: 0.5757 - precision: 0.8161 - recall: 0.3999 - val_loss: 1.0990 - val_accuracy: 0.6511 - val_precision: 0.8414 - val_recall: 0.4759
Epoch 11/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2714 - accuracy: 0.5816 - precision: 0.8159 - recall: 0.4031 - val_loss: 1.0977 - val_accuracy: 0.6596 - val_precision: 0.8422 - val_recall: 0.4920
Epoch 12/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2637 - accuracy: 0.5836 - precision: 0.8155 - recall: 0.4089 - val_loss: 1.0859 - val_accuracy: 0.6604 - val_precision: 0.8410 - val_recall: 0.4938
Epoch 13/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2494 - accuracy: 0.5868 - precision: 0.8183 - recall: 0.4167 - val_loss: 1.0703 - val_accuracy: 0.6648 - val_precision: 0.8348 - val_recall: 0.4985
Epoch 14/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2330 - accuracy: 0.5938 - precision: 0.8197 - recall: 0.4224 - val_loss: 1.0489 - val_accuracy: 0.6691 - val_precision: 0.8357 - val_recall: 0.5191
Epoch 15/30
1144/1144 [==============================] - 96s 84ms/step - loss: 1.2279 - accuracy: 0.5959 - precision: 0.8214 - recall: 0.4249 - val_loss: 1.0393 - val_accuracy: 0.6672 - val_precision: 0.8415 - val_recall: 0.5139
Epoch 16/30
1144/1144 [==============================] - 96s 84ms/step - loss: 1.2168 - accuracy: 0.6002 - precision: 0.8205 - recall: 0.4339 - val_loss: 1.0334 - val_accuracy: 0.6716 - val_precision: 0.8333 - val_recall: 0.5102
Epoch 17/30
1144/1144 [==============================] - 96s 84ms/step - loss: 1.2109 - accuracy: 0.6030 - precision: 0.8211 - recall: 0.4356 - val_loss: 1.0244 - val_accuracy: 0.6713 - val_precision: 0.8360 - val_recall: 0.5290
Epoch 18/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1995 - accuracy: 0.6064 - precision: 0.8253 - recall: 0.4414 - val_loss: 1.0293 - val_accuracy: 0.6728 - val_precision: 0.8418 - val_recall: 0.5196
Epoch 19/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.1916 - accuracy: 0.6112 - precision: 0.8249 - recall: 0.4470 - val_loss: 1.0129 - val_accuracy: 0.6773 - val_precision: 0.8439 - val_recall: 0.5389
Epoch 20/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.1815 - accuracy: 0.6113 - precision: 0.8275 - recall: 0.4502 - val_loss: 1.0108 - val_accuracy: 0.6787 - val_precision: 0.8576 - val_recall: 0.5371
Epoch 21/30
1144/1144 [==============================] - 97s 84ms/step - loss: 1.1813 - accuracy: 0.6126 - precision: 0.8275 - recall: 0.4539 - val_loss: 1.0102 - val_accuracy: 0.6811 - val_precision: 0.8439 - val_recall: 0.5381
Epoch 22/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1736 - accuracy: 0.6167 - precision: 0.8274 - recall: 0.4549 - val_loss: 1.0081 - val_accuracy: 0.6812 - val_precision: 0.8421 - val_recall: 0.5373
Epoch 23/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1659 - accuracy: 0.6194 - precision: 0.8313 - recall: 0.4607 - val_loss: 0.9983 - val_accuracy: 0.6847 - val_precision: 0.8404 - val_recall: 0.5449
Epoch 24/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1608 - accuracy: 0.6222 - precision: 0.8307 - recall: 0.4639 - val_loss: 0.9941 - val_accuracy: 0.6837 - val_precision: 0.8330 - val_recall: 0.5566
Epoch 25/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1559 - accuracy: 0.6223 - precision: 0.8315 - recall: 0.4666 - val_loss: 1.0145 - val_accuracy: 0.6793 - val_precision: 0.8276 - val_recall: 0.5456
Epoch 26/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1526 - accuracy: 0.6242 - precision: 0.8295 - recall: 0.4681 - val_loss: 0.9947 - val_accuracy: 0.6849 - val_precision: 0.8349 - val_recall: 0.5558
Epoch 27/30
1144/1144 [==============================] - 96s 85ms/step - loss: 1.1429 - accuracy: 0.6272 - precision: 0.8329 - recall: 0.4739 - val_loss: 0.9949 - val_accuracy: 0.6854 - val_precision: 0.8358 - val_recall: 0.5561
Epoch 28/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1303 - accuracy: 0.6302 - precision: 0.8330 - recall: 0.4783 - val_loss: 0.9941 - val_accuracy: 0.6871 - val_precision: 0.8365 - val_recall: 0.5567
Epoch 29/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.1357 - accuracy: 0.6311 - precision: 0.8336 - recall: 0.4760 - val_loss: 0.9916 - val_accuracy: 0.6922 - val_precision: 0.8394 - val_recall: 0.5646
Epoch 30/30
1144/1144 [==============================] - 98s 85ms/step - loss: 1.1306 - accuracy: 0.6321 - precision: 0.8344 - recall: 0.4805 - val_loss: 0.9899 - val_accuracy: 0.6872 - val_precision: 0.8403 - val_recall: 0.5549
614/614 [==============================] - 20s 32ms/step - loss: 0.9899 - accuracy: 0.6872 - precision: 0.8403 - recall: 0.5549
Epoch 1/30
1144/1144 [==============================] - 104s 87ms/step - loss: 1.7357 - accuracy: 0.4166 - precision: 0.7389 - recall: 0.1841 - val_loss: 1.2998 - val_accuracy: 0.5746 - val_precision: 0.8115 - val_recall: 0.3486
Epoch 2/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.4879 - accuracy: 0.4944 - precision: 0.7671 - recall: 0.2657 - val_loss: 1.2304 - val_accuracy: 0.5984 - val_precision: 0.8268 - val_recall: 0.3648
Epoch 3/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.4127 - accuracy: 0.5285 - precision: 0.7809 - recall: 0.3264 - val_loss: 1.1419 - val_accuracy: 0.6342 - val_precision: 0.8366 - val_recall: 0.4380
Epoch 4/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.3736 - accuracy: 0.5394 - precision: 0.7864 - recall: 0.3579 - val_loss: 1.1245 - val_accuracy: 0.6307 - val_precision: 0.8314 - val_recall: 0.4660
Epoch 5/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.3460 - accuracy: 0.5502 - precision: 0.7930 - recall: 0.3753 - val_loss: 1.0869 - val_accuracy: 0.6512 - val_precision: 0.8453 - val_recall: 0.4768
Epoch 6/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.3256 - accuracy: 0.5579 - precision: 0.7999 - recall: 0.3871 - val_loss: 1.1117 - val_accuracy: 0.6432 - val_precision: 0.8270 - val_recall: 0.4763
Epoch 7/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.3096 - accuracy: 0.5652 - precision: 0.7978 - recall: 0.3979 - val_loss: 1.0739 - val_accuracy: 0.6593 - val_precision: 0.8378 - val_recall: 0.5019
Epoch 8/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2934 - accuracy: 0.5728 - precision: 0.7996 - recall: 0.4085 - val_loss: 1.0775 - val_accuracy: 0.6556 - val_precision: 0.8403 - val_recall: 0.4992
Epoch 9/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2806 - accuracy: 0.5768 - precision: 0.8022 - recall: 0.4137 - val_loss: 1.0622 - val_accuracy: 0.6570 - val_precision: 0.8318 - val_recall: 0.5148
Epoch 10/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2693 - accuracy: 0.5828 - precision: 0.8061 - recall: 0.4219 - val_loss: 1.0411 - val_accuracy: 0.6674 - val_precision: 0.8364 - val_recall: 0.5280
Epoch 11/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2645 - accuracy: 0.5843 - precision: 0.8062 - recall: 0.4251 - val_loss: 1.0521 - val_accuracy: 0.6634 - val_precision: 0.8201 - val_recall: 0.5332
Epoch 12/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2600 - accuracy: 0.5871 - precision: 0.8042 - recall: 0.4279 - val_loss: 1.0387 - val_accuracy: 0.6736 - val_precision: 0.8368 - val_recall: 0.5254
Epoch 13/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2526 - accuracy: 0.5901 - precision: 0.8062 - recall: 0.4332 - val_loss: 1.0310 - val_accuracy: 0.6714 - val_precision: 0.8269 - val_recall: 0.5391
Epoch 14/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2446 - accuracy: 0.5938 - precision: 0.8079 - recall: 0.4373 - val_loss: 1.0310 - val_accuracy: 0.6757 - val_precision: 0.8338 - val_recall: 0.5366
Epoch 15/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2390 - accuracy: 0.5953 - precision: 0.8086 - recall: 0.4431 - val_loss: 1.0470 - val_accuracy: 0.6716 - val_precision: 0.8166 - val_recall: 0.5464
Epoch 16/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2390 - accuracy: 0.5946 - precision: 0.8081 - recall: 0.4412 - val_loss: 1.0422 - val_accuracy: 0.6725 - val_precision: 0.8092 - val_recall: 0.5634
Epoch 17/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2276 - accuracy: 0.5999 - precision: 0.8079 - recall: 0.4483 - val_loss: 1.0440 - val_accuracy: 0.6742 - val_precision: 0.8313 - val_recall: 0.5408
Epoch 18/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2283 - accuracy: 0.6013 - precision: 0.8104 - recall: 0.4490 - val_loss: 1.0237 - val_accuracy: 0.6794 - val_precision: 0.8378 - val_recall: 0.5422
Epoch 19/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2230 - accuracy: 0.6036 - precision: 0.8102 - recall: 0.4527 - val_loss: 1.0057 - val_accuracy: 0.6861 - val_precision: 0.8282 - val_recall: 0.5300
Epoch 20/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2222 - accuracy: 0.6028 - precision: 0.8108 - recall: 0.4529 - val_loss: 1.0412 - val_accuracy: 0.6752 - val_precision: 0.8204 - val_recall: 0.5552
Epoch 21/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2191 - accuracy: 0.6055 - precision: 0.8110 - recall: 0.4572 - val_loss: 1.0257 - val_accuracy: 0.6824 - val_precision: 0.8168 - val_recall: 0.5595
Epoch 22/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2109 - accuracy: 0.6101 - precision: 0.8138 - recall: 0.4609 - val_loss: 1.0484 - val_accuracy: 0.6742 - val_precision: 0.8128 - val_recall: 0.5611
Epoch 23/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2120 - accuracy: 0.6074 - precision: 0.8110 - recall: 0.4605 - val_loss: 1.0493 - val_accuracy: 0.6774 - val_precision: 0.8099 - val_recall: 0.5584
Epoch 24/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2056 - accuracy: 0.6108 - precision: 0.8139 - recall: 0.4619 - val_loss: 1.0368 - val_accuracy: 0.6790 - val_precision: 0.8221 - val_recall: 0.5568
Epoch 25/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.2050 - accuracy: 0.6119 - precision: 0.8156 - recall: 0.4650 - val_loss: 1.0101 - val_accuracy: 0.6844 - val_precision: 0.8191 - val_recall: 0.5771
Epoch 26/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1994 - accuracy: 0.6155 - precision: 0.8152 - recall: 0.4697 - val_loss: 1.0383 - val_accuracy: 0.6750 - val_precision: 0.8124 - val_recall: 0.5634
Epoch 27/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2008 - accuracy: 0.6132 - precision: 0.8143 - recall: 0.4679 - val_loss: 1.0294 - val_accuracy: 0.6839 - val_precision: 0.8255 - val_recall: 0.5625
Epoch 28/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1971 - accuracy: 0.6136 - precision: 0.8158 - recall: 0.4701 - val_loss: 1.0420 - val_accuracy: 0.6763 - val_precision: 0.8011 - val_recall: 0.5799
Epoch 29/30
1144/1144 [==============================] - 98s 86ms/step - loss: 1.2015 - accuracy: 0.6143 - precision: 0.8149 - recall: 0.4680 - val_loss: 1.0280 - val_accuracy: 0.6855 - val_precision: 0.8317 - val_recall: 0.5605
Epoch 30/30
1144/1144 [==============================] - 97s 85ms/step - loss: 1.1929 - accuracy: 0.6158 - precision: 0.8133 - recall: 0.4725 - val_loss: 1.0201 - val_accuracy: 0.6809 - val_precision: 0.8256 - val_recall: 0.5668
614/614 [==============================] - 20s 31ms/step - loss: 1.0201 - accuracy: 0.6809 - precision: 0.8256 - recall: 0.5668
Test Accuracy (LR=0.001): 64.49%
Test Accuracy (LR=0.0001): 68.72%
Test Accuracy (LR=0.0004): 68.09%
Training Time (LR=0.001): 48.35 minutes
Training Time (LR=0.0001): 48.42 minutes
Training Time (LR=0.00045): 49.02 minutes
Best Learning Rate: 0.0001 (Test Accuracy: 68.72%)
```

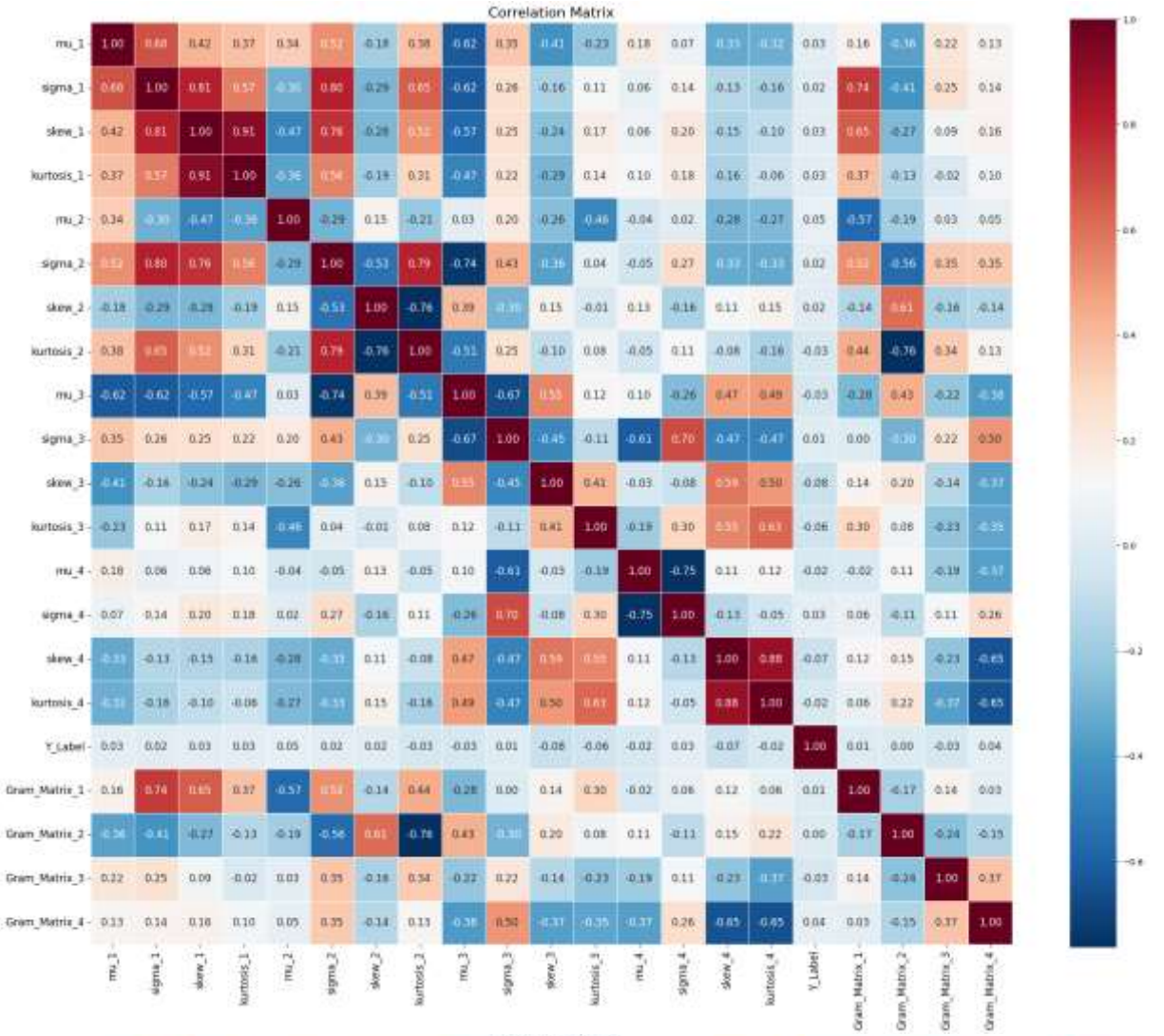
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_8 (UpSamplin  (None, None, None, None   0
 g2D)                        )

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d_8  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_16 (Dense)            (None, 256)               524544

 dropout_8 (Dropout)         (None, 256)               0

 dense_17 (Dense)            (None, 10)                2570

=================================================================
Total params: 24114826 (91.99 MB)
Trainable params: 527114 (2.01 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

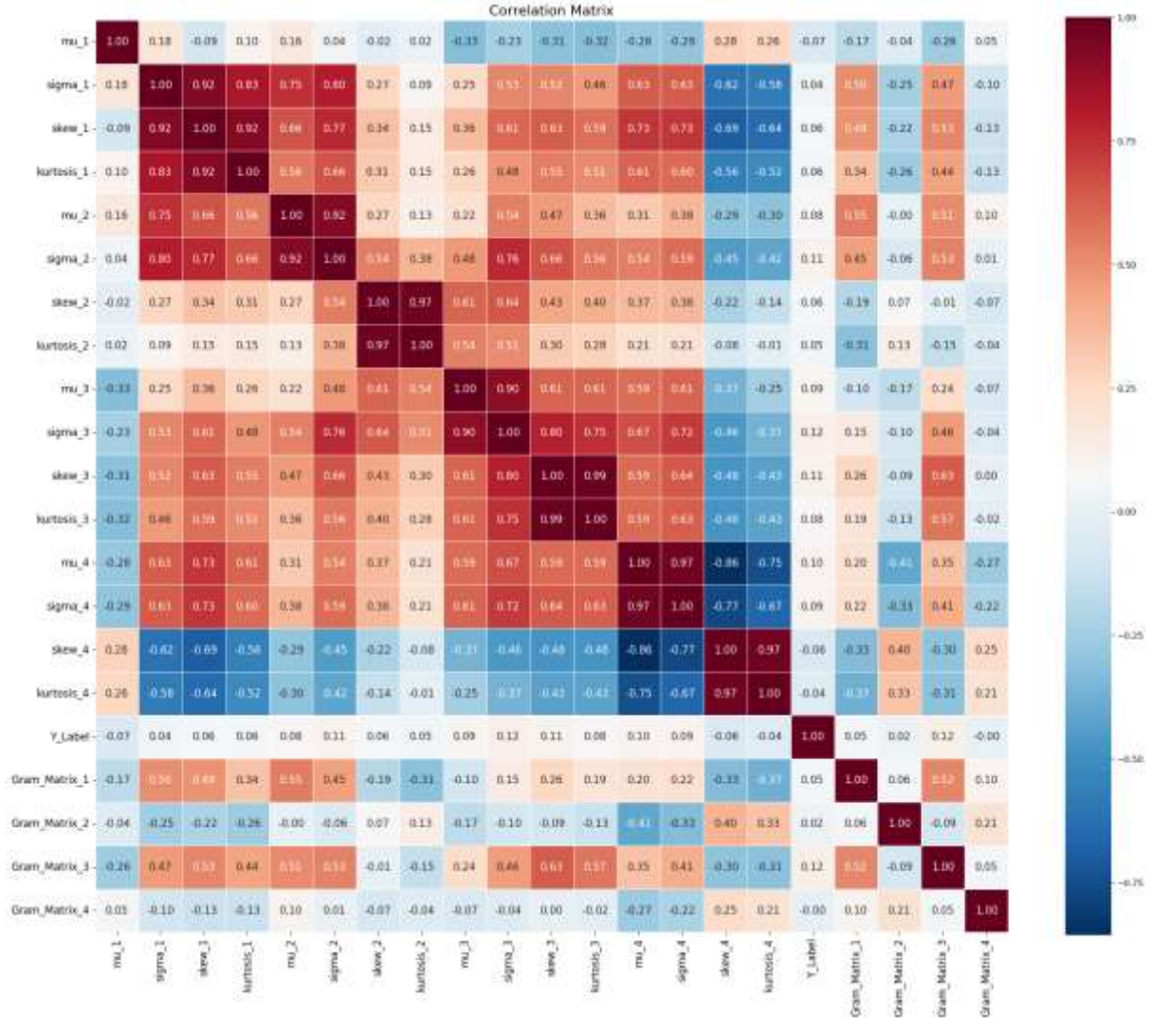

Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
mu_5
sigma_5
skew_5
kurtosis_5
Y_Label
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4
Gram_Matrix_5


Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
Y_Label
('image', '')
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4


Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
Y_Label
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4


Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
mu_5
sigma_5
skew_5
kurtosis_5
Y_Label
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4
Gram_Matrix_5


Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
Y_Label
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4


Correlation Matrix
mu_1
sigma_1
skew_1
kurtosis_1
mu_2
sigma_2
skew_2
kurtosis_2
mu_3
sigma_3
skew_3
kurtosis_3
mu_4
sigma_4
skew_4
kurtosis_4
Y_Label
Gram_Matrix_1
Gram_Matrix_2
Gram_Matrix_3
Gram_Matrix_4

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa