Deep Learning Models for Image Classification

Giovanni Beraldi Martin Miesbauer

December 18, 2024

Contents

1 Introduction

This report describes the implementation and experimentation of deep learning models for image classification using the CIFAR10 dataset. The objective is to understand and compare different models, data processing techniques, and training strategies.

2 Data Processing and Dataset Structure

The CIFAR10 dataset contains 60,000 images distributed among 10 classes. The dataset is balanced, with each class containing the same number of images. This dataset was divided into training, validation, and test sets with the following structure:

- Training set: 40,000 samples, used to train machine learning models
- Validation set: 10,000 samples, used to monitor training progress, tune hyperparameters, and prevent overfitting
- Test set: 10,000 samples, used to evaluate the final performance of trained models on unseen data

The images have a shape of (32, 32, 3), with RGB color channels. As an example, the first 8 labels for the training set were `[6:frog, 9:truck, 9:truck, 4:deer, 1:car, 1:car, 2:bird, 7:horse]`.

Figure 1: First 8 training samples

Before training, we preprocessed the data, transforming them into PyTorch tensors and normalizing pixel values.

3 Loss Functions and Metrics

3.1 Loss Function

The chosen loss function is Cross-Entropy, which measures the divergence between predicted class scores and ground-truth labels. In this context, PyTorch's built-in cross-entropy function was applied to evaluate the model's predictions against the true class labels.

3.2 Metrics

The evaluation metric used is accuracy, A custom `Accuracy` class was implemented to measure the model's performance. The class tracks overall accuracy and per-class accuracy to ensure balanced performance across all classes. The accuracy class has methods to reset its internal state, update the counts of correct predictions, and compute both overall and per-class accuracy. The update method is responsible for comparing predicted classes to ground-truth labels, ensuring valid input and incrementing counts accordingly.

4 Model Implementations

4.1 ResNet18

The ResNet18 model from `torchvision.models` was implemented to establish a baseline for performance. This model has a typical convolutional neural network architecture with residual blocks. We had to modify the output dimension to work with our task. Otherwise, we left it as is.

4.2 Convolutional Neural Network

The Convolutional Neural Network (CNN) model implemented for this project is designed to classify images into one of 10 classes. It follows a common architecture pattern with convolutional layers followed by a fully connected classification section. The hyperparameters, such as the dimensions of the convolutional and fully connected layers, as well as the dropout rate, are derived through hyperparameter tuning to configure the CNN to deliver robust performance on the given task.

4.2.1 Convolutional layers

The first part of the model consists of three convolutional layers, each followed by a batch normalization, ReLU activation, and max-pooling. This section is designed to extract high-level features from the input images.

4.2.2 Flatten and Fully Connected Layers

After the convolutional layers, the model uses a `Flatten` layer to convert the output into a one-dimensional vector. This vector is fed into Multi-Layer Perceptron (MLP) with two layers for classification. The final output layer performs the final classification, transformsing the feature vector into 10 output classes (corresponding to the CIFAR-10 dataset). A softmax activation function is used for normalize the output values of this layer into a probability distribution over the 10 classes.

4.3 Vision Transformer (ViT)

The Vision Transformer (ViT) model implemented for this project based on a mixture of code from the Machine Learning for Visual Computing lecture and the implementation of Dosovitskiy et al. (2021).

4.3.1 Linear Projection of Flattened Patches

At first the model splits the image into patches, flattens these and performs a linear projection. This can be performed using a convolutional layer followed by flattening.

4.3.2 Position Embeddings

Because the model is invariant to the order of the patches, we add a position embedding to it. The choice of a simple addition in combination with learnable parameters have been directly taken from Dosovitskiy et al. (2021).

4.3.3 Position Embeddings

The heart of the Vision Transformer is the Transformer Encoder Layer. We implemented this from scratch using the building blocks provided by PyTorch. We followed the structure provided by the diagrams in the lecture and fixed the number of hidden layers in the MLP to one.

5 Training Loop and Strategies

The main training loop consists of computing the forward pass for the whole pass at once, then perform the backward propagation. After 5 epochs, the performance is tested on the validation set. If the performance increased, we save the model weights. Data augmentation can be performed in this step by adding random transformations to the dataloader.

6 Experimentation and Results

6.1 Hyperparameter Tuning

To optimize the model's performance, a sweep was implemented using Weights and Biases (WandB) to performe hyperparametrs tuning and find the best combination of parameters. The tuning process was configured to maximize Validation Accuracy. To facilitate the tuning process, a `WandBHyperparameterTuning` class was defined, which encapsulates the logic of sweep creation and execution. By tracking these experiments with WandB, it was possible to effectively analyze the results, enabling easy comparison and visualization of the tuning process, as can be seen from the following charts.

6.1.1 Resnet tuning

We decided against tuning the resnet model to have a strong, simple baseline. Therefore nothing is done here.

6.1.2 CNN tuning

For the CNN, the hyperparameters to be tuned were defined as follows:

- **Batch Size**: The number of samples processed before the model's internal parameters are updated. A range of `[128, 256, 512]` was tested.
- **Dropout Rate**: The proportion of neurons to be randomly dropped during training to avoid overfitting. This was varied across `[0.3, 0.5, 0.7]`.
- **Convolutional Layer Dimensions**: The number of filters in the three convolutional layers, with different options for each layer
- **Fully Connected Layer Dimensions**: The number of neurons in the two MLP layers

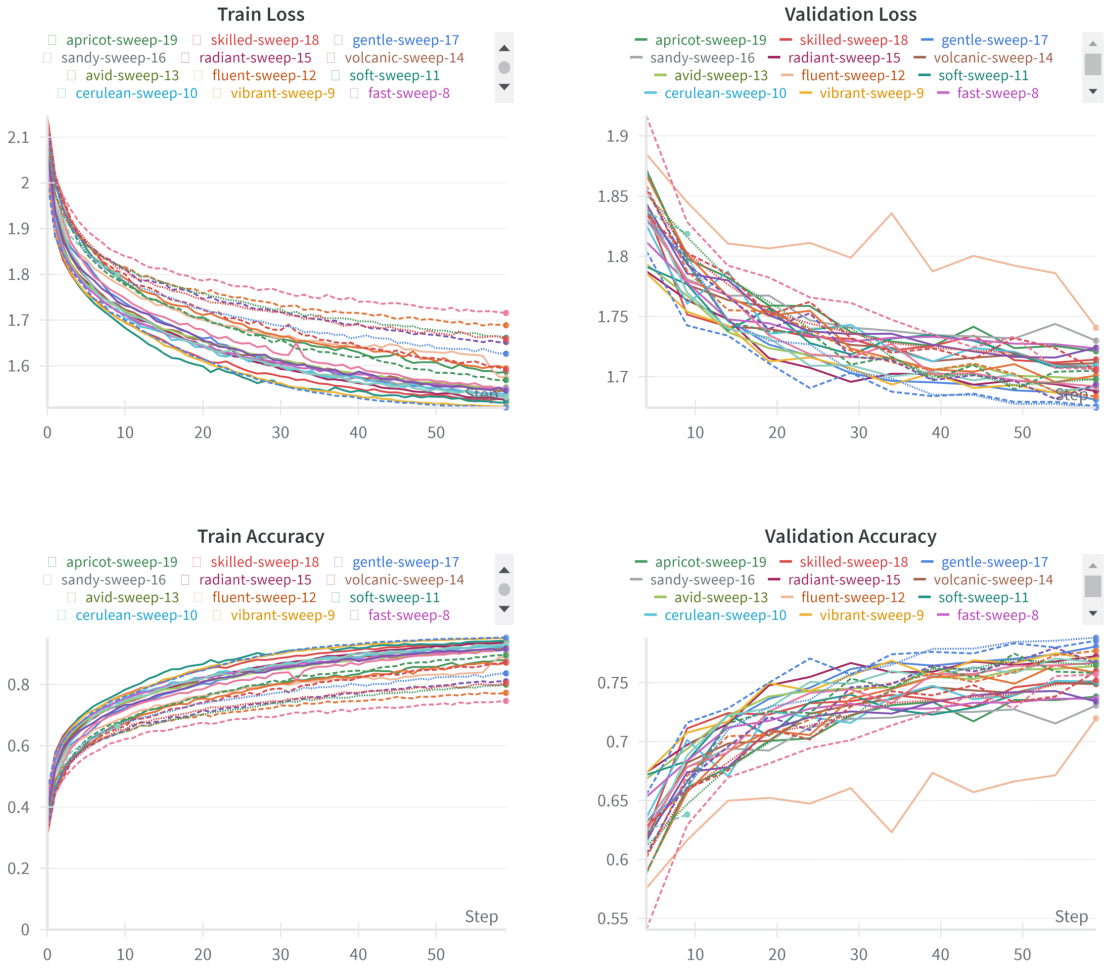

Figure 2: CNN tuning metrics

6.1.3 ViT tuning

For the ViT we had many more parameters. Therefore we decided on the following tuning strategy: At first we create a quite large range of hyperparameter values and choose random combinations. After letting it run overnight we restrict the range of the hyperparameters to the most promising values, and train a few more different models. In total we tried 83 different combinations of the following parameters:

- **Batch Size**: The number of samples processed before the model's internal parameters are updated.
- **Patch Size**: The size of the image patches
- **Dropout Rate**: The proportion of neurons to be randomly dropped during training to avoid overfitting.
- **lr**: The rate at which the model learns
- **gamma**: A parameter regulating the decay of the learn rate
- **optimizer**: The optimizer used for finding the weights
- **embed_dim**: The dimension of the projections of the patches.
- **num_encoder_layers**: The number of encoder layers

- **hidden_layer_depth**: The depth of the hidden layer in the MLP in the transformer layer
- **head_dim**: The latent dimension H of the heads
- **num_heads**: The number of different heads
- **mlp_head_hidden_layers_depth**: The depth of the hidden layer in the final MLP.
- **weight_decay**: A value regulating a decay of the weights. This might prevent overfitting.
- **dropout**

Initial tests quickly showed that the optimizer used was not important for the performance on the validation set. Therefore we decided to always use the AdamW optimizer. The full results from the initial and the second tuning sweeps can be seen in the csv files provided. Figure 3 shows the resulting metrics from the best 30 runs according to their validation accuracy. You can see a quite wide range of different losses and accuracies.

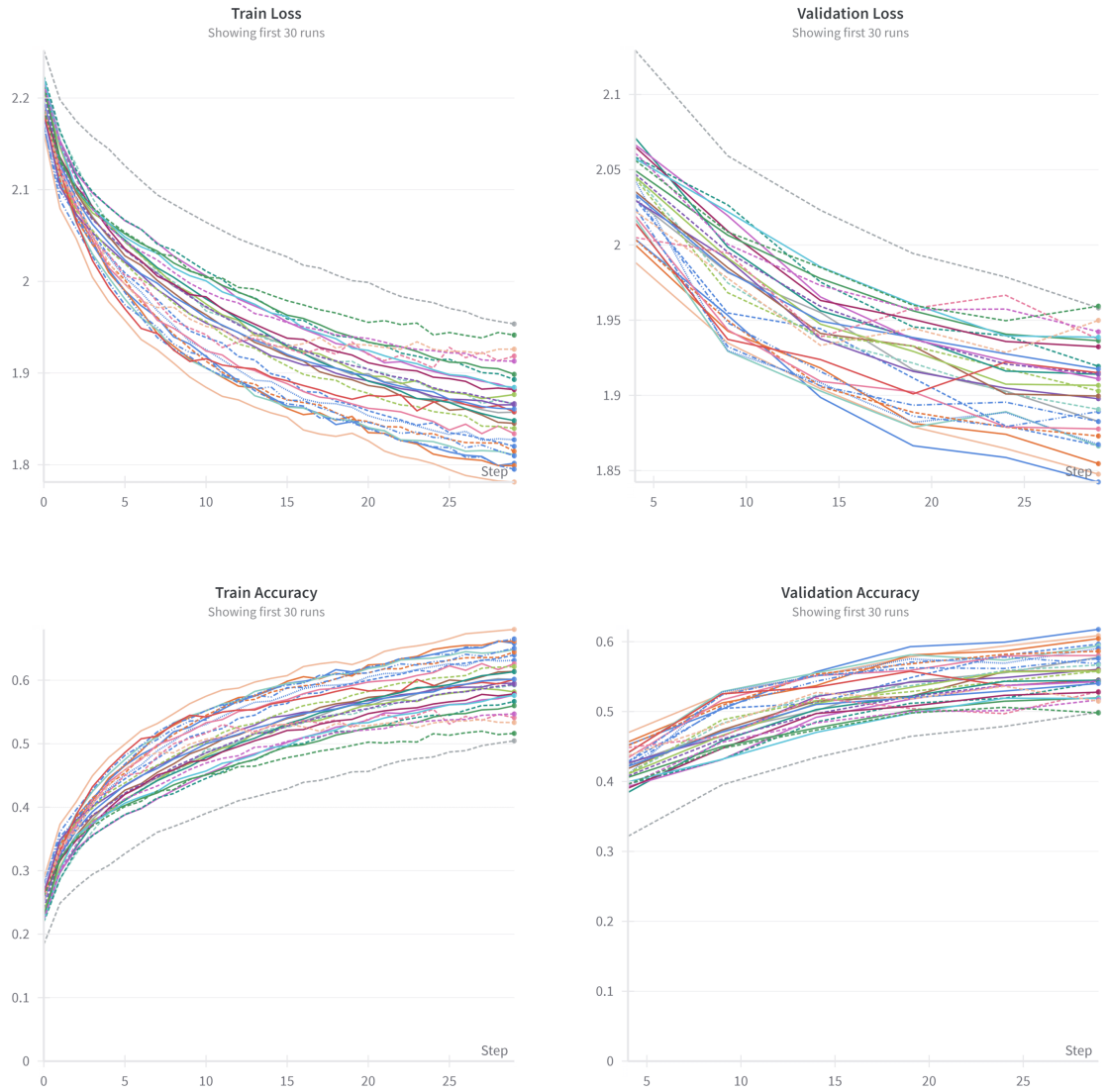

Figure 3: ViT tuning metrics

6.2 Regularization and Data Augmentation

We performed data augmentation expanding the variety of the training dataset by introducing random transformations to the existing data. This approach enhances the generalization capability of the model and helps mitigate over-fitting by exposing the model to a broader spectrum of data variations. To control the application of augmentation, we introduced a hyperparameter to define the probability of data augmentation, with a range from 0% to 100%. This adjustment allows us to explore the impact of varying degrees of augmentation on the model's performance. During training, the transformation is applied with a probability determined by this hyperparameter, enabling the model to experience a mix of original and augmented data across epochs. The transformation used are `Horizontal Flip`, `Random Rotation` and `Random Resized Crop`. In addition to the augmentation probability, we also incorporated a hyperparameter for regularization using dropout, with a range from 0 to 0.7. This setup provides flexibility to experiment with different combinations of augmentation and regularization, including scenarios where augmentation is applied without regularization, and vice versa, allowing us to understand the unique effects of each strategy and their optimal combination on model performance and generalization. This approach turned out to create quite a large overhead at the loading of each batch.

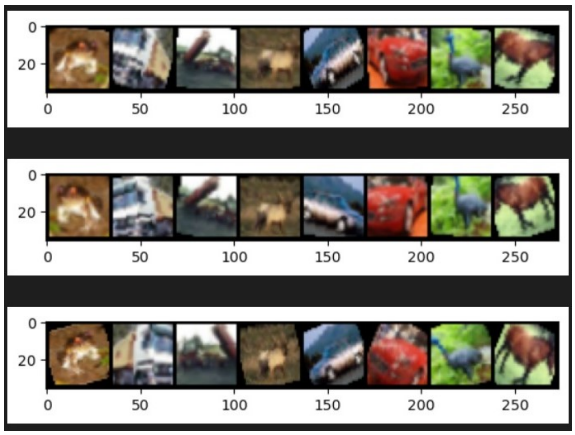

Figure 4: Data augmentation

6.3 Training and Validation

To train our deep learning models, we began by using Weights and Biases to obtain the optimal set of hyperparameters (only for the models for which the tuning was done). After identifying the best configuration, we trained the models for a certain number of epochs. We trained all the models using the AdamW optimizer, a variant of Adam that includes decoupled weight decay, and the ExponentialLR learning rate scheduler to adjust the learning rate exponentially during training. Throughout this process, all training metrics were logged on Weights and Biases for comprehensive tracking and analysis, ensuring a thorough evaluation of the model's performance.

6.3.1 ResNet training

We trained the ResNet model for 60 epochs. The following plots show the metrics recorded during the training.

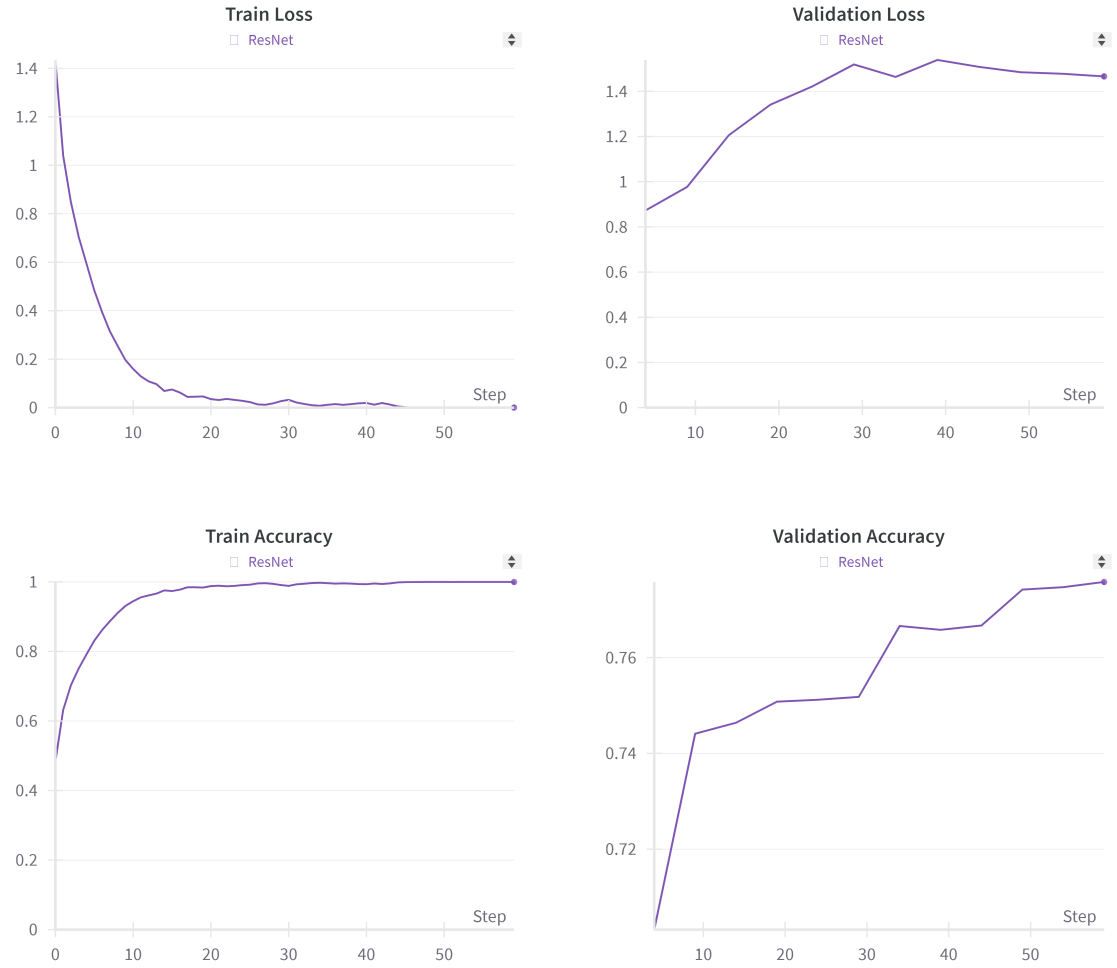

Figure 5: ResNet training metrics

6.3.2 CNN training

We trained our CNN model for 120 epochs. The following plots show the metrics recorded during the training.

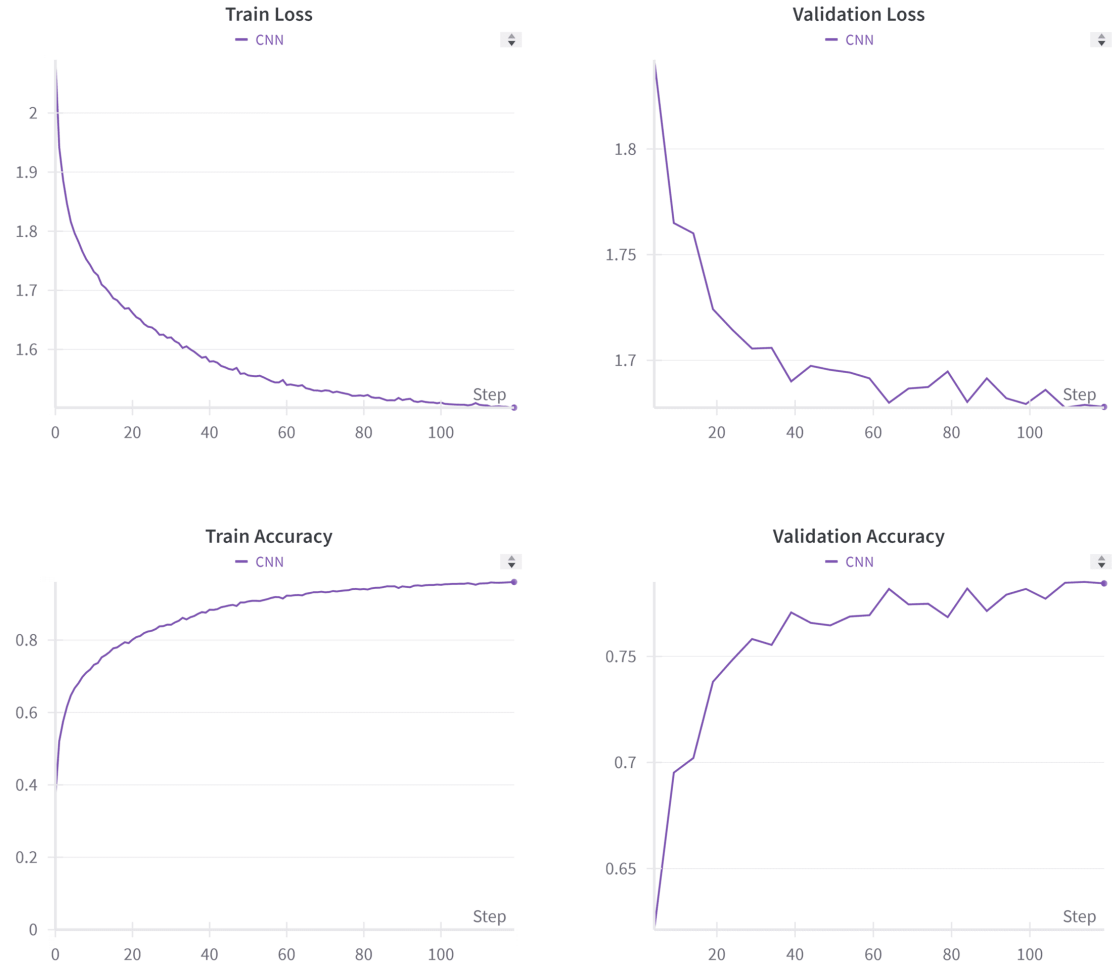

Figure 6: CNN training metrics

6.3.3 ViT training

The optimal parameters were: lr = 0.000365, gamma = 0.9, patch_size = 2, embed_dim = 32, num_encoder_layers = 4, hidden_layer_depth = 1024, head_dim = 16, num_heads = 20, dropout=0.25, mlp_head_hidden_layer_depth = 1024, batch_size = 128, and weight_decay = 0.1. We trained our ViT model twice for 60 epochs, once without and once with data augmentation. The run without data augmentation achieved a higher accuracy on the validation set. The following plots show the metrics recorded during the training.

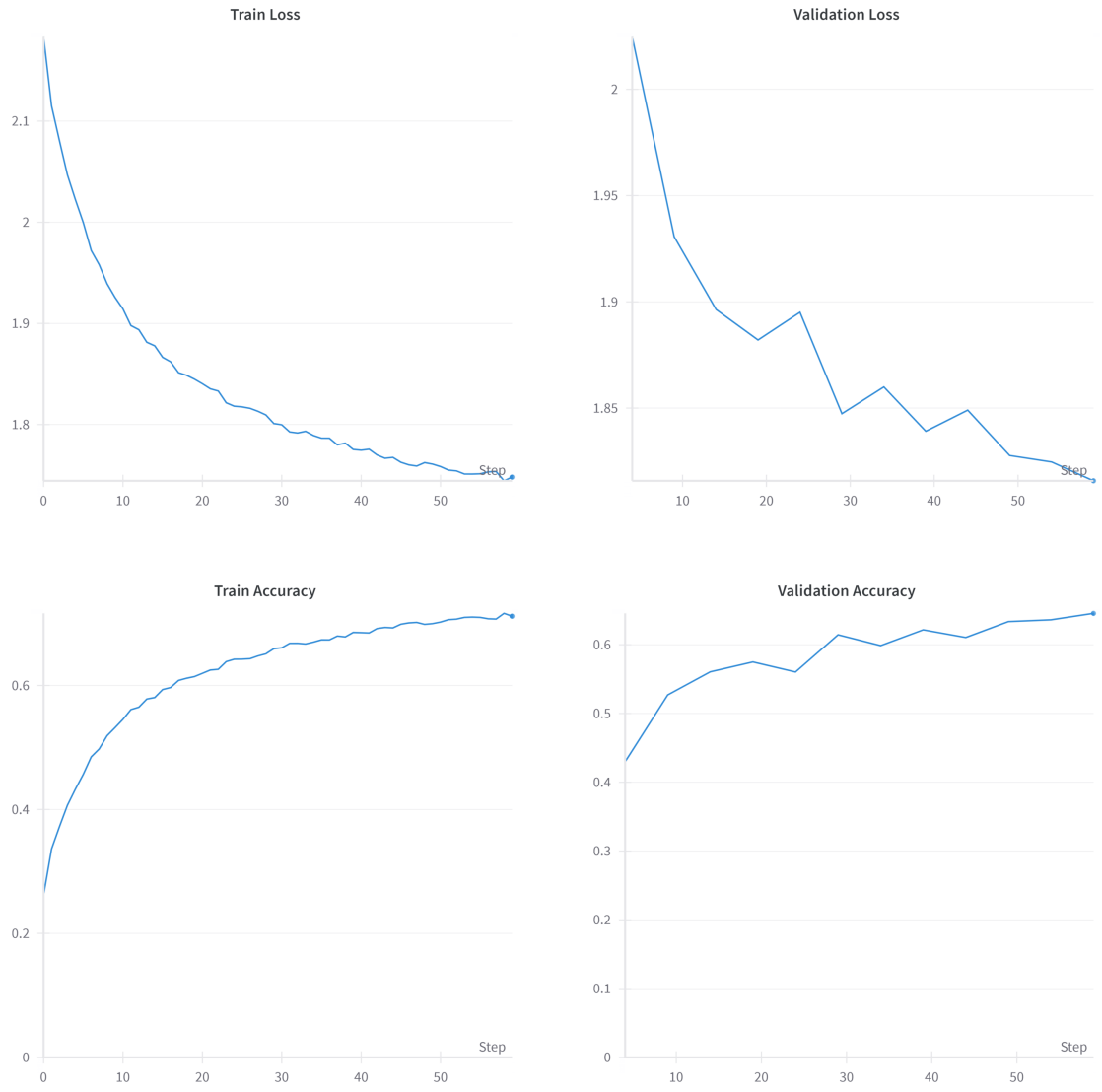

Figure 7: ViT training metrics

We can see that there could have been a benefit in training for more epochs, but this would have required more computing power.

6.4 Test Results

Following the training phase, we evaluated the models on a designated test set to assess their performance. We generated confusion matrices to visualize the accuracy of predictions and identify any misclassifications. Additionally, we plotted ROC curves for each model to examine their sensitivity and specificity, allowing for a deeper understanding of their discriminative capabilities. These evaluations provided crucial insights into the models' effectiveness and helped compare the various models.

6.4.1 ResNet test results

The ResNet model achieved a test loss of `1.49` and a test accuracy of `0.78`, indicating a good balance between prediction errors and overall correctness. Below are the confusion matrix and ROC curve for the ResNet model.

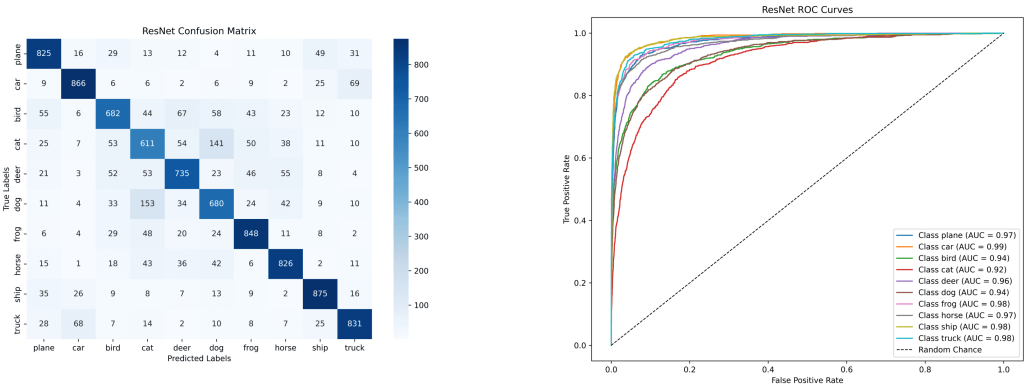

Figure 8: ResNet test results

6.4.2 CNN test results

Our CNN model achieved a test loss of `1.68` and a test accuracy of `0.77`. The performance can be compared to ResNet model, but it should be kept in mind that our model trained for twice as many epochs. Below are the confusion matrix and ROC curve for the CNN model.

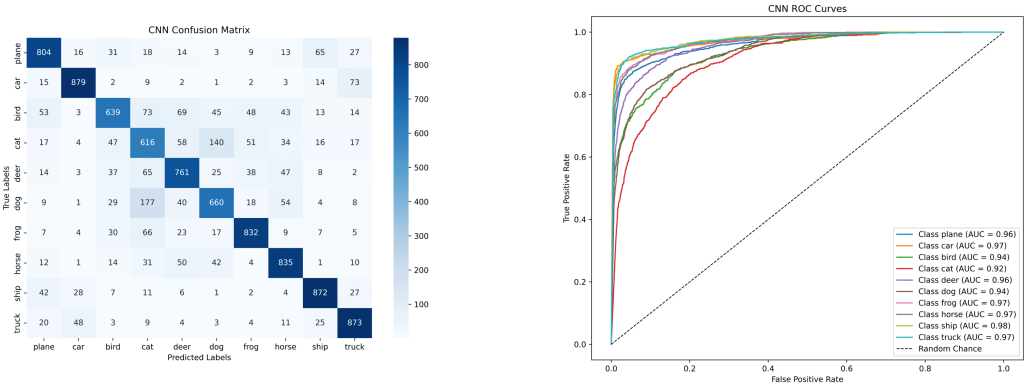

Figure 9: CNN test results

6.4.3 ViT test results

Our ViT model achieved a test loss of `1.81` and a test accuracy of `0.73`. The performance seems slightly worse than the performance of the other two models, but we seem to have fully utilized the maximum number of epochs for training. Below are the confusion matrix and ROC curve for the ViT model.

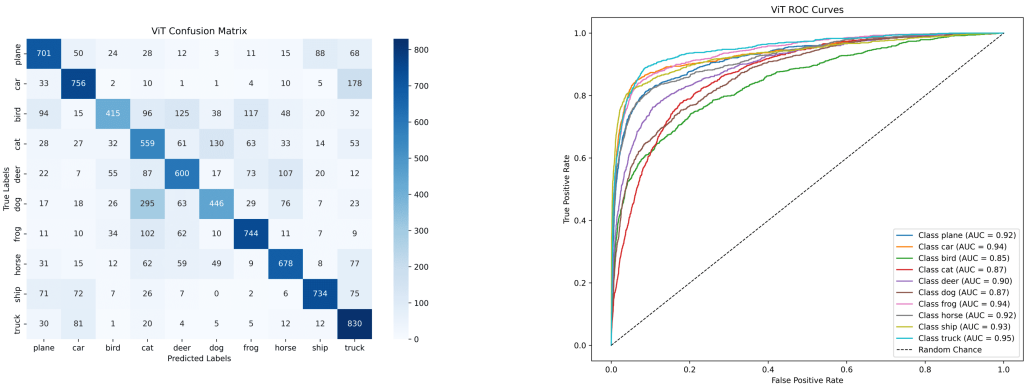

Figure 10: ViT test results

6.5 Comparison of Models

To compare the performance of various models during training, we plotted key metrics on a single graph. This visualization allowed us to examine trends in loss and accuracy across different models, facilitating a direct comparison of their learning progress.

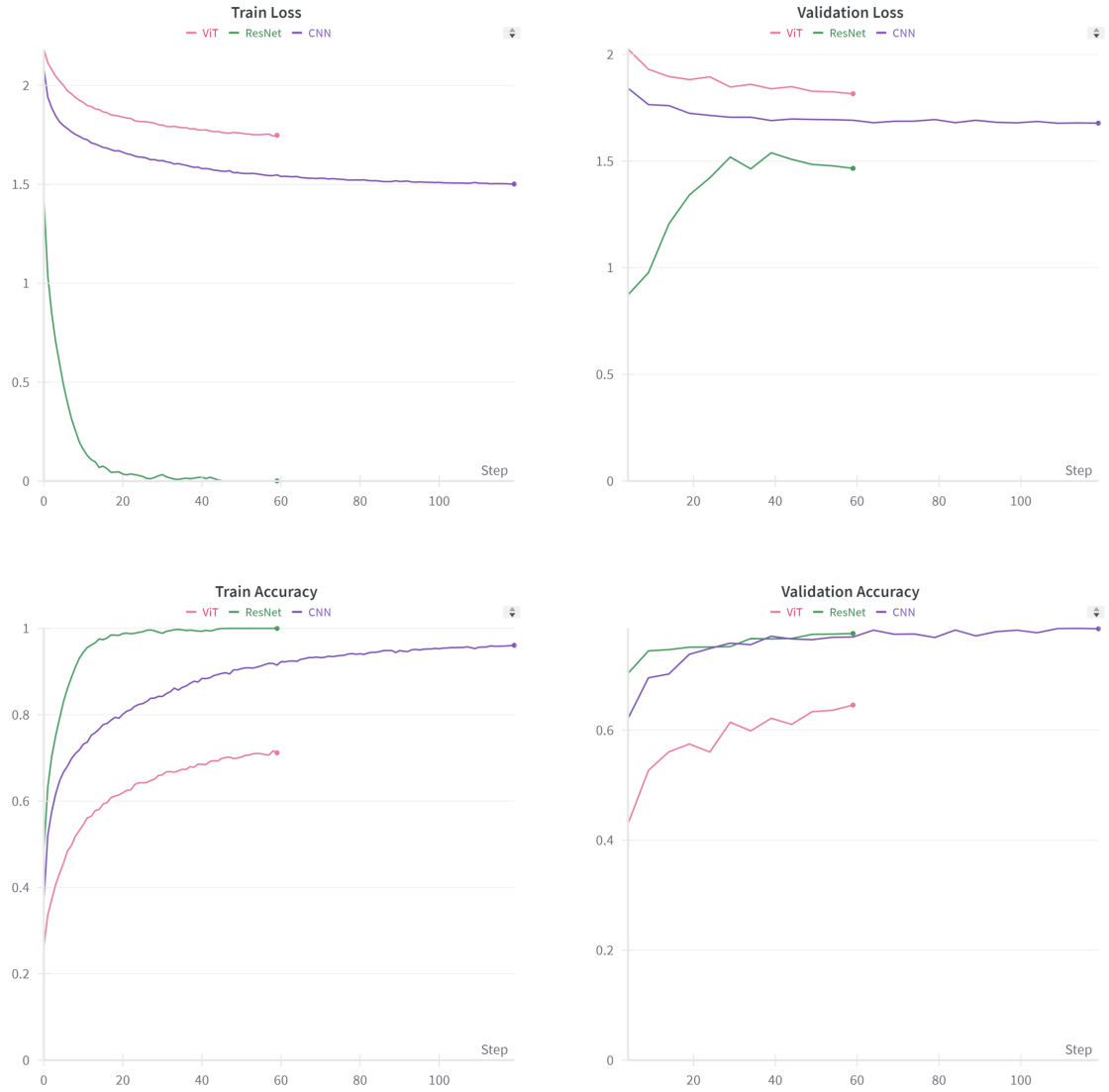

Figure 11: Comparison of training metrics

We notice that the ResNet model seems to overfit the data quite fast. This can also be seen with the validation loss compared to the training loss. The ViT seems to converge slower than the other models.

7 Discussion and Conclusions

In conclusion, the tuning of hyperparameters and the application of data augmentation strategies played a significant role in optimizing our models. The careful adjustment of learning rates, batch sizes, and other key parameters, along with the use of techniques data augmentation, contributed to enhancing the model's robustness and accuracy. Our CNN performed well, achieving results comparable to a ResNet on the test set. The ViT has many different hyperparameters to select, and the training process seemed to take longer for a similar performance, if this is even possible. We achieved good performance with all three models.

References

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... Houlsby,

N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*.