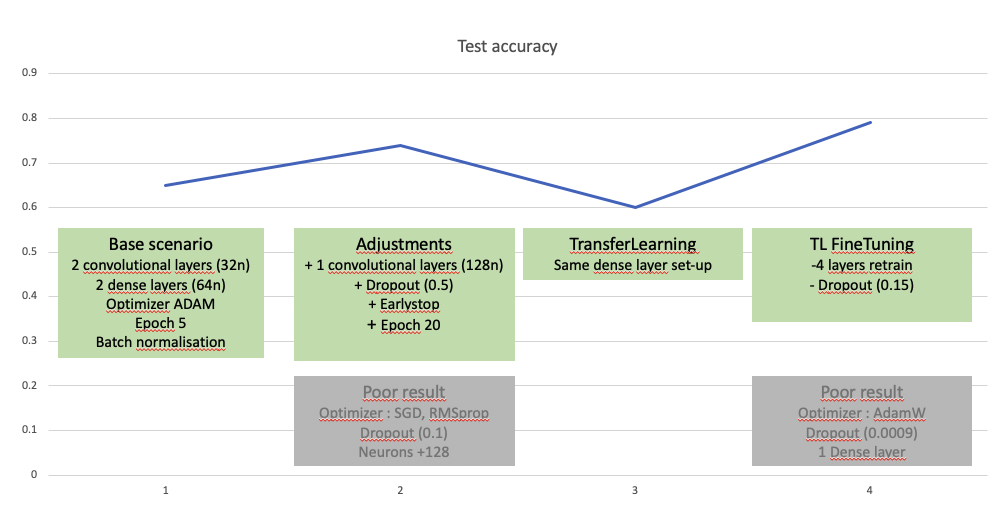
# IronHack project 1: Deep learning image classification with CNN

Objective of this project is to build a convolutional neural network model, classify images and predict new images against the trained model.

Final result can be accessed below

Access the page from your home: <https://af2f-87-209-237-170.ngrok-free.app/>

Github: <https://github.com/paulvble/project-1-deep-learning-image-classification-with-cnn.git>



# Project report

## Dataset

The data used for this project is the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. (<https://www.cs.toronto.edu/~kriz/cifar.html>)

## Tools

* Python
* Visual Studio Code, Google Colab, NGROK
* Libraries: Numpy, Scikit-Learn, Matplotlib, Tensorflow, Keras

# Data exploration

## Data exploration

* Total images = 60.000 (32p x 32p)
  + Training set = 50.000
  + Test set = 10.000
* 10 elements = 10 classes
* Images already splitted for training and test, so no train\_test\_split function required

|  |  |
| --- | --- |
| # Check images and their shapes  print(f"Training data shape: {train\_images.shape}")  print(f"Training labels shape: {train\_labels.shape}")  print(f"Testing data shape: {test\_images.shape}")  print(f"Testing labels shape: {test\_labels.shape}")  # Check the unique classes and their counts  unique\_classes, class\_counts = np.unique(train\_labels, return\_counts=True)  print(f"Classes: {unique\_classes}")  print(f"Class counts: {class\_counts}") | Training data shape: (50000, 32, 32, 3)  Training labels shape: (50000, 1)  Testing data shape: (10000, 32, 32, 3)  Testing labels shape: (10000, 1)  Classes: [0 1 2 3 4 5 6 7 8 9]  Class counts: [5000 5000 5000 5000 5000 5000 5000 5000 5000 5000] |

# Model selection

3 approaches were taken into account

* CNN training from the start
* CNN with transfer learning
* CNN with transfer learning + finetuning

## CNN training from the start

Base attempt & running locally

* 2 convolutional layers (1 input, 1 hidden)
  + Conv2D: we are working with images where the input and the following spatial hierarchies is required to understand the patterns in the images. Leveraging the kernels/filters reduces complexity and makes it work train for efficiently.
* 2 dense layers
  + Started of with 128 to absorb image complexity in general. Also because the images appeared to be small and vague.
* Activation layers
  + Selected ReLU to have non-linearity to the output. Did not go for sigmoid, because that’s mainly usefull for binary classifications.
  + Softmax used in the output layer to be able to define the probability scores of the classification.
  + Batch normalization, to enhance efficiency between the layers.
* Epochs: started of with 5 for the speed to test the code
* Optimizer: selected ADAM. Shortly tested with SGD, but that didn’t go anywhere.

|  |
| --- |
| def build\_cnn\_model():  cnn\_model = models.Sequential([  # Define convolutional layer nbr.1  layers.Conv2D(filters=32, kernel\_size=(3, 3), activation='relu', input\_shape=(32, 32, 3)),  layers.MaxPool2D(pool\_size=(2, 2)),  layers.BatchNormalization(),  # Define convolutional layer nbr.2  layers.Conv2D(filters=64, kernel\_size=(3, 3), activation='relu'),  layers.MaxPool2D(pool\_size=(2, 2)),  layers.BatchNormalization(),  # Flatten and Dense layers  layers.Flatten(),  layers.Dense(128, activation='relu'),  layers.Dense(10, activation='softmax')  ])    return cnn\_model  cnn\_model = build\_cnn\_model()  # Print the summary of the layers in the model.  print(cnn\_model.summary())  optimizer = tf.keras.optimizers.Adam()  cnn\_model.compile(optimizer=optimizer,  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy']) |

Accuracy: 0.6527

Precision: 0.678541004259845

Recall: 0.6527

F1 Score: 0.6541426761277805

Classification Report:

precision recall f1-score support

airplane 0.64 0.78 0.70 1000

automobile 0.75 0.85 0.80 1000

bird 0.41 0.73 0.53 1000

cat 0.52 0.46 0.49 1000

deer 0.67 0.50 0.58 1000

dog 0.62 0.47 0.54 1000

frog 0.71 0.77 0.74 1000

horse 0.81 0.65 0.72 1000

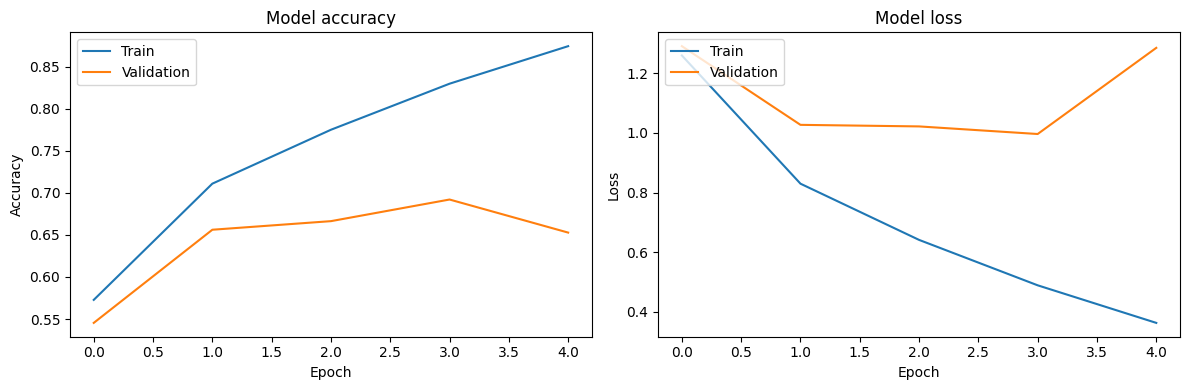
ship 0.77 0.74 0.75 1000

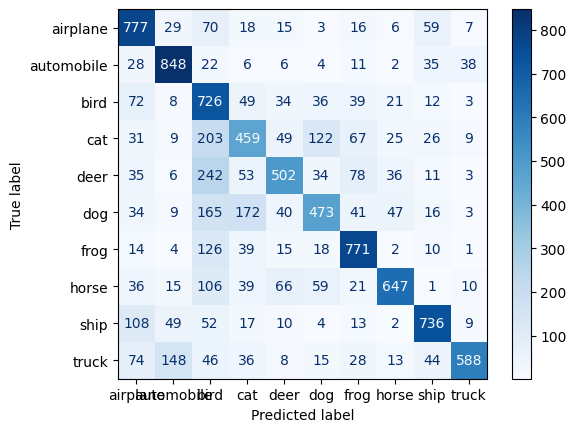
truck 0.88 0.59 0.70 1000

accuracy 0.65 10000

macro avg 0.68 0.65 0.65 10000

weighted avg 0.68 0.65 0.65 10000





Conclusion: starting point to tackle

* The model is overfitting where the training loss is much lower compared to the test.
* Low accuracy on the test set

Options explored

* Adding convolutional layer -> to address the model memorizing noise data since it’s blurry and small.
* Adding dropout -> randomly remove neurons to hopefully remove unnecessary features
* Early stopping -> stop the learning with cycle to prevent the model learning things that do not matter.
* Batch normalization -> improves the learning efficiency of the model
* Adjustment of the learning rate -> decrease the rate so it does not try to learn everything.
* Data augmentation -> increase the data set

Result of the options explored

Increased the accuracy rate, and reduced the overfitting. Though not there yet.

Accuracy: 0.7446

Precision: 0.7517972128563107

Recall: 0.7446

F1 Score: 0.7467254999132825

Classification Report:

precision recall f1-score support

airplane 0.72 0.83 0.77 1000

automobile 0.91 0.79 0.84 1000

bird 0.66 0.65 0.65 1000

cat 0.56 0.60 0.58 1000

deer 0.69 0.76 0.72 1000

dog 0.63 0.66 0.64 1000

frog 0.83 0.80 0.81 1000

horse 0.81 0.76 0.79 1000

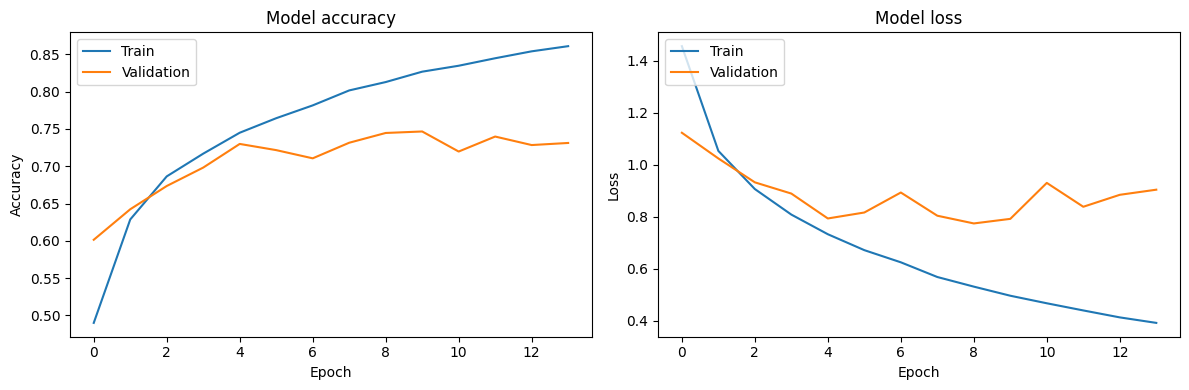
ship 0.85 0.81 0.83 1000

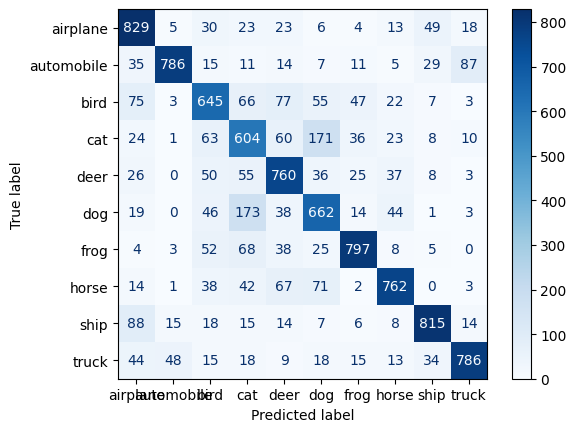
truck 0.85 0.79 0.82 1000

accuracy 0.74 10000

macro avg 0.75 0.74 0.75 10000

weighted avg 0.75 0.74 0.75 10000





## Transfer learning with VGG16

Explore VGG16 and Inception

One of my limitation was the processing power. Since VGG16 was relatively simple in set-up and more efficient than others, I choose to go for VGG16. In articles it was mentioned that it was prone for overfitting with smaller datasets, so that was something to take into account going forward. I need to control this factor in the finetuning.

Potential action: open up part of the layer to retrain with the CIFAR dataset.

Result

Accuracy: 0.6096

Precision: 0.6099438891771447

Recall: 0.6096

F1 Score: 0.6065246061066646

Classification Report:

precision recall f1-score support

airplane 0.77 0.60 0.68 1000

automobile 0.62 0.70 0.66 1000

bird 0.56 0.50 0.53 1000

cat 0.48 0.40 0.44 1000

deer 0.58 0.52 0.55 1000

dog 0.54 0.54 0.54 1000

frog 0.63 0.69 0.66 1000

horse 0.69 0.66 0.67 1000

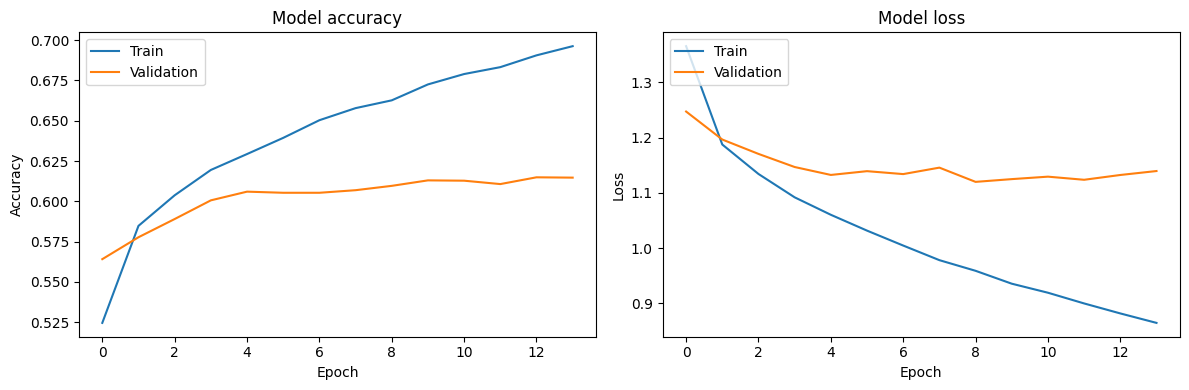
ship 0.66 0.80 0.72 1000

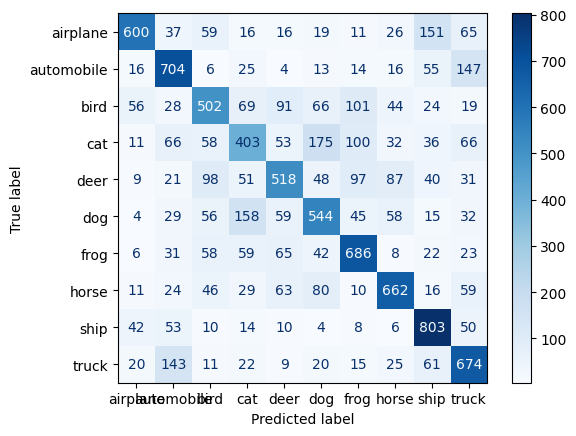
truck 0.58 0.67 0.62 1000

accuracy 0.61 10000

macro avg 0.61 0.61 0.61 10000

weighted avg 0.61 0.61 0.61 10000





## CNN training with transfer learning VGG16 + finetuning

* Re-train weights of the last 4 layers of the base VGG16
* Adjusted learning rate 20 -> 15%

Result

Test Accuracy: 0.7979000210762024

Test Precision: 0.8047041390003009

Test Recall: 0.7979

Test F1 Score: 0.7986512694685315

Classification Report:

precision recall f1-score support

0 0.74 0.90 0.81 1000

1 0.91 0.90 0.91 1000

2 0.78 0.67 0.72 1000

3 0.56 0.69 0.62 1000

4 0.80 0.72 0.76 1000

5 0.79 0.67 0.72 1000

6 0.82 0.85 0.84 1000

7 0.86 0.85 0.86 1000

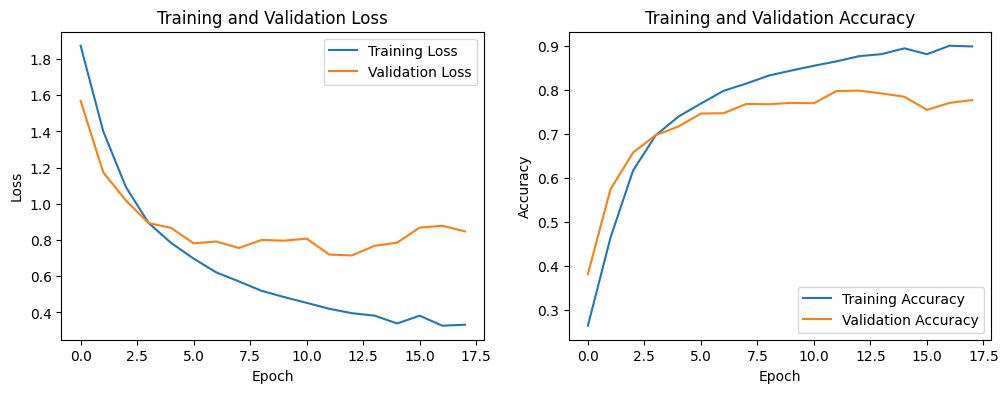
8 0.89 0.85 0.87 1000

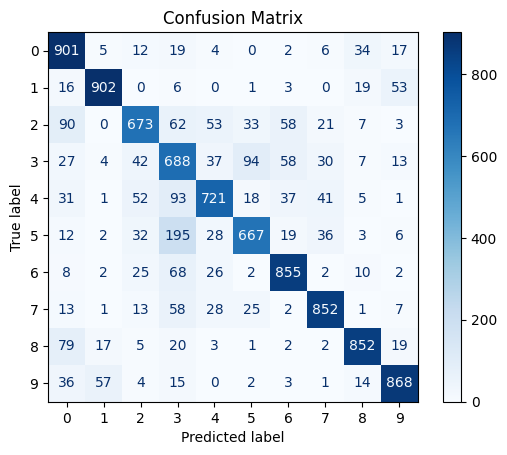
9 0.88 0.87 0.87 1000

accuracy 0.80 10000

macro avg 0.80 0.80 0.80 10000

weighted avg 0.80 0.80 0.80 10000





* Adjusted the optimizer to AdamW. Apparently this helps preventing overfitting by penelizing large parameter values therefore improving generalization performance. However the results did not increase for this dataset.
* Test Accuracy: 0.7703999876976013
* Test Precision: 0.7813474139705266
* Test Recall: 0.7704
* Test F1 Score: 0.7714482591095648



## Deployment

* From Google Colab; generated a H5 file stored in google drive.
* Set-up the environment locally by using Flask
* Leverage NGROK to tunnel the application onto the internet. So the application runs locally from the laptop.