UNIVERSITÀ DI PISA

Department of Information Engineering
MSc Computer Engineering

Intelligent Systems

Traffic Signs Classification

Author:
Matteo Biondi

Academic Year: 2022/2023

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction to the context

Traffic signs classification is one of the foremost important integral parts of autonomous vehicles and advanced driver assistance systems (ADAS). Most of the time drivers missed traffic signs due to different obstacles and lack of attentiveness. Automating the process of classification of the traffic signs would help reduce accidents. [1][2][3] Traffic sign recognition systems (TSRS) are essential in many real-world applications such as autonomous driving, traffic surveillance, driver safety and assistance, road network maintenance, and analysis of traffic scenes. Normally, a TSRS concerns two related subjects which are traffic sign detection (TSD) and traffic sign recognition (TSR)[4]. Classification of traffic signs is not so simple task, images are affected by adverse variations due to illumination, orientation, the speed variation of vehicles, etc. Normally wide angle camera is mounted on the top of a vehicle to capture traffic signs and other related visual features for ADAS. [1] These images are distorted due to several external factors including vehicle speed, sunlight, rain, etc. Images are affected by variability, such as scale variations, bad viewpoints, motion-blur, faded colors, occlusions, and lightning conditions. [4][5] Traffic signs are always designed in specific shapes and highly saturated colors, such that the symbols or text are distinctive to its surrounding background.[5][2]

1.2 Problem Description

Traffic Sign Classification involves the application of classification techniques to classify traffic signs in real-world scenarios. An image representing a photo of a traffic sign is fed into the deep neural network which, after processing it, will classify its content in one of the sign classes. The task is to develop a robust model that can accurately identify various traffic signs from images. Unlike generic object classification, traffic sign recognition requires a model capable of understanding the specific visual symbols, colors, and shapes that convey critical information to drivers.

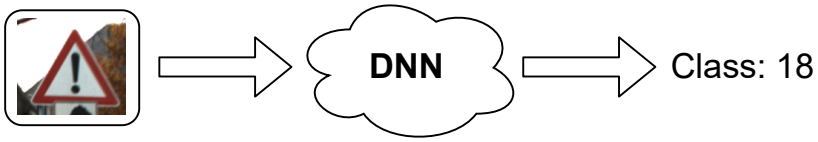

Figure 1: Traffic Sign Classification task

1.3 Dataset Description

The data used for classification comes from the GTSRB (German Traffic Sign Recognition Benchmark) dataset. The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. The dataset contains more than 50,000 images of different sizes (featuring different types of traffic signs) and exactly 43 signal classes. Each image is annotated with the corresponding class label, representing a specific type of traffic sign. Dataset includes:

- **Raw Image Data**: Images containing various traffic signs.
- **Class Labels**: Numeric representations of traffic sign categories.

The data needs an analysis and pre-processing phase before being fed into the network. For example:

- Images must be resized to a shared dimension
- Images must be organized in appropriate folders.
- The classes must be analyzed to study the balance of their samples and, if necessary, carry out appropriate balancing operations.

1.4 Use Cases

Traffic sign recognition has a wide range of potential applications in real-world scenarios. Some of the most common use cases include:

- **Driver assistance systems (ADAS)**: Traffic sign recognition can be used to improve the capabilities of ADAS systems, such as lane departure warning, adaptive cruise control, and blind spot detection. By accurately identifying traffic signs, ADAS systems can provide drivers with more timely and accurate warnings about upcoming road conditions.
- **Advanced driver monitoring systems (DMS)**: Traffic sign recognition can also be used to improve the capabilities of DMS systems, which monitor driver behavior to detect fatigue, distraction, and other potential hazards. By identifying traffic signs, DMS systems can provide drivers with alerts or interventions to help them stay focused and attentive on the road.

- **Intelligent transportation systems (ITS)**: Traffic sign recognition can also be used to develop ITS applications, such as traffic signal control and traffic monitoring systems. By accurately identifying traffic signs, ITS systems can optimize traffic flow and improve traffic safety.
- **Emergency vehicle navigation**: Traffic sign recognition can also be used to assist emergency vehicles in navigating to their destinations. By accurately identifying traffic signs, emergency vehicles can avoid congestion and detours and can arrive at the scene of an emergency more quickly.

In addition to these specific applications, traffic sign recognition has the potential to be used in a wide range of other applications, such as autonomous vehicles, parking assistance systems, and educational tools.

The development of accurate and robust traffic sign recognition models is an important area of research, with the potential to improve road safety and traffic efficiency for all road users.

1.5 Related Works and State Of The Art

There are many papers relating to this topic and there is a tendency to grow in the coming years. Among the many, some are mentioned below. The main objective of research in this field is to identify a solution capable of obtaining increasingly greater classification accuracy and the identification of increasingly less complex models. With the reduction of the number of parameters, a lesser number of computations and memory will be used, and therefore the possibility of using these models in devices with greater constraints (e.g. battery-powered) in line with the trend of the Internet Of Things (IoT)[1][6].

1.5.1 Classification Approaches

In the paper [1], the authors use a spatial transformer layer to make the network more robust to deformations such as translation, rotation, and scaling of input images. Instead of using a single-sized filter for one convolutional layer, they use multiple-sized filters and concatenate those convolutional filter responses to get more abstract representations in one layer. They have used SGD with momentum (0.9 with weight decay of 0.0918), with a minibatch size of 20 images and a learning rate of 0.00032, dropout (40%) for the fully connected layer. For activation Parametric Rectified Linear Unit (PReLU) is used. The total number of parameters is around 10.5 Million.

In the paper [4], the authors measure the impact of diverse factors with the end goal of designing a Convolutional Neural Network that can improve the state-of-the-art traffic sign classification task. First, different adaptive and non-adaptive stochastic gradient descent optimization algorithms such as SGD, SGDNesterov, RMSprop, and Adam are evaluated. Subsequently, multiple combinations of Spatial Transformer Networks placed at distinct positions within the main neural network are analyzed. They propose a traffic sign recognition system that carries out fine-grained classification of traffic sign images through a CNN whose main blocks are convolutional and spatial transformer modules. To find an accurate and efficient CNN for such a purpose, the proposed method for the recognition of traffic signs is a single CNN that combines several types of layers: convolutional, spatial transformer, Rectified Linear Units (ReLU), local contrast normalization, and max-pooling. These layers act as a feature extractor that maps raw pixel information of the input image to a tensor which is classified later into a particular traffic sign category by two fully connected layers. The total number of parameters for this CNN is around 14 Million.

As cited in paper [2], it is possible to categorise traffic sign recognition in different aspects, but one of the most accepted first stage categorisations is to split sign recognition into two categories, Support Vector Machines (SVMs) and learned-features classification methods. A support vector machine is primarily partnered with the use of Histogram of Oriental Gradients (HOG) features or with the use of a Gaussian kernel. The notable advantage of using SVMs compared to other classification methods is the fact that they are easier to train and they are additionally less prone to over-fitting as regularisation is used. A more recent solution is based on Deep Learning and in particular the application of CNN. In their paper, the authors propose a solution based on Deep Learning Methods and in particular CNNs and Three-Dimensional Image Depth. The CNN is simply composed of a sequence of convolution layers and pooling layers (the simplicity of this network inspires the structure used in the networks of this project)

In the literature, it is also possible to find papers like [7] that provide a comprehensive overview of the latest advancements in the field of traffic sign recognition, covering various key areas, including preprocessing techniques, feature extraction methods, classification techniques, datasets, and performance evaluation. The paper also delves into the commonly used traffic sign recognition datasets and their associated challenges. More in deep, the main contributions of this paper are as follows:

- A comprehensive review of state-of-the-art traffic sign recognition work, categorizing studies into conventional machine learning and deep learning approaches.

- A discussion of widely adopted traffic sign recognition datasets, their challenges, and limitations, as well as the future research prospects in this field.

As [7] says, the classification algorithms used in traffic sign recognition can be broadly divided into two categories: machine learning and deep learning.

Machine learning algorithms include traditional methods, such as Support Vector Machines (SVMs), k-Nearest Neighbor (k-NN), and decision trees. These algorithms are based on the concept of training a model on a dataset and then using that model to make predictions on new data. They have been widely used in traffic sign recognition due to their simplicity and efficiency.

On the other hand, deep learning algorithms use neural networks to model complex relationships between inputs and outputs. They have recently gained popularity in traffic sign recognition due to their ability to automatically learn high-level features from raw data, reducing the need for manual feature extraction. Deep learning algorithms, such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have been applied to the problem of traffic sign recognition with promising results.

Traffic sign classification has become a mature area with an increasing focus on autonomous driving research. Notable research work exists on the detection and classification of traffic signs for advanced driver assistance systems. Most of the works attempted to address the challenges involved in real-life problems due to scaling, rotation, blurring etc.[1]

The article [8] presents a model based on which some characteristics of the one presented in this report were created.
To further explore the state of the art you can also consult the articles [9], [10], [11], [12], [13].

1.5.2 Datasets

Regarding the database used, the 'German Traffic Sign Recognition Benchmark' (GTSRB) is one of the reliable datasets for testing and validating traffic sign classification and detection algorithms. In the competition of GTSRB, the top-performing algorithm exceeds the best human classification accuracy. GTSRB is the standard state-of-the-art revelation benchmark for traffic sign recognition/classification[1]. There are multiple reasons for choosing this dataset over the others, including the fact that it is highly accepted and is used for comparing traffic sign recognition approaches in the literature. Moreover, its authors and the organization behind them held a public competition ([14] to learn more about the topic) whereby scientists from different fields contributed with their results and tested the GSTRB dataset. The GTSRB

dataset contains traffic sign samples with different resolutions and image distortions that were extracted from video sequences. These samples each belong to one of the 43 existing classes. Traffic sign samples are raw RGB images whose size varies from 15×15 to 250×250 pixels [4]. The usage of this database limits the generalization of models trained on the GTSRB dataset and may result in a decreased performance when applied to other regions. Despite this limitation, the GTSRB dataset remains a popular resource due to its size, high-quality annotations, and real-world scenario representation, making it an excellent resource for researchers in the field of traffic sign recognition[7].

1.6 Project Goals

The objective of the project is to implement models capable of classifying road signs with an acceptable accuracy (at least accuracy $\geq 90\%$, higher is better) but with a reduced size compared to state-of-the-art models. In particular, the number of parameters that make up the model will be taken as a complexity metric. Finally, completely customized solutions will be compared with solutions based on the use of pre-trained networks. The solutions will be analyzed from the point of view of different metrics.

1.7 Work Environment

The project has been developed on a standard Google Colab environment that offers the following resources:

- Dual-core CPU
- About 12 GB of main memory
- About 78 GB of secondary storage
- Nvidia Tesla T4 GPU for limited time-sessions
- Only interactive sessions, no background operations admitted

The time-sessions GPU usage and only-interactive operations limit represented the main issues that prevented the possibility of running long operations needed during model design and training. The deep learning framework adopted is Keras along with Tensorflow as the back-end.

2 Dataset Analysis And Preprocessing

2.1 Analysis

This phase is dedicated to the analysis of the raw dataset and its preprocessing to obtain the final prepared dataset. The official website provides three different non-intersecting datasets. These can be used for training, validation, and testing.

As a preliminary phase, the three datasets and a CSV file associated with the test dataset are downloaded into the notebook. These resources are then filtered from unwanted files and finally reunited in a single zip file which represents the raw dataset from which the analysis and preprocessing phase starts.

By analyzing the contents of the "raw" dataset, it can be identified that all the datasets but 'Test' contains subfolders (one for each class) with associated images inside of them. The unzipped dataset also contains one CSV file ('Test.csv') that contains information about images(including their class) in the associated directory.

A first phase of reorganizing the data into subfolders for the Test dataset was, therefore, necessary so that each image was contained in a directory that represented its class. Below is the structure that represents the organization of the raw dataset, with subdivision between Train, Validation, and Test and for each the subfolders for each class of signs (identified by id from 0 to 42)

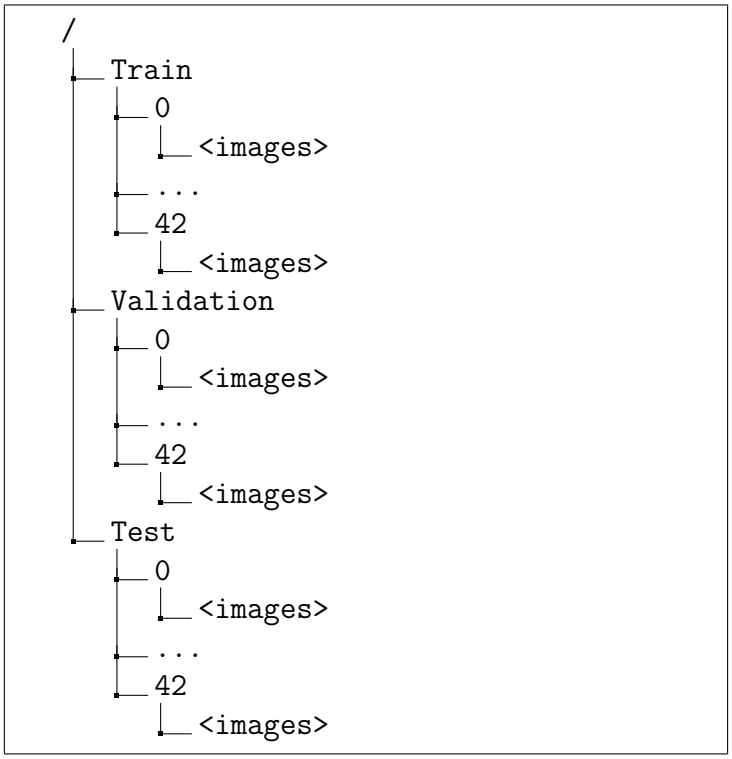

Figure 2: Hierarchical organization of raw dataset folders

As mentioned above, it can be noticed the images have different sizes so we need to resize all images in the Training, Validation, and Test datasets to specified dimensions. To do this it was chosen to fulfill this task in the model section at the loading data phase. Rescaling is fulfilled through a layer in the model.

Figure 3: Example images with different sizes

By subsequently analyzing the contents of the subfolders for each dataset to identify the distribution of images for each class, it can be seen that the dataset is highly unbalanced, both in the training samples, in the validation samples and in the test samples.

Figure 4: Histograms about classes unbalanced for Training, Validation, and Test sets

The class distribution in the Train and Validation sets was not distorted. The test set presents a different distribution of images per class.

2.2 Random Undersampling Balancing

In an attempt to achieve better training and fairer evaluation, different techniques can be applied to reduce imbalance. When it was provided, the dataset was already composed of some data that were highly dependent on each other because they were obtained from different frames of the same video track. With a more in-depth analysis, it can be deduced that each image has about 30 instances obtained by sampling the same video track multiple times.

It is important to balance the dataset if it is very unbalanced. An unbalanced dataset can lead to a machine learning model that is too good at classifying the most represented classes and too bad at classifying the least represented classes. This can lead to unsatisfactory results, such as low model accuracy or discrimination against certain classes.

In general, it is advisable to balance all three sets, the training set, the validation set, and the test set.

- Balancing the training set is particularly important because it is the set used to train the model. A model trained on an unbalanced dataset can be too good at classifying the most represented classes and too bad at classifying the least represented classes.
- Balancing the validation set can help to ensure that the model is able to generalize well to unseen data.
- Balancing the test set is useful for evaluating the actual performance of the model. A model tested on a balanced dataset is more likely to provide an accurate estimate of the model's performance in the real world.

To balance the images per class distribution there is the need to apply a resampling technique. Resampling is a process that changes the distribution of classes in the dataset. There are two main resampling techniques:

- Oversampling: This technique creates samples from underrepresented classes.
- Undersampling: This technique eliminates samples from overrepresented classes.

The resampling technique to use depends on the dataset and the model used.

In the case in question, applying oversampling techniques (SMOTE or image modification) on a dataset on which data are highly dependent on each other to balance the classes would increase (in the worst-case scenario) the number of images by more than 10 times. Images coming from the same video track are of different

sizes and present more significant differences when images are not time-adjacent to each other(coming from non-adjacent frames of the same video track). In fact, the images are the result of a temporal sequence of a traffic sign sampled in a video track. Initially, the traffic sign will be far away (and therefore the image will be small) and gradually it will be closer and therefore clearer and larger. However, the images created would be very similar and would create a high dependency between the images. This approach would lead to the creation of a dataset in which even a simple model could achieve a very high level of accuracy. However, these metrics would be untrue due to the dataset on which the model is trained and tested.

In order to obtain a model whose evaluation metrics were more truthful, it was chosen to balance the datasets through random undersampling, thus reducing the number of 'similar' images and enhancing the differences between the images of a single class. This makes the model more robust and therefore more suitable for practical use. Random undersampling technique still produces a considerable amount of data. This technique eliminates a certain number of random samples from overrepresented classes. By doing so, classes subfolders will contain as many images as there are images in the class with fewer images

In the next table the representation of the amount of data before and after Random Undersampling. Considering the samples in the different sets, the training set

Set	**Images Before Undersampling**	**Images After Undersampling**	**Images/Class After Undersampling**
Train	26640	6450	150
Validation	12569	2580	60
Test	12630	2580	60

Table 1: Table representing how the application of the Random Undersampling technique impacts the composition of the sets

is composed of approximately 56% of the images, while the validation and test sets consist of approximately 22%. Although the aspect ratios are slightly different from those usually used, these percentages have not been modified (moving images from one set to another) to respect the subdivision established by the creators of the challenge associated with the GTSRB dataset.

3 CNN From Scratch

Let's now create different models to train on the dataset obtained by means undersampling technique. The aim is to evaluate which of the models will obtain the best performance while still containing the complexity of the model as much as possible considering what is explained in chapter 1. Performance and simplicity will therefore guide the choice of the structure of the custom models presented below. This model will then be compared in performance with pre-trained models. For each custom model, some hyperparameters were established and fixed a priori, while others were subject to hyperparameter tuning. In particular, the hyperparameters established a priori and fixed for each model are:

- **Batch Size**: 32
- **Epochs**: 30
- **Early Stopping Patience**: 5
- **Early Stopping Metric**: Validation Loss
- **Input Shape**: (48, 48, 3)

If these hyperparameters undergo variations in one of the subsequent models, it will be explicitly indicated in the chapter referring to it.

The metrics taken into consideration during training are:

- **Training Loss**
- **Training Accuracy**
- **Validation Loss**
- **Validation Accuracy**

The final numeric performance evaluation metrics are:

- **Precision**: For a given class in multi-class classification is the fraction of instances correctly classified as belonging to a specific class out of all instances the model predicted to belong to that class.

  $$Precision_{ClassA} = \frac{TP_{ClassA}}{TP_{ClassA} + FP_{ClassA}}$$

  In other words, precision measures the model's ability to identify instances of a particular class correctly.

- **Recall**: in multi-class classification is the fraction of instances in a class that the model correctly classified out of all instances in that class.

$$Recall_{ClassA} = \frac{TP_{ClassA}}{TP_{ClassA} + FN_{ClassA}}$$

  In other words, recall measures the model's ability to identify all instances of a particular class.

- **F1-Score**: It is a metric that combines both precision and recall. It is defined as a simple weighted average (harmonic mean) of precision and recall. If we denote precision using P and recall using R, we can represent the F1 score as:

$$F1\_Score_{ClassA} = 2\frac{P_{ClassA} * R_{ClassA}}{P_{ClassA} + R_{ClassA}}$$

- **Loss**: It indicates the amount of error committed by the model. For all models, categorical cross-entropy was used as the loss function.

- **Accuracy**: It measures the proportion of correctly classified cases from the total number of objects in the dataset. To compute the metric, divide the number of correct predictions by the total number of predictions made by the model.

$$Accuracy = \frac{CorrectPredictions}{AllPredictions}$$

- **TopK (K=3) Accuracy**: This metric computes the number of times when the correct label is among the top k labels predicted (ranked by predicted scores). It is represented as a normalized value in the interval [0,1]

Where TP indicates True Positives, TN indicates True Negatives, FP indicates False Positives, and FN indicates False Negatives.
Precision, recall and F1-Score metrics can be used in multi-class classification problems and they can be represented in different ways.
There are different approaches:

- In the first case, metrics can be computed for each class individually

- In the second case, metrics are computed as "average" across all the classes to get a single number. It can be used different methods to average them, such as macro- or micro-averaging.

The approach to compute average precision, recall and F1-Score of multi-class classification problems are two:

- Macro-Average: Average the metric across all classes to get the final macro-averaged metric scores. In other words, calculates each class's performance metric (e.g., precision, recall) and then takes the arithmetic mean across all classes. So, the macro-average gives equal weight to each class, regardless of the number of instances.

  $$MetricX_{Macro-AVG} = \frac{\sum_{i=0}^{N} MetricX_{Class\_i}}{N}$$

  Where N is the total number of classes

- Micro-Average: It aggregates the counts of true positives, false positives, and false negatives across all classes and then calculates the performance metric based on the total counts. So, the micro-average gives equal weight to each instance, regardless of the class label and the number of cases in the class.

Considering that all classes are equally important and that our dataset is balanced, this report shows only the values of these metrics obtained through Macro-Average. To observe the values for the individual classes refer to the notebook in [15].
Furthermore, Confusion Matrix and ROC Curve (One vs All, which compares each class against all the others, assumed as one) graphs are presented for each model.
For training graphs refer to chapter 6 or notebook [15].

Unless otherwise specified, the ReLU function will be employed in the inner layers, and the Softmax function will be used in the output layer. Moreover, following the input layer, a Rescaling layer will always be included. This layer is a preprocessing layer that rescales input values to a new range, in our case to rescale an input in the [0, 255] range to be in the [0, 1] range.

3.1 Hyperparameters Tuning: Hyperband

As a first step in the development of the custom models, an initial hyperparameters tuning phase was performed to configure some hyperparameters to their optimal value. This phase was carried out by considering the structure of a basic model, from which all subsequent models were developed. Refer to 3.2 for more details on the basic model.

The choice to perform hyperparameters tuning on these hyperparameters was dictated by the papers analyzed to understand the state of the art (refer to section 1.5) and in particular to paper [4].

Hyperparameter	Analyzed Values	Optimal Value
Kernel Size	[2, 3, 4]	4
Optimizer	[Adam, RMSprop, SGD]	Adam
Learning Rate	[1e-2, 1e-3, 1e-4, 1e-5]	1e-3

Table 2: Table representing the Hyperparameters Tuning

3.2 Base Model

As an initial model, a basic CNN composed of three Conv2D layers was developed to process the input images. These three layers have 32, 64, and 128 filters respectively, and are followed by one MaxPooling2D layer each. No Dropout, Regularization, or Hidden Dense layers were used. The architecture is illustrated in the figure 5.

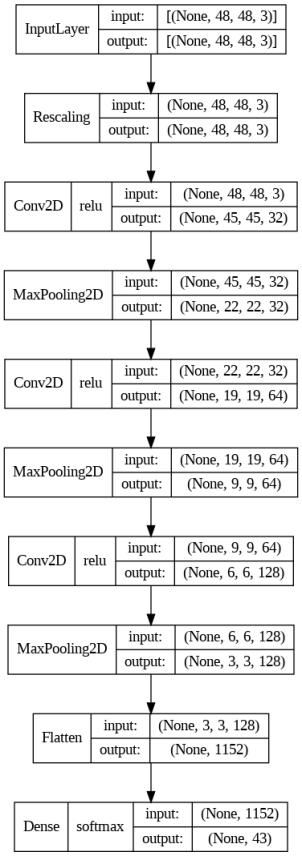

Figure 5: Custom base model architecture

This architecture has:

- **Total parameters**: 215179 (840.54 KB)
- **Trainable parameters**: 215179 (840.54 KB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 3 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
19	0.0050	0.9984	0.3870	0.9426	Yes

Table 3: Table representing the results obtained for the custom base model in training phase

Table 4 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.9123	0.9012	0.8997	0.7051	0.9012	0.9512

Table 4: Table representing the results obtained for the custom base model in testing phase

The confusion matrix and ROC curve are now reported.

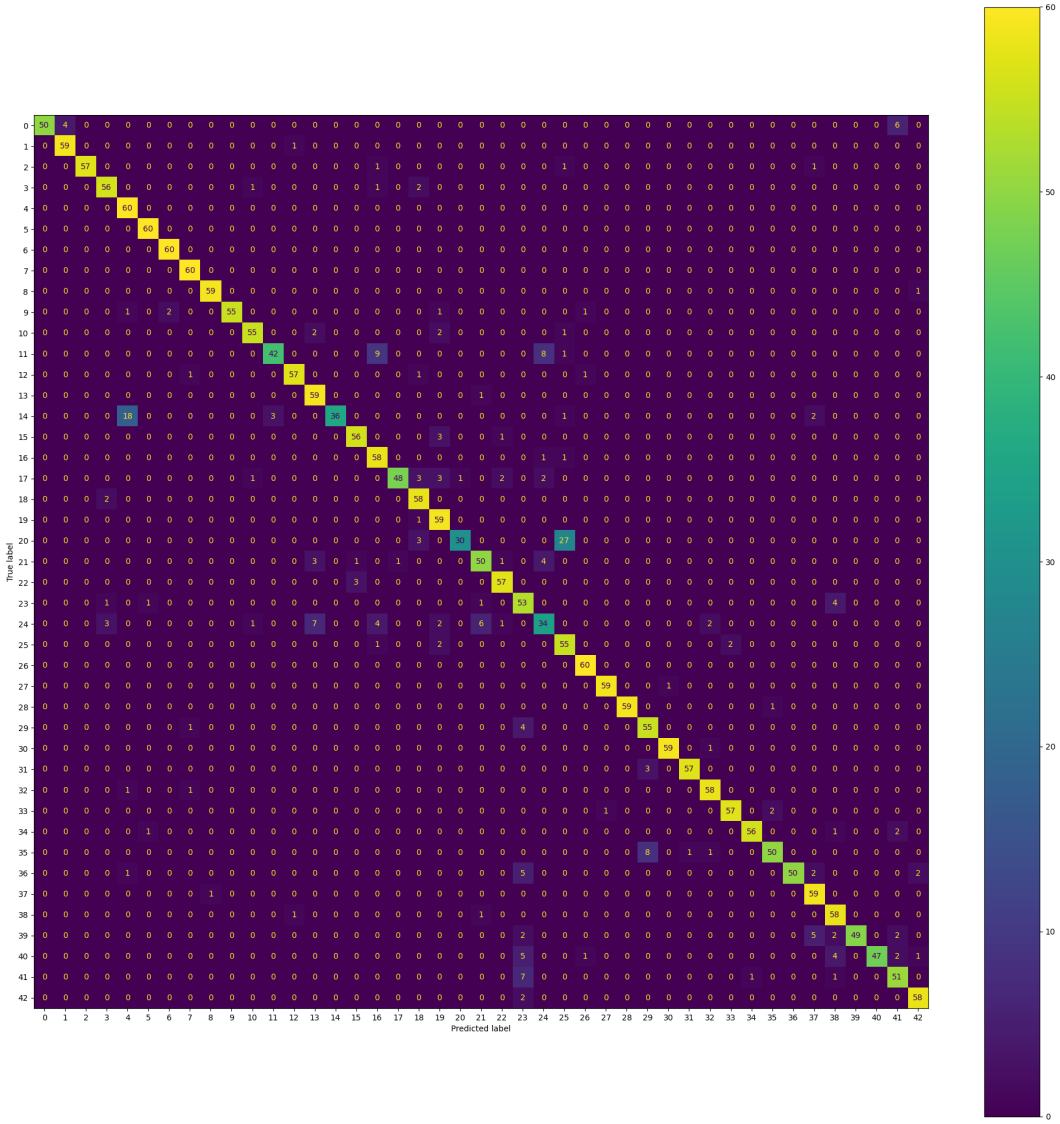

Figure 6: Custom base model confusion matrix

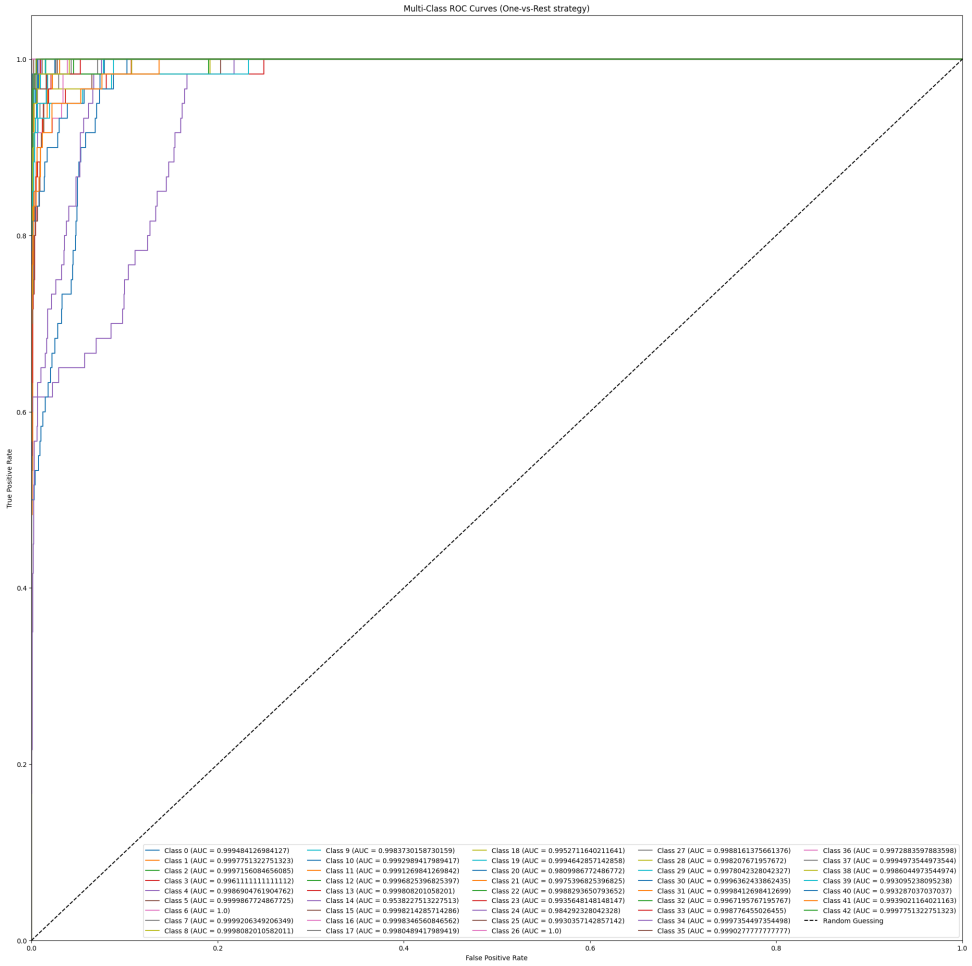

Figure 7: Custom base model ROC curve

The result of the base model is an accuracy of approximately 90.1% and a Top-K score of approximately 0.951. The model was trained for 19 epochs, after which the early stopping mechanism was activated for overfitting. Let's try to increase the number of filters to increase the network's ability to learn through new features.

3.3 Base Model With Filters Increased

This model is based on the initial CNN, so it is composed of three Conv2D layers to process the input images, but these three layers have different numbers of filters: 64, 128, and 256 filters respectively. As in the previous case, each Conv2D layer is followed by one MaxPooling2D layer. No Dropout, Regularization, or Hidden Dense

layers were used. The architecture is illustrated in the figure 8.

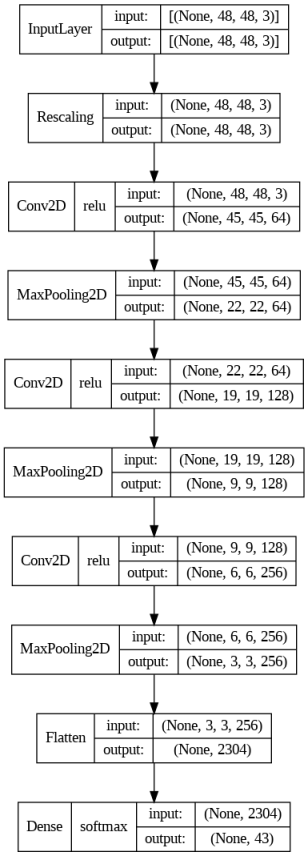

Figure 8: Custom base model with increased filters architecture

This architecture has:

- **Total parameters**: 757995 (2.89 MB)
- **Trainable parameters**: 757995 (2.89 MB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 5 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
14	0.0229	0.9943	0.2685	0.9357	Yes

Table 5: Table representing the results obtained for the custom base model with increased filters in training phase

Table 6 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.9149	0.9035	0.9019	0.4818	0.9035	0.9636

Table 6: Table representing the results obtained for the custom base model with increased filters in testing phase

The confusion matrix and ROC curve are now reported.

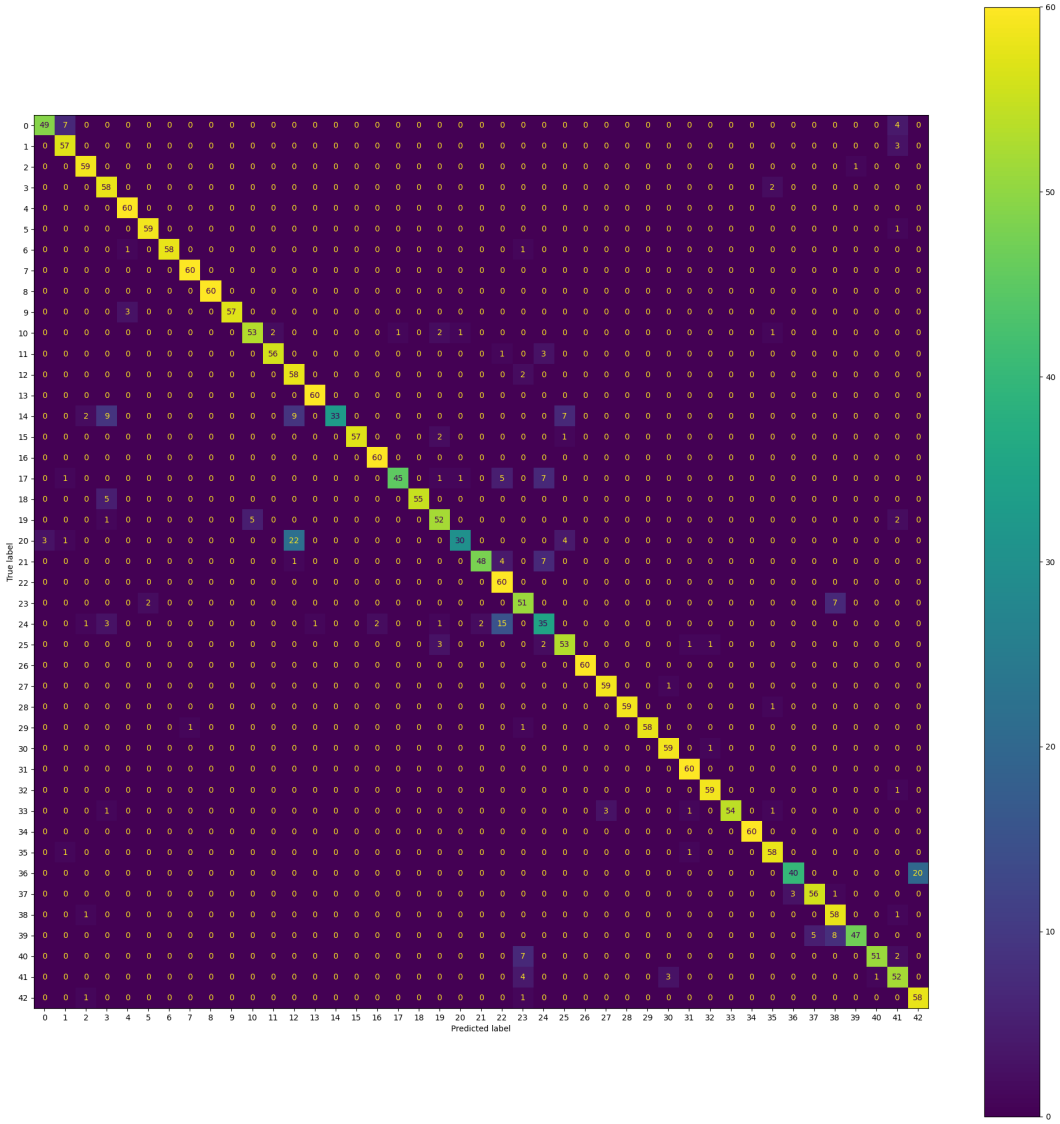

Figure 9: Custom base model with increased filters confusion matrix

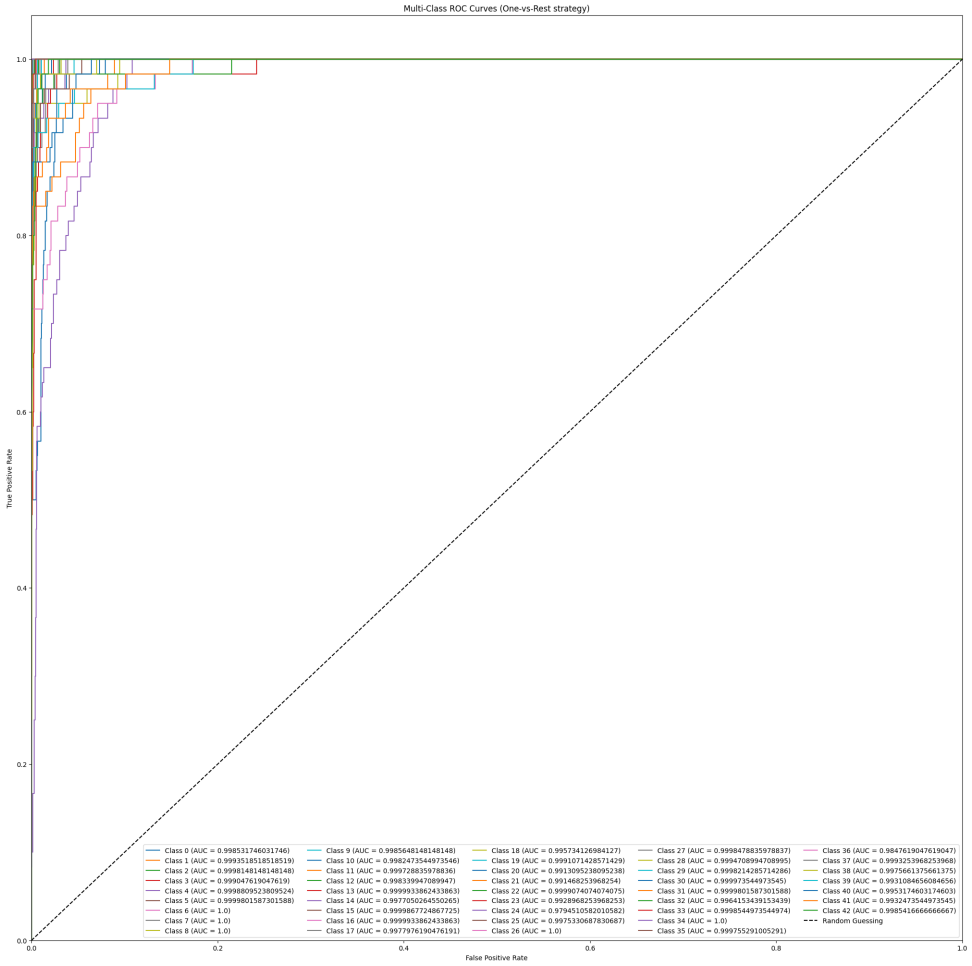

Figure 10: Custom base model with increased filters ROC curve

The basic model with the number of filters increased reaches an accuracy of approximately 90.3% and a Top-K score of approximately 0.964. The model was trained for 14 epochs, after which the early stopping mechanism was activated for overfitting. The worsening of overfitting is probably due to the increase in complexity of the network. Maintaining the newly tested structure, let's try to apply some techniques that mitigate overfitting.

3.4 Model With Dropout

This model is based on the previous one, so it is composed of three Conv2D layers to process the input images. These three layers have 64, 128, and 256 filters respectively.

As in the previous case, each Conv2D layer is followed by one MaxPooling2D layer. Now Dropout is added with a rate of 0.2 after the first and the second Conv2D-MaxPooling pairs and a rate of 0.5 before the last layer. No Regularization or Hidden Dense layers were used. The architecture is illustrated in the figure 11.

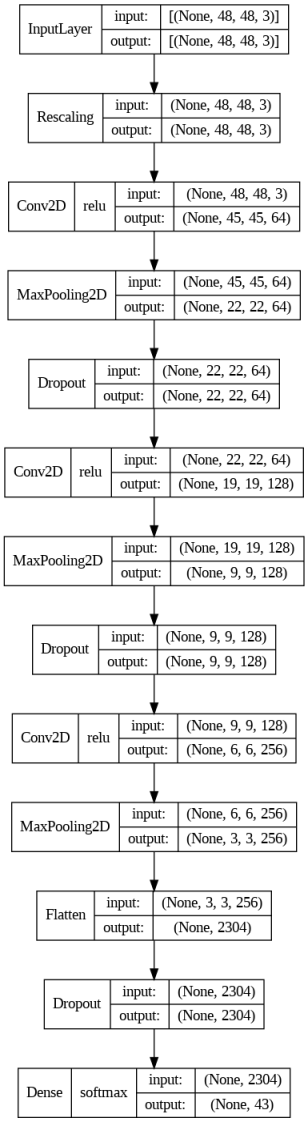

Figure 11: Custom model with dropout architecture

This architecture has:

- **Total parameters**: 757995 (2.89 MB)

- **Trainable parameters**: 757995 (2.89 MB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 7 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
23	0.0271	0.9918	0.2078	0.9535	Yes

Table 7: Table representing the results obtained for the custom model with dropout in training phase

Table 8 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.9493	0.9453	0.9447	0.2987	0.9453	0.9775

Table 8: Table representing the results obtained for the custom model with dropout in testing phase

The confusion matrix and ROC curve are now reported.

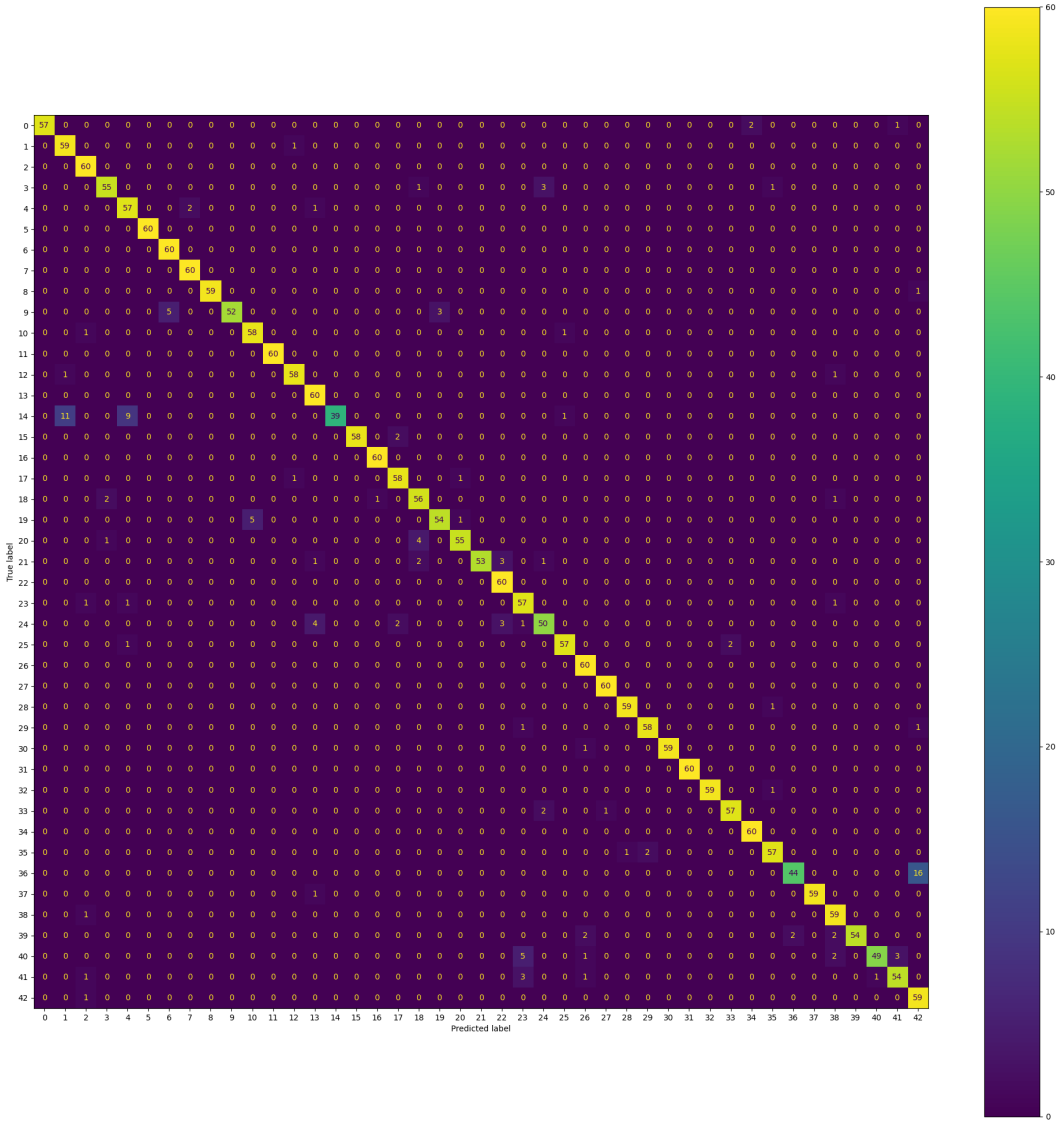

Figure 12: Custom model with dropout confusion matrix

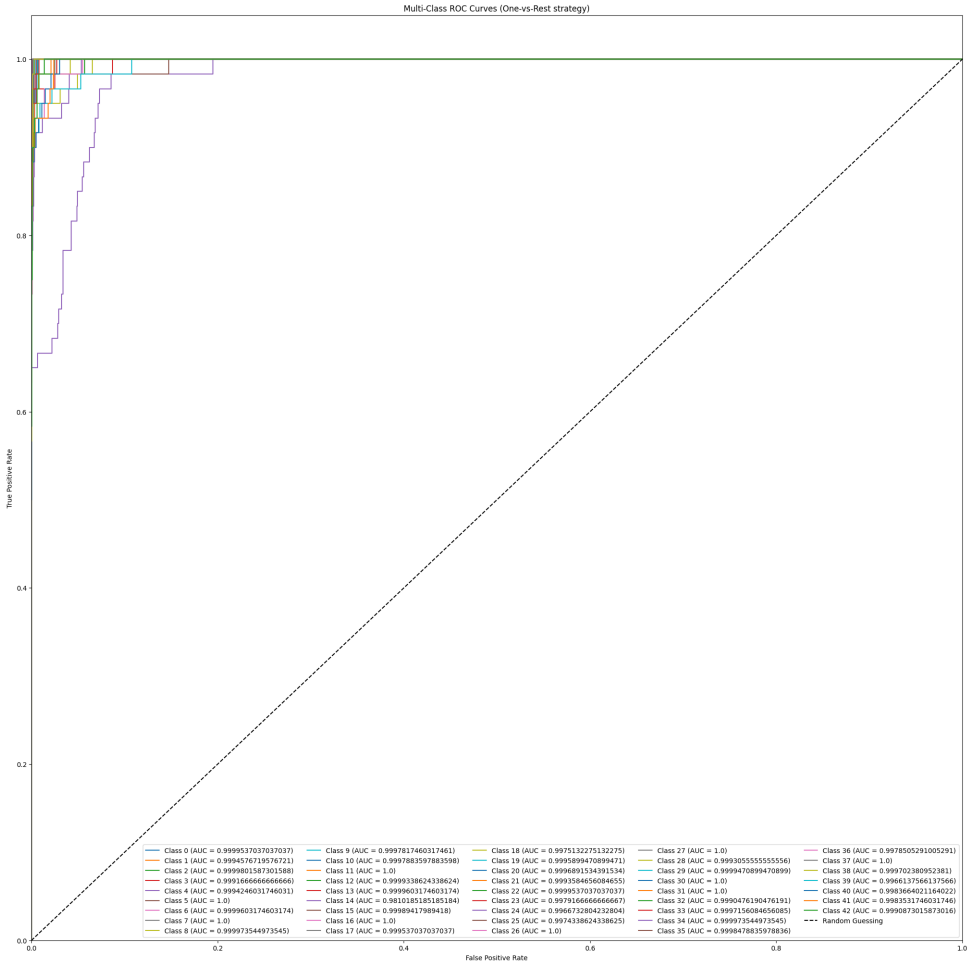

Figure 13: Custom model with dropout ROC curve

Thanks to the dropout application the model achieves an accuracy of approximately 94.5% and a Top-K score of approximately 0.978. The model was trained for 23 epochs, after which the early stopping mechanism was activated for overfitting. The dropout improved the overfitting condition.

3.5 Model With Dropout, L2-Regularization: Version 1

This model is based on the previous one, so it is composed of three Conv2D layers to process the input images. These three layers have 64, 128, and 256 filters respectively. As in the previous case, each Conv2D layer is followed by one MaxPooling2D layer. As before, Dropout is added with a rate of 0.2 after the first and the second Conv2D-

MaxPooling pairs and a rate of 0.5 before the last layer. Unlike the previous model, a hidden dense layer with 64 neurons and an L2-Regularization value of 0.01 has now been added before the last layer. A Dropout of rate 0.3 precedes this layer. The number of maximum epochs is increased to 50. The architecture is illustrated in the figure 14.

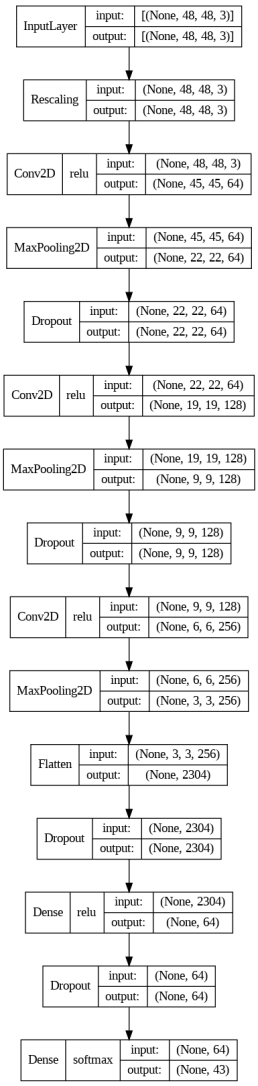

Figure 14: Custom model with dropout and L2-Regularization architecture: Version 1

This architecture has:

- **Total parameters**: 809195 (3.09 MB)

- **Trainable parameters**: 809195 (3.09 MB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 9 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
39	0.3323	0.9374	0.2409	0.9779	Yes

Table 9: Table representing the results obtained for the custom model with dropout and L2-Regularization in training phase: Version 1

Table 10 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.9660	0.9643	0.9639	0.2958	0.9643	0.986

Table 10: Table representing the results obtained for the custom model with dropout and L2-Regularization in the testing phase: Version 1

The confusion matrix and ROC curve are now reported.

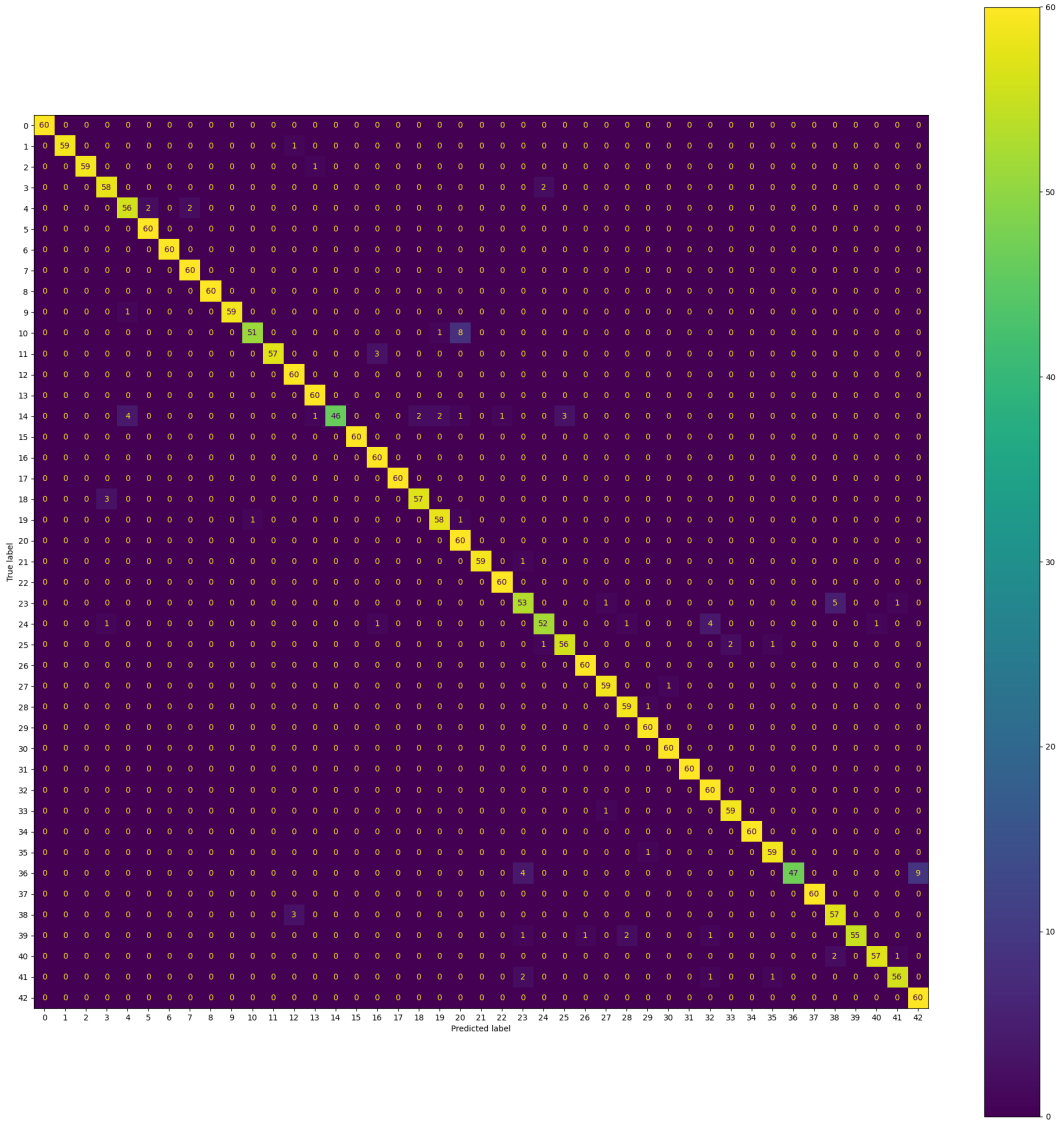

Figure 15: Custom model with dropout and L2-Regularization confusion matrix: Version 1

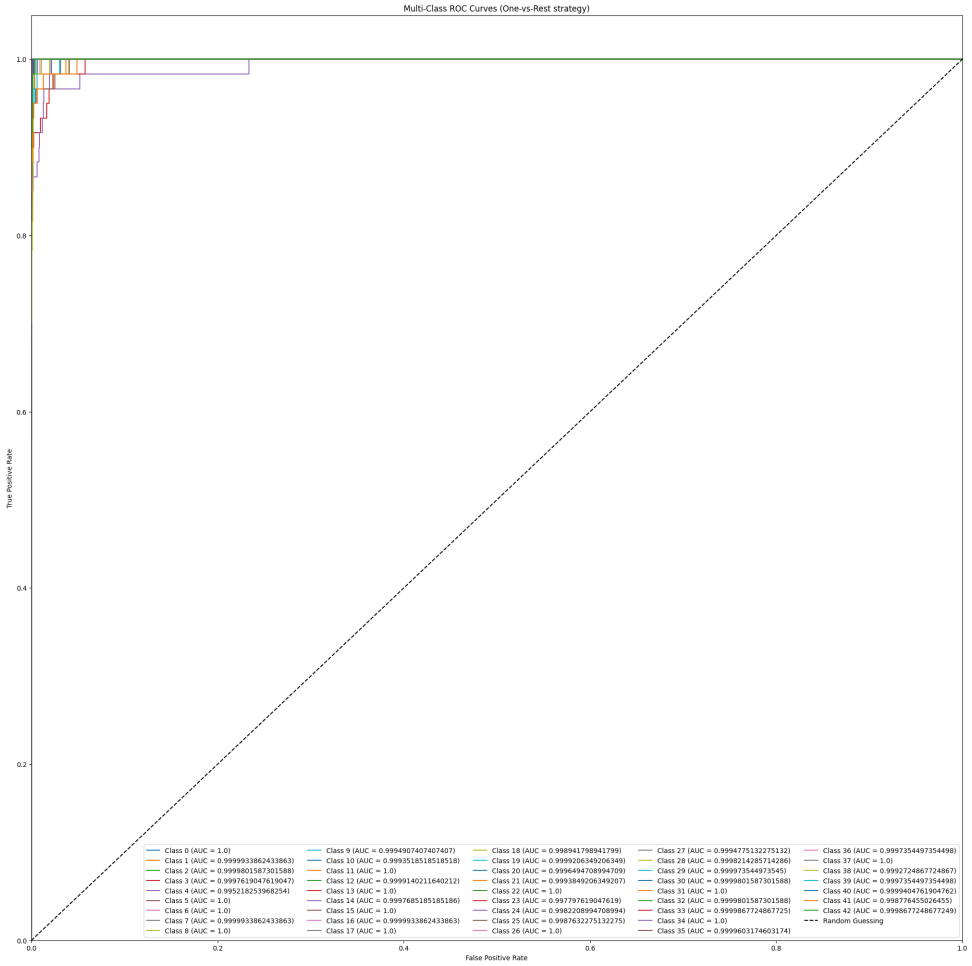

Figure 16: Custom model with dropout and L2-Regularization ROC curve: Version 1

Thanks to the application of L2-Regularization the model achieves an accuracy of approximately 96.4% and a Top-K (K=3) score of approximately 0.986. The model was trained for 39 epochs, after which the early stopping mechanism was activated for overfitting. The dropout and application of L2-Regularization improved the overfitting condition.

3.6 Model With Dropout, L2-Regularization: Version 2

This model is based on the previous one, so it is composed of three Conv2D layers to process the input images. These three layers have 64, 128, and 256 filters respectively. As in the previous case, each Conv2D layer is followed by one MaxPooling2D layer.

As before, Dropout is there with a rate of 0.2 after the first and the second Conv2D-MaxPooling pairs and a rate of 0.5 before the last layer. Unlike the previous model, the hidden dense layer before the last layer has 128 neurons with an L2-Regularization value of 0.01. A Dropout of rate 0.3 precedes this layer. The number of maximum epochs is increased to 50. The architecture is illustrated in the figure 17.

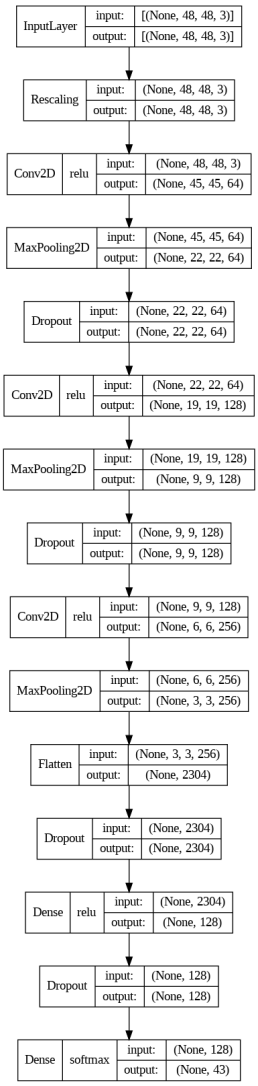

Figure 17: Custom model with dropout and L2-Regularization architecture: Version 2

This architecture has:

- **Total parameters**: 959467 (3.66 MB)

- **Trainable parameters**: 959467 (3.66 MB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 11 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
40	0.2865	0.9820	0.3068	0.9760	Yes

Table 11: Table representing the results obtained for the custom model with dropout and L2-Regularization in training phase: Version 2

Table 12 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.9587	0.9531	0.9527	0.3809	0.9531	0.9822

Table 12: Table representing the results obtained for the custom model with dropout and L2-Regularization in the testing phase: Version 2

The confusion matrix and ROC curve are now reported.

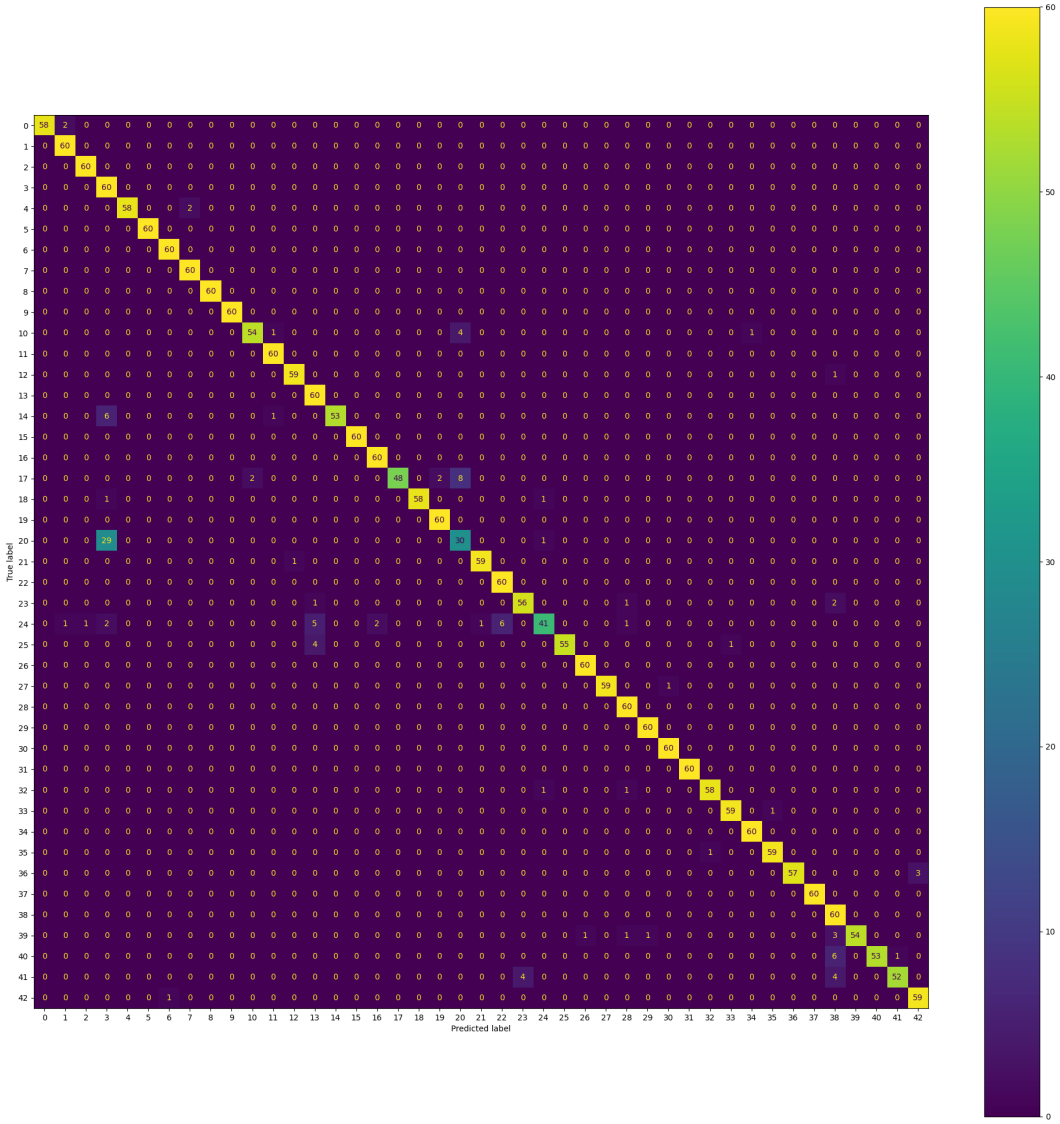

Figure 18: Custom model with dropout and L2-Regularization confusion matrix: Version 2

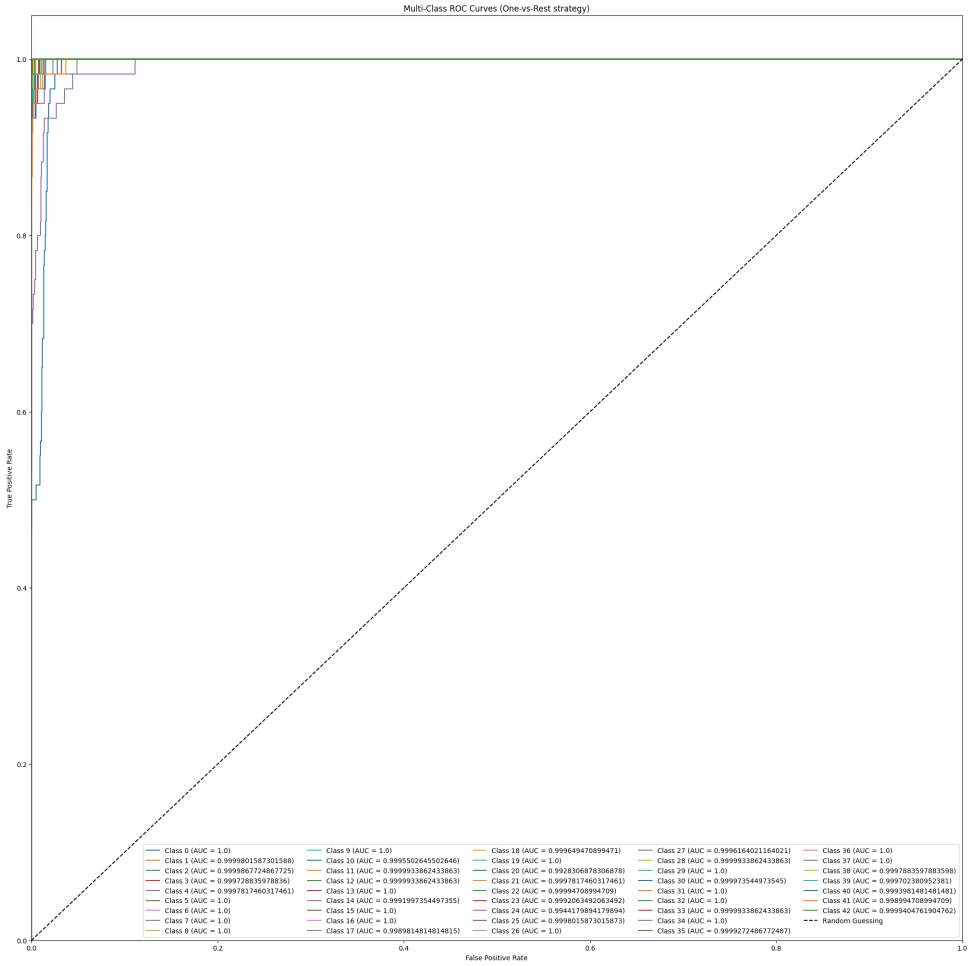

Figure 19: Custom model with dropout and L2-Regularization ROC curve: Version 2

The model achieves an accuracy of approximately 95.3% and a Top-K score (K=3) of approximately 0.982. The model was trained for 40 epochs, after which the early stopping mechanism was activated for overfitting. The model presents a slightly worse performance than the previous ones.

3.7 Model With Dropout, L2-Regularization: Version 3

This model is based on the model in chapter 3.5, so it is composed of three Conv2D layers to process the input images. These three layers have 64, 128, and 256 filters respectively. As in the previous case, each Conv2D layer is followed by one Max-Pooling2D layer. Now Dropout is there with a rate of 0.2 after the first Conv2D-

MaxPooling pairs, a rate of 0.3 after the second, a rate of 0.5 before the hidden dense layer, and a rate of 0.5 before the last layer. Unlike the previous models, the hidden dense layer has 64 neurons with an L2-Regularization value of 0.001. The number of maximum epochs is increased to 50. The architecture is illustrated in the figure 20.

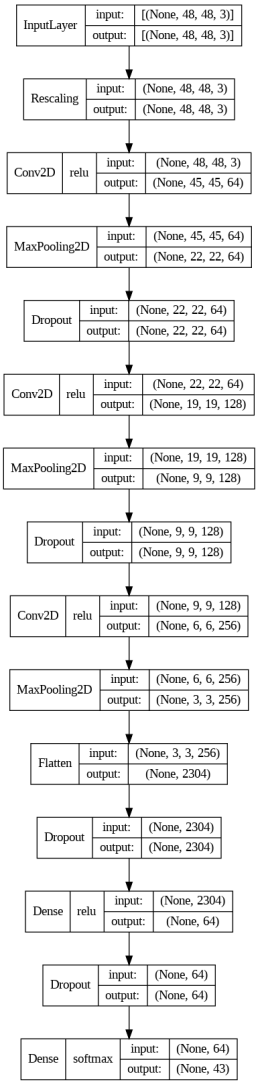

Figure 20: Custom model with dropout and L2-Regularization architecture: Version 3

This architecture has:

- **Total parameters**: 809195 (3.09 MB)

- **Trainable parameters**: 809195 (3.09 MB)
- **Non-trainable parameters**: 0 (0.00 Byte)

Table 13 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
30	0.3226	0.9416	0.2672	0.9682	Yes

Table 13: Table representing the results obtained for the custom model with dropout and L2-Regularization in training phase: Version 3

Table 14 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.9726	0.9705	0.9707	0.2635	0.9705	0.988

Table 14: Table representing the results obtained for the custom model with dropout and L2-Regularization in the testing phase: Version 3

The confusion matrix and ROC curve are now reported.

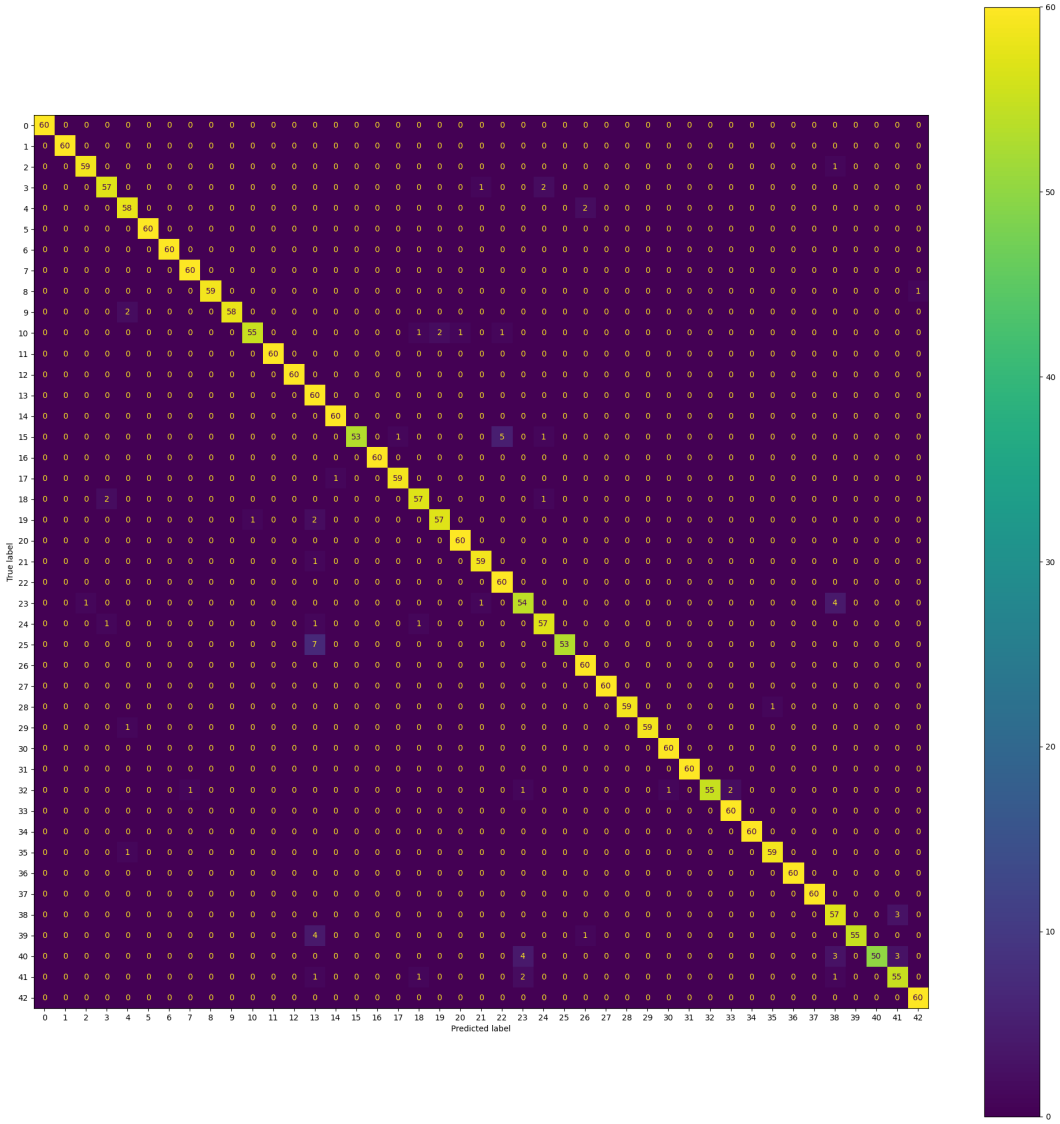

Figure 21: Custom model with dropout and L2-Regularization confusion matrix: Version 3

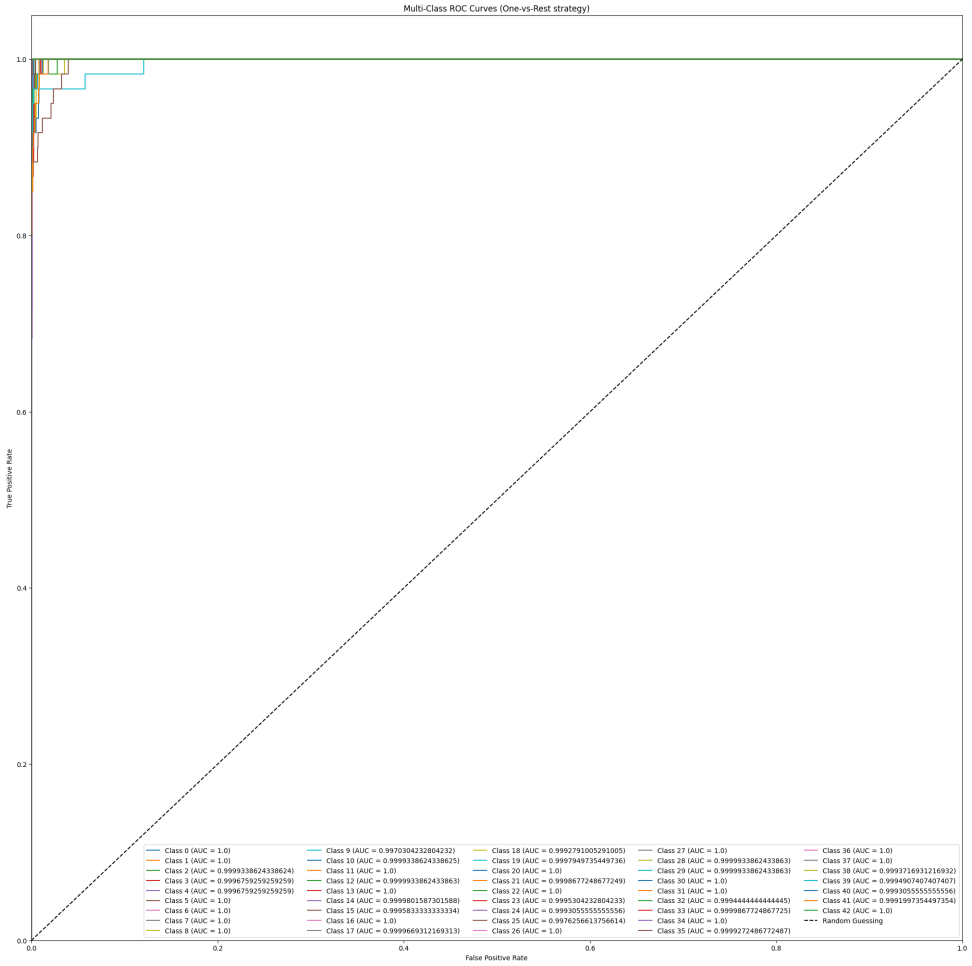

Figure 22: Custom model with dropout and L2-Regularization ROC curve: Version 3

Thanks to the application of L2-Regularization the model achieves an accuracy of approximately 97.1% and a Top-K (K=3) score of approximately 0.988. The model was trained for 30 epochs, after which the early stopping mechanism was activated for overfitting. This model is the one that achieves the best performance.

3.8 Summary Of Results And Considerations

In this chapter, various models have been explored to address the classification challenge. Through the analysis of the results, it was observed that the use of techniques such as Dropout and L2-Regularization played a crucial role in mitigating the overfitting issues in the problem context. After careful evaluation, Model 6 was chosen

as the best model to address the problem at hand. This decision is based on its ability to find an optimal balance between accuracy, loss, and simplicity. The performance of this model aligns well with our goal of achieving effective classification while maintaining a simple, maintainable, and interpretable architecture.

4 Pre-Trained Models

Leveraging a pre-trained network is a very common and highly effective approach. A pre-trained network is simply a saved network previously trained on a large dataset. The spatial hierarchy of features learned by the pre-trained network can act as a generic model for our case. The portability of learned features across different problems is a key factor. In the case in question, the VGG16 network from Keras was chosen.

VGG16 is a deep neural network architecture known for its simplicity and effectiveness in image classification. It consists of 16 layers, mostly using 3x3 convolutional filters and 2x2 max-pooling layers. VGG16 captures features at different scales, making it useful for tasks like recognizing objects in images. Its pre-trained weights enable quick adaptation to new tasks, making it a popular choice for transfer learning.

In the beginning, the basic convolutional of the VGG16 model was instantiated without including the top, with the weights coming from pre-training on ImageNet and an input shape of (48,48,3) as in the previous models. Next, before training the models, let's freeze the convolutional base. "Freezing" layers means preventing their weights from getting updated during training. The hyperparameter configuration discussed at the beginning of the 3 chapter is still valid. It is important to point out that the hidden dense layers, for each model below, have a linear activation function.

4.1 Base Model

The convolutional base starts the feature extraction phase. In the first model (the simplest) let's use the convolutional base in conjunction with a dense layer without L2 regularization (with 256 nodes) and the last dense layer for classification (with softmax activation function). No dropouts are used. This first model will be analyzed as a base model and further improvements will subsequently be added if necessary. The optimizer chosen is the same as the custom models described above. This model has such a simple structure to let the network learn and obtain a basic model with which to then perform comparisons. The architecture is illustrated in the figure 23.

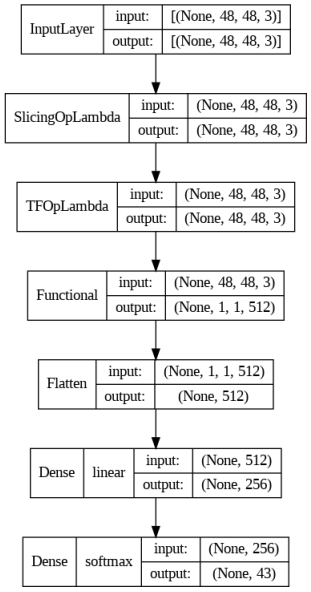

Figure 23: Pretrained base model architecture

This architecture has:

- **Total parameters**: 14857067 (56.68 MB)
- **Trainable parameters**: 142379 (556.17 KB)
- **Non-trainable parameters**: 14714688 (56.13 MB)

Table 15 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
7	2.7326	0.6936	9.0312	0.3919	Yes

Table 15: Table representing the results obtained for the pre-trained base model in training phase

Table 16 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.3947	0.3748	0.3654	10.1802	0.3748	0.5969

Table 16: Table representing the results obtained for the pre-trained base model in testing phase

The confusion matrix and ROC curve are now reported.

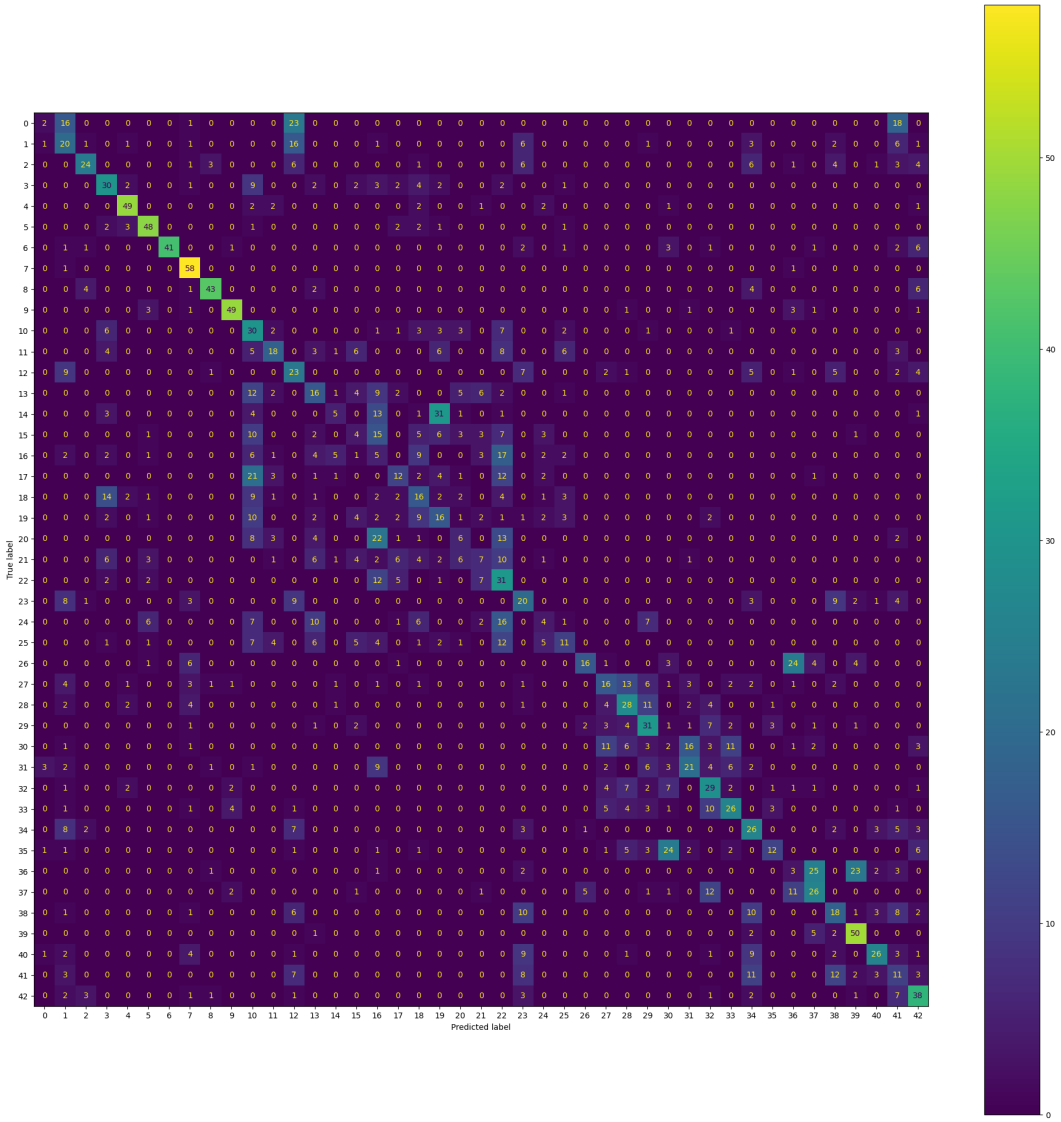

Figure 24: Pretrained base model confusion matrix

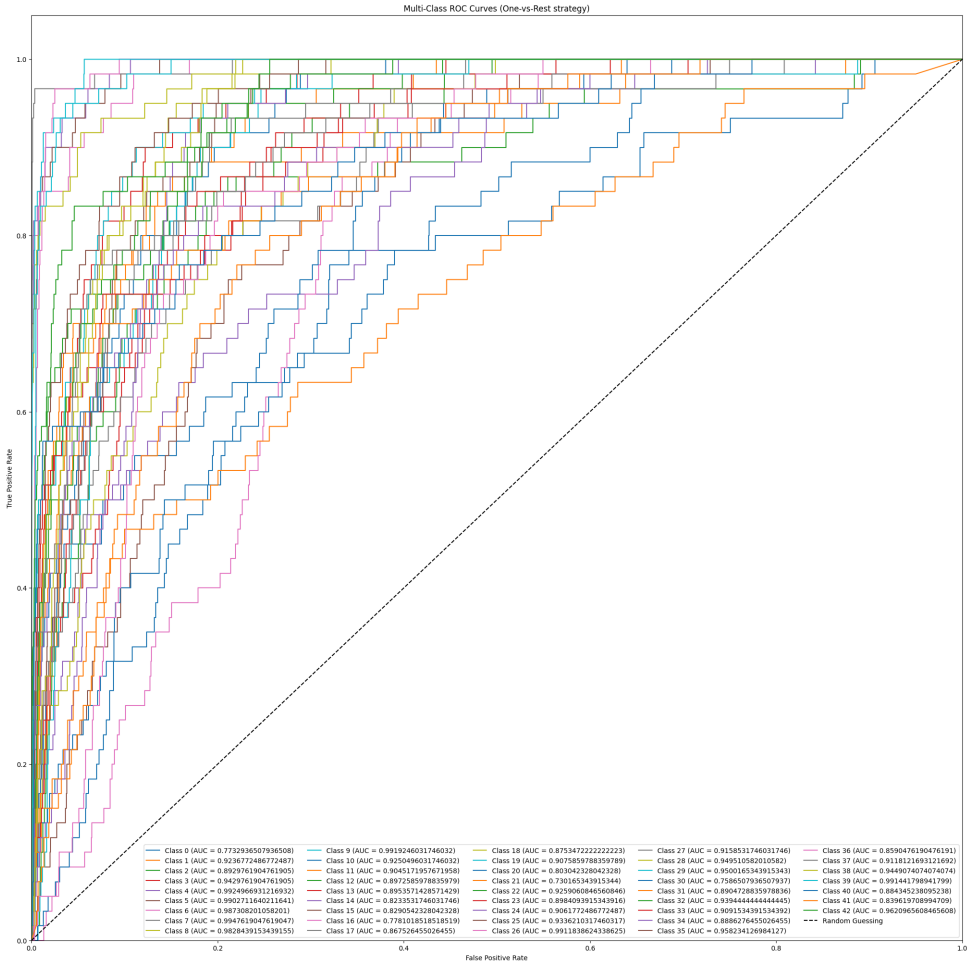

Figure 25: Pretrained base model ROC curve

The result of the base pre-trained model is an accuracy of approximately 37.5% and a Top-K (K=3) score of approximately 0.597. The model was trained for only 7 epochs, after which the early stopping mechanism was activated for overfitting. Maintaining the newly tested structure, let's try to apply some techniques that mitigate overfitting.

4.2 Model With Dropout, L2-Regularization: Version 1

The convolutional base starts the feature extraction phase. In the second model, as in the first one, let's use the convolutional base in conjunction with a dense layer with 256 nodes and the last dense layer for classification (with softmax activation

function). Now the hidden dense layer has L2-Regularization with a value of 0.01 and there is a dropout with a rate of 0.3 before the hidden dense layer and also before the last layer. The architecture is illustrated in the figure 26.

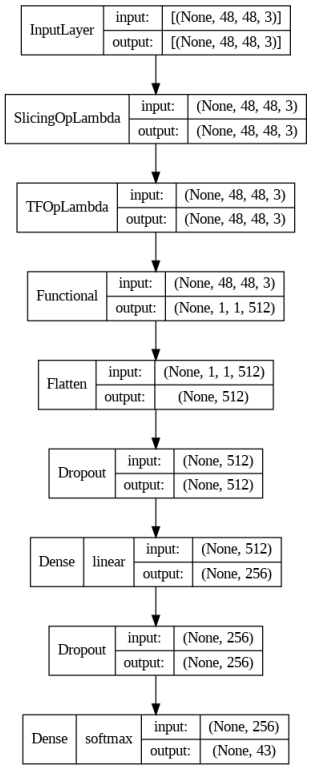

Figure 26: Pretrained model architecture with dropout and L2-Regularization: Version 1

This architecture has:

- **Total parameters**: 14857067 (56.68 MB)
- **Trainable parameters**: 142379 (556.17 KB)
- **Non-trainable parameters**: 14714688 (56.13 MB)

Table 17 summarizes the results of the training phase (maximum number of epochs is now 50) at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
35	1.9341	0.6048	2.9923	0.4070	Yes

Table 17: Table representing the results obtained for the pre-trained model with Dropout and L2-Regularization in training phase: Version 1

Table 18 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.4266	0.3981	0.3957	3.1365	0.3981	0.6663

Table 18: Table representing the results obtained for the pre-trained model with Dropout and L2-Regularization in the testing phase: Version 1

The confusion matrix and ROC curve are now reported.

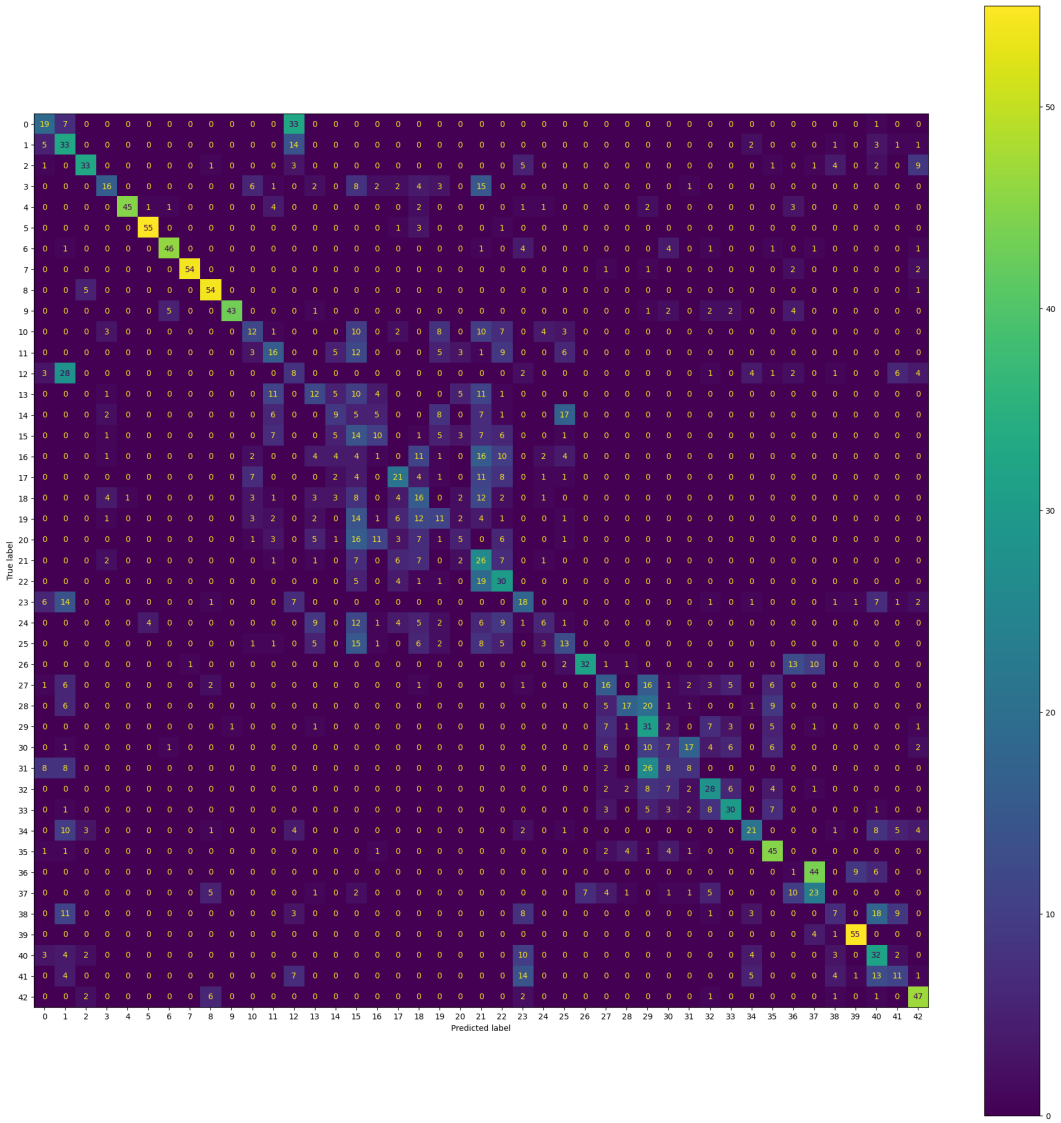

Figure 27: Pretrained model with Dropout and L2-Regularization confusion matrix: Version 1

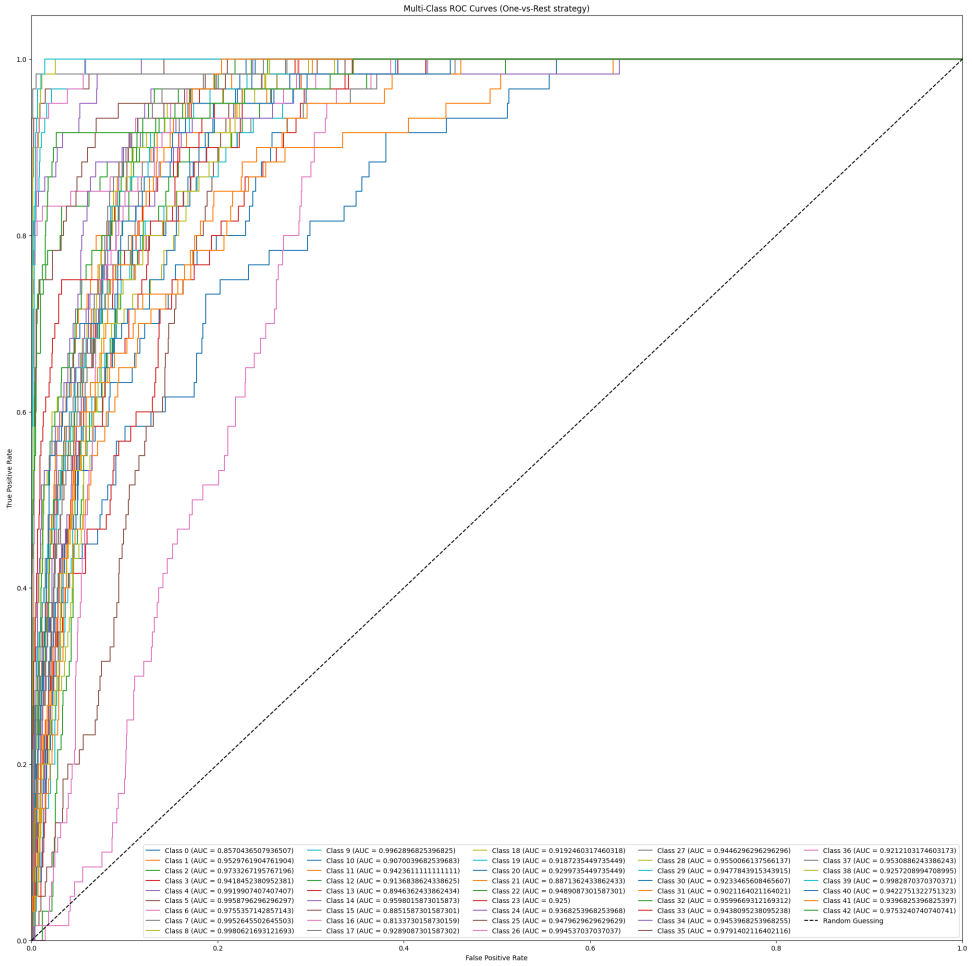

Figure 28: Pretrained model with Dropout and L2-Regularization ROC curve: Version 1

Thanks to the application of dropout and L2-Regularization the model achieves an accuracy of approximately 39.8% and a Top-K (K=3) score of approximately 0.666. The model was trained for 35 epochs, after which the early stopping mechanism was activated for overfitting. The mitigation techniques applied have had the desired effect but the network seems unable to learn well. It is important to underline a greater instability of the model compared to those previously discussed

4.3 Model With Dropout, L2-Regularization: Version 2

The convolutional base starts the feature extraction phase. In the third model, as in the previous one, let's use the convolutional base in conjunction with a dense layer

(that now is composed by 512 neurons to try to improve the learning capabilities of the network) and the last dense layer for classification (with softmax activation function). Now the hidden dense layer has L2-Regularization with a value of 0.01 and there is a dropout with a rate of 0.3 before the hidden dense. The architecture is illustrated in the figure 29.

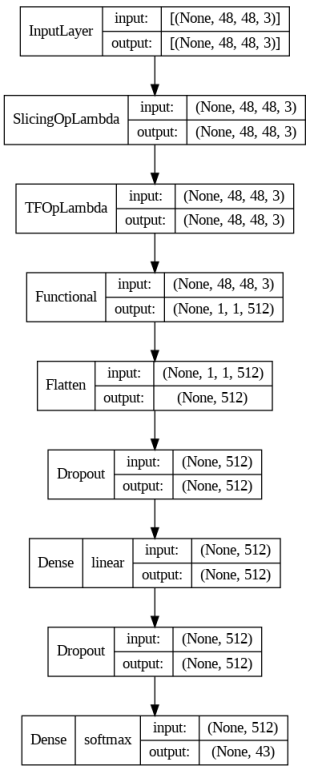

Figure 29: Pretrained model architecture with dropout and L2-Regularization: Version 2

This architecture has:

- **Total parameters**: 14999403 (57.22 MB)
- **Trainable parameters**: 284715 (1.09 MB)
- **Non-trainable parameters**: 14714688 (56.13 MB)

Table 19 summarizes the results of the training phase (maximum number of epochs is now 50) at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
41	2.5714	0.5667	3.8615	0.4035	Yes

Table 19: Table representing the results obtained for the pre-trained model with Dropout and L2-Regularization in training phase: Version 2

Table 20 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.4048	0.3729	0.3662	3.9391	0.3729	0.6256

Table 20: Table representing the results obtained for the pre-trained model with Dropout and L2-Regularization in the testing phase: Version 2

The confusion matrix and ROC curve are now reported.

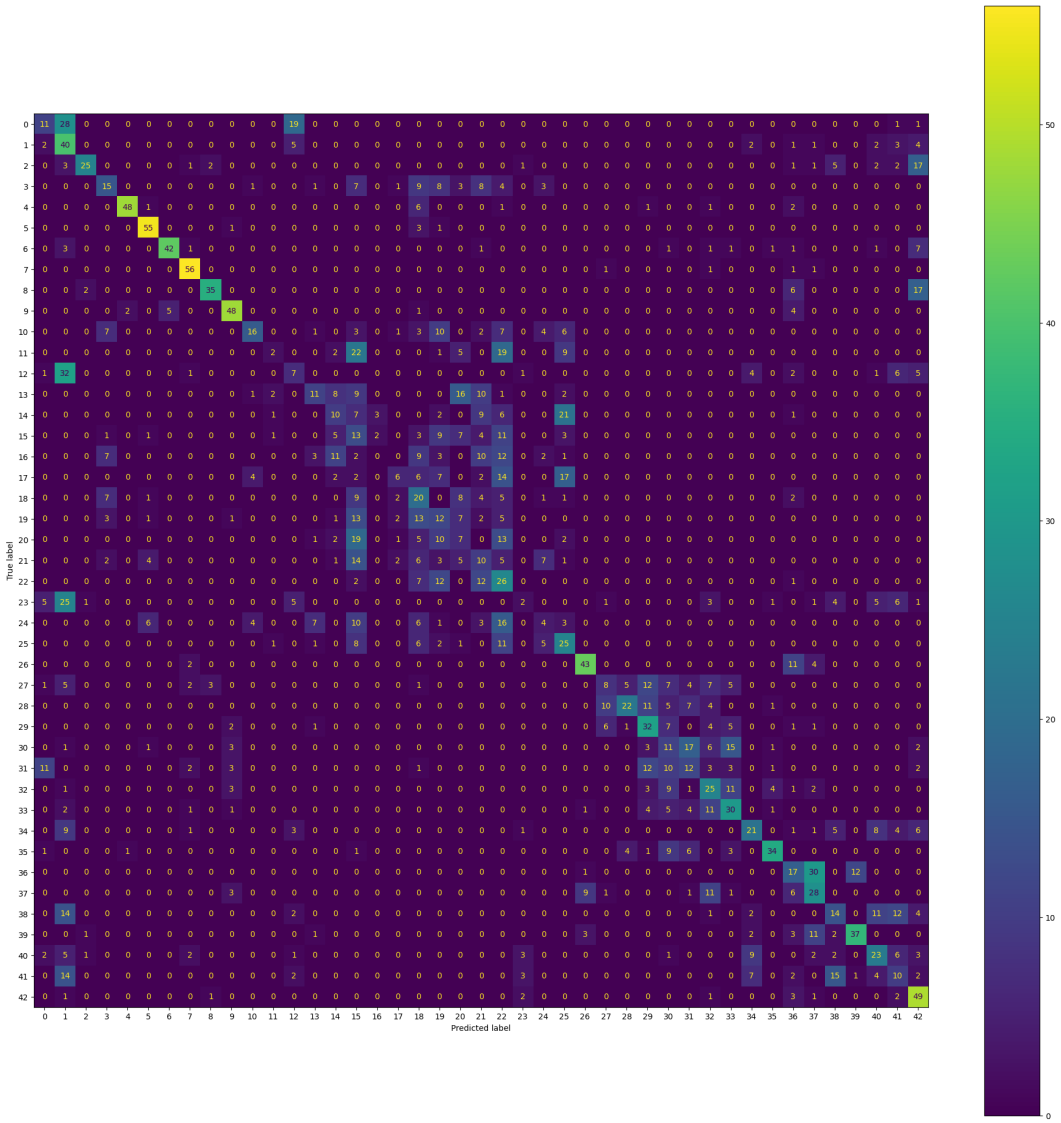

Figure 30: Pretrained model with Dropout and L2-Regularization confusion matrix: Version 2

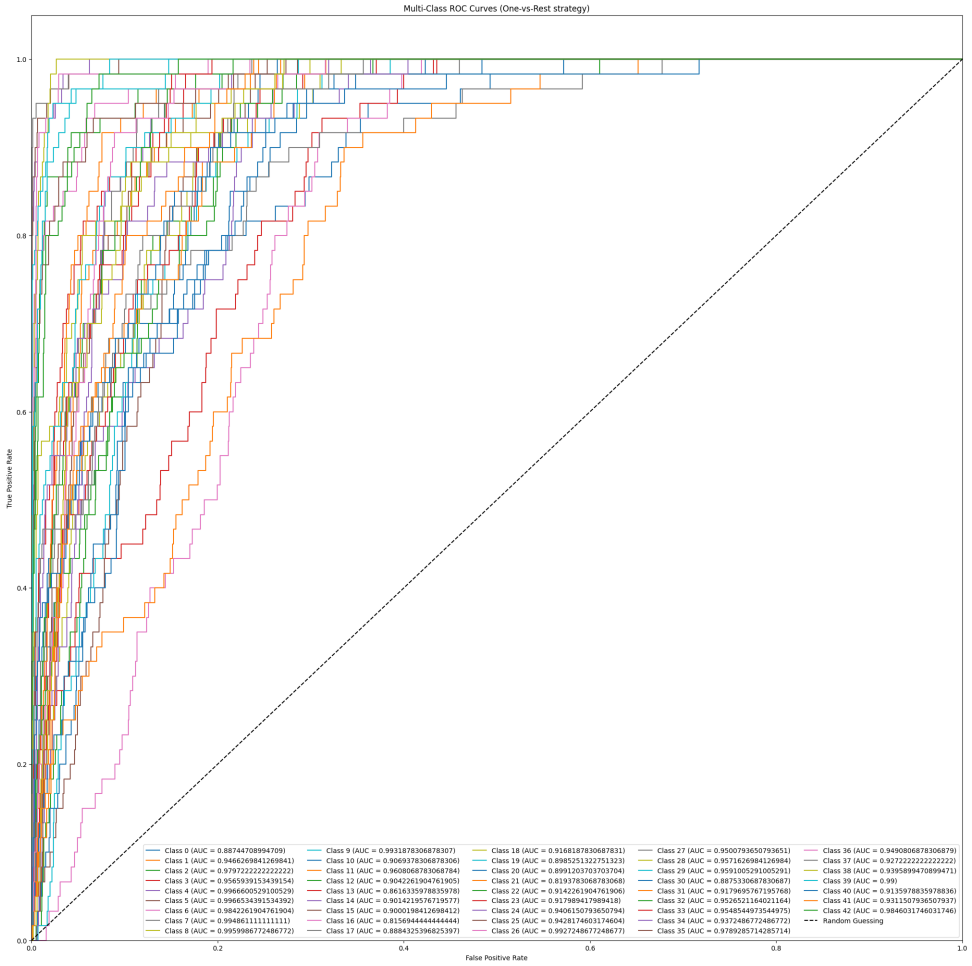

Figure 31: Pretrained model with Dropout and L2-Regularization ROC curve: Version 2

The model achieves an accuracy of approximately 37.3% and a Top-K (K=3) score of approximately 0.626, slightly worse than before. The model was trained for 41 epochs, after which the early stopping mechanism was activated for overfitting. The network seems incapable of learning well from features as specific as those output from VGG16.

4.4 Model With Feature Extraction From Block 4

In an attempt to increase the learning capabilities of the network, VGG16 features are now extracted from layers before the last one. The network presents a different structure compared to those previously observed (when "all" VGG16 was used) to

obtain a network capable of learning and limiting overfitting if possible. For this last objective, the network is also equipped with dropout and L2-Regularization. In this model, the features are extracted from the layer "block4_pool" to use more general features than the ones previously used. After the 'Flatten' layer the architecture has a dropout with a rate of 0.5, a dense layer with 128 nodes and L2-Regularization with a value of 0.01, a dropout layer with a rate of 0.3 and the last dense layer for classification (with softmax activation function). The architecture is illustrated in the figure 32.

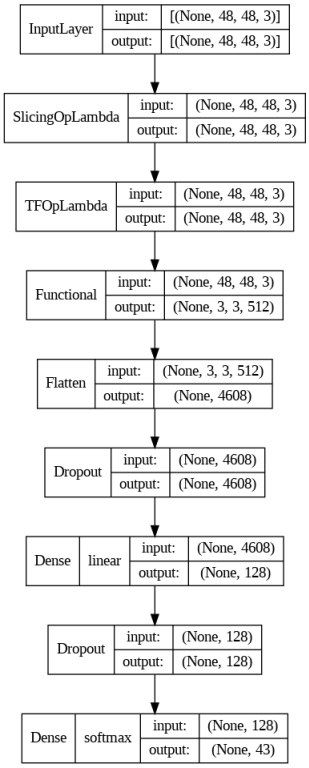

Figure 32: Pretrained model architecture with features extraction from Block4

This architecture has:

- **Total parameters**: 8230763 (31.40 MB)
- **Trainable parameters**: 595499 (2.27 MB)
- **Non-trainable parameters**: 7635264 (29.13 MB)

Table 21 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	**Train Loss**	**Train Acc.**	**Valid. Loss**	**Valid. Acc.**	**Early St.**
11	92.3214	0.8564	233.5148	0.7031	Yes

Table 21: Table representing the results obtained for the pre-trained model with features extraction from Block4 in training phase

Table 22 summarizes the results of the testing phase.

Precision	**Recall**	**F1-Score**	**Loss**	**Accuracy**	**TopK Accuracy**
0.7168	0.6829	0.6819	263.0163	0.6829	0.7198

Table 22: Table representing the results obtained for the pre-trained model with features extraction from Block4 in testing phase

The confusion matrix and ROC curve are now reported.

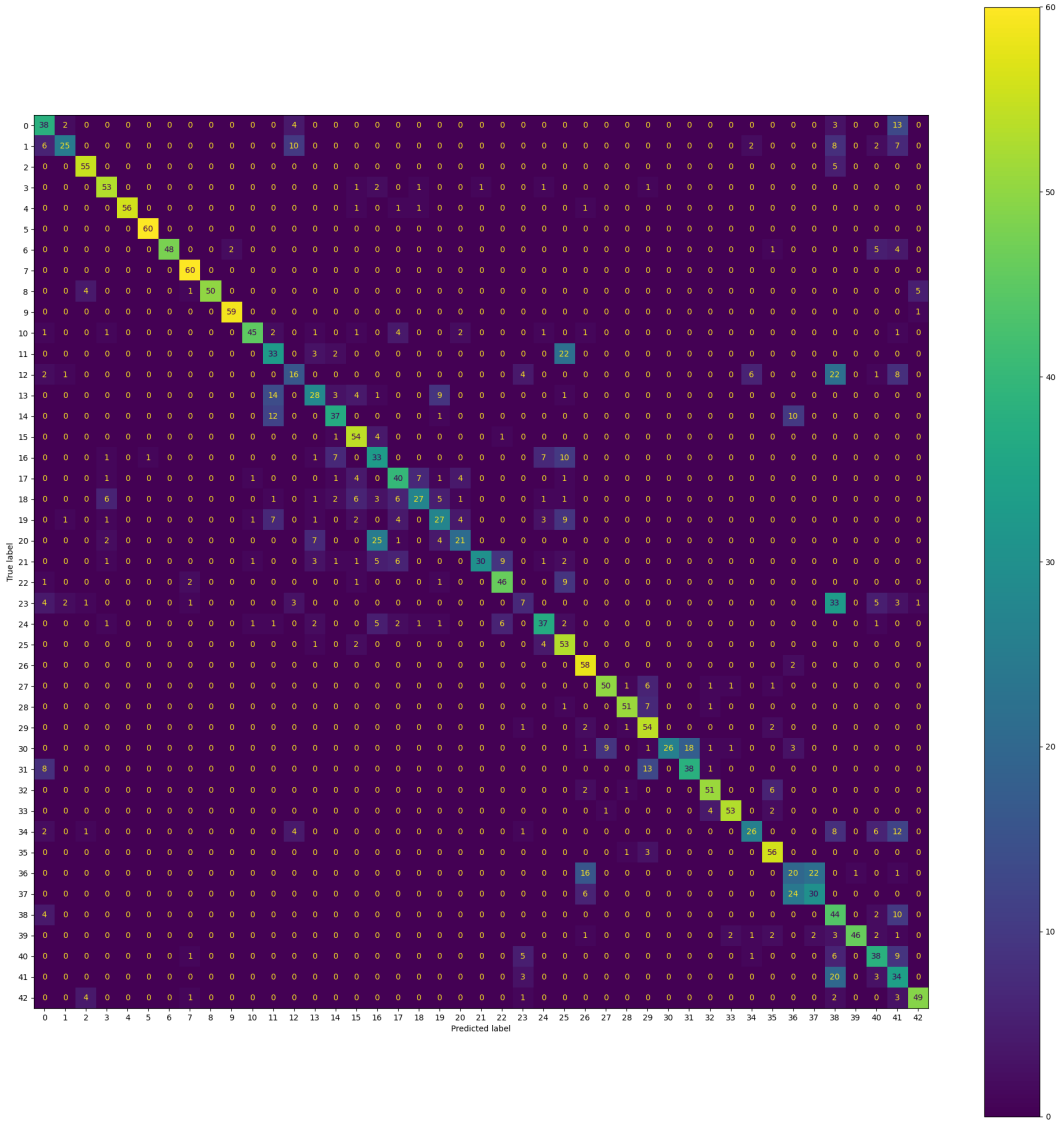

Figure 33: Pretrained model with features extraction from Block4 confusion matrix

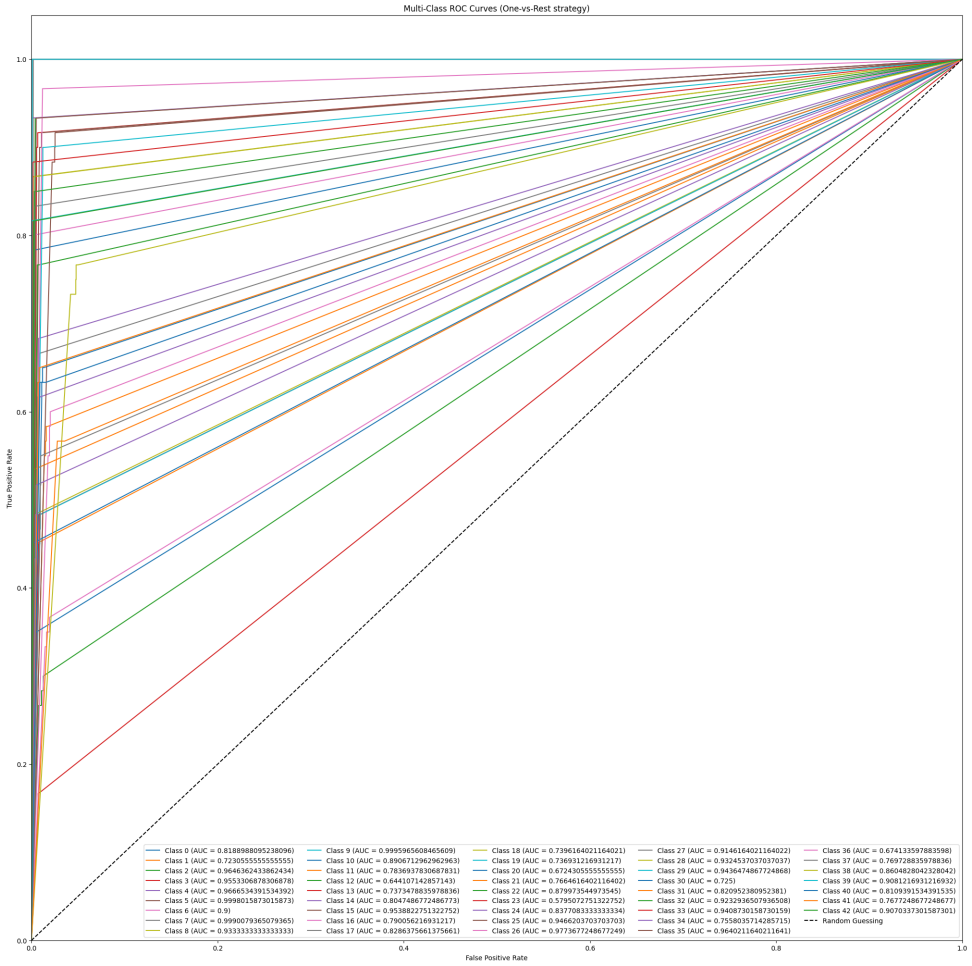

Figure 34: Pretrained model with features extraction from Block4 ROC curve

Thanks to the use of features extracted from 'block4_pool', the model achieves an accuracy of around 68.3% and a Top-K score (K=3) of around 0.72. The model was trained for 11 epochs, after which the early stopping mechanism was activated for overfitting. This result shows the effectiveness of intuition, greatly increasing the performance obtained. The instability of the network should be highlighted.

4.5 Model With Feature Extraction From Block 3

In order to evaluate how the network behaves with even more general features, in this model, the features are extracted from the layer "block3_pool". The network presents the same structure compared to the one previously observed. After the

'Flatten' layer the architecture has a dropout with a rate of 0.5, a dense layer with 128 nodes and L2-Regularization with a value of 0.01, a dropout layer with a rate of 0.3 and the last dense layer for classification (with softmax activation function). The architecture is illustrated in the figure 35.

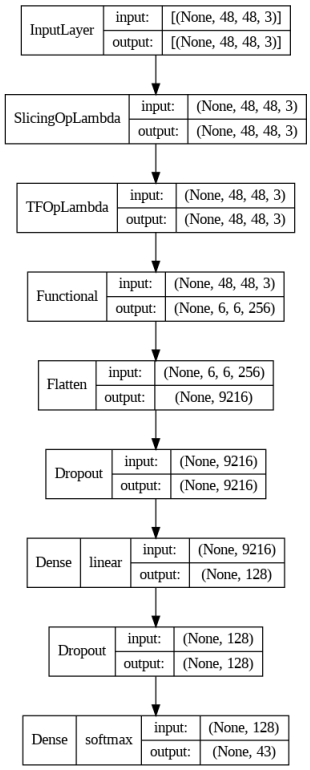

Figure 35: Pretrained model architecture with features extraction from Block3

This architecture has:

- **Total parameters**: 2920811 (11.14 MB)
- **Trainable parameters**: 1185323 (4.52 MB)
- **Non-trainable parameters**: 1735488 (6.62 MB)

Table 23 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
11	410.8558	0.9295	925.6184	0.8690	Yes

Table 23: Table representing the results obtained for the pre-trained model with features extraction from Block3 in training phase

Table 24 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.8995	0.8736	0.8730	708.1760	0.8736	0.8806

Table 24: Table representing the results obtained for the pre-trained model with features extraction from Block3 in testing phase

The confusion matrix and ROC curve are now reported.

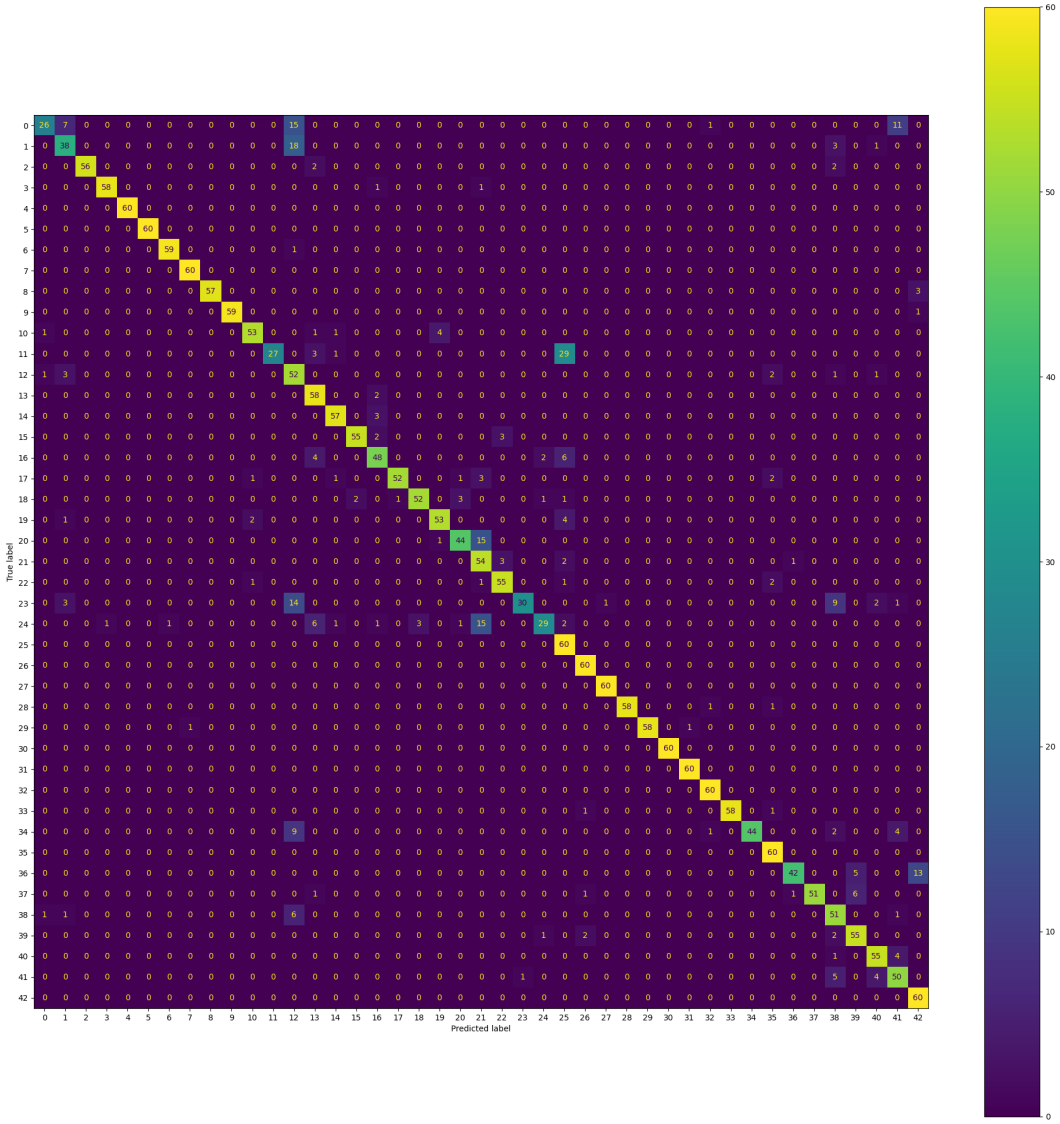

Figure 36: Pretrained model with features extraction from Block3 confusion matrix

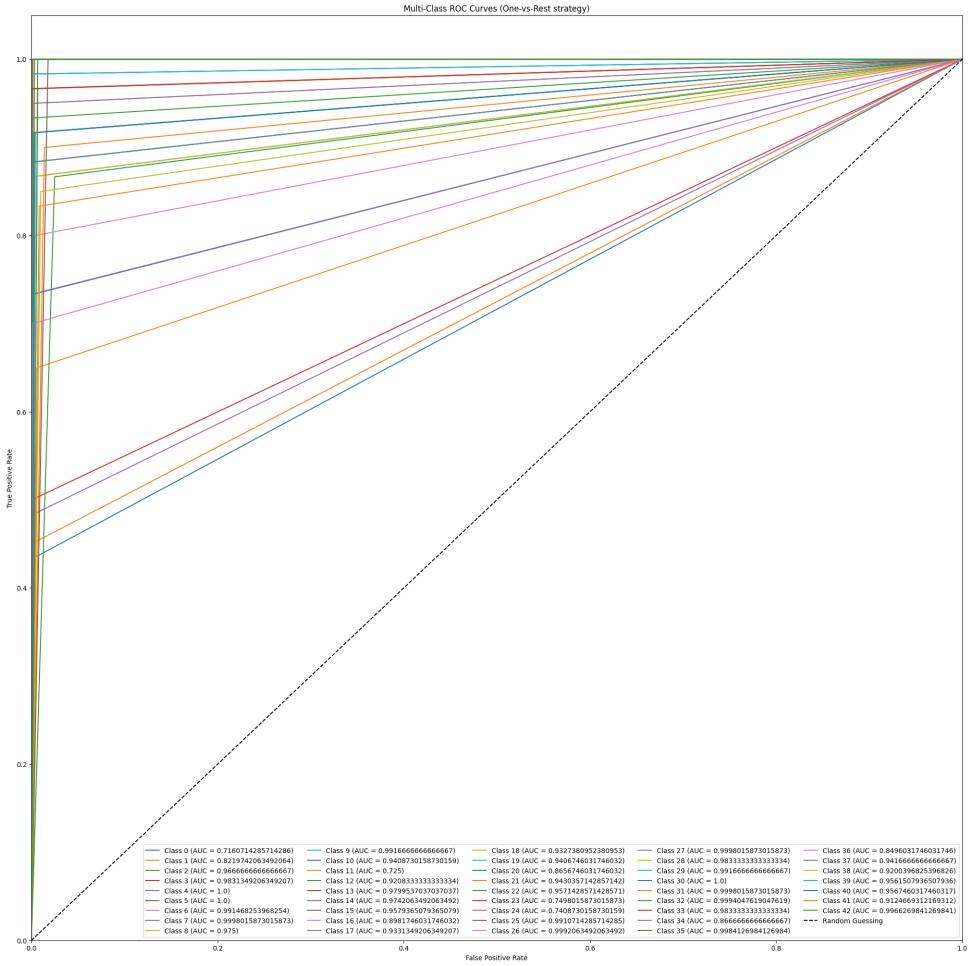

Figure 37: Pretrained model with features extraction from Block3 ROC curve

Thanks to the use of features extracted from 'block3_pool', the model achieves an accuracy of around 87.4% and a Top-K score (K=3) of around 0.881. The model was trained for 11 epochs, after which the early stopping mechanism was activated for overfitting. This result shows the effectiveness of intuition, greatly increasing the performance obtained. The instability of the network should be highlighted.

4.6 Model With Feature Extraction From Block 2

To evaluate how the network behaves with even more general features, in this model, the features are extracted from the layer "block2_pool". The network presents the same structure compared to the one previously observed. After the 'Flatten' layer

the architecture has a dropout with a rate of 0.5, a dense layer with 128 nodes and L2-Regularization with a value of 0.01, a dropout layer with a rate of 0.3 and the last dense layer for classification (with softmax activation function). The architecture is illustrated in the figure 38.

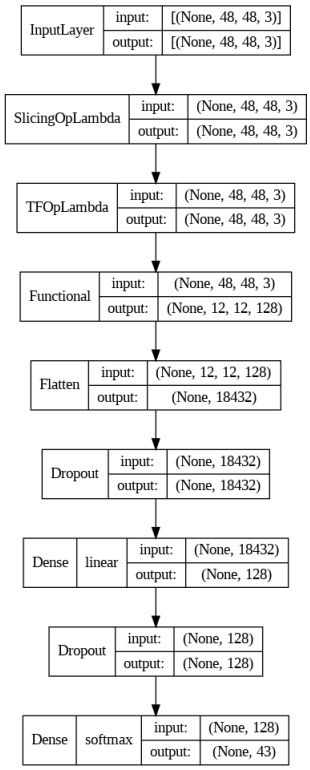

Figure 38: Pretrained model architecture with features extraction from Block2

This architecture has:

- **Total parameters**: 2625131 (10.01 MB)
- **Trainable parameters**: 2364971 (9.02 MB)
- **Non-trainable parameters**: 260160 (1016.25 KB)

Table 25 summarizes the results of the training phase at the last epoch in which the model performed an improvement (in case of early stopping at epoch M, the results of the M-Patience epoch will be reported, but "Epochs" Field is M):

Epochs	Train Loss	Train Acc.	Valid. Loss	Valid. Acc.	Early St.
8	598.7974	0.9555	1727.6056	0.8884	Yes

Table 25: Table representing the results obtained for the pre-trained model with features extraction from Block2 in training phase

Table 26 summarizes the results of the testing phase.

Precision	Recall	F1-Score	Loss	Accuracy	TopK Accuracy
0.9017	0.8802	0.8801	1937.6002	0.8802	0.8857

Table 26: Table representing the results obtained for the pre-trained model with features extraction from Block2 in testing phase

The confusion matrix and ROC curve are now reported.

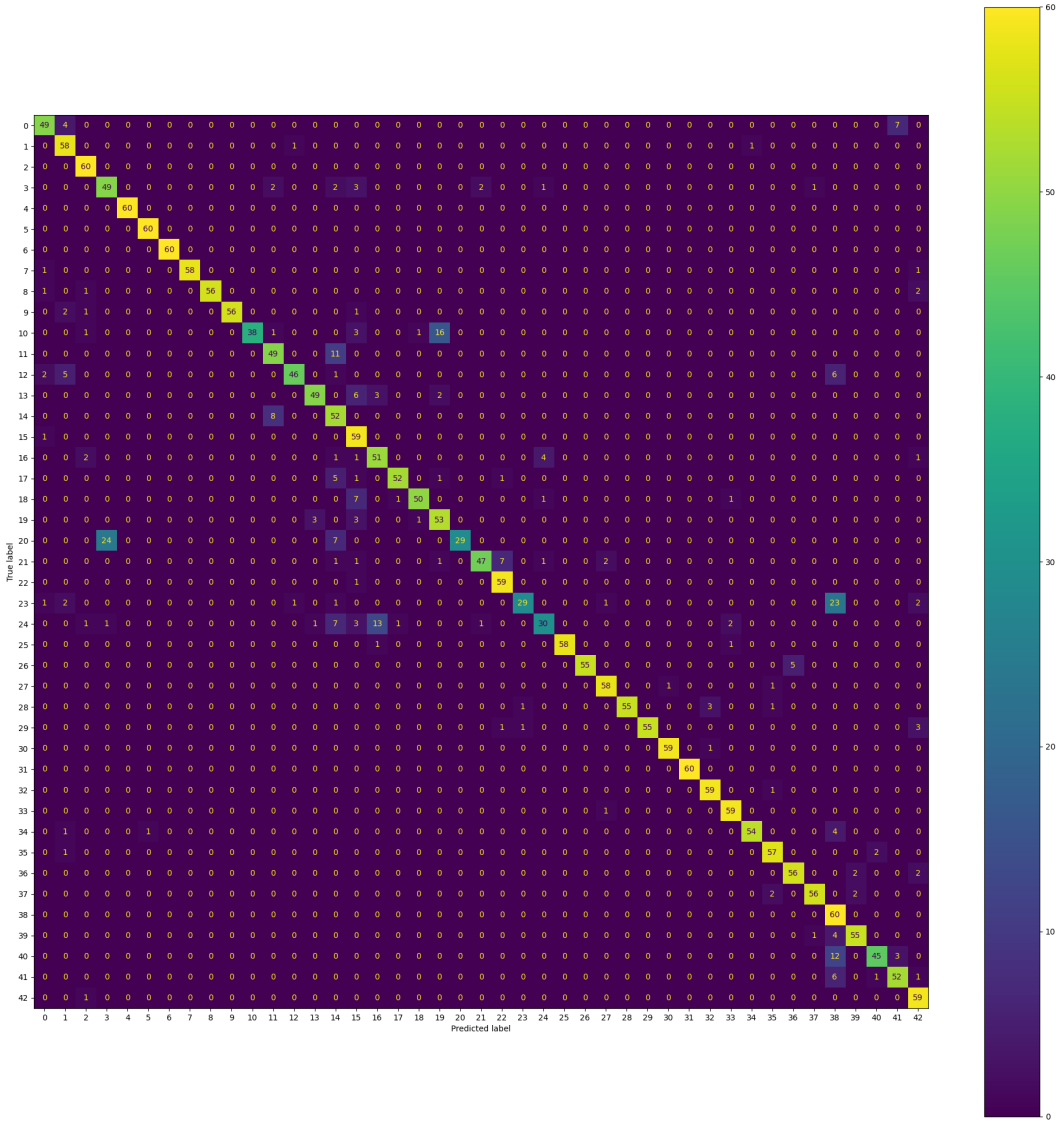

Figure 39: Pretrained model with features extraction from Block2 confusion matrix

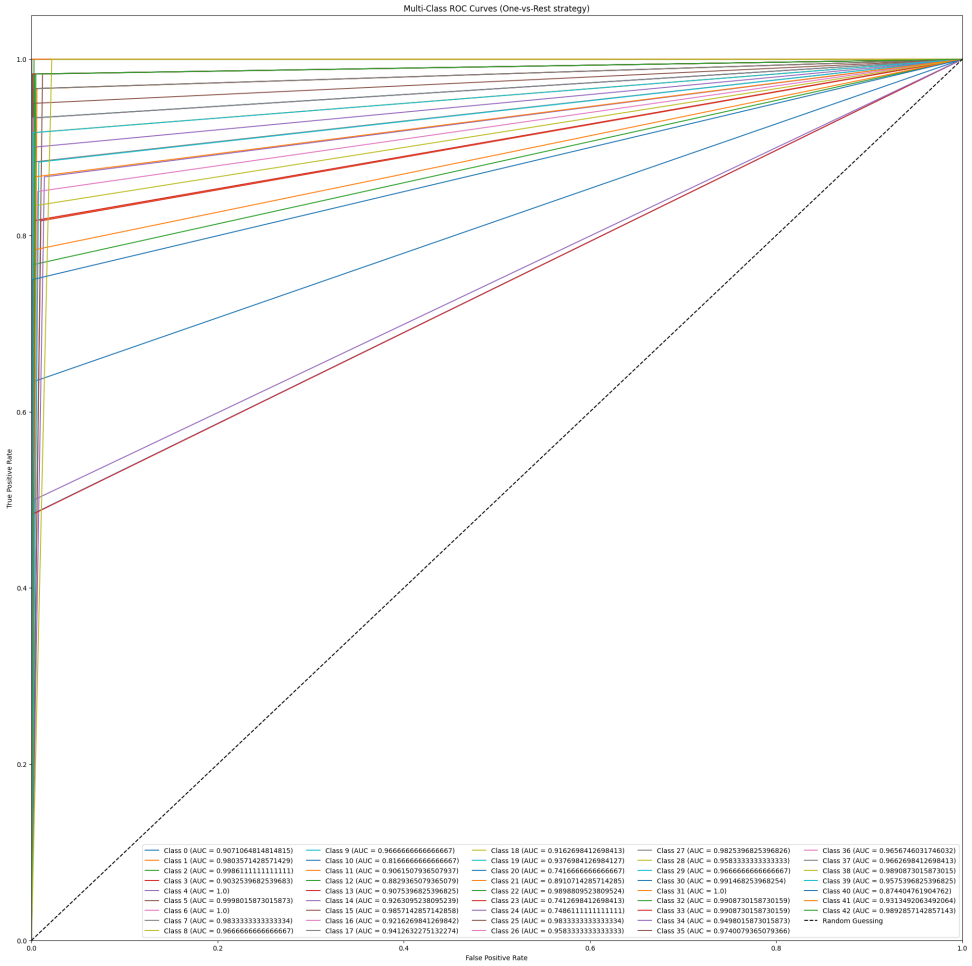

Figure 40: Pretrained model with features extraction from Block2 ROC curve

Thanks to the use of features extracted from 'block2_pool', the model achieves an accuracy of around 88% and a Top-K score (K=3) of around 0.886. The model was trained for 8 epochs, after which the early stopping mechanism was activated for overfitting. The instability of the network should be highlighted.

4.7 Summary Of Results And Considerations

In this chapter, various models were explored to address the classification challenge using VGG16 as a pre-trained network to extract features. Through the analysis of the results, it was observed that, although the use of techniques such as Dropout and L2-Regularization played a crucial role in mitigating the overfitting problems in

the context, the network was not able to adequately learn to solve the classification problem presented. Having identified this problem, the performance of the network was analyzed when it is trained by extracting features from previous layers (therefore more general features). This insight more than doubled the goodness of model classification. After careful evaluation, Model 5 was chosen as the best model to address the problem at hand. This decision is based on a trade-off between accuracy, loss, and simplicity. The performance of this model, however, does not align well with our objective of obtaining an effective classification (the accuracy remains less than 90%) and obtaining a simple architecture (the number of parameters is approximately 46% larger compared to the best custom model)

5 Conclusions

During this project, several CNN models have been developed from scratch and used several pre-trained models based on the VGG16 architecture to classify traffic sign images from the GTSRB challenge datasets. At the end, a CNN model from scratch has been successfully built that showed exceptional performance, even outperforming pre-trained models.

All custom models achieved excellent performance while maintaining simple architectures with less than 1 million trainable parameters. In particular, the best custom model created (model 6) reaches accuracy levels greater than 97% (comparable or slightly lower than those present in the state of the art) by maintaining an architecture with only 809195 trainable parameters (much lower than most state-of-the-art architectures, even more than 10 times). This model fully meets the objectives set at the beginning of the project. From the point of view of every metric considered (precision, loss, F1-Score, ...) this model is absolutely competitive.

However, models based on pre-trained networks did not satisfy requirements, presenting greater difficulty in learning with specific features and therefore the need to extract more generic ones. Despite this, the results obtained are not competitive and do not satisfy the previously set requirements (the accuracy does not reach 90% and the architecture is more complex than the custom one).

As a further development of this project, the following ideas are given:

- Preprocess the dataset with different techniques
- Compare the performance obtained with color images versus that obtained using grayscale images
- Use larger images than those used (48,48,3)
- Analyze the impact of introducing the macro class (obligation, attention, ..) of the sign to be classified as an additional input to the network
- Analyze the impact of using different hyperparameter tuning techniques to search for better configurations

6 Attachments

This section contains training graphs of the models presented as the results have been extracted from these and presented in a more concise format in the previous chapters of the report. For a more detailed view of the graphs, refer to the Notebook associated with the report which can be found at the link [15].

6.1 Custom Models

6.1.1 Base Model

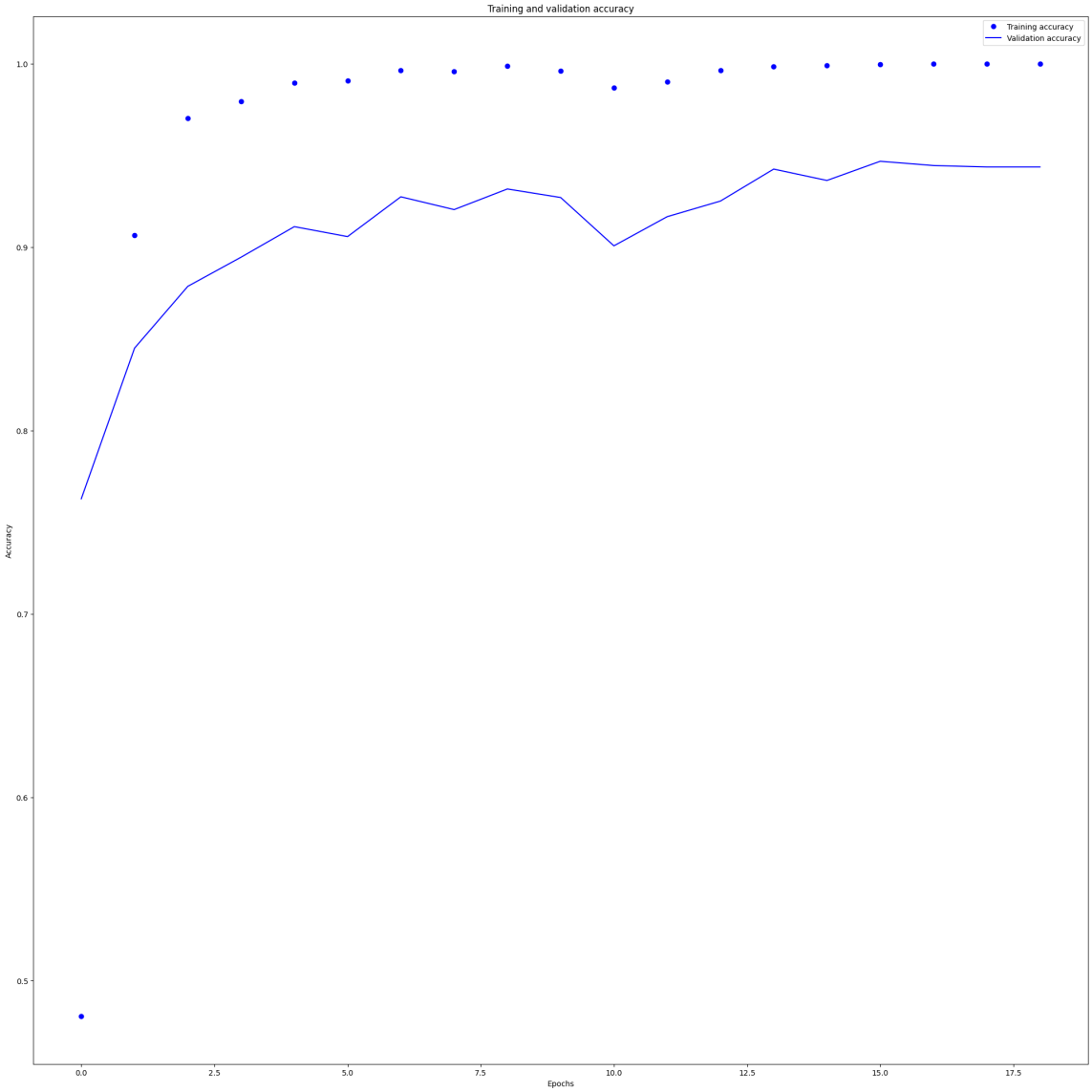

Figure 41: Training accuracy for custom base model

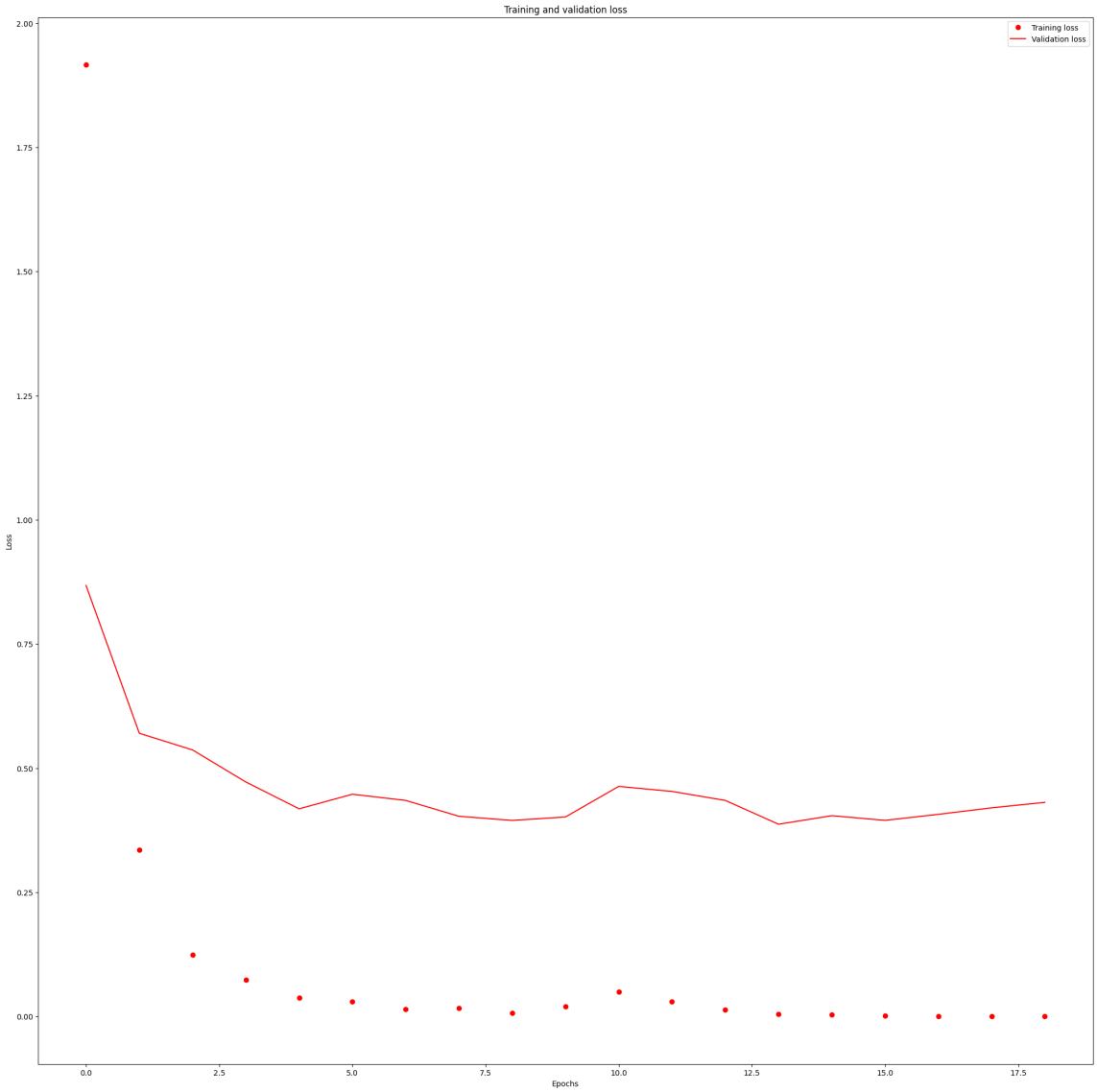

Figure 42: Training loss for custom base model

6.1.2 Base Model With Filters Increased

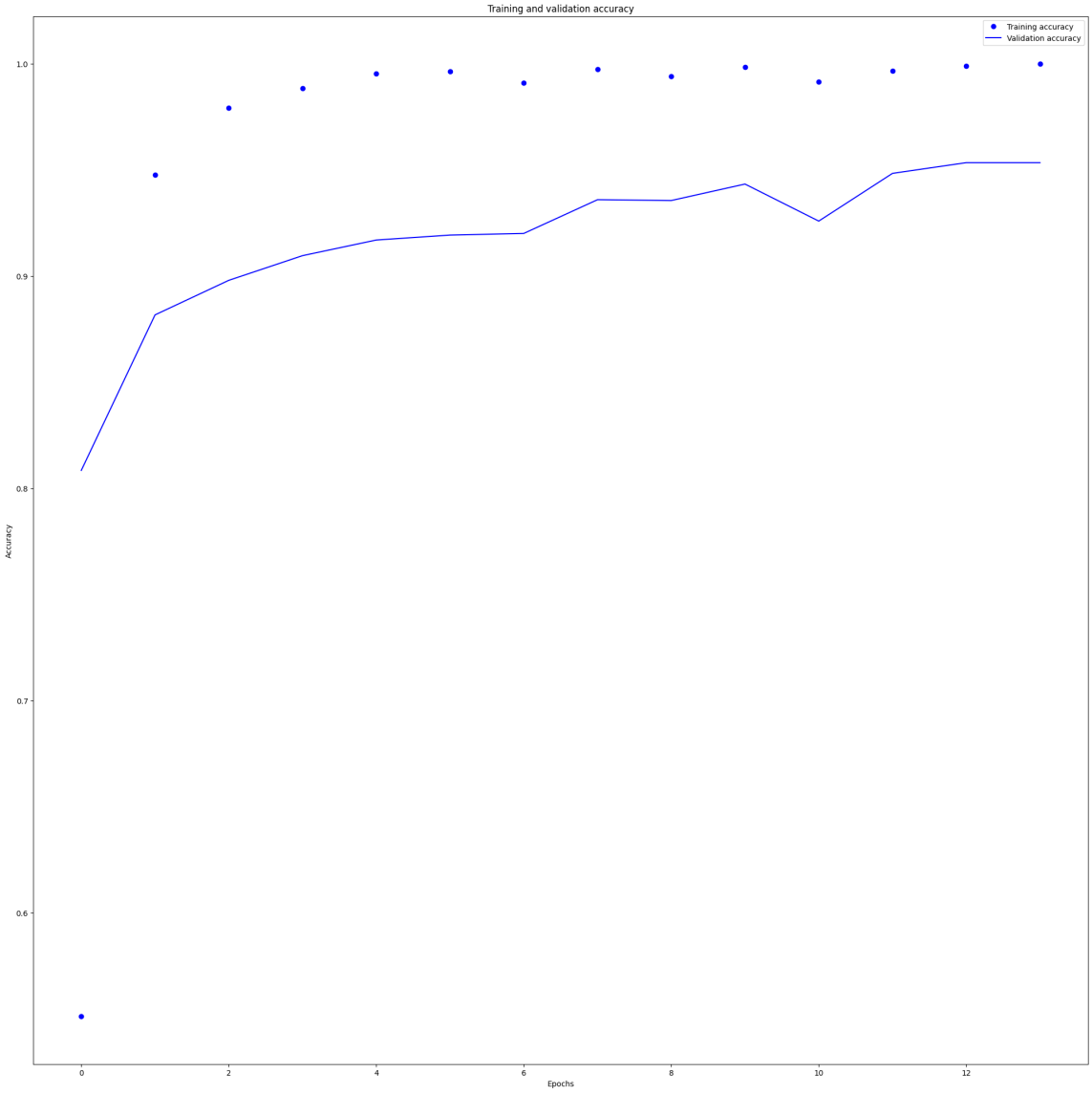

Figure 43: Training accuracy for custom base model with increased filters

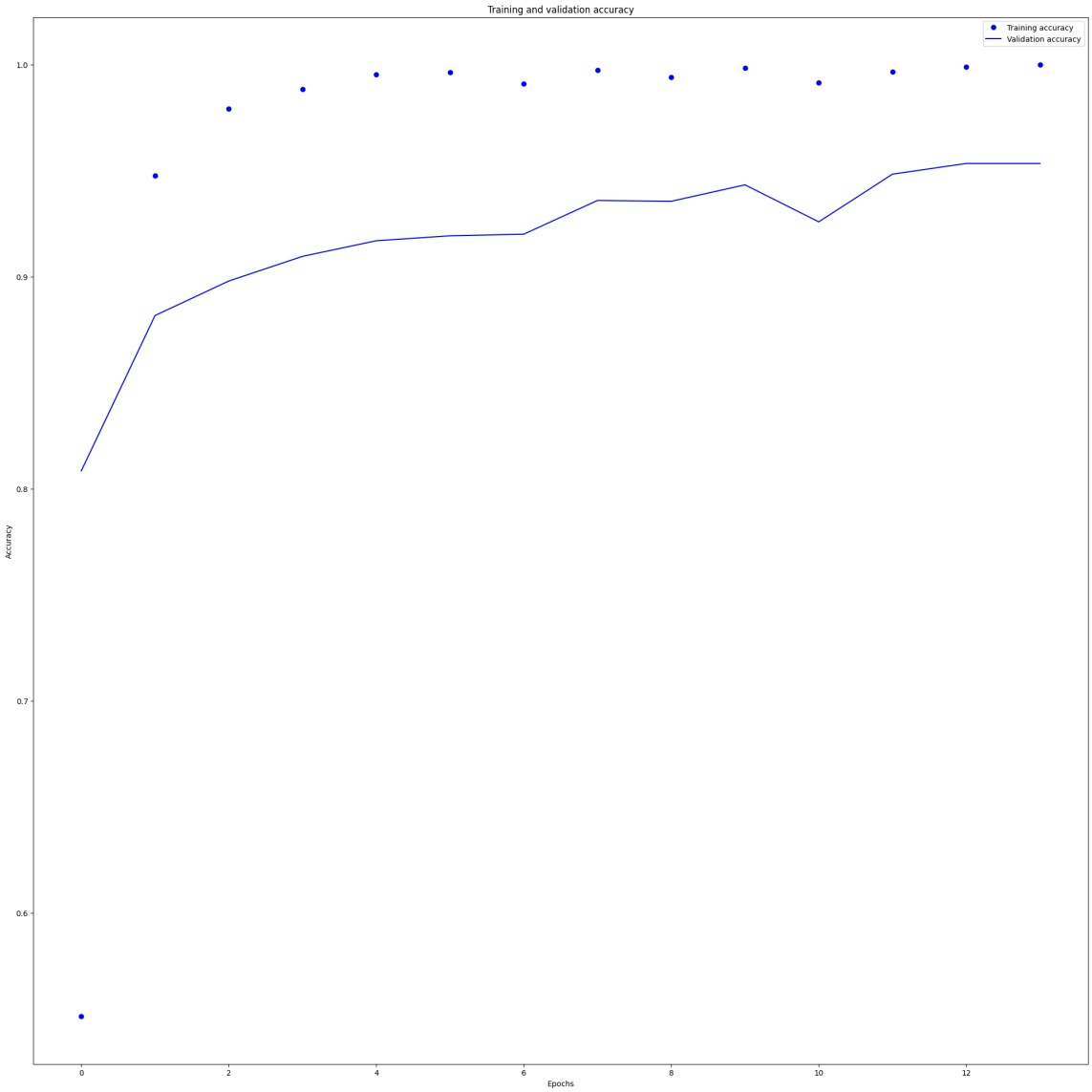

Figure 44: Training loss for custom base model with increased filters

6.1.3 Model With Dropout

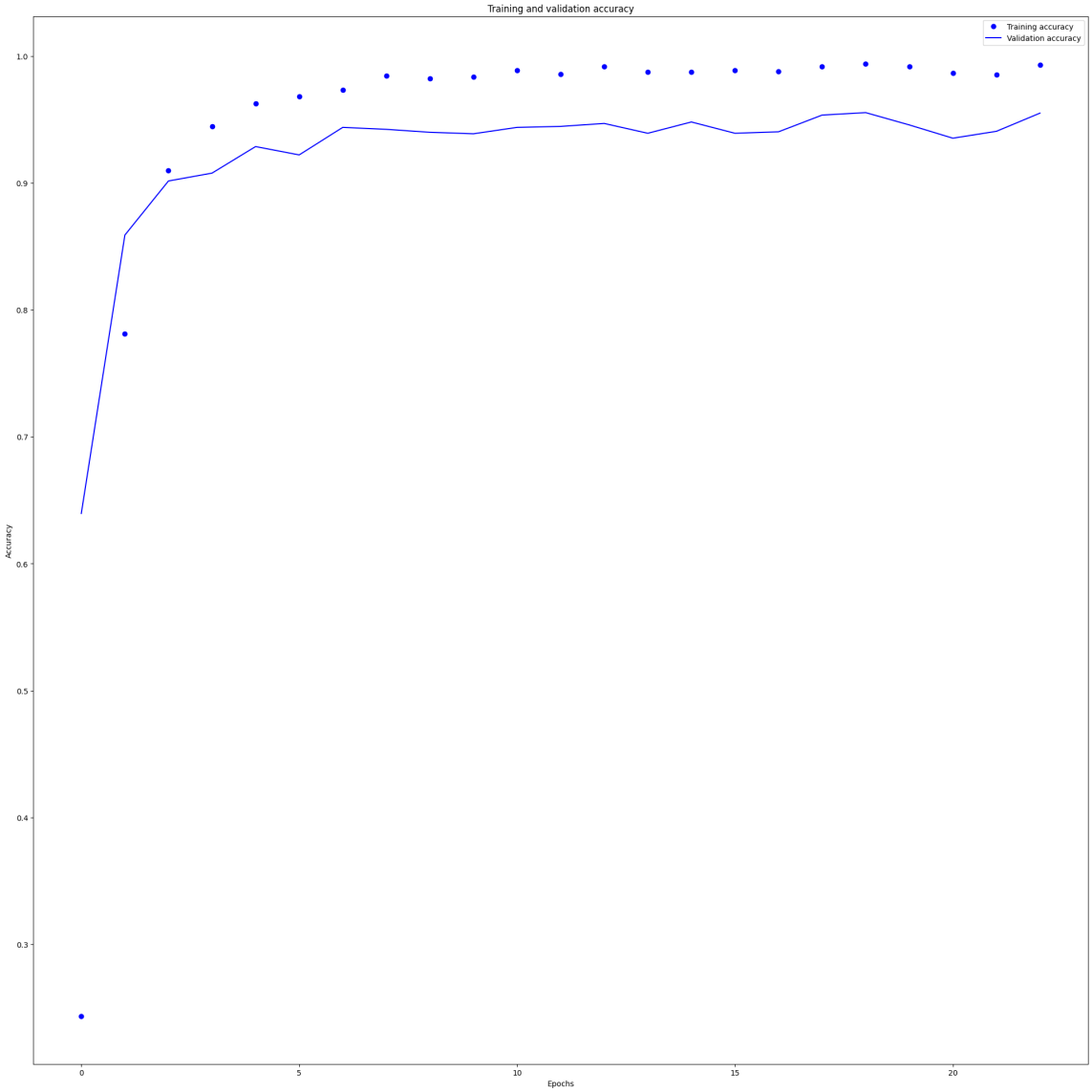

Figure 45: Training accuracy for custom model with only dropout

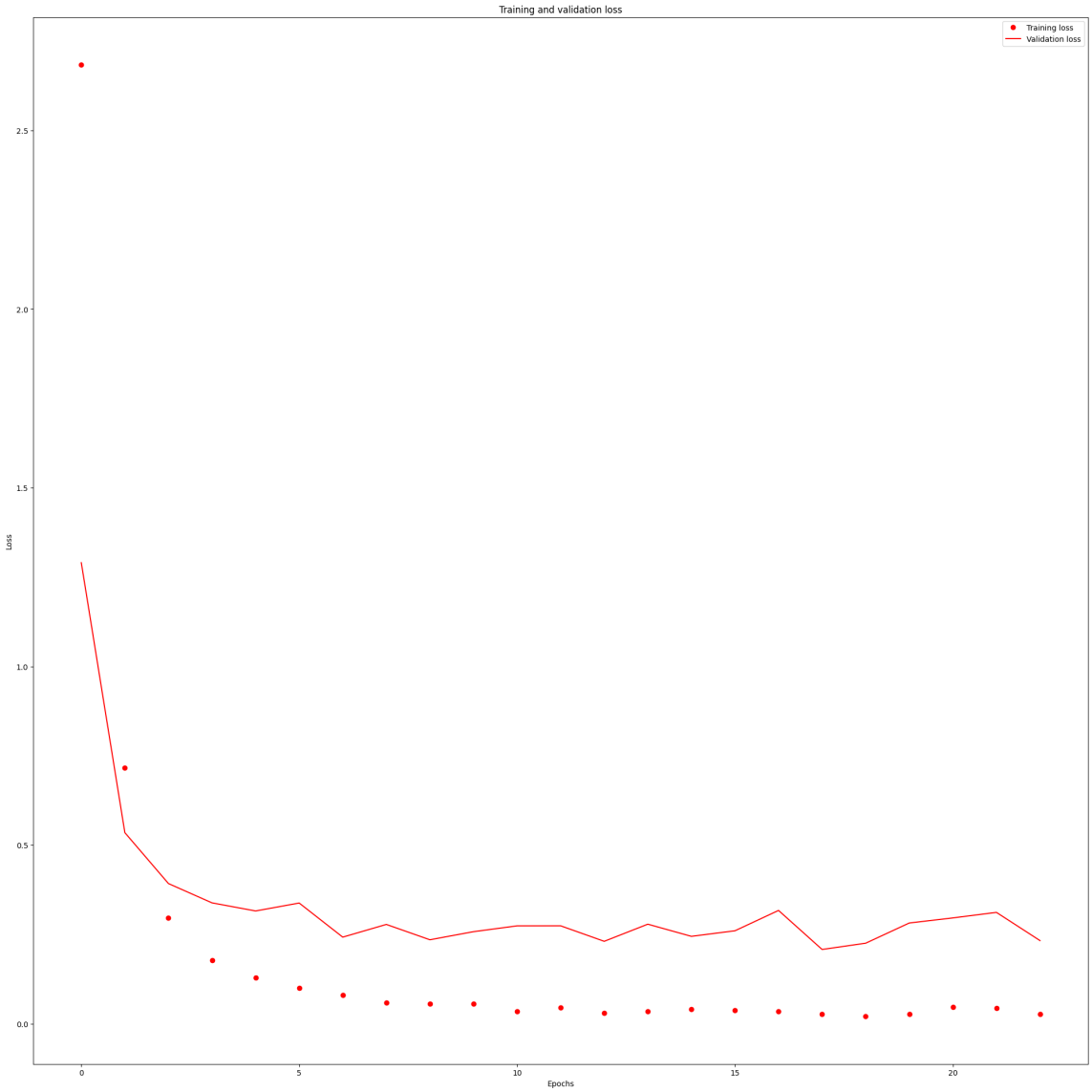

Figure 46: Training loss for custom model with only dropout

6.1.4 Model With Dropout, L2-Regularization: Version 1

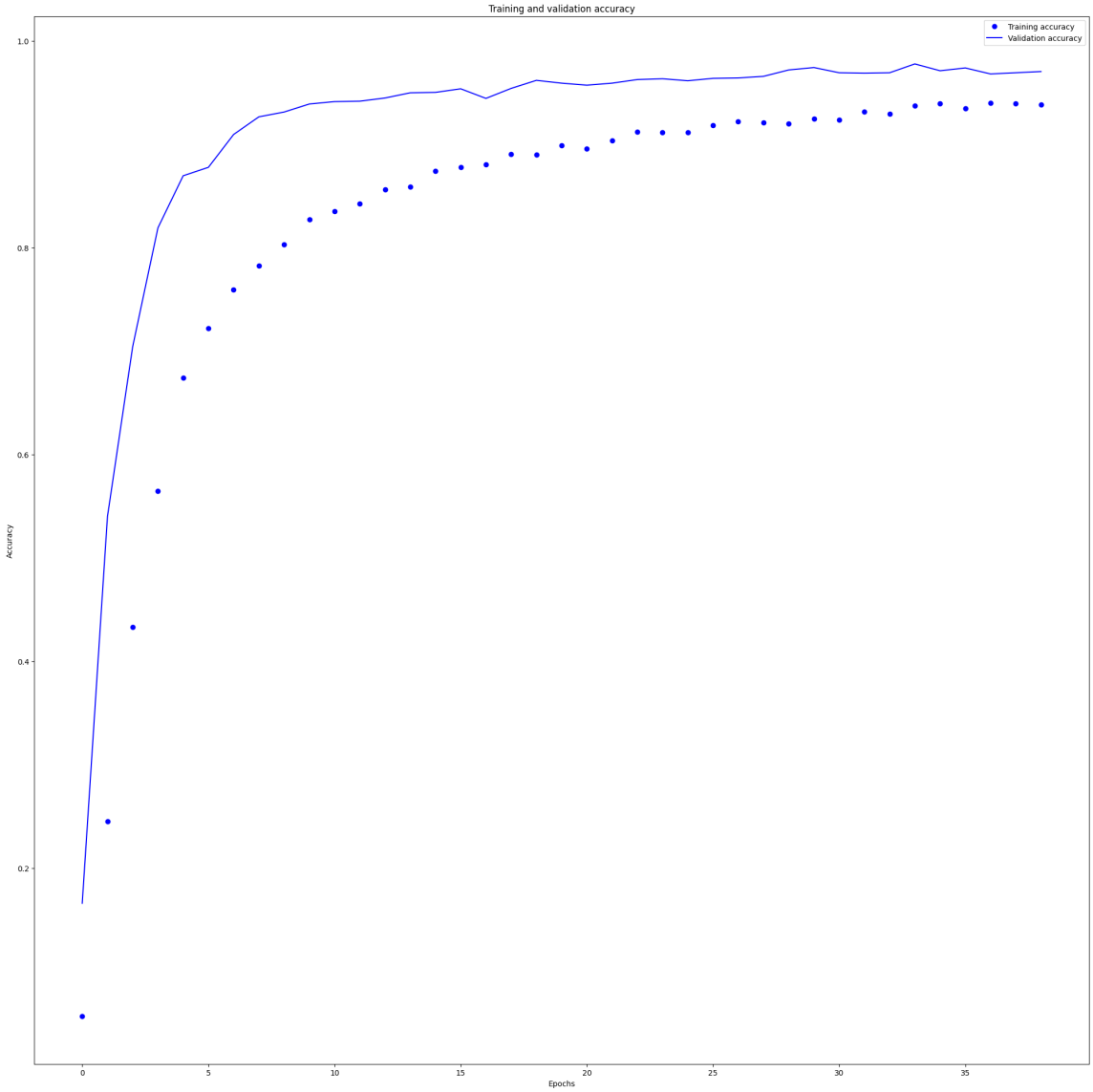

Figure 47: Training accuracy for custom model with dropout and L2-Regularization Version 1

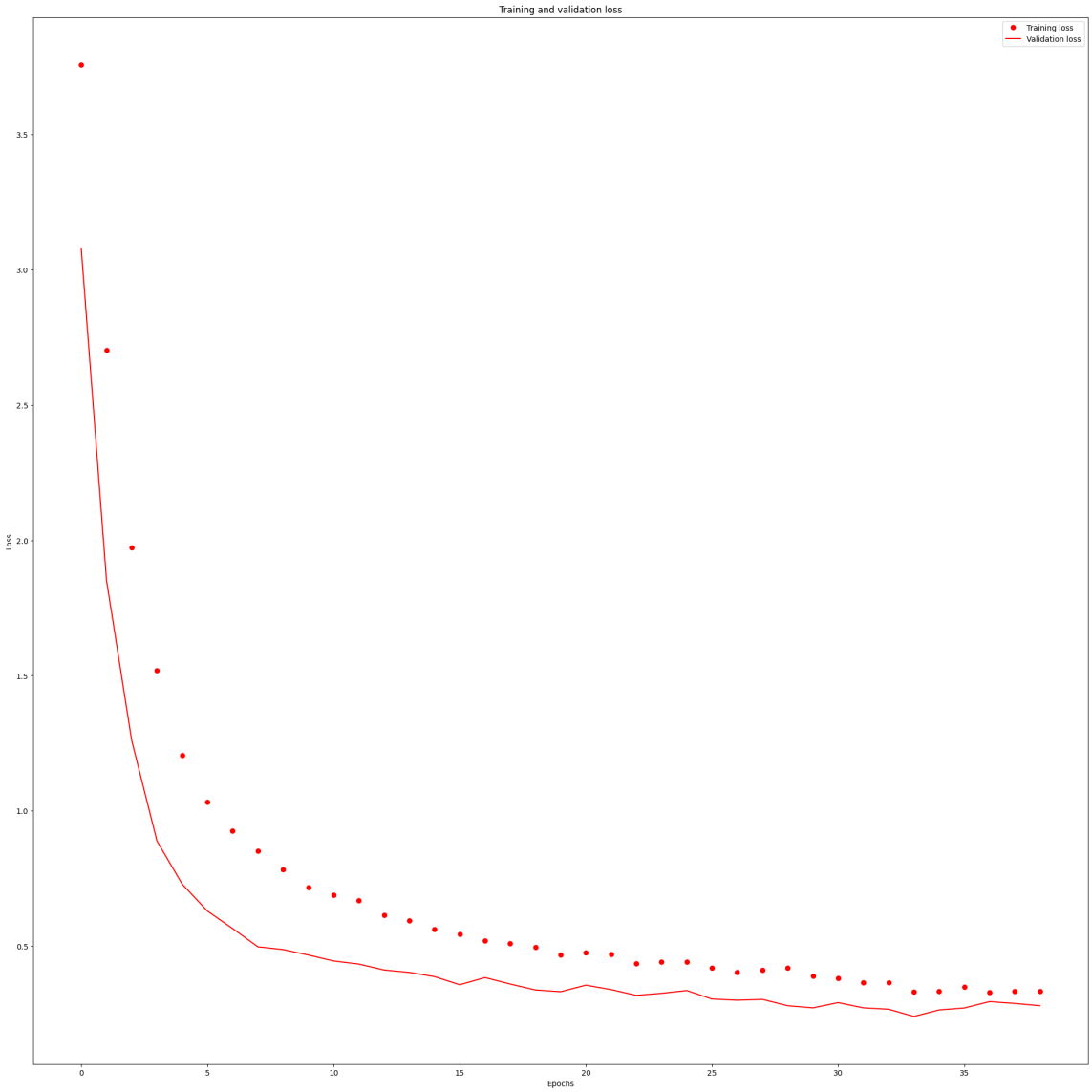

Figure 48: Training loss for custom model with dropout and L2-Regularization Version 1

6.1.5 Model With Dropout, L2-Regularization: Version 2

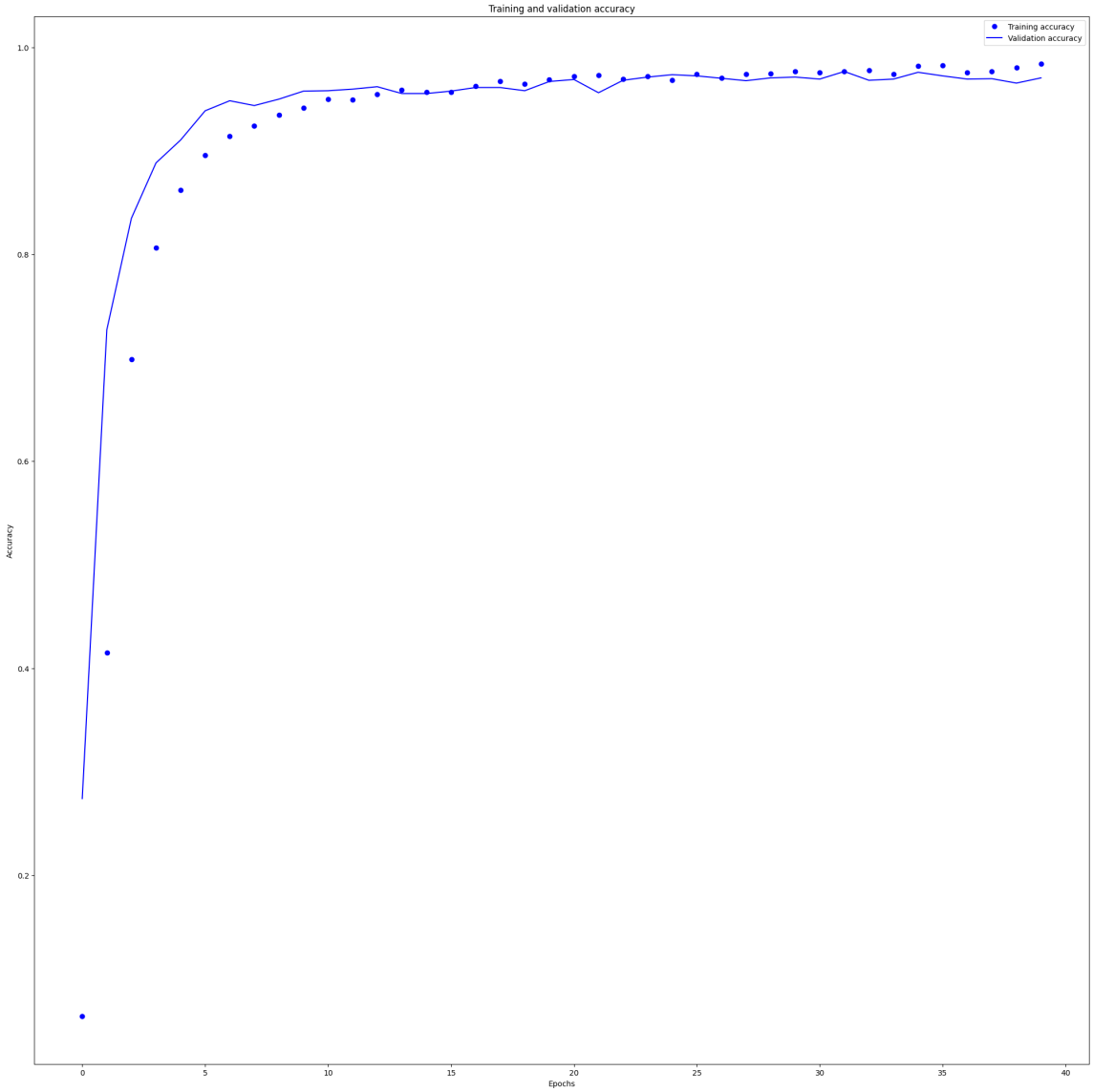

Figure 49: Training accuracy for custom model with dropout and L2-Regularization Version 2

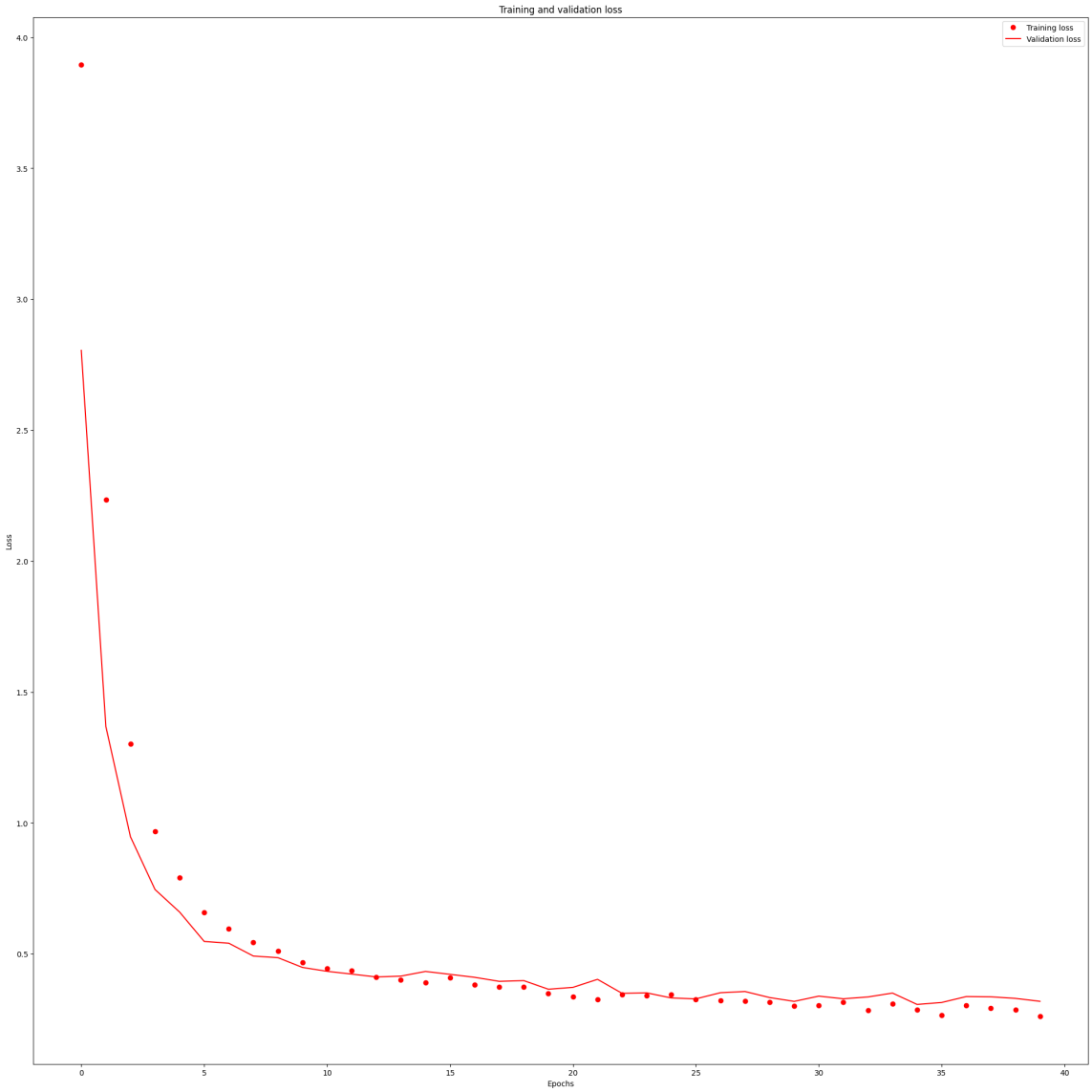

Figure 50: Training loss for custom model with dropout and L2-Regularization Version 2

6.1.6 Model With Dropout, L2-Regularization: Version 3

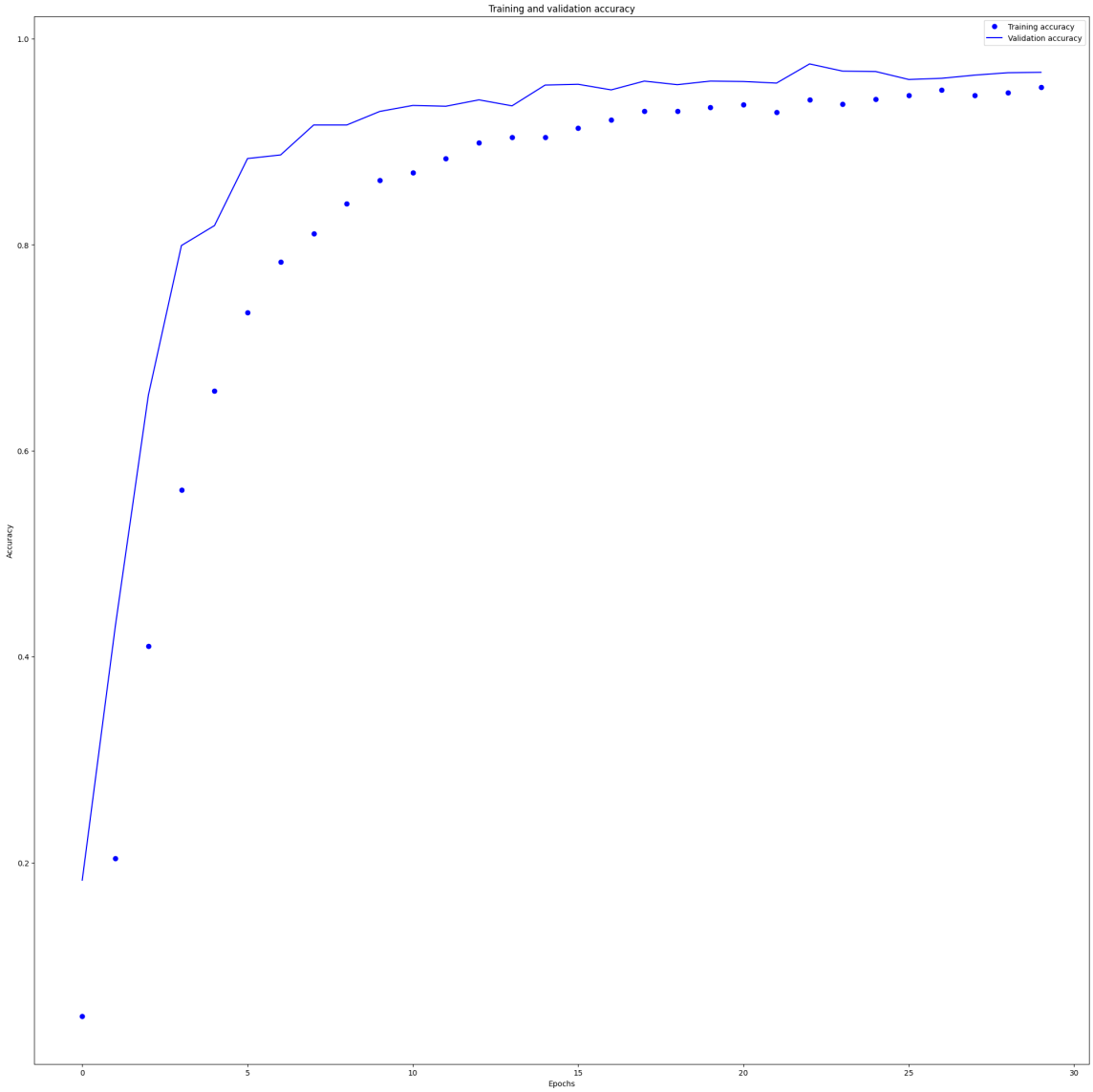

Figure 51: Training accuracy for custom model with dropout and L2-Regularization Version 3

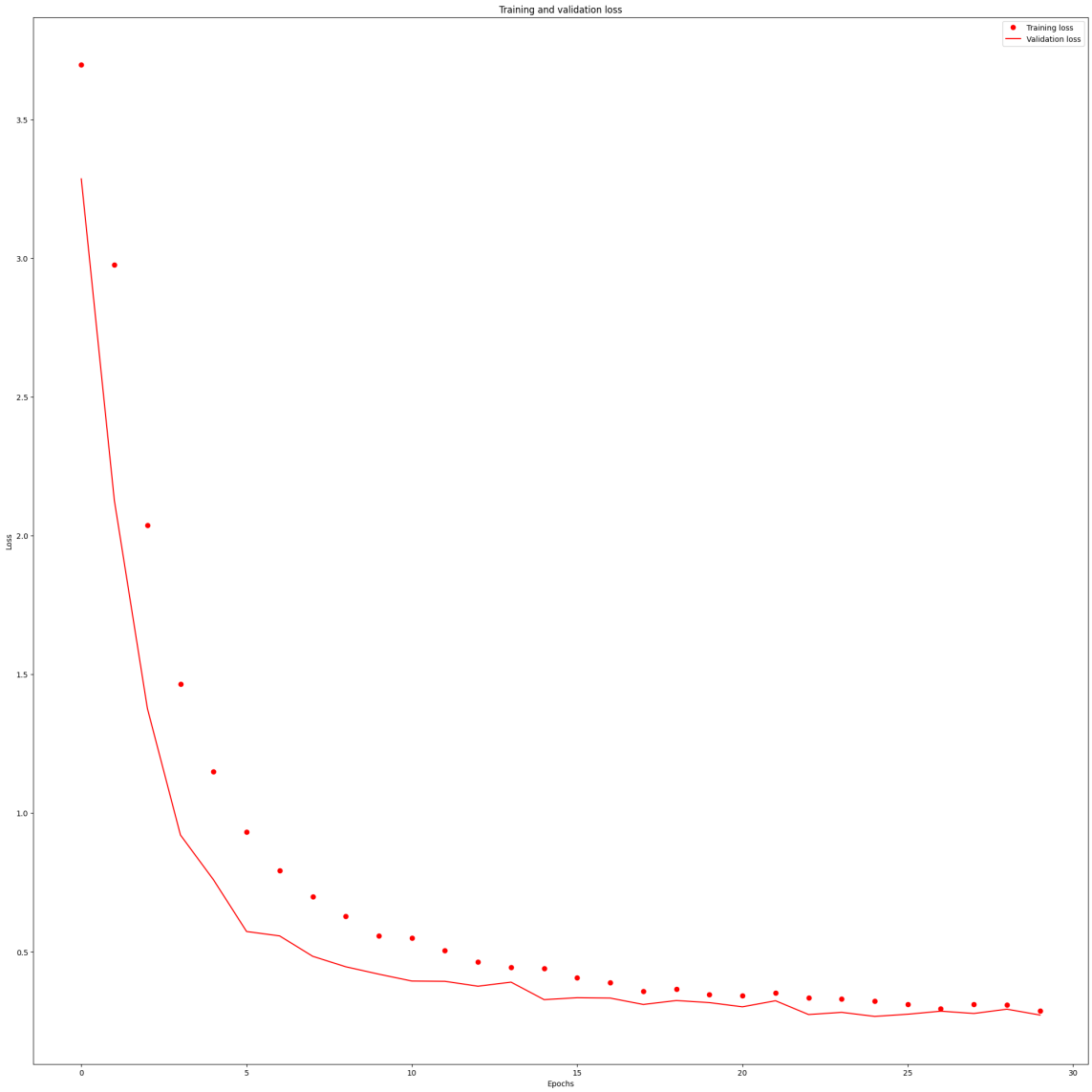

Figure 52: Training loss for custom model with dropout and L2-Regularization Version 3

6.2 Pretrained Models

6.2.1 Base Model

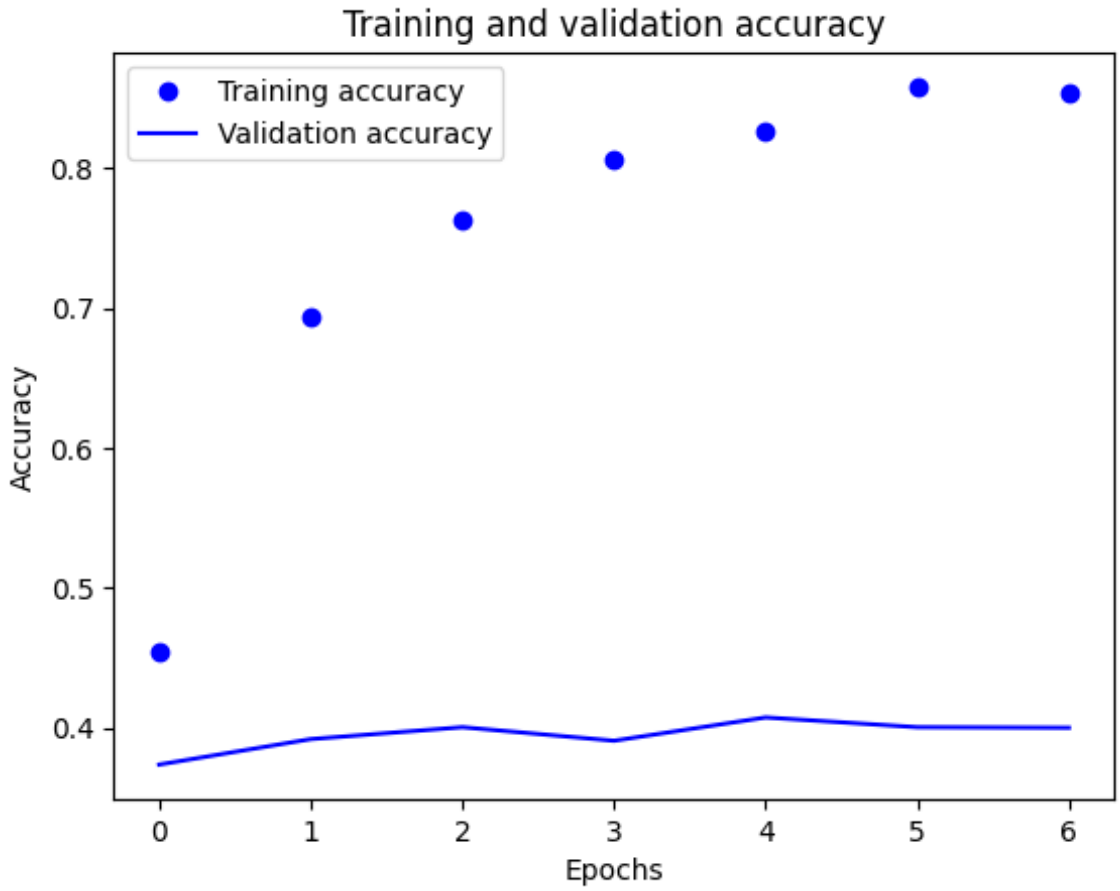

Figure 53: Training accuracy for pre-trained base model

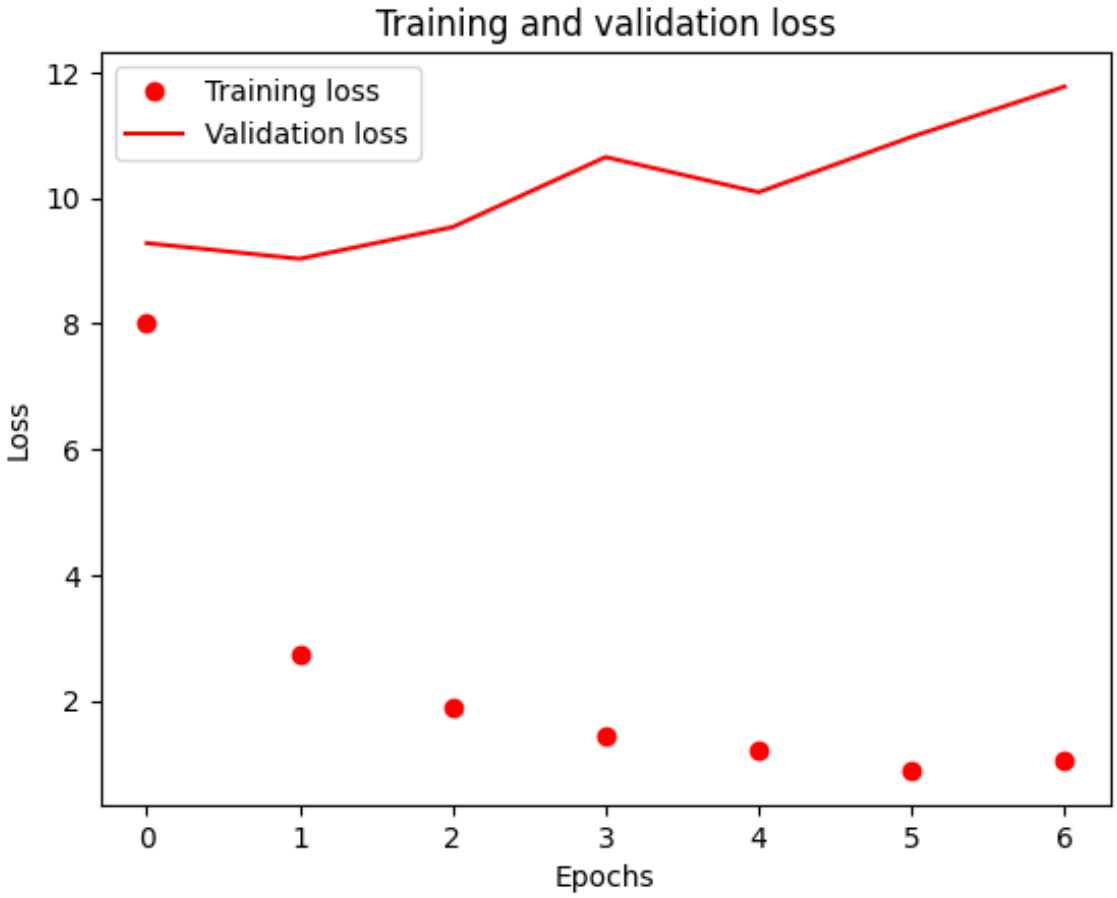

Figure 54: Training loss for pre-trained base model

6.2.2 Model With Dropout, L2-Regularization: Version 1

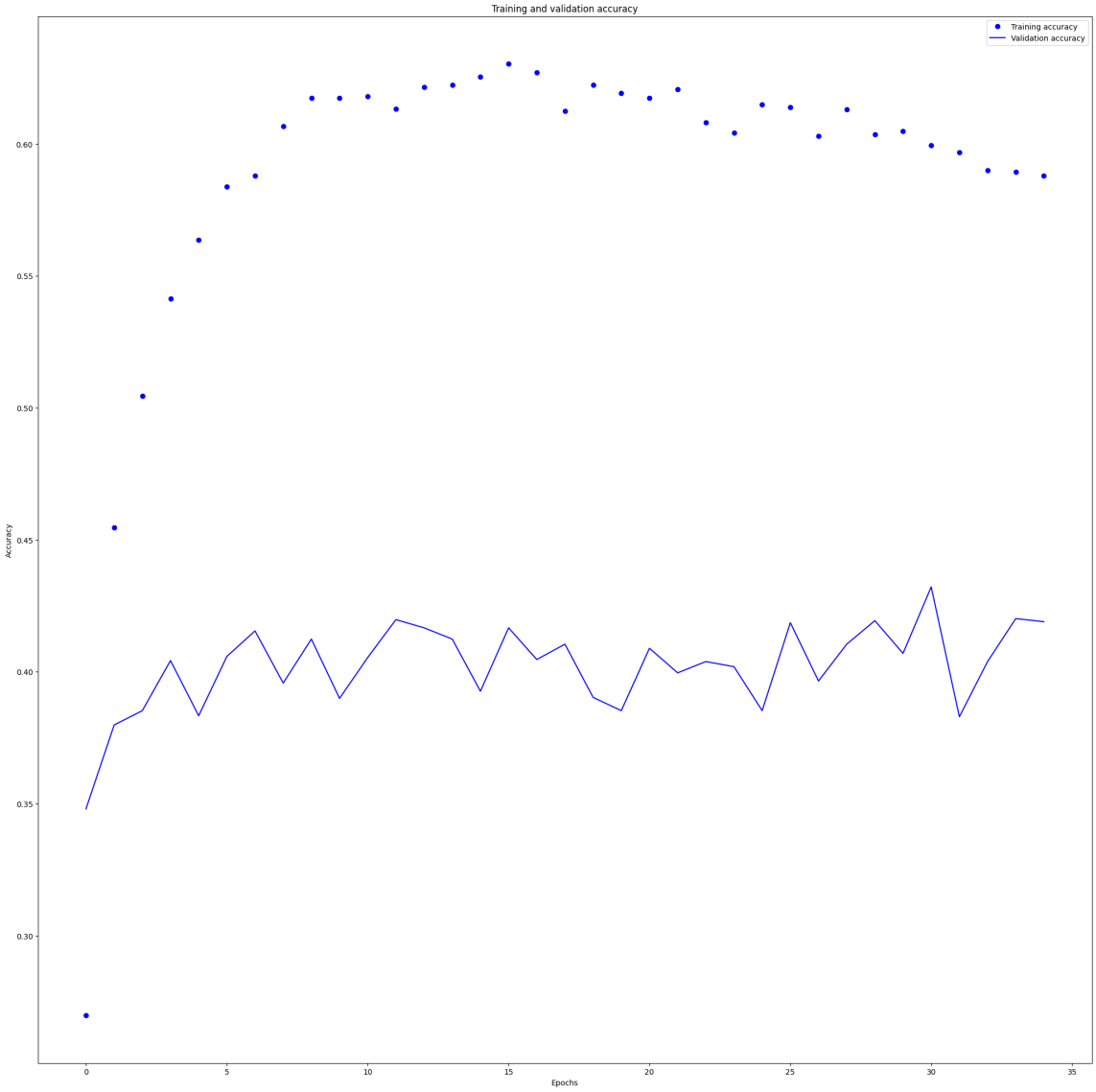

Figure 55: Training accuracy for pre-trained model with dropout and L2-Regularization: Version 1

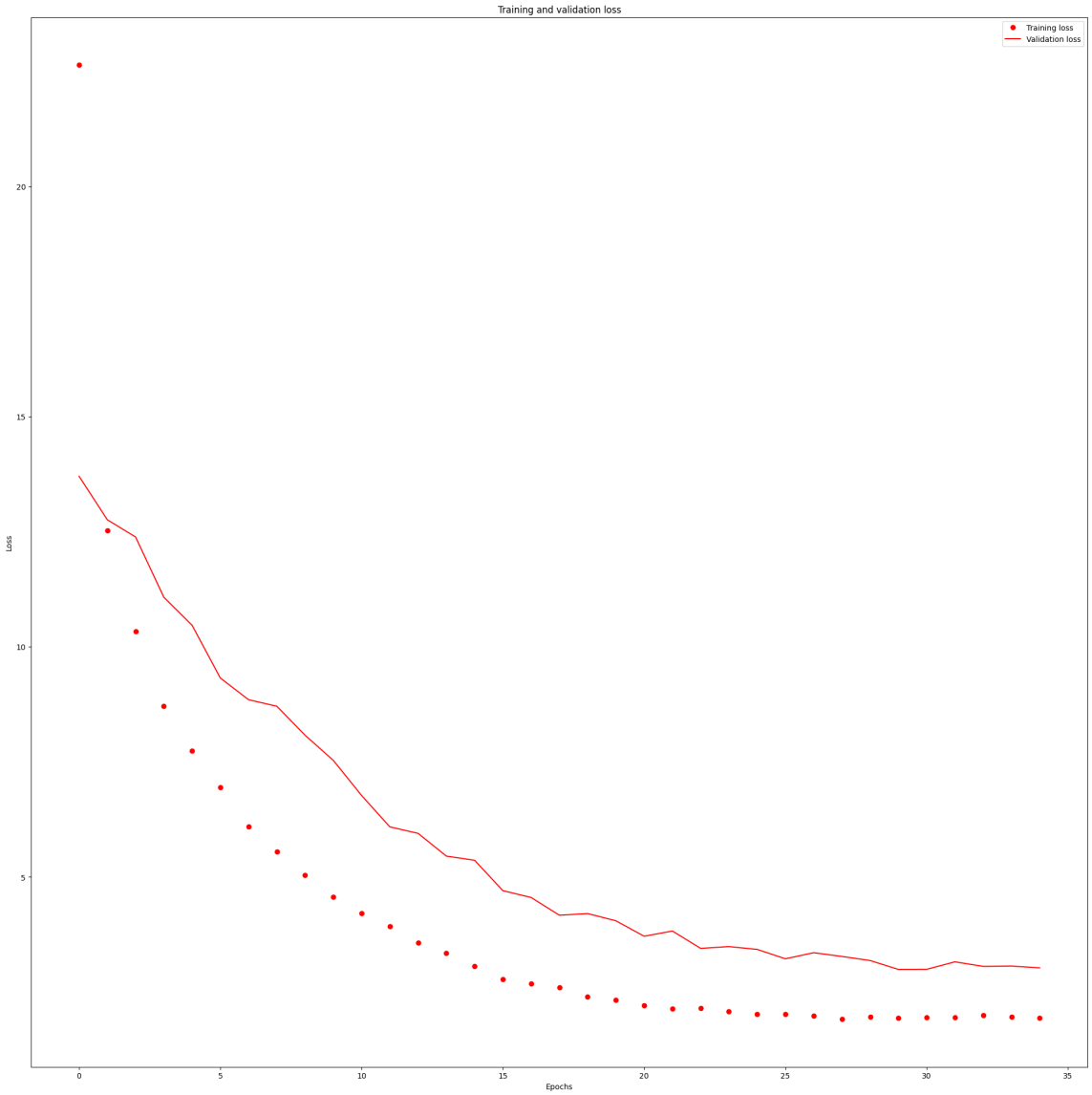

Figure 56: Training loss for pre-trained model with dropout and L2-Regularization: Version 1

6.2.3 Model With Dropout, L2-Regularization: Version 2

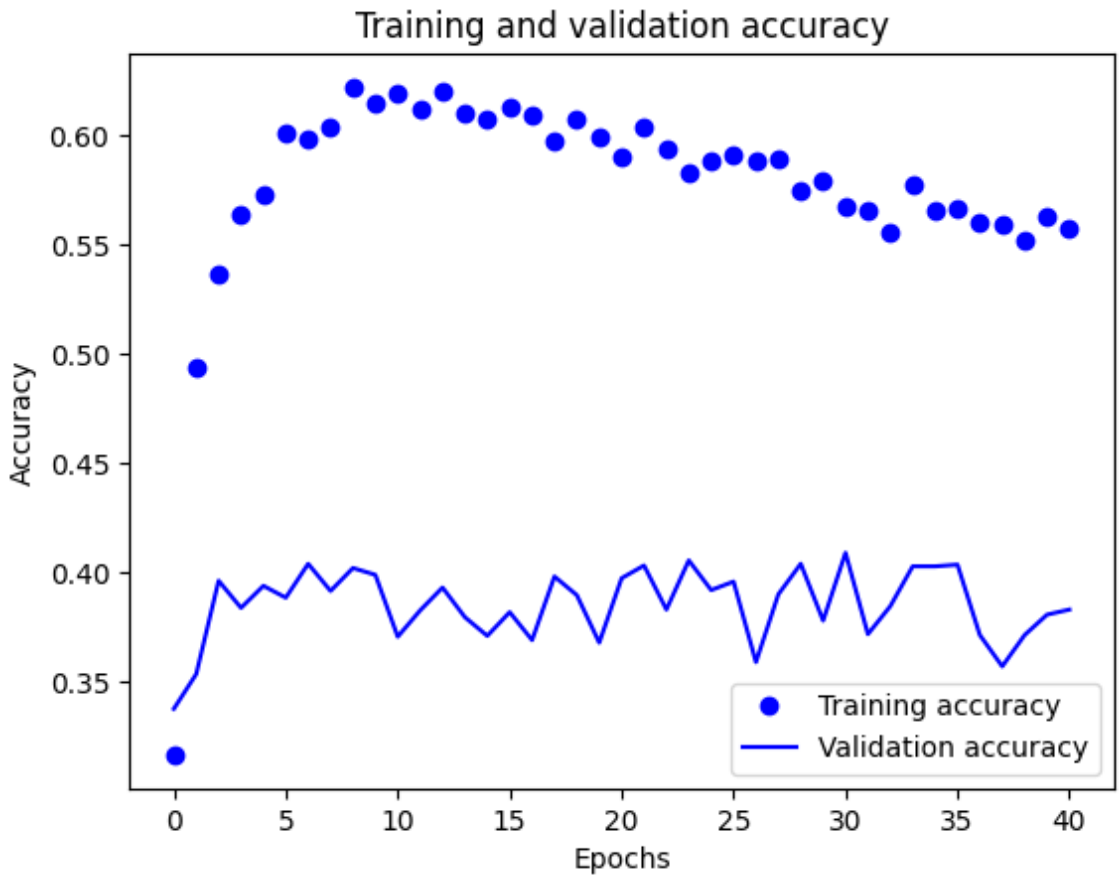

Figure 57: Training accuracy for pre-trained model with dropout and L2-Regularization: Version 2

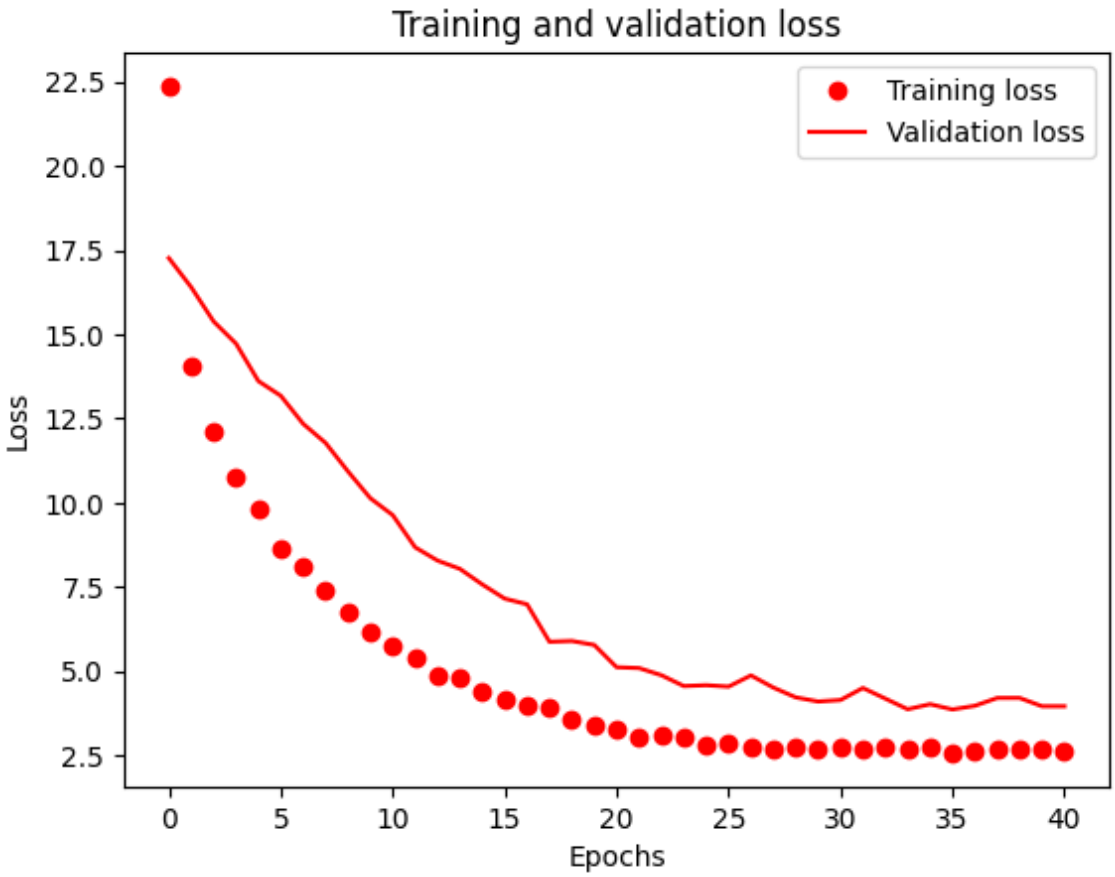

Figure 58: Training loss for pre-trained model with dropout and L2-Regularization: Version 2

6.2.4 Model With Feature Extraction From Block 4

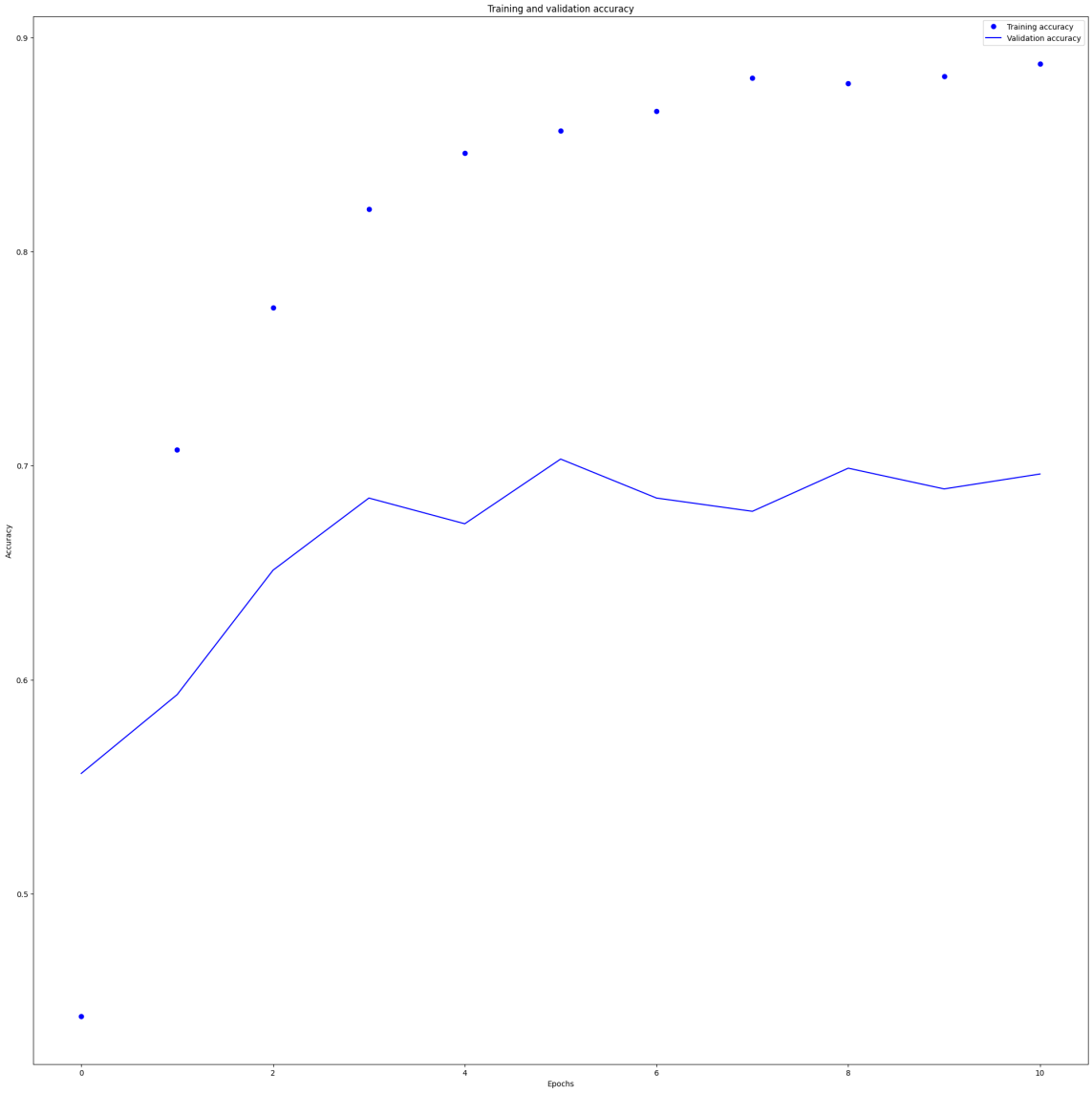

Figure 59: Training accuracy for pre-trained model with feature extraction from block 4

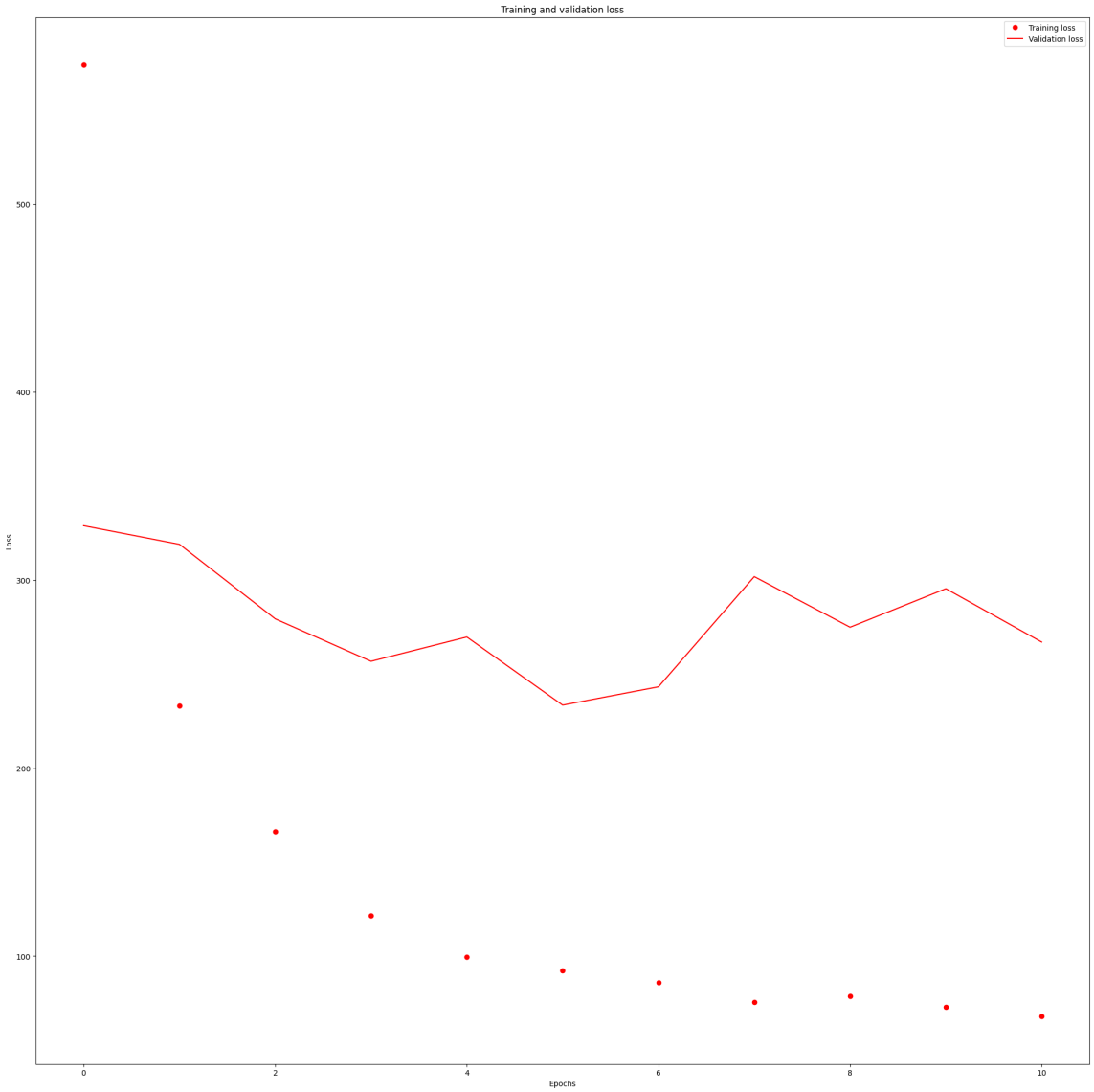

Figure 60: Training loss for pre-trained model with feature extraction from block 4

6.2.5 Model With Feature Extraction From Block 3

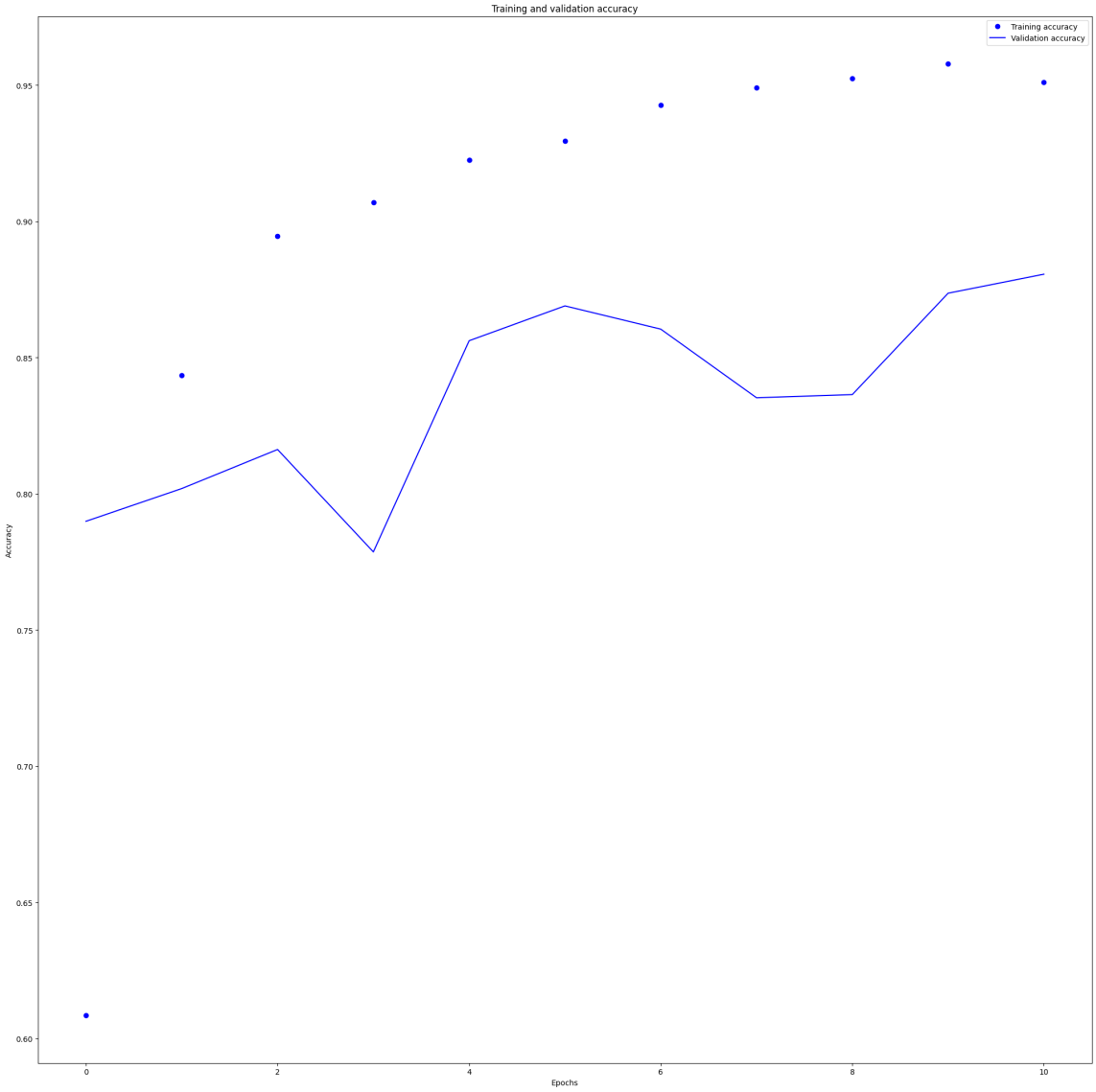

Figure 61: Training accuracy for pre-trained model with feature extraction from block 3

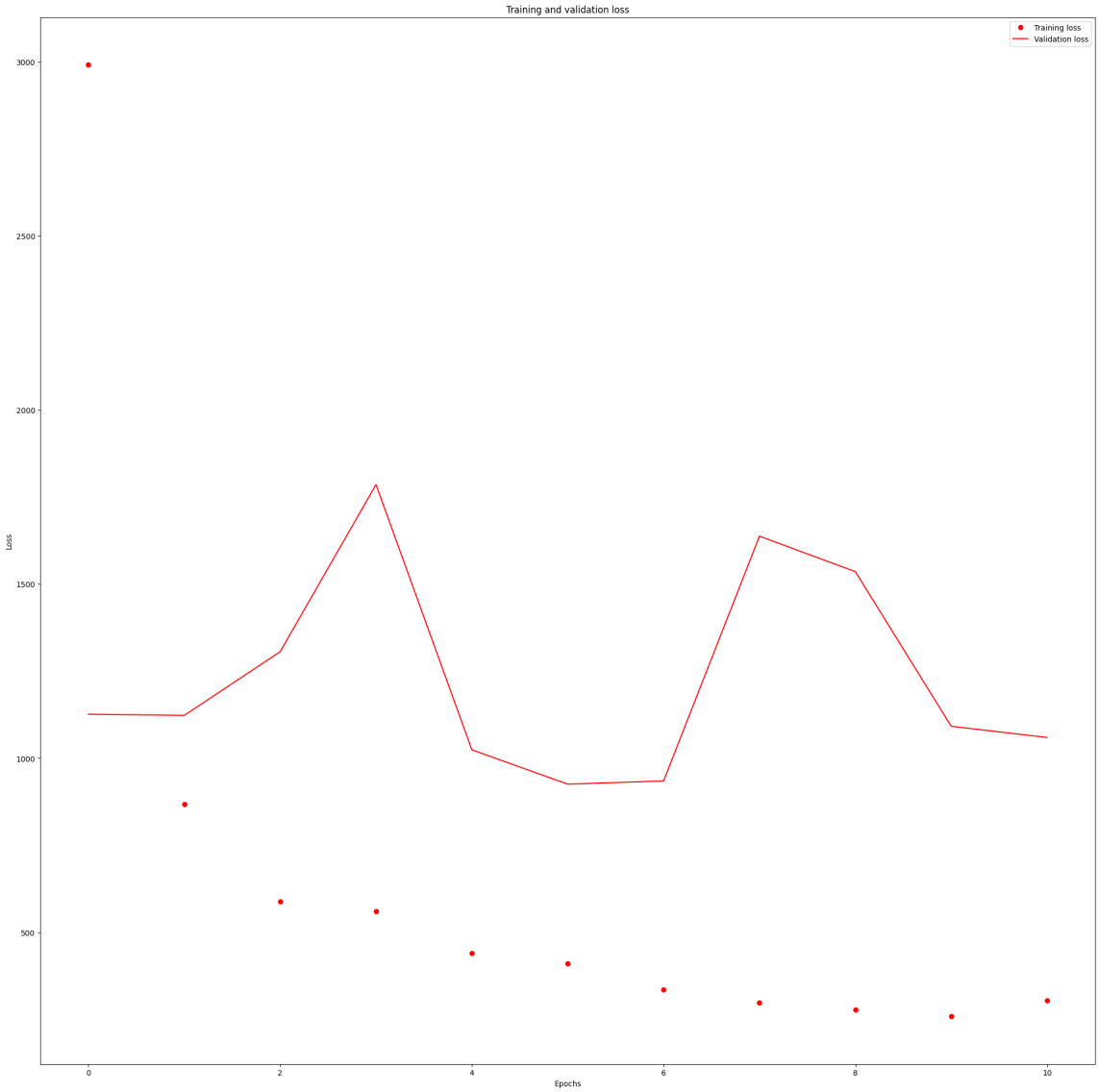

Figure 62: Training loss for pre-trained model with feature extraction from block 3

6.2.6 Model With Feature Extraction From Block 2

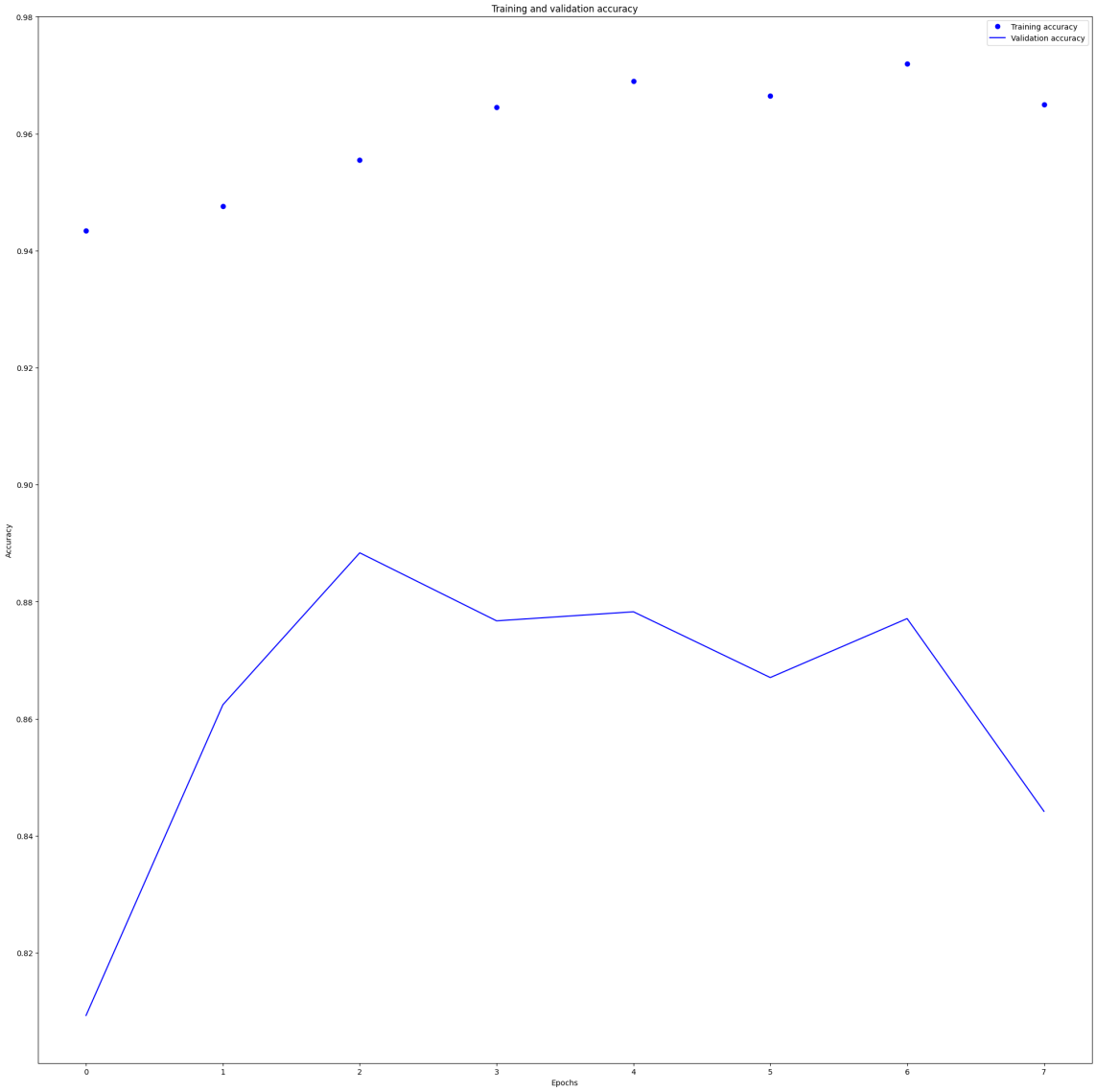

Figure 63: Training accuracy for pre-trained model with feature extraction from block 2

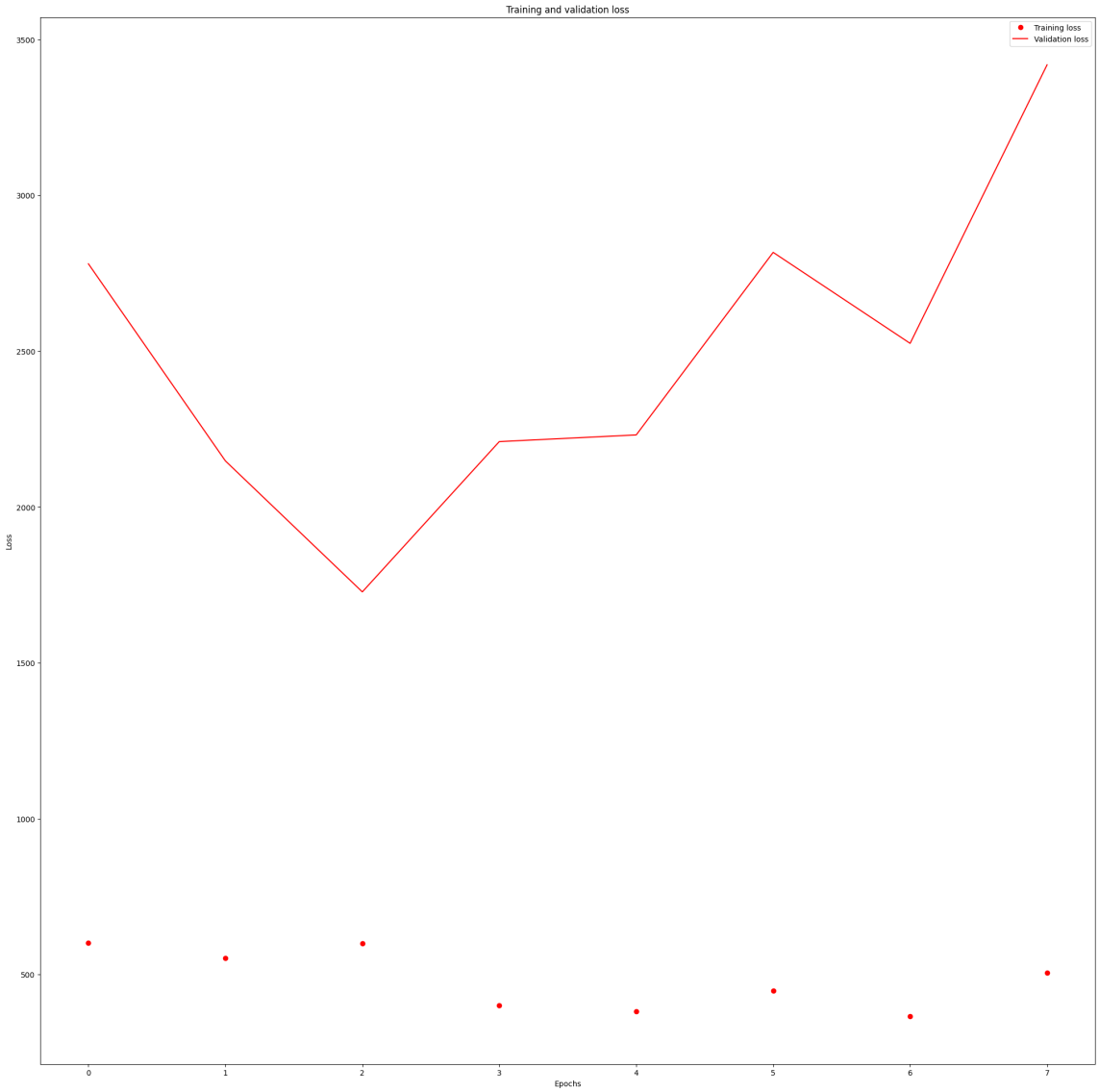

Figure 64: Training loss for pre-trained model with feature extraction from block 2

7 References

References

[1] Mrinal Haloi. Traffic sign classification using deep inception based convolutional networks. *CoRR*, abs/1511.02992, 2015.

[2] Alexandros Stergiou, Grigorios Kalliatakis, and Christos Chrysoulas. Traffic sign recognition based on synthesised training data. *Big Data and Cognitive Computing*, 2(3), 2018.

[3] Citlalli Gámez Serna and Yassine Ruichek. Classification of traffic signs: The european dataset. *IEEE Access*, 6:78136–78148, 2018.

[4] Álvaro Arcos-García, Juan A. Álvarez García, and Luis M. Soria-Morillo. Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. *Neural Networks*, 99:158–165, 2018.

[5] Qing Tang, Laksono Kurnianggoro, and Kang-Hyun Jo. Traffic sign classification with dataset augmentation and convolutional neural network. page 222, 04 2018.

[6] Muneeb Khan, Heemin Park, and Jinseok Chae. A lightweight convolutional neural network (cnn) architecture for traffic sign recognition in urban road networks. *Electronics*, 12:1802, 04 2023.

[7] Xin Roy Lim, Chin Poo Lee, Kian Ming Lim, Thian Song Ong, Ali Alqahtani, and Mohammed Ali. Recent advances in traffic sign recognition: Approaches and datasets. *Sensors*, 23(10), 2023.

[8] Ramya Sree Pothineni, Srinivas Inampudi, Lakshmi Yesaswini Gudavalli, and T. Lakshmi Surekha. Traffic sign classification using deep learning. In *2023 Third International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, pages 527–531, 2023.

[9] Yi Yang, Hengliang Luo, Huarong Xu, and Fuchao Wu. Towards real-time traffic sign detection and classification. *IEEE Transactions on Intelligent Transportation Systems*, 17(7):2022–2031, 2016.

[10] Seungjong Yoon and Eungtae Kim. Temporal classification error compensation of convolutional neural network for traffic sign recognition. *Journal of Physics: Conference Series*, 806(1):012007, feb 2017.

[11] Zhenli He, Fengtao Nan, Xinfa Li, Shin-Jye Lee, and Yun Yang. Traffic sign recognition by combining global and local features based on semi-supervised classification. *IET Intelligent Transport Systems*, 14(5):323–330, 2020.

[12] Ervin Miloš, Aliaksei Kolesau, and Dmitrij Šešok. Traffic sign recognition using convolutional neural networks / kelio ženklu atpažinimas naudojant neuronini tinkla. *Mokslas - Lietuvos ateitis*, 10:1–5, 12 2018.

[13] Ali Youssef, Dario Albani, Daniele Nardi, and Domenico Bloisi. Fast traffic sign recognition using color segmentation and deep convolutional networks. volume 10016, 10 2016.

[14] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. pages 1453 – 1460, 09 2011.

[15] Matteo Biondi. In *https://github.com/MatteBiondi/Traffic_Sign_Recognition*.