# Convolutional Neural Networks for CIFAR-10 Dataset

Miguel Cruzeiro, 50%
*Machine Learning Fundamentals*
*Student 107660, DETI*
*Universidade de Aveiro*
Aveiro, Portugal
miguelcruzeiro@ua.pt

Miguel Figueiredo, 50%
*Machine Learning Fundamentals*
*Student 108287, DETI*
*Universidade de Aveiro*
Aveiro, Portugal
miguel.belchior@ua.pt

Pétia Georgieva
*Machine Learning Fundamentals*
*Course Instructor, DETI*
*Universidade de Aveiro*
Aveiro, Portugal
petia@ua.pt

***Abstract*—This report presents a comprehensive analysis of the CIFAR-10 dataset, which constitutes a single-label multiclass image classification problem between 10 different classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The main goal of this report is to evaluate and compare the implementation and performance of various Convolutional Neural Network (CNN) architectures on the CIFAR-10 dataset, which are particularly efficient for image classification tasks. The study emphasizes the importance of data preprocessing, parameter tuning and train-test splitting in optimizing the model's performance. Additionally, it discusses the utilization of various techniques to try to minimize overfitting and improve the model's performance, including early stopping and learning rate scheduling. Three distinct CNN models are implemented and tested, following an iterative approach that builds upon the improvements made in each subsequent model. Furthermore, a Transfer Learning approach with fine-tuning was applied to leverage pre-trained models and their learned feature representations.**

***Index Terms*—CIFAR-10 Dataset, Machine Learning, Convolutional Neural Networks, Multiclass Classification, Loss Function, Accuracy, Transfer Learning, Fine-tuning, Data Preprocessing, ROC, AUC, Confusion Matrix**

## I. Introduction

This report presents the second project undertaken as part of the Machine Learning Fundamentals course. The goal of this project [1], is to apply an appropriate machine learning algorithm to address a specific problem using the chosen dataset. From the available topics proposed to complete the assignment, the CIFAR-10 dataset [2] was selected as image classification is a particularly area of interest for the group, where the results feel more visible and tangible when compared to other problems within the broad field of machine learning. Furthermore, this allows the group to focus on a different problem on this assignment, as the first assignment dealt with the Bank Marketing Dataset [3], and to implement a new model, such as Convolutional Neural Networks (CNN's).

The CIFAR-10 dataset is a widely used collection of images in the field of machine learning for training and testing image classification models. It contains 60,000 32x32 color images across 10 distinct classes, with each class representing one of the following categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. This way, the dataset is evenly balanced with each class having an equal number of images, ensuring that no category is disproportionately represented. The goal is to implement a model to identify one of the various classes, which are mutually exclusive. Therefore, this problem constitutes a **single-label multiclass classification problem**.

The project proposal outlined the use of Convolutional Neural Networks (CNNs) to address the problem, as CNN's constitute one of the most widely used and effective algorithms for image classification and object recognition tasks. Three different Convolutional Neural Networks were implemented, tested and evaluated, following an iterative approach that builds upon the improvements made in each subsequent model. All the models were evaluated based on the progression of the loss function and accuracy across epochs for both the training and validation sets. Additionally, performance was assessed using the ROC curve and AUC for each class, along with metrics such as accuracy, precision, recall, and F1-score. The evaluation also included the corresponding confusion matrix for a comprehensive analysis of each model's performance.

Additionally, the VGG16 model, a widely recognized pre-trained architecture, was implemented as part of the experimentation process. However, it was not extensively experimented with or analyzed in great detail compared to the custom models.

## II. State of the Art Analysis

The CIFAR-10 dataset, a widely used benchmark in image classification, has inspired extensive research and innovation in Convolutional Neural Networks (CNNs). These networks have proven highly effective in extracting meaningful features from images without the need for manual feature engineering, a process that is time-consuming.

This section provides a detailed analysis of the state-of-the-art methods employed for image classification using CNNs on CIFAR-10. It explores various CNN architectures and techniques, focusing on their design, performance metrics, and approaches to overcoming common challenges such as overfitting and high computational demands.

## *A. Convolutional Neural Network Implementation for Image Classification using CIFAR-10 Dataset*

In the study by Ajala Sunday Adeyinka (2021) [4], three CNN architectures were developed and their performance was evaluated:

- **Net I**: A simple CONV-POOL layer stack followed by fully connected layers resulting in a total of 8 layers. It is exhibiting significant overfitting with a training accuracy of 96.82% but a test accuracy of only 72.25% (Figure 1).

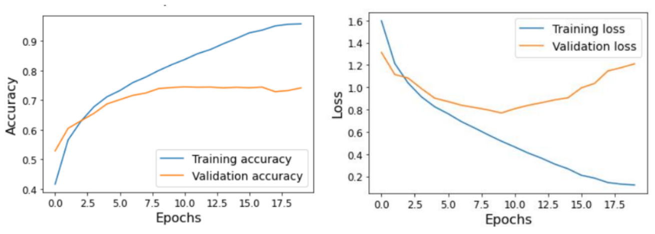


Fig. 1. Net I: Training Accuracy and Validation Accuracy, Training Loss and Validation Loss over 20 epoch

- **Net II**: Similar to Net I with the same number of convolution layers(3) and fewer pooling layers(2), which increased the model's complexity and led to more severe overfitting (training accuracy: 97.40%, test accuracy: 69.18%) (Figure 2).

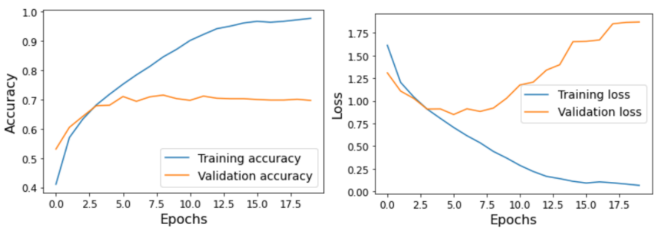


Fig. 2. Net II: Training Accuracy and Validation Accuracy, Training Loss and Validation Loss over 20 epoch

- **Net III**: A deeper network incorporating a 50% dropout rate after each pooling layer and L2 regularization to mitigate overfitting. It achieved the best balance between training and test performance, with a training accuracy of 77.04% and a test accuracy of 78.29% after 100 epochs (Figure 3).

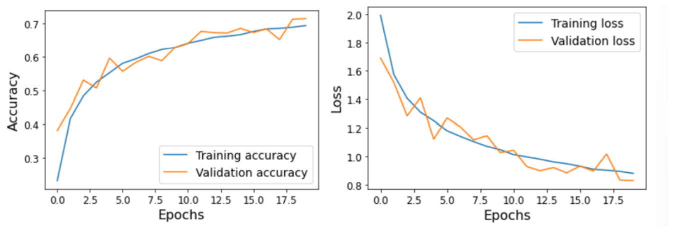


Fig. 3. Net III: Training Accuracy and Validation Accuracy, Training Loss and Validation Loss over 20 epochs

This work demonstrated that incorporating regularization techniques and increasing network depth could improve generalization while controlling overfitting.

## *B. Effect of filter sizes on image classification in CNN: a case study on CFIR10 and Fashion-MNIST datasets*

This study from Owais Mujtaba Khanday and Samad Dadvandipour (2021) [5] aims to analyze the impact of filter sizes on the performance of Convolutional Neural Networks (CNNs) using the CIFAR-10 dataset. The research explored three commonly used filter sizes: 3×3, 5×5, and 7×7, while keeping other network parameters constant. The primary objective was to evaluate how varying filter sizes affect model accuracy and computational efficiency.

The study utilized a 13-layer CNN architecture with two Conv2D layers, batch normalization, max-pooling, dropout, and dense layers. Each configuration was trained for 50 epochs with 32 filters per layer, while maintaining the same overall network structure to isolate the effects of filter size. The results indicated that smaller filter sizes (3×3) outperformed larger ones (5×5 and 7×7) in terms of accuracy, achieving a test accuracy of 73.04% on CIFAR-10, as seen in Figure 4.

| Filter Size | Training Data | Validation Data | Test Data |
|---|---|---|---|
| $3 \times 3$ | 0.942625 | 0.7275 | 0.7304 |
| $5 \times 5$ | 0.923275 | 0.7261 | 0.7297 |
| $7 \times 7$ | 0.87725 | 0.7067 | 0.635 |

Fig. 4. Accuracy of the CIFAR-10 dataset using different filter sizes

This work demonstrated an inverse relationship between filter size and accuracy, where smaller filters captured more detailed spatial features, leading to better generalization. However, using smaller filters also increased computational costs due to a higher number of parameters.

## *C. Transfer Learning: Performance Analysis of VGG-16 onCIFAR-10*

This study by Vijay Kumar Samyal, Harsh Gupta, Karamveer and Dinesh Puri [6] analyzed the impact of transfer learning on image classification using the CIFAR-10 dataset, comparing the performance of a baseline CNN model with the pre-trained VGG-16 architecture. Transfer learning uses pre-trained models to extract generic features from large datasets and adapts them to new ones, significantly improving performance and reducing training time.

The study make use of the VGG-16 model, pre-trained on the ImageNet dataset, as a feature extractor. Custom classification layers were added to adapt the model to CIFAR-10, including dense layers with ReLU activation and a softmax output layer. Data augmentation techniques were also applied to improve generalization. The VGG-16 model's convolutional layers were frozen to focus only on training the custom layers.

The CNN was overfitting as it achieved a training accuracy of 83% an a validation accuracy of only 71%. As for the VGG-16 model, it achieved a trainig and validating accuracy of 90% and 85.42% respectively (Figure 5), showing a better overall generalization compared to the baseline CNN.

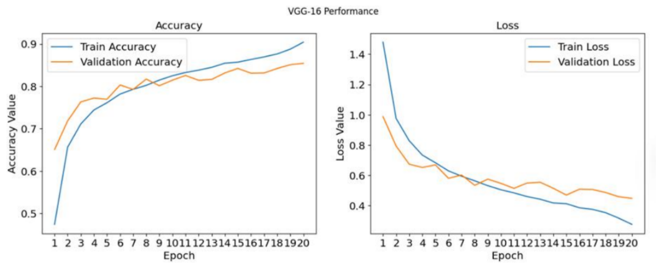


Fig. 5. VGG-16 accuracy and loss over 20 epochs

The results showed a clear advantage of transfer learning over training from scratch with a basic CNN. The baseline CNN achieved a test accuracy of 69%, while the VGG-16 model attained 84.46%, demonstrating superior generalization.

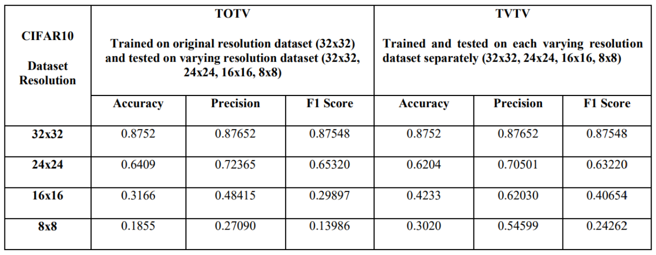
### *D. Effects of Varying Resolution on Performance of CNN based Image Classification: An Experimental Study*

Suresh Prasad Kannojia and Gaurav Jaiswal conducted a study [7] examining how varying image resolutions impact CNN classification performance on MNIST and CIFAR-10 datasets. Their experimental methodology involved two distinct approaches:

- Training on original resolution (32x32) and testing on varying resolutions (TOTV).
- Training and testing on each resolution separately (TVTV)

For this study, CIFAR10 dataset is rescaled into 8x8, 16x16, 24x24 and 32x32 pixel resolution image dataset and a CNN based image classifier implemented in sk-learn and keras with the backend of Tensorflow was utilized.

As seen by Figure 6, their experiments on CIFAR-10 revealed that classification performance degradates significantly as image resolution decreases. Using their base CNN architecture, they achieved 87.52% accuracy on the original 32x32 resolution. However, performance dropped to 64.09% when testing on 24x24 images, and further declined to 31.66% and 18.55% for 16x16 and 8x8 resolutions respectively when using the TOTV approach.



| CIFAR10 Dataset Resolution | TOTV Trained on original resolution dataset (32x32) and tested on varying resolution dataset (32x32, 24x24, 16x16, 8x8) | | | TVTV Trained and tested on each varying resolution dataset separately (32x32, 24x24, 16x16, 8x8) | | |
|---|---|---|---|---|---|---|
| | Accuracy | Precision | F1 Score | Accuracy | Precision | F1 Score |
| 32x32 | 0.8752 | 0.87652 | 0.87548 | 0.8752 | 0.87652 | 0.87548 |
| 24x24 | 0.6409 | 0.72365 | 0.65320 | 0.6204 | 0.70501 | 0.63220 |
| 16x16 | 0.3166 | 0.48415 | 0.29897 | 0.4233 | 0.62030 | 0.40654 |
| 8x8 | 0.1855 | 0.27090 | 0.13986 | 0.3020 | 0.54599 | 0.24262 |

Fig. 6. Performance result for different resolution images on CIFAR10 dataset

Their TVTV method showed slightly better robustness to lower resolutions. The degradation was less severe suggesting that CNNs can adapt better to lower resolutions when explicitly trained on them.

The study highlights the importance of maintaining adequate resolution for image classification, as the complex visual information in these natural images is particularly sensitive to resolution reduction.

### *E. Advancing Image Classification on CIFAR-10: Exploring Architectural Depth, Transfer Learning, and Innovative Computational Approaches*

The study by Kaixin Wu [8] presents a comprehensive analysis of different image classification techniques applied to CIFAR-10 dataset, highlighting the progression from traditional CNNs to advanced architectures and innovative computational methods.

Combined with stardard data augmentation techniques, this study introduced a Generative Adversarial Network (GAN) approach to generate synthetic image data to increase the diversity of the training set.

The research documented a clear progression in classification performance across different architectural approaches. Traditional CNNs established a baseline accuracy of 75.0%, while deeper architectures like ResNet, ResNeXt and attention reaching 85%, 87% and 88.5% respectively. Transfer Learning achieved the highest performance (90%) (Figure 7).

| model | Accuracy | Summons rate | F1 score | Training time |
|---|---|---|---|---|
| CNN | 75.0% | 74.5% | 74.7% | 2 hours |
| ResNet | 85.0% | 84.5% | 84.7% | 4 hours |
| ResNeXt | 87.0% | 86.5% | 86.7% | 5 hours |
| Attention | 88.5% | 88.0% | 88.2% | 6 hours |
| Transfer Learning | 90.0% | 89.5% | 89.7% | 3 hours |

Fig. 7. Model-Performance Comparison

The results demonstrated a clear trade-off between model complexity and performance. While deeper architectures consistently achieved better accuracy, they also required significantly more training time with the exception of Transfer Learning. Notably, transfer learning emerged as a particularly efficient approach, achieving the highest accuracy while maintaining relatively low computational requirements.

The research emphasized the practical implications of architectural choices, noting that while advanced models showed superior classification performance, they demanded significantly greater computational resources.

## III. Dataset Information

### A. Data Description

The CIFAR-10 dataset contains 60,000 32x32 color images across 10 distinct classes, with each class representing one of the following categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The 10 distinct classes, along with 10 different random samples from each class are represented on Figure 8.

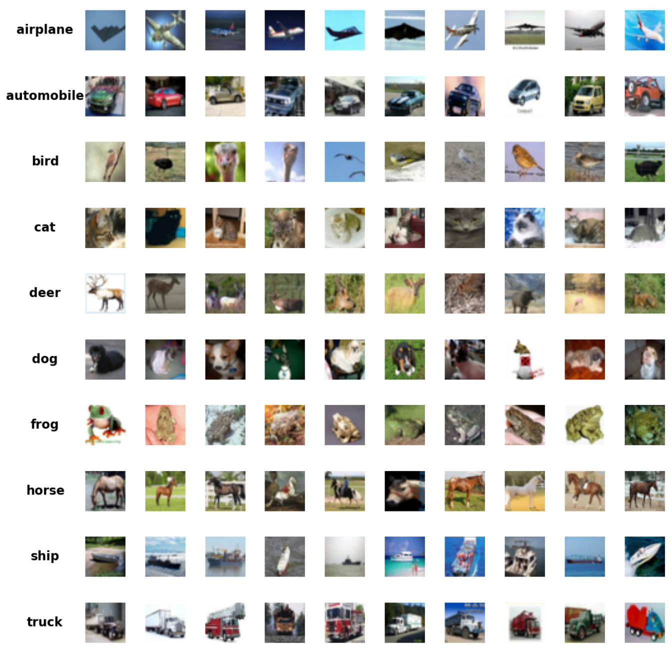


Fig. 8. Example images from each of the 10 classes in the CIFAR-10 dataset

As previously mentioned, the images in the CIFAR-10 dataset are clearly colored, with each pixel values ranging from 0 to 255 for the three color channels: Red, Green, and Blue (RGB). To facilitate effective model training, it is crucial to normalize these pixel values to a range between 0 and 1, by dividing each pixel value by 255. The data preprocessing process, including this pixel value normalization, will be further discussed later in the report.

### B. Statistical Analysis

Since the task at hand is a multiclass classification problem, it is important to examine the distribution of the target classes across the entire dataset, which is highlighted in Figure 9. As previously mentioned, the figure showcases an even distribution, with 6000 images (10% of the full dataset) for each of the 10 classes, showcasing the **balanced nature of the dataset**.

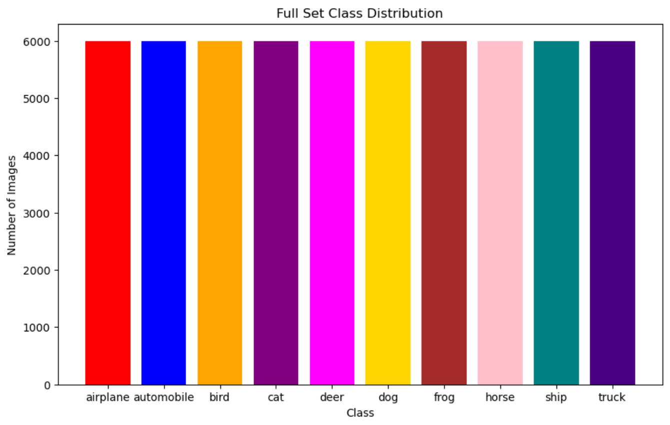


Fig. 9. Class Distribution across the entire dataset

As it will later be discussed, in order to ensure each subset of the data is representative of the overall dataset during the train-validation-test split, we used **stratified sampling**, which ensures that the class distribution in each subset (train, validation, and test sets) is the same as the distribution in the original dataset. The class distribution for the train, test and validation sets after the 80-10-10 train-validation-test split is displayed in Figure 10.

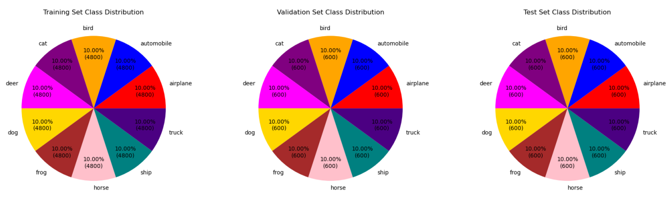


Fig. 10. Enter Caption

By analysis of Figure 10 it is possible to conclude that by using **stratified sampling**, the **resulting training, validation and test sets are also balanced** with 10% of entries corresponding to each class, which amounts to 4800 samples (10% of 80% of 60000) for the training set and 600 samples for the validation and test sets (10% of 10% of 60000).

## IV. Methodology

In the following section, we will outline the methodology employed to solve the problem, including a detailed analysis of the steps involved: data preprocessing, the data splitting strategy and the machine learning models implemented.

### A. Data Preprocessing

Effective preprocessing is crucial to prepare the dataset for modeling and ensure optimal performance of the machine learning algorithms. The following preprocessing steps were carried out.

*1) Data Normalization:* To facilitate effective model training, it is crucial to normalize the RGB (Red, Green, Blue) pixel values, which range from 0 to 255, to a range between 0 and 1, by dividing each pixel value by 255. The data preprocessing process, including this pixel value normalization, will be further discussed later in the report.

*2) One Hot Encoding of the Target Labels:* The target labels in this multiclass classification task represent the class of each image. These labels were converted into a one-hot encoded format using the `to_categorical` function [9]. In this format, each label is represented as a vector where the correct class is marked with a 1, and all other classes are marked with a 0. For this dataset, as there are 10 classes, each label is represented as a 10-dimensional vector.

## B. Splitting the dataset

The dataset was split using stratified sampling, by utilizing parameter `stratify` of the `train_test_split` function [10], to ensure that the class distribution remains consistent between the various subsets. A train-validation-test split was performed, following an 80-10-10 ratio, which divided the initial dataset into the following subsets:

- **Training Set** - The subset of data used to train the Convolutional Neural Network (CNN). The model learns the underlying features and patterns from this data through multiple forward and backward passes during each training epoch;
- **Validation Set** - A separate dataset used during training to validate the model's performance. After each epoch, the model's performance on the validation set is evaluated to ensure generalization, and techniques such as early stopping are employed to save the best-performing model (only on the third implemented CNN);
- **Test Set** - A distinct, previously unseen dataset used to evaluate the final model after training is complete. It serves as an unbiased measure of the model's ability to generalize to new, real-world data. Thus, the most credible outcome of the model training is the performance metric with the test data, not used for training or validation of the model.

## C. Model Implementation - Convolutional Neural Networks

To tackle the image classification problem, Convolutional Neural Networks (CNNs) were selected as the model of choice. Convolutional Neural Networks are a class of deep learning models particularly effective for processing data with grid-like topology, such as images. They have three main types of layers:

- **Convolutional Layers** which apply a set of filters to the input, capturing spatial hierarchies in the data and identifying key features like edges, textures, and shapes;
- **Pooling Layers** - Pooling layers, also known as downsampling, conduct dimensionality reduction, reducing the number of parameters in the input. There are two main types of pooling:
  1) **Max pooling** - As the filter moves across the input, it selects the pixel with the maximum value to send to the output array. This approach tends to be used more often compared to average pooling, and was the one used in the showcased implementations as well;
  2) **Average pooling** - As the filter moves across the input, it calculates the average value within the receptive field to send to the output array;
- **Fully Connected Layers** - In Fully-connected layers, each node in the output layer connects directly to a node in the previous layer, usually serving as the final steps before output, transforming the spatial features into a class score. While convolutional and pooling layers tend to use ReLu functions, FC layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability from 0 to 1.

In summary, the architecture of CNN's is designed to automatically learn and extract hierarchical features from images, making them well-suited for tasks like object detection and image classification, which is precisely the problem at hand with the CIFAR-10 dataset.

Another notable application of CNNs is in transfer learning, where pre-trained models are adapted for new but related tasks. This is done by adding new layers to the pre-trained model to accommodate the specific outputs required for the new task and then fine-tuning the entire model by training it on the new dataset. This approach significantly reduces training time and improves performance, especially when the new dataset is smaller or less diverse, leveraging the rich feature representations learned from the original, larger dataset. As previously mentioned, the VGG16 model is an example of this and it was implemented as part of the experimentation process.

### 1) CNN Version 1:

The first version of the implemented Convolutional Neural Network was designed at establishing a baseline performance metric and was heavily based on a kaggle notebook by Bhuvan Chennoju [11], which also focused on the CIFAR-10 dataset. The model's architecture is highlighted on Figures 11 and 12.

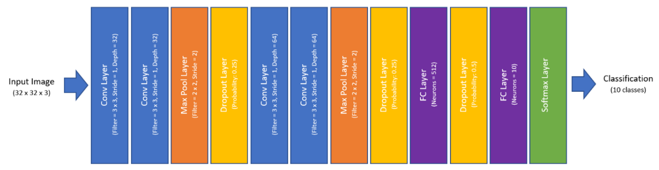
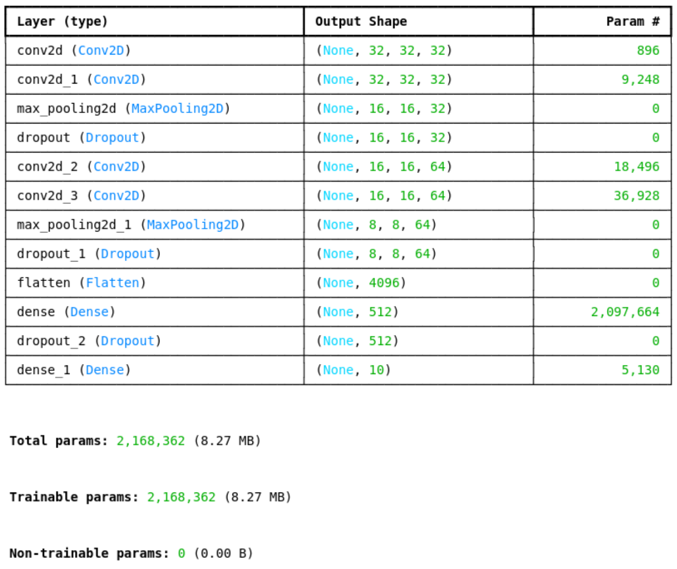


Fig. 11. CNN V1 Architecture



| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 32) | 896 |
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 9,248 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| dropout (Dropout) | (None, 16, 16, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 16, 16, 64) | 18,496 |
| conv2d_3 (Conv2D) | (None, 16, 16, 64) | 36,928 |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| dropout_1 (Dropout) | (None, 8, 8, 64) | 0 |
| flatten (Flatten) | (None, 4096) | 0 |
| dense (Dense) | (None, 512) | 2,097,664 |
| dropout_2 (Dropout) | (None, 512) | 0 |
| dense_1 (Dense) | (None, 10) | 5,130 |

**Total params:** 2,168,362 (8.27 MB)

**Trainable params:** 2,168,362 (8.27 MB)

**Non-trainable params:** 0 (0.00 B)

Fig. 12. CNN V1 Architecture - Detailed

Figure 11 provides a visual summary of the neural network architecture, showing the flow from the input image to the final classification. The input image shape is 32 × 32 × 3, matching the previously mentioned 32 × 32 image size with three color channels (RGB - Red, Green, Blue). Figure 12 gives a more detailed technical breakdown of each layer, including the type, output shape, and the number of parameters.

The network consists of two blocks of 2 convolutional layers — to extract local features from the input image or previous layer — with a ReLU activation function followed by a max pooling layer — to reduce spatial dimensionality — and a dropout layer — to help preventing overfitting by randomly dropping a fraction of the neurons during training, making the network's architecture dynamically different for each training batch. Afterwards, the flattened output from the convolutional layers is fed into two fully connected dense layers. The first dense layer has 512 neurons and ReLU activation with a dropout probability of 0.8. The last (dense) layer has 10 neurons, representing the 10 output classes, and utilizes the Softmax activation function, which converts the raw output scores into a probability distribution. Each neuron's output represents the probability that the input image belongs to one of the 10 classes. This softmax layer ensures that the model outputs class probabilities suitable for multi-class classification tasks.

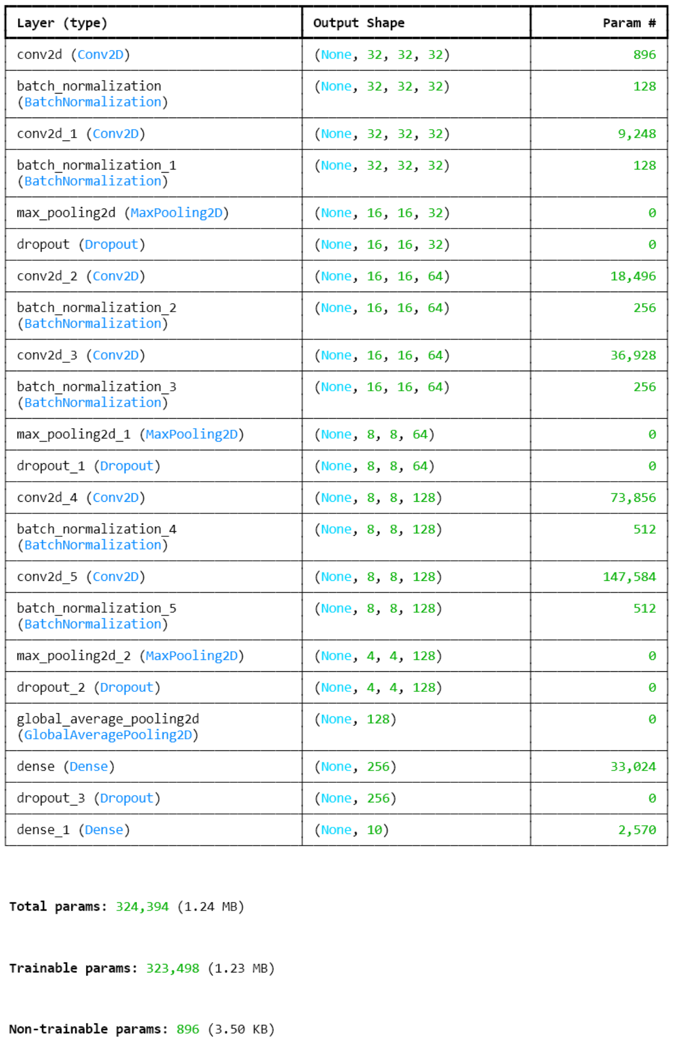
The model was compiled (`model.compile` function from keras [12]) using the following parameters:

- **Optimizer** - The **Adam Optimizer** was utilized with an initial small learning rate of 0.0009. The Adam optimizer is well suited for this problem due to its adaptive learning rate capabilities;
- **Loss Function** - The `categorical_crossentropy` loss function was used, which is a loss function that's commonly used for multi-class classification problems, such as the CIFAR-10 dataset;
- **Metrics** - Accuracy was specified as the performance metric to be tracked and evaluated during training and testing.

The model was trained using a batch size of 64 over the course of 75 epochs. During training, the validation data `X_val` and `y_val` was used to monitor the model's performance and adjust the learning process, helping to prevent overfitting and improving generalization.

*2) CNN Version 2:*

The second version of the Convolutional Neural Network introduces a more complex architecture with the aim of reducing the overfitting observed in Version 1 and improving generalization. As seen by the Figure 13, the new model increases depth by introducing a third block of convolutional layers with 128 filters.



| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d (Conv2D) | (None, 32, 32, 32) | 896 |
| batch_normalization (BatchNormalization) | (None, 32, 32, 32) | 128 |
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 9,248 |
| batch_normalization_1 (BatchNormalization) | (None, 32, 32, 32) | 128 |
| max_pooling2d (MaxPooling2D) | (None, 16, 16, 32) | 0 |
| dropout (Dropout) | (None, 16, 16, 32) | 0 |
| conv2d_2 (Conv2D) | (None, 16, 16, 64) | 18,496 |
| batch_normalization_2 (BatchNormalization) | (None, 16, 16, 64) | 256 |
| conv2d_3 (Conv2D) | (None, 16, 16, 64) | 36,928 |
| batch_normalization_3 (BatchNormalization) | (None, 16, 16, 64) | 256 |
| max_pooling2d_1 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| dropout_1 (Dropout) | (None, 8, 8, 64) | 0 |
| conv2d_4 (Conv2D) | (None, 8, 8, 128) | 73,856 |
| batch_normalization_4 (BatchNormalization) | (None, 8, 8, 128) | 512 |
| conv2d_5 (Conv2D) | (None, 8, 8, 128) | 147,584 |
| batch_normalization_5 (BatchNormalization) | (None, 8, 8, 128) | 512 |
| max_pooling2d_2 (MaxPooling2D) | (None, 4, 4, 128) | 0 |
| dropout_2 (Dropout) | (None, 4, 4, 128) | 0 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 128) | 0 |
| dense (Dense) | (None, 256) | 33,024 |
| dropout_3 (Dropout) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 10) | 2,570 |

**Total params:** 324,394 (1.24 MB)

**Trainable params:** 323,498 (1.23 MB)

**Non-trainable params:** 896 (3.50 KB)

Fig. 13. CNN V2 Architecture - Detailed

Batch normalization was also added after each Conv2D layer to remove biases on the batch dimension, meaning that the network can easily identify differences between the features produced for different images of a batch. This version introduces Global Average Pooling that replaces the Flattern layer, so, instead of flattening the feature maps and connecting them to a dense layer, global average pooling directly feeds the averaged values into the output layer. It aids in reducing model complexity and preventing overfitting by reducing the number of parameters in the network. From the Figure 13 we observe that the total params reduced from over 2 milion in the first implementation of the CNN to around 300 thousand in the second version. The dropouts was also reduced from 0.8 to 0.5 in the first dense layer balancing regularization without overly penalizing the network.

*3) CNN Version 3:*

The third version of the Convolutional Neural Network retains the same architecture as version 2, with improvements made in the model's compilation and training phases. The initial learning rate for the Adam Optimized was increased to 0.001. When fitting the model, the learning rate scheduler `ReduceLROnPlateau` [13] was used to dynamically adjusting the learning rate during training to optimize convergence. This helps the model to make large updates at the beginning of training when the parameters are far from their optimal values, and smaller updates later when the parameters are closer to their optimal values, allowing for more fine-tuning. The following parameters were configured for the scheduler:

- **Monitor**: Monitors the validation loss to decide when to reduce the learning rate.
- **Factor**: Reduces the learning rate by multiplying it by 0.5 (halving it) when triggered.
- **Patience**: Waits for 3 epochs without improvement in the validation loss before reducing the learning rate.
- **Min_lr**: Specifies a minimum learning rate of $1 \times 10^{-6}$ to prevent it from going too low.

Early stopping was also included to stop training early if the model's performance on the validation set stops improving, preventing overfitting and saving time. The parameters used for early stopping include:

- **Monitor**: Monitors the validation loss to decide when to stop training.
- **Patience**: Waits for 5 epochs without improvement before stopping training.
- **Restore_best_weights**: Restores the model's weights from the epoch with the best validation loss after stopping.

## V. Results and analysis

### A. CNN Version 1

Figure 14 showcases the evolution of the loss function (on the left) and accuracy (on the right) over the epochs for the training and validation data. On the Loss Function plot, it's clear that initially the training and validation loss decrease. However, it appears that after epoch 20/30, the validation loss doesn't decrease any further, while the loss for the training data keeps decreasing. A similar behaviour is exhibited on the accuract plot. Initially both the accuracy with the training and validation data increases. However, it appears that after epoch 20/30, the accuracy for the validation data plateaus, while it keeps increasing for the training data.

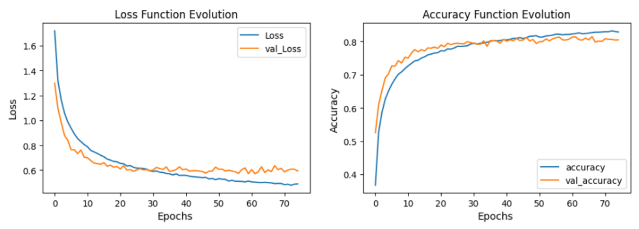


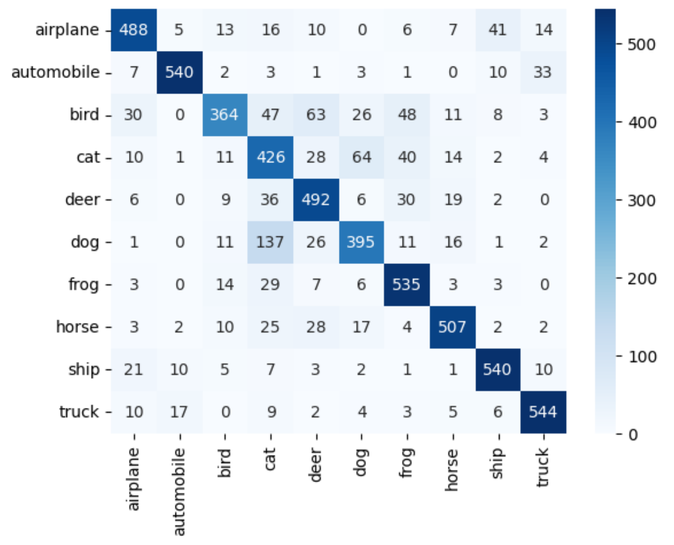
Fig. 14. Accuracy and Loss of the Training and Validation Data over the Epochs — CNN Version 1

Table I presents the accuracy performance of the model across the training, validation, and test sets. The model achieved accuracies of 94.14% on the training set, 80.50% on the validation set, and 80.52% on the test set. There is a noticeable drop in accuracy between the training set and validation and test sets suggesting that the model is overfitting, struggling to generalize to unseen data. However, the validation and test set accuracies are very similar, which suggests that the model's performance on new, unseen data is consistent with its performance during validation.

TABLE I
Accuracy Comparison On Training, Validation and Test Sets
CNN Version 1

| **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|
| 94.14% | 80.50% | 80.52% |

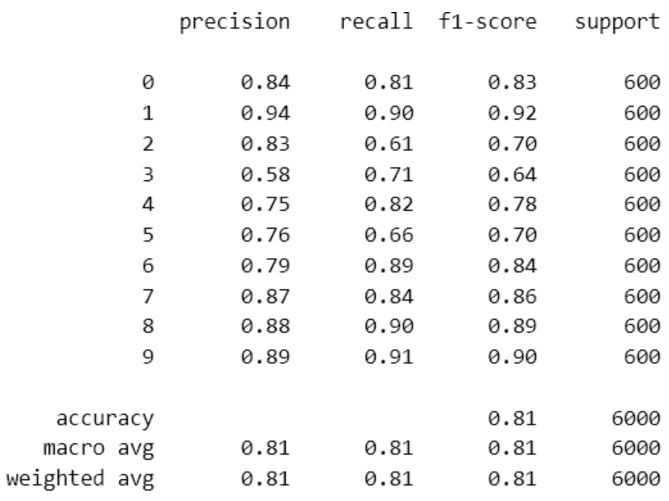
Figure 15 displays the confusion matrix for the first implemented CNN. Correct classifications, where predictions align with the actual values, are reflected in the diagonal cells with higher counts. The classes truck, automobile and ship are classified very accurately. The worst performing classes are birds, dogs and cats that are misclassified 236, 205 and 174 times, respectively. The bird class is frequently misclassified with deer (63), frog (48), cat (47) and airplane (30) classes. Most of the misclassified instances in the dog class were incorrectly identified as belonging to the cat class (137 out of 205). In the same way, the cat class frequently gets confused with dog (64) and frog (40) classes. It's also of note that airplanes are sometimes confuded with the ship class and the automobile with the truck class, showcasing their visual similarities.



|  | airplane | automobile | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| airplane | 488 | 5 | 13 | 16 | 10 | 0 | 6 | 7 | 41 | 14 |
| automobile | 7 | 540 | 2 | 3 | 1 | 3 | 1 | 0 | 10 | 33 |
| bird | 30 | 0 | 364 | 47 | 63 | 26 | 48 | 11 | 8 | 3 |
| cat | 10 | 1 | 11 | 426 | 28 | 64 | 40 | 14 | 2 | 4 |
| deer | 6 | 0 | 9 | 36 | 492 | 6 | 30 | 19 | 2 | 0 |
| dog | 1 | 0 | 11 | 137 | 26 | 395 | 11 | 16 | 1 | 2 |
| frog | 3 | 0 | 14 | 29 | 7 | 6 | 535 | 3 | 3 | 0 |
| horse | 3 | 2 | 10 | 25 | 28 | 17 | 4 | 507 | 2 | 2 |
| ship | 21 | 10 | 5 | 7 | 3 | 2 | 1 | 1 | 540 | 10 |
| truck | 10 | 17 | 0 | 9 | 2 | 4 | 3 | 5 | 6 | 544 |

Fig. 15. Confusion Matrix — CNN Version 1

Figure 16 displays a classification report of the trained model on the testing set. Yet again the best performing classes (with the best metrics) are classes 1, 8 and 9 corresponding to classes automobile, ship and truck. As previously observed, the metrics values for classes 2 (bird), 3 (cat) and 5 (dog) are considerably lower than for the rest of the classes with class 2 and 5 exhibiting worrying values of recall (0.61 and 0.66, respectively) indicating a high number of false negatives (many instances of this class are misclassified as others) and class 3 showcasing a very low precision value of 0.58, indicating a high number of false positives.



```
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       600
           1       0.94      0.90      0.92       600
           2       0.83      0.61      0.70       600
           3       0.58      0.71      0.64       600
           4       0.75      0.82      0.78       600
           5       0.76      0.66      0.70       600
           6       0.79      0.89      0.84       600
           7       0.87      0.84      0.86       600
           8       0.88      0.90      0.89       600
           9       0.89      0.91      0.90       600

    accuracy                           0.81      6000
   macro avg       0.81      0.81      0.81      6000
weighted avg       0.81      0.81      0.81      6000
```

Fig. 16. Classification Report — CNN Version 1

Figure 25 showcases the ROC curve and AUC for each of the classe sin the multi-class classification problem. The ROC curve plots the True Positive Rate (TPR) over the False Positive Rate (FPR) at various threshold settings. The diagonal line in each graph represents the ROC curve for a purely random classifier, where TPR = FPR. A perfect classifier's curve would hug the top-left corner, maximizing TPR while minimizing FPR. In the figure, AUC values are also displayed which quantifies the overall performance of the model and represents the probability that the model, if given a randomly chosen positive and negative example, will rank the positive higher than the negative. All classes present high values of AUC with the top performers being classes airplane, automobile, frog, horse, ship and truck, which exhibit near perfect classification values (0.99). As previously observed, the worst performers include again (in order) the bird, cat and dog classes.

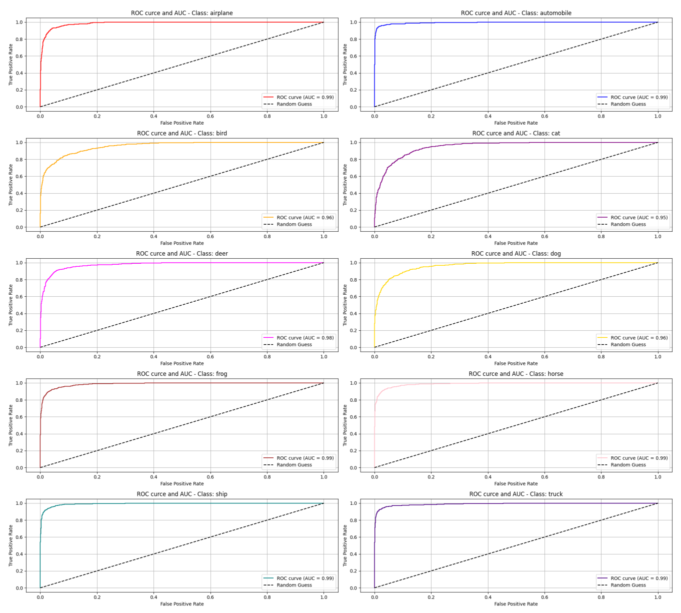


Fig. 17. ROC Curve and AUC — CNN Version 1

## B. CNN Version 2

The charts in Figure 18 ilustrate the accuracy and loss for both training and validation sets over 50 epochs. Analysing the charts we observe that the orange line (accuracy/loss in validation set) stabilizes at a higher value that the training loss and at a lower value at the accuracy, indicating possible overfitting.

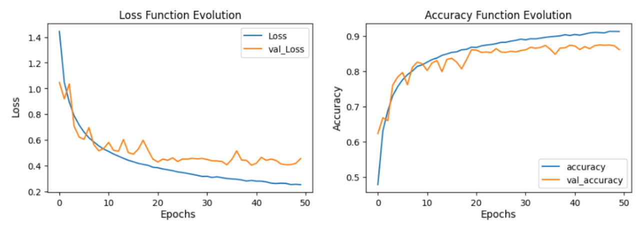


Fig. 18. Accuracy and Loss of the Training and Validation Data over the Epochs — CNN Version 2

The performance of the CNN version 2 was measured using the train, test and validation sets. We observe a significant improvement over Version 1, achieving an accuracy of 96.03% on training set, 86.07% on validation set and 85.92% on test set. Notably, the gap between the training and test set accuracies has been reduced by nearly 5%, indicating a considerable reduction in overfitting.

TABLE II
ACCURACY COMPARISON ON TRAINING, VALIDATION AND TEST SETS CNN VERSION 2

| **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|
| 96.03% | 86.07% | 85.92% |

In the confusion matrix of Figure 19, we observe fewer misclassifications compared to Version 1, indicating improved overall performance in terms of both accuracy and generalization. The diagonal cells, representing correct predictions, consistently show higher values than in the previous version. However, the model still struggles to distinguish between certain classes, such as cats and dogs, with 78 dogs misclassified as cats and 61 cats misclassified as dogs. Additionally, there is some confusion between birds and airplanes (10 misclassifications), cats (17), deer (26), and frogs (24), though these errors are less frequent than in Version 1. Similarly, the model occasionally confuses automobiles with trucks, as seen in 33 misclassifications, though these errors have also been reduced.

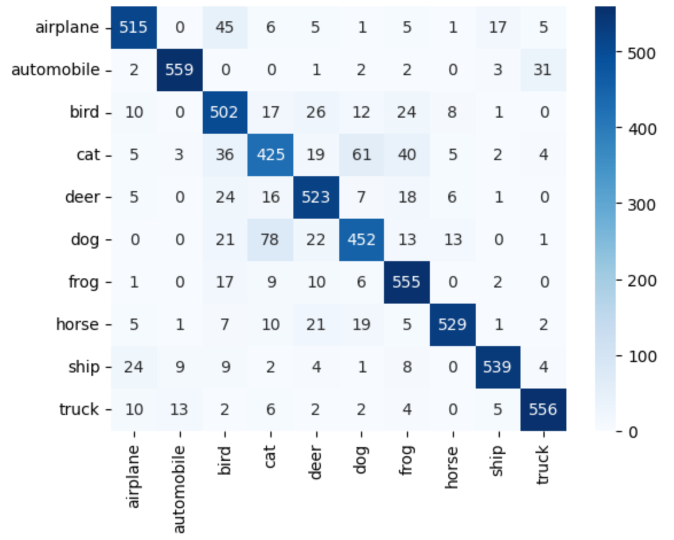
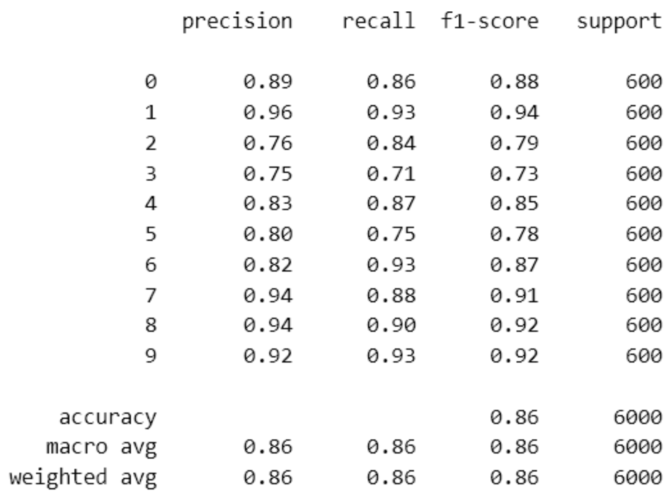


Fig. 19. Confusion Matrix — CNN Version 2

From the classification report in Figure 20, we observe that the model has improved in classifying nearly every class compared to Version 1. The only class that did not improve in all metrics was the bird class, with a precision of 0.76 compared to 0.83 in Version 1. However, despite the drop in precision, the recall and F1-score for the bird class have shown improvements indicating that the model is better at identifying more true positives, even if it misclassifies some other classes as birds.



```
              precision    recall  f1-score   support

           0       0.89      0.86      0.88       600
           1       0.96      0.93      0.94       600
           2       0.76      0.84      0.79       600
           3       0.75      0.71      0.73       600
           4       0.83      0.87      0.85       600
           5       0.80      0.75      0.78       600
           6       0.82      0.93      0.87       600
           7       0.94      0.88      0.91       600
           8       0.94      0.90      0.92       600
           9       0.92      0.93      0.92       600

    accuracy                           0.86      6000
   macro avg       0.86      0.86      0.86      6000
weighted avg       0.86      0.86      0.86      6000
```

Fig. 20. Classification Report — CNN Version 2

As shown in Figure 21, the AUC values, which were already very high in the first version, has either improved or stayed the same for all the classes. The worst performers in the last version were cat(0.95), dog(0.96) and bird(0.96) that are now having an AUC value of 0.97, 0.98 and 0.98 respectively.

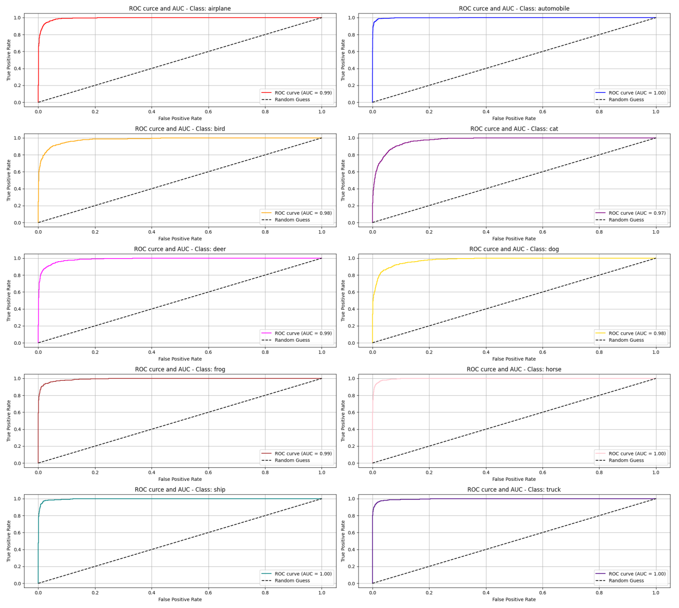


Fig. 21. ROC Curve and AUC — CNN Version 2

## C. CNN Version 3

Figure 22 shows the evolution of the loss function and accuracy over 50 epochs on validation and training sets. When comparing to the previous versions we observe that, when both of the validation and training lines stabilize, the gap between them remains almost the same, meaning the overfitting issue is still present.

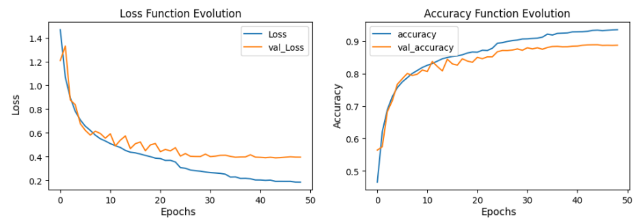


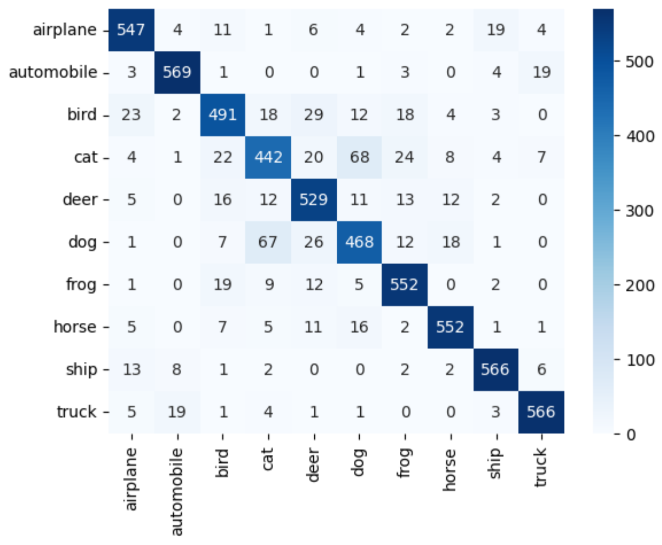
Fig. 22. Accuracy and Loss of the Training and Validation Data over the Epochs — CNN Version 3

From the Table III, we observe a significant improvement over Version 1 and Version 2, achieving an accuracy of 98.74% on training set, 88.83% on validation set and 88.03% on test set. However, the gap between the accuracy in training set and test set remains almost the same when comparing to the Version 2, indicating the model didn't improve the overfitting issue.

TABLE III
ACCURACY COMPARISON ON TRAINING, VALIDATION AND TEST SETS
CNN VERSION 3

| **Training Set** | **Validation Set** | **Test Set** |
|---|---|---|
| 98.74% | 88.83% | 88.03% |

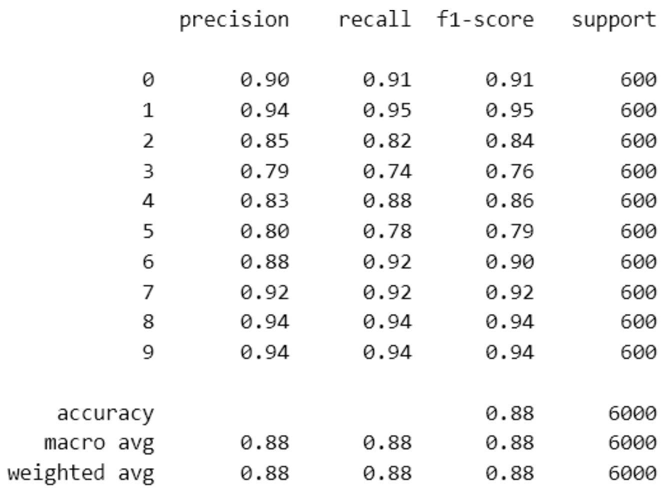
From the Confusion Matrix in Figure 23, we observe an overall improvement in correct predictions compared to Version 2. However, the bird and frog classes show a slight decrease in accuracy, with fewer correct predictions. Misclassifications have also improved in certain areas, such as automobiles being less frequently misclassified as trucks (31 in Version 2 compared to 19 in Version 3) and airplanes being less frequently confused with birds (45 compared to 11). On the other hand, some misclassifications have increased, such as cats being confused with dogs (61 in Version 2 compared to 68 in Version 3) and birds being misclassified as airplanes (10 compared to 23).



| | airplane | automobile | bird | cat | deer | dog | frog | horse | ship | truck |
|---|---|---|---|---|---|---|---|---|---|---|
| airplane | 547 | 4 | 11 | 1 | 6 | 4 | 2 | 2 | 19 | 4 |
| automobile | 3 | 569 | 1 | 0 | 0 | 1 | 3 | 0 | 4 | 19 |
| bird | 23 | 2 | 491 | 18 | 29 | 12 | 18 | 4 | 3 | 0 |
| cat | 4 | 1 | 22 | 442 | 20 | 68 | 24 | 8 | 4 | 7 |
| deer | 5 | 0 | 16 | 12 | 529 | 11 | 13 | 12 | 2 | 0 |
| dog | 1 | 0 | 7 | 67 | 26 | 468 | 12 | 18 | 1 | 0 |
| frog | 1 | 0 | 19 | 9 | 12 | 5 | 552 | 0 | 2 | 0 |
| horse | 5 | 0 | 7 | 5 | 11 | 16 | 2 | 552 | 1 | 1 |
| ship | 13 | 8 | 1 | 2 | 0 | 0 | 2 | 2 | 566 | 6 |
| truck | 5 | 19 | 1 | 4 | 1 | 1 | 0 | 0 | 3 | 566 |

Fig. 23. Confusion Matrix — CNN Version 3

Comparing the classification report in Figure 24 to that of Version 2, we observe that the values are largely similar across most classes. However, the precision values for the automobile, horse, and truck classes have slightly decreased. Despite this, the overall average of all metrics has improved, increasing from 0.86 to 0.88. This indicates that Version 3 has enhanced the model's overall performance.



```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91       600
           1       0.94      0.95      0.95       600
           2       0.85      0.82      0.84       600
           3       0.79      0.74      0.76       600
           4       0.83      0.88      0.86       600
           5       0.80      0.78      0.79       600
           6       0.88      0.92      0.90       600
           7       0.92      0.92      0.92       600
           8       0.94      0.94      0.94       600
           9       0.94      0.94      0.94       600

    accuracy                           0.88      6000
   macro avg       0.88      0.88      0.88      6000
weighted avg       0.88      0.88      0.88      6000
```

Fig. 24. Classification Report — CNN Version 3

In Figure 25, the ROC curves show a slight improvement in AUC values compared to the previous version, which was already near-perfect. The AUC values for the airplane, bird, cat, and frog classes have each increased by 0.01. Currently, the only classes that do not achieve a perfect AUC value of 1 are bird, cat, deer, and dog.

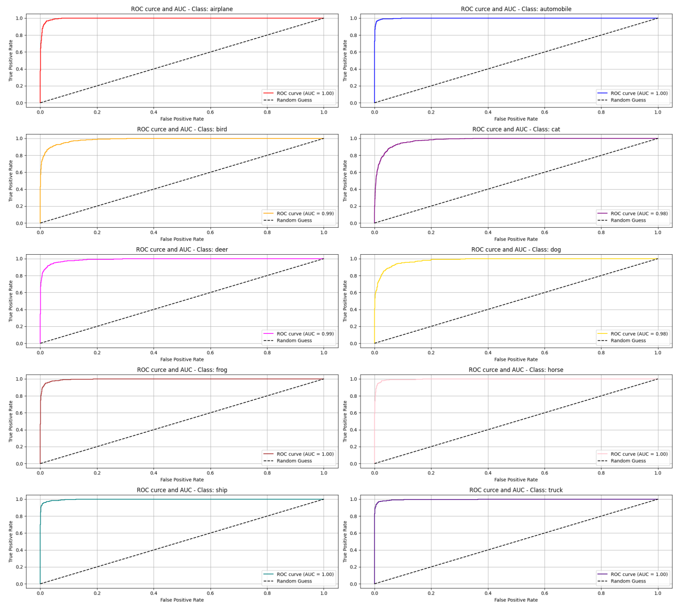


Fig. 25. ROC Curve and AUC — CNN Version 3

## VI. Comparing with solutions from other references

This section includes comparisons with the works referenced in the State of the Art Analysis section. It aims to highlight the similarities and differences in approaches, techniques, and results, emphasizing areas where the current work outperformed or underperformed compared to these studies.

In the work of Ajala Sunday Adeyinka [4], three CNN architectures were designed, but all of their models underperformed compared to ours. Their best-performing architecture achieved a test accuracy of 78.29%, which is significantly lower than all three of our models. This improvement in our models can be attributed to key enhancements, including the use of global average pooling, dynamic learning rate adjustment, and batch normalization. Furthermore, the total number of parameters in their best model architecture is 1,448,682, which is substantially higher than the 324,394 parameters in our model. This demonstrates that our models are not only more accurate but also more efficient in terms of computational complexity.

The study from Owais Mujtaba Khanday and Samad Dadvandipour (2021) [5] showed that smaller filters (3x3) achieved an higher test accuracy of 73.04%, while larger filters underperformed. This has been incorporated in the designed architectures, with the implemented CNN models utilizing 3x3 filters in combination with other architectural innovations, allowing significantly higher performance.

The work of Vijay Kumar Samyal [6] achieved a test accuracy of 84.46% using transfer learning with VGG-16. Our best custom CNN surpassed this result with a test accuracy of 88.03%. Their implementations showed signs of overfitting with a training accuracy of 90%, validation accuracy of 85.42% and test accuracy of 84.46% which suggests their regularization strategy might not have been as effective. While VGG-16's transfer learning approach can be powerful, Samyal's custom model, which utilized multiple layers with varying depths (1024, 512, 256, and 128 units), tried to show the potential of deeper architectures. However, deeper models like this one can be prone to overfitting, especially when the dataset is limited or when regularization techniques are not carefully tuned.

The work of Suresh Prasad Kannojia and Gaurav Jaiswal [7] showcases the adverse impact of reduced image resolution on classification accuracy of the CIFAR-10 dataset, with values ranging from 87.52% (original $32 \times 32$ resolution) to 18.55% ($8 \times 8$ resolution), utilizing the same base CNN architecture. The $32 \times 32$ original resolution of the CIFAR-10 dataset was preserved throughout the training and evaluation process in our work, ensuring that the models operated on images with full visual detail and avoiding the performance degradation observed at lower resolutions. The 87.52% accuracy mentioned for the CIFAR-10 dataset in this work has also been surpassed by the implemented CNN Version 3, with an accuracy with the test set of 88.03%.

Unlike the previously mentioned papers, the study by Kaixin Wu [8] presents a comprehensive analysis of different image classification techniques, from traditional CNNs to advanced architectures and innovative computational methods, on the CIFAR-10 dataset. The baseline accuracy of 75.0% of the traditional CNN implemented on this paper was surpassed by all the three implemented CNN versions. The accuracies of 85% and 87% for deeper architectures like ResNet and ResNeXt were also surpassed by the second and third version of the CNN with 85.92% and 88.03% accuracy, which were trained in a fraction of the time (supposedly took 4 and 5 hours to train the ResNet and ResNeXt models). However, the Attention and Transfer Learning models outperformed our best model (CNN version 3) with accuracies of 88.5% and 90%, respectively. However, the results shown throughout this report reflect a well-balanced trade-off between model complexity and performance, specially when compared to these more complex models.

## VII. Disclaimer regarding the VGG16 model

The report frequently refers to the fine-tuning process applied to the pre-trained VGG16 model. However, the results of this process weren't further analysed in this report as the obtained result accuracy on the test set of 84.02% was worse than the custom implemented models. This outcome can be attributed to the relatively simple set of layers that were added to the pre-trained model to fine-tune it, consisting of Global Average Pooling, a Dense layer with 256 units with a dropout probability of 0.5%, and a final Dense layer with 10 units for classification with a softmax activation function. Future work is necessary to further address and improve the model's performance.

## VIII. Conclusion

This report provides a comprehensive analysis of the CIFAR-10 dataset, covering everything from an in-depth examination of the dataset to the application of different convolutional neural network models to address the multi-class classification problem. It also encompasses the full pipeline to train, test and evaluate each of the proposed models as well as the impact of utilizing learning rate schedulers and early stopping techniques to maximize the model's performance. Unfortunately, although the results improved as changes were made to the convolutional neural network, the final version of the CNN (version 3) still didn't show improvements on the overfitting problem with both the second and third model version showcasing a gap of roughly 10% discrepancy between the accuracy with training and validation/test sets. Other techniques such as data augmentation were also tested, but to no avail. Future work is necessary to further address and solve this problem.

## References

[1] P. Georgieva, 'PROJECT 2 2024/2025 – Instructions," [Online]. Available: https://elearning.ua.pt/mod/resource/view.php?id=1198366. [Accessed: Jan. 13, 2025].

[2] A. Krizhevsky, "CIFAR-10 and CIFAR-100 datasets," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html. [Accessed: Jan. 13, 2025].

[3] J. Bachmann, "Bank Marketing Dataset," Kaggle, 2022. [Online]. Available: https://www.kaggle.com/datasets/janiobachmann/bank-marketing-dataset. [Accessed: Jan. 13, 2025]

[4] S. Kumar, "Convolutional Neural Network Implementation for Image Classification using CIFAR-10 Dataset," ResearchGate, 2022. [Online]. Available: https://www.researchgate.net/publication/355972159_Convolutional_Neural_Network_Implementation_for_Image_Classification_using_CIFAR-10_Dataset. [Accessed: Jan. 13, 2025]

[5] Owais Mujtaba Khanday, Samad Dadvandipour, Mohd Aaqib Lone, 'Effect of filter sizes on image classification in CNN: A case study on CIFAR-10 and Fashion-MNIST datasets," ResearchGate, 2022. [Online]. Available: https://www.researchgate.net/publication/356686393_Effect_of_filter_sizes_on_image_classification_in_CNN_a_case_study_on_CFIR10_and_Fashion-MNIST_datasets. [Accessed: Jan. 14, 2025]

[6] Vijay Kumar Samyal, Harsh Gupta, Karamveer, Dinesh Puri, 'Transfer Learning: Performance Analysis of VGG-16 onCIFAR-10," International Journal of Recent Papers in Research, vol. 5, no. 12, 2022. [Online]. Available: https://ijrpr.com/uploads/V5ISSUE12/IJRPR36421.pdf. [Accessed: Jan. 14, 2025]

[7] Suresh Prasad Kannojia, Gaurav Jaiswal, 'Effects of Varying Resolution on Performance of CNN based Image Classification: An Experimental Study," CafeProzhe, 2022. [Online]. Available: https://app.cafeprozhe.com/storage/files/project/Zpx6dS588sgrzp59BCVIF4qMUEekFI0mx2jKphM2.pdf. [Accessed: Jan. 14, 2025]

[8] S. S. S. R. Depuru, L. Wang, and V. Devabhaktuni, "Advancing Image Classification on CIFAR-10: Exploring Architectural Depth, Transfer Learning, and Innovative Computational Approaches," in *Proceedings of the 2022 IEEE International Conference on Big Data (Big Data)*, Osaka, Japan, Dec. 2022, pp. 1234-1241. [Online]. Available: https://ieeexplore.ieee.org/document/10604269. [Accessed: Jan. 15, 2025]

[9] TensorFlow, "tf.keras.utils.to_categorical," TensorFlow, 2025. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/utils/to_categorical. [Accessed: Jan. 15, 2025].

[10] Scikit-learn, "sklearn.model_selection.train_test_split," Scikit-learn, 2025. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.model_selection.train_test_split.html. [Accessed: Jan. 15, 2025].

[11] B. Chennoju, "CIFAR-10 Image Classification with CNN," Kaggle, 2023. [Online]. Available: https://www.kaggle.com/code/bhuvanchennoju/cifar-10-image-classification-with-cnn. [Accessed: Jan. 15, 2025].

[12] Keras, "Model Training APIs," Keras, 2025. [Online]. Available: https://keras.io/api/models/model_training_apis/. [Accessed: Jan. 15, 2025].

[13] Keras Team, "ReduceLROnPlateau," Keras Documentation. [Online]. Available: https://keras.io/api/callbacks/reduce_lr_on_plateau/. [Accessed: Jan. 15, 2025]