Basic Of Deep Learning - Final Project

Ori Malca (ID. 315150599), Kfir Sitalkil (ID. 208722660)

Submitted as final project report for Basic Of Deep Learning course, Colman, 2023

1 Introduction

Our project addresses a multi-class classification problem involving 196 different types of cars from the "cars196" dataset. We plan to solve this problem using three different approaches: Transfer Learning, Image Retrieval, and End-to-End CNN.

1.1 Data

We received a labeled dataset consisting of 16,185 images of different car types, with varying sizes. The dataset contains 196 types of cars. The initial data split was evenly divided between training and testing, but we decided to adjust the split to 80 percent for training (with 20 percent reserved for validation) and 20 percent for testing. Our rationale for this split was that 20 percent of the data would be sufficient for testing purposes.

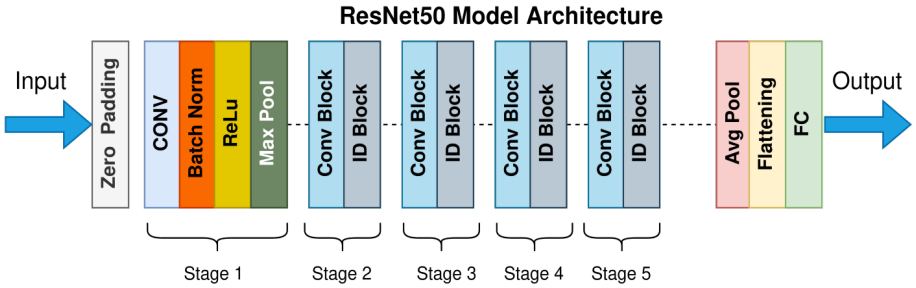
1.2 Problem

Our project involves a multi-class classification problem among 196 different types of cars in the "cars196" dataset. We plan to solve this problem using three approaches: Transfer Learning, Image Retrieval, and End-to-End CNN.

2 Transfer Learning Task

2.1 First Experiment

For our first experiment model we followed TensorFlow/Keras API guide on transfer learning. We decided to use ResNet50 weights (without the fully connected layers on top).



We added a flatten layer (embedding layer) and a dense layer with 196 neurons and a softmax activation. As for our optimizer we used SGD with momentum as the optimizer, setting the learning rate to 0.0001 and the momentum hyperparameter (beta) value to 0.9, as suggested in the paper. We fine-tuned the pre-trained ResNet50 model/weights, with the aim of adjusting the feature extraction at the convolutional part of the network to our specific cars classification problem. Additionally, we augmented the data to increase the variety of images, improve model robustness, and reduce overfitting.
We standardized the data by applying the pre-processing layer of ResNet50 in the Keras documentation, which involves zero-centering the data to have a mean of 0 and a variance of 1. This was done to prevent an elongated cost function and to achieve a more symmetric cost function, which speeds up the learning process. As Andrew Ng explains here, a more symmetric cost function is easier for the loss function to converge to. Additionally, standardizing the data makes the network more robust to changes in the data distribution (makes the network more stable to **covariate shifts**).

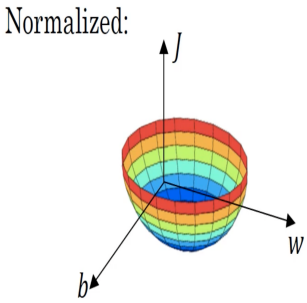


Figure 1: Loss function when normalizing the data

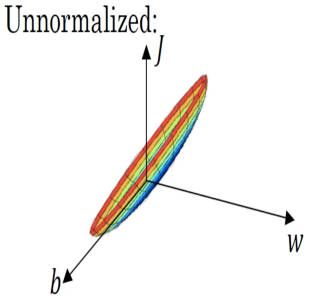


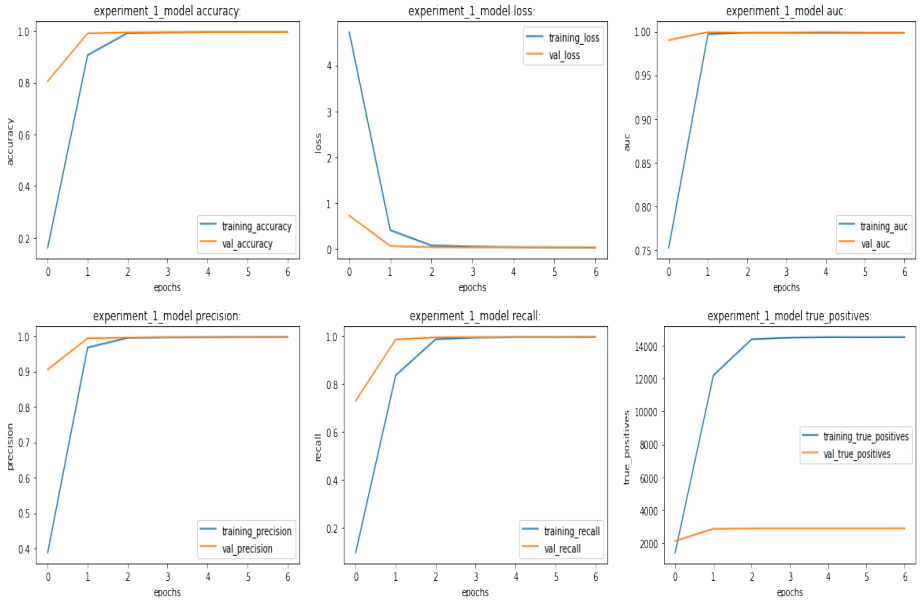
Figure 2: Loss function when **not** normalizing the data

We decided to use a batch size of 32 for two main reasons.
First, a larger batch size helps to reduce the impact of outliers on the training process.

Second, a small batch size adds a regularization effect when using batch normalization. Since batch normalization calculates the mean and variance of the **current** mini-batch, the smaller the batch size, the noisier the calculated mean and variance will be, and thus we get a gentle element of regularization when using batch normalization with a small batch size because we are subtracting off a **noisy** mean from the neuron value and dividing the neuron value by a **noisy** standard deviation (this is described in the above and linked video of Andrew Ng. about batch normalization).
We chose a batch size that is not too small (to reduce the impact of outliers on the training process) but also not too large (to retain the regularization effect). A smaller batch size would likely increase the regularization effect, but we are already using dropout for regularization, so we thought a batch size of 32 would be sufficient. Note that we are referring to the batch normalization layers in the pre-trained ResNet50 base model that we used for the transfer learning, but this regularization technique also applies to the fully connected part on top of the pre-trained ResNet50 model in our next experiment.

We chose to use categorical cross-entropy as our loss function because it is a convex function, which facilitates convergence to a global minimum and helps to prevent the model from getting stuck in local minimum during training.
We trained the model for 10 epochs with early stopping.

The metrics for the first experiment:



Loss	Accuracy	AUC	Precision	Recall
2.2895	0.4688	0.9257	0.6979	0.3767

F1 Score	TPR	TNR	FPR	FNR
0.4893	0.3767	0.9991	0.0008	0.6232

2.2 Second Experiment

For our second experiment, we built upon the first model by making some modifications. Specifically, we replaced the flatten layer with a global average pooling 2D layer. Additionally, we added a sequence of layers twice, consisting of a dense layer with 4096 neurons, a leaky ReLU activation, batch normalization, and dropout.

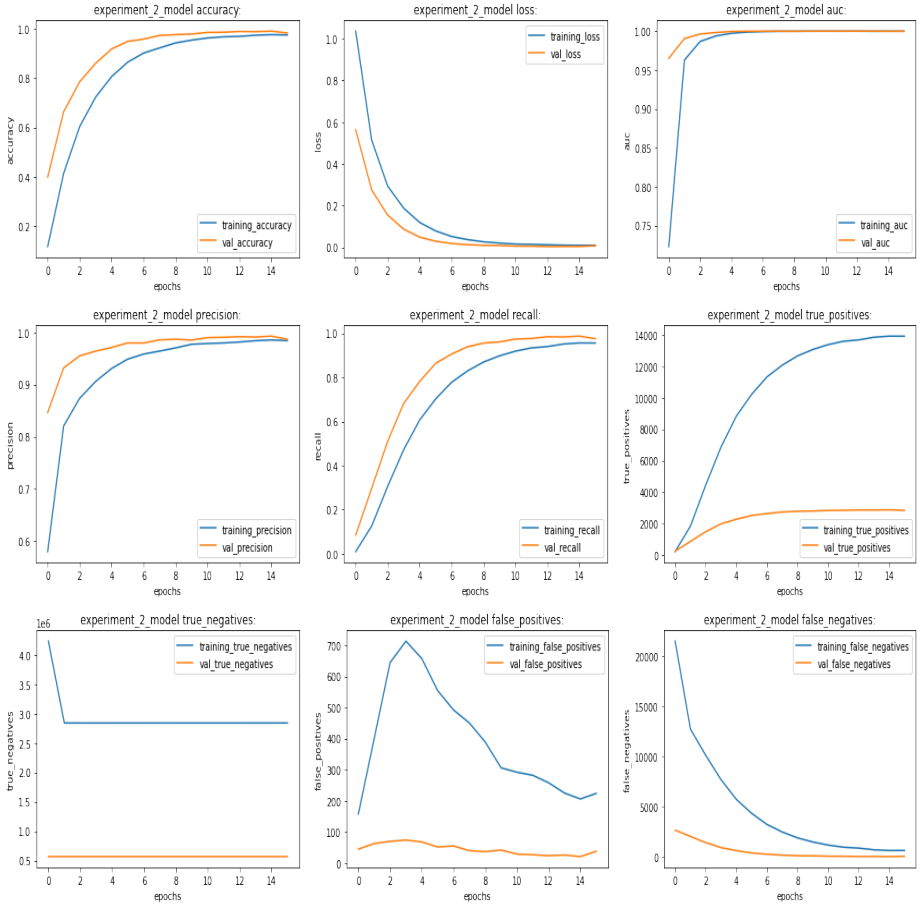
The reason for adding the dropout layer was to prevent the network from relying too heavily on the same set of neurons during training, which can hinder the learning process of other neurons. Additionally, the use of dropout introduces an element of ensemble learning, as described in the paper, because the architecture of the network changes from iteration to iteration due to some neurons being "shut down", resulting in a different architecture each time.

The motivation for incorporating a batch normalization layer is to alleviate the issue of **covariate shift** between layers, thus ensuring that each layer is not affected by this shift. The purpose is to enhance the resilience of each layer to variations and promote **independence**, as Andrew Ng discusses in this video. We chose not to include a bias vector at the dense layer in our model due to the use of batch normalization. As Andrew Ng describes here, the bias term is eliminated by the calculated mini-batch mean and can be replaced by the beta hyperparameter of the batch normalization. Hence, including a bias vector is redundant and not necessary in this case (a more detailed explanation of how the bias term is cancelled out by the calculated mini-batch mean can be found here).

To address the issue of the dying ReLU problem and prevent vanishing gradients, we utilized a leaky ReLU activation layer in our model. Leaky ReLU is a variant of the ReLU activation function that introduces a small slope to negative inputs, thereby preventing the neuron from becoming inactive.

We utilized a categorical cross-entropy loss function in our model because it is a convex function that enables the loss to converge to a global minimum and avoid local minima. Additionally, we incorporated the focal loss variant to address the issue of class imbalance and concentrate on hard examples. Focal loss assigns more weight to misclassified examples, enabling the network to focus on challenging samples and improve its accuracy.
We selected the Adam optimizer for our model, which is a combination of SGD with momentum and RMSProp. Adam combines the exponential moving average update technique of the gradients used in SGD with momentum and RMSProp (as explained in this source).
In this experiment, we chose to use a batch size of 32 for the same reasons mentioned in the first experiment section.
We trained the model for 50 epochs with early stopping.

The metrics for the second experiment:



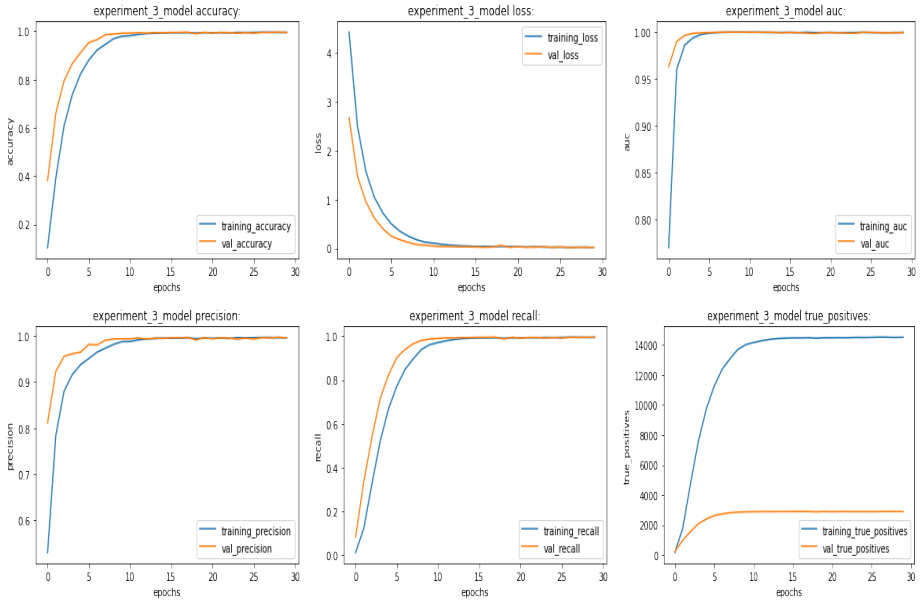
Loss	Accuracy	AUC	Precision	Recall
0.138	0.7887	0.9919	0.8911	0.7029

F1 Score	TPR	TNR	FPR	FNR
0.7859	0.7029	0.9995	0.0004	0.2971

2.3 Third Experiment

In the third experiment, we utilized the same architecture as the second experiment, with the only difference being the use of a standard categorical cross-entropy loss function instead of the focal loss. Interestingly, we observed that this modification led to an improvement in our model's performance.
We trained the model for 50 epochs with a batch size of 32 (for the same reasons mentioned in the first experiment) and early stopping.

The metrics for the third experiment:



Loss	Accuracy	AUC	Precision	Recall
0.6296	0.8363	0.9813	0.8787	0.8054

F1 Score	TPR	TNR	FPR	FNR
0.8404	0.8054	0.9994	0.0005	0.1945

3 Image Retrieval Task

For the image retrieval task, we have chosen to utilize our best transfer learning model without the fully connected layer on top as our embedding model. This will allow us to generate embeddings for each sample/image, which can then be fed into a KNN classifier.

3.1 First Experiment

For the first experiment, we selected a hyperparameter value of K=2 for our K-Nearest Neighbors (KNN) model.

The performance evaluation metrics of the first experiment:

Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
0.7566	0.7526	0.8943	0.7566	0.8011

3.2 Second Experiment

For the second experiment, we selected a hyperparameter value of K=4 for our K-Nearest Neighbors (KNN) model.

The performance evaluation metrics of the second experiment:

Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
0.7535	0.7526	0.9089	0.7535	0.8072

3.3 Third Experiment

For the third experiment, we selected a hyperparameter value of K=3 for our K-Nearest Neighbors (KNN) model.

The performance evaluation metrics of the third experiment:

Accuracy	Balanced Accuracy	Precision	Recall	F1 Score
0.8035	0.8043	0.8524	0.8035	0.8125

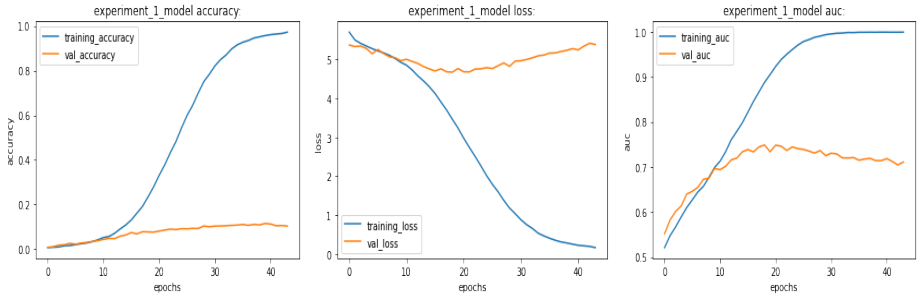
4 End-To-End CNN Learning Task

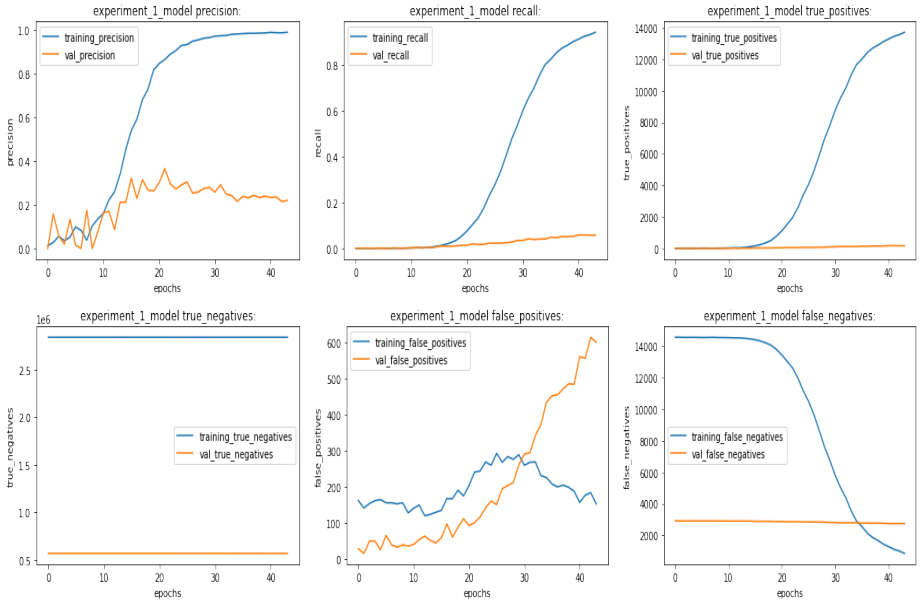
To address the End-To-End CNN task, we chose to develop a widely recognized artificial neural network (ANN) architecture from scratch (such as ResNet50) or adapt a widely recognized architecture by incorporating a few custom changes of our own.

4.1 First Experiment

In the first experiment, we constructed ResNet50 **from scratch** and added a global average pooling 2D layer, along with the same fully connected layers used in the third experiment of the transfer learning task. Our goal was to apply the same reasoning and intuition as in the third experiment of the transfer learning task.
We applied data augmentation techniques and utilized the built-in data standardization function of ResNet50 provided by Keras documentation.
We trained the model for 100 epochs with a batch size of 32 and early stopping.

The metrics for the first experiment:



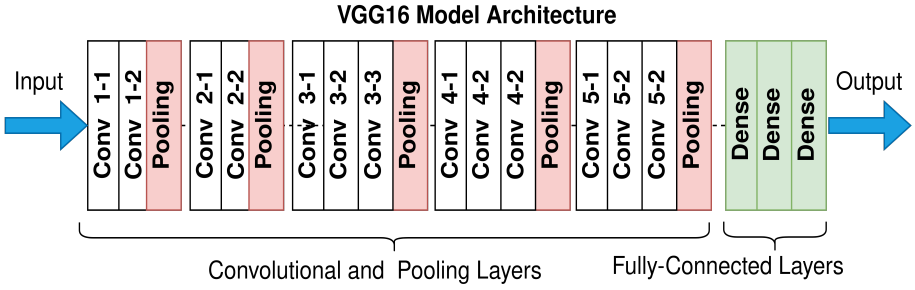


Loss	Accuracy	AUC	Precision	Recall
6.6786	03034	0.607	0.0815	0.016

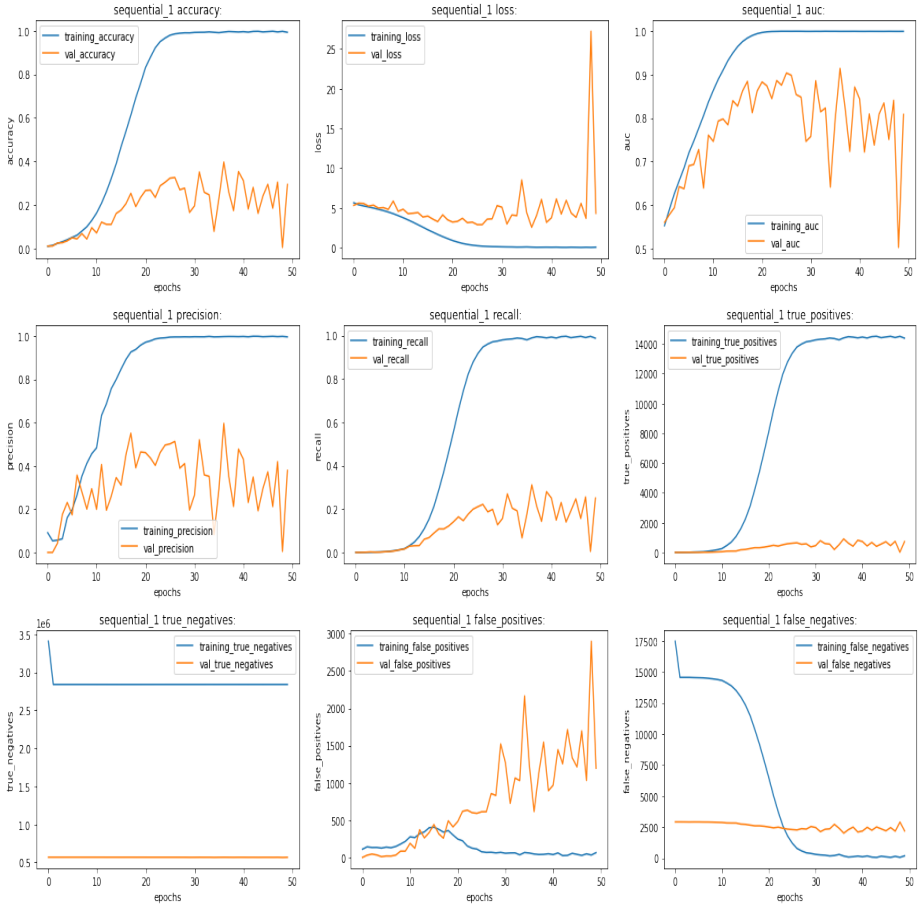
F1 Score	TPR	TNR	FPR	FNR
0.0268	0.016	0.999	0.0009	0.9839

4.2 Second Experiment

In the second experiment, we chose to construct VGG16 **from scratch** and incorporated the same fully connected layers (on top) as used in the first experiment.
We applied data augmentation techniques and utilized the built-in data standardization function of VGG16 provided by Keras documentation.
We trained the model for 50 epochs with a batch size of 32 and early stopping.



The metrics for the second experiment:



Loss	Accuracy	AUC	Precision	Recall
6.401	0.1537	0.7003	0.2013	0.1272

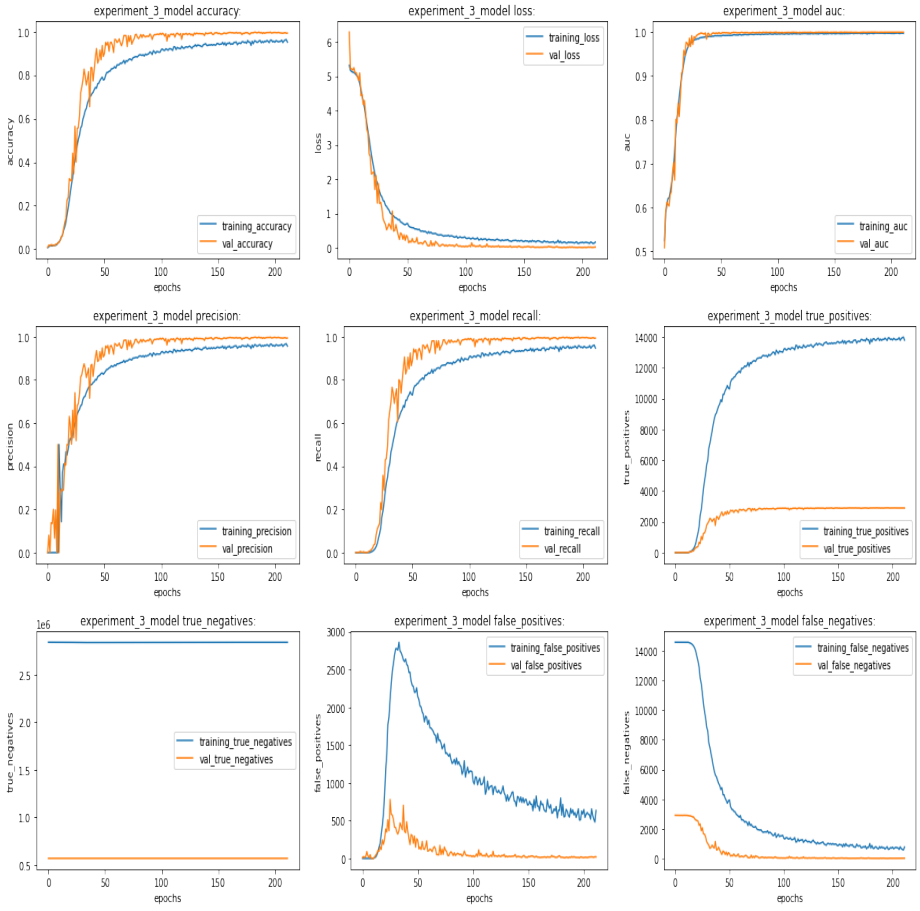
F1 Score	TPR	TNR	FPR	FNR
0.1559	0.1272	0.9974	0.0025	0.8727

4.3 Third Experiment

In our third experiment, we created a custom artificial neural network (ANN) that drew inspiration from other popular models. For the convolutional portion of the network, our aim was to increase the depth dimension of the tensor while reducing the vertical and horizontal dimensions. We then flattened the tensor using a flatten layer, which is a common technique used in popular ANN models

such as AlexNet.
For the fully connected part of the network, we followed the same approach as in our previous experiments, incorporating dropout, batch normalization, and Leaky ReLU activation. However, in this experiment, we also focused on decreasing the dimension size of each layer until reaching the output layer, similar to many other well-known ANN models.
The architecture for our model is available in the notebook (a detailed description of the architecture can be found in the "Presentation of the Architecture" section of the experiment).
We trained the model for 250 epochs with a batch size of 32 and early stopping.

The metrics for the third experiment:



Loss	Accuracy	AUC	Precision	Recall
7.0711	0.2452	0.7351	0.2971	0.2279

F1 Score	TPR	TNR	FPR	FNR
0.2579	0.2279	0.9972	0.0027	0.7720

5 Platform and Tools

We carried out our experiments on the Google Colaboratory platform. NumPy library was used for metric calculations, scikit-learn library was used to construct the KNN model for the image retrieval task and calculate metrics on the test set, and Matplotlib library was used for visualization of different metrics. TensorFlow/Keras was utilized for constructing the models in the Transfer Learning and End-To-End CNN tasks.

6 Discussion

This project was an enjoyable and satisfying experience. We were able to tackle a challenging multi-class classification problem involving 196 different car types using three different approaches: Transfer Learning, Image Retrieval, and End-To-End CNN. It was exciting to see how each approach performed and to compare the results.

7 Code

To access the notebook click here