# Narrative Machine Learning Technical Test

Dhvani Shah

September 2023

## 1 Introduction

### 1.1 CIFAR-10 Photo Classification Dataset

CIFAR is an acronym that stands for the Canadian Institute For Advanced Research and the CIFAR-10 dataset [Kri17] was developed along with the CIFAR-100 dataset by researchers at the CIFAR institute.

The dataset comprises 60,000 32×32 pixel color photographs of objects from 10 classes, such as frogs, birds, cats, ships, etc. The class labels and their standard associated integer values are listed below.

0: airplane
1: automobile
2: bird
3: cat
4: deer
5: dog
6: frog
7: horse
8: ship
9: truck

The training set contains 5,000 samples from each class, totaling 50,000 samples. Since the dataset is labeled and each class contains an equal number of samples, the dataset is balanced and is considered a supervised classification problem. These are very small images, much smaller than a typical photograph, and the dataset was intended for computer vision research.

The state-of-the-art architectures can be found here [Res].

## 2 Level 1

1. A python script(s) which trains a model from scratch and saves it to a file: Link to the repo: Train Script

2. A python script(s) which tests the generated model: Test Script

3. Accompanying documentation on how to run both scripts: ReadMe

4. A document that describes the chosen model and structure and the result of the model, i.e., several measures of accuracy and performance: I have implemented three blocks of VGG architecture [SZ14] and the model summary, results without hyperparameter tuning could be found here: Results

## 2.1 Performance on Validation and Test Sets

## 2.2 Model Quality on Validation set

Out of 50,000 train samples, 10,000 samples were included in the validation set.

In this case 1, we can see that the model rapidly overfits the test dataset. This is clear in the loss plot (top plot). We can see that the model's performance on the training dataset (blue) continues to improve, whereas the performance on the validation dataset (orange) doesn't improve.

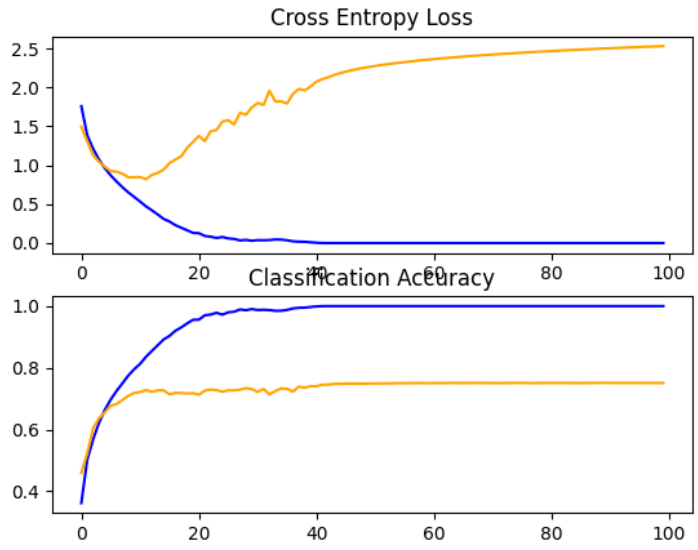


Figure 1: Baseline Model

In this case 2, we can see that the model rapidly overfits the test dataset. This is clear if we look at the plot of loss (top plot); we can see that the model's performance on the training dataset (blue) is low, but the perfor-

mance on the validation dataset (orange) has increased. Most importantly, the model shows less overfitting.

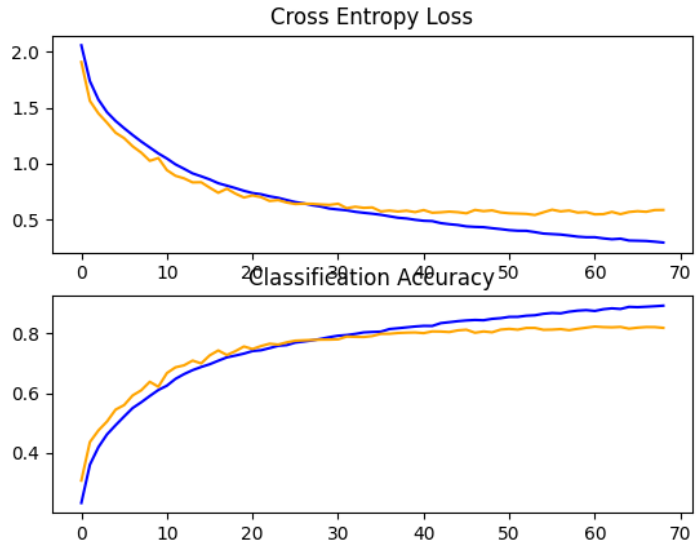


Figure 2: Model with dropout

### 2.3 Model Performance on Test set

Table 1: Performance Evaluation on Test Samples

| ***Sr No.*** | ***Model*** | ***Parameters*** | ***Performance (Tesy Accuracy)*** |
|---|---|---|---|
| 1 | Baseline Model | 73.530 | $100 epochs, SimpleCNN$ |
| 2 | Model with Dropout | 80.7 | $100 epochs, Droput, ReduceLronPlateau, EarlyStopping$ |
| 2 | Model with Dropout | 82 | $100 epochs, Droput, ReduceLronPlateau, EarlyStopping$ |

### 2.4 Level 2

1. Optimise the model to achieve high accuracy with a small model. The optimised model must contain less than 1 million parameters:

   To perform hyperparameter tuning, the following hyperparameters can be tuned:

   (a) the dropout rate for different dropout layers

   (b) the number of filters for the convolutional layers

(c) the number of units for the dense layer

(d) its activation function

(e) the kernel size

(f) the learning rate

For this problem, considering time constraints, I finally decided to optimise learning and dropout rates with few epochs and fewer execution trials.

The hyperparameter tuning code can be found here: Hyperparamter Script. Please scroll to the end of the notebook since I have executed different versions (considering more hyperparameters but stopped them since they took a lot of processing time) to reduce the training time.

2. Expand on the details of the model performance with a confusion matrix and breakdown accuracy for each label/class : The confusion matrix and performance metrics for each class can be found here: Results. This is the result with 3 blocks of VGG network, dropout, early stopping, and reduced learning rate when on plateau. I have not retrained the model with the found hyperparameters at this stage.

3. Measure the inference time of the model in milliseconds per image per CPU: I used the following code to capture the inference time per CPU thread:

```
import os
import tensorflow as tf
tf.config.threading.set_inter_op_parallelism_threads(1)
tf.config.threading.set_intra_op_parallelism_threads(1)
os.environ["OMP_NUM_THREADS"] = "1"
```

Inference time per CPU thread per sample: 1.3383677005767822 ms.

Inference time per CPU thread for 10000 test samples: 13.383677005767822 s.

4. Measure the number of parameters of the model Total params: 550570 (2.10 MB)
Trainable params: 550570 (2.10 MB)
Non-trainable params: 0 (0.00 Byte)

# 3 Level 3

1. Show your work: Document your thinking process, findings, and experiments.

### 3.0.1 Thinking Process

When I received the test, I first checked out the state-of-the-architectures on the cifar10 dataset [Res]. Then, considering the given time frame, the constraints that the model should have less than 1M parameters and no transfer learning framework should be utilised, I started to research the initial research done for image classification. I came across a medium article that stated that the VGG networks from Oxford were the first to use much smaller 3×3 filters in each convolutional layers and also combined them as a sequence of convolutions [Cul17]. The same article also mentioned the downside of this network. VGG used large feature sizes in many layers, and thus, inference was quite costly at run-time. Reducing the number of features, as done in Inception bottlenecks, will save some of the computational cost. So, I implemented just three blocks of VGG architecture for simplicity and faster execution.

After implementing the baseline model and model with dropout, as mentioned above, I started looking for ways to reduce the model size further and maintain accuracy. At this time, I came across the concept of the DepthwiseConv2D layer from Keras documentation and MobileNet architecture. MobileNet is a streamlined architecture that uses depthwise separable convolutions to construct lightweight deep convolutional neural networks and provides an efficient model for mobile and embedded vision applications [Wan+20]. So, I decided to implement DepthwiseConv2D with CNN nets. This experiment can be found here: DepthwiseConvolutionsCNN Script.

I achieved an accuracy of around 69%. Compared with the baseline model with an accuracy of around 73% and less than 1M parameters, I decided to proceed with VGG architecture.

At this point, I investigated modifications to the model and the training algorithm that seek to improve. Top techniques that could reduce overfitting of the model include regularisation techniques, weight decay, reducing learning when reached on the plateau, and data augmentation, amongst others. After the baseline model, I only considered dropout, early stopping, and ReduceLROnPlateau to increase model performance. The performance increased from 73% to 80%. This further built my confidence in this VGG architecture.

### 3.0.2 Experiments and Tools

For this technical test of cifar10 dataset classification, the following experiments are conducted:

(a) 3 Block VGG architecture

(b) 3 blocks VGG architecture dropout, early stopping, and ReduceLROnPlateau

(c) CNN with DepthwiseConv2D

(d) Hyperparameter tuning using the Hyperband from Keras Tuner - Number of filters, learning rate, dropout.

(e) Hyperparameter tuning using the Hyperband from Keras Tuner - learning rate, dropout.

(f) Hyperparameter tuning using the Hyperband from Keras Tuner learning rate, dropout for only one layer and with fewer epochs. The results can be found here: HyperparameterTuning results.

(g) Building the entire pipeline - train and test

To note, I interrupted the first two experiments of the hyperparameter since it was taking a lot of time.

Tools used: Python 3.10, Tensorflow 2, Keras Tuner, Sklearn Pipeline, Matplotlib, Visual Studio code, Jupyter Notebook, Latex (Overleaf), Github and Github Desktop I ran these experiments on the MAC M2 machine.

### 3.0.3 Findings

Considering the confusion matrix and the classification report as described in Results, it can be observed that the model is more accurate in identifying vehicular objects such as airplanes, automobiles, ships, and trucks except for horses. One reason for this is the sharp edges of these objects. Consider the raw images shown in Fig 3; the vehicular objects are easier to differentiate from their background than the animals because of their sharp boundaries and contrast. The color contrast in the case of horses is also very good, and the boundary between the object and background is clear, resulting in better accuracy of horses as well.

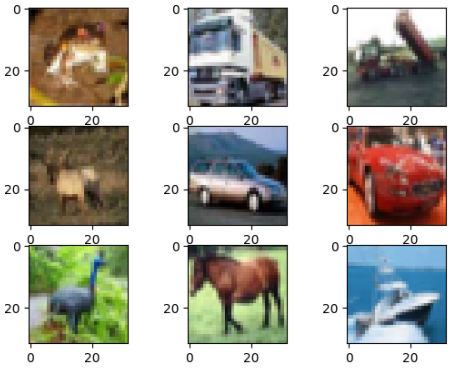

Figure 3: Class Samples

2. Imagine if you had 3 months to work on this model: Describe areas of future research, techniques you might implement, experiments you might run, and specific areas where you believe the model could be improved.

   (a) First, I would complete the hyperparameter tuning with all the required parameter settings with the aim to achieve above 90% accuracy as most of the state-of-the-art architectures have reached so far (maximum being 99.6%).

   (b) Currently, the model contains 550570 total params, which is far less than 1M parameters. So, I will add more blocks of VGG or use Inception nets, ResNet, and EfficientNet

   (c) The size of the images in this dataset is very small and hence could be one of the reasons for low performance. I will resize the image to increase size and choose a neural network architecture accordingly.

   (d) Most importantly, I will use techniques to enhance image quality, GAN [Ahm+22], or research into some image processing techniques. I think this would be crucial as image enhancement is the core of the photography domain.

   (e) From a coding perspective, I have developed the complete pipeline where parameters are read from the configuration file, but I think this can be improved further with the additions of comments, defining the functions with the description of input parameters and type of output, small updates in the code to remove two user warnings.

   (f) Initialisation of seed variable from Tensorflow backend for reproducibility in the future.

   (g) Code update to align the for better visualisation of graphs

3. Reference any academic research that you have consulted when working on this test: All the required citations are provided in the text.

# 4 Requirements checklist

1. The test solution must be entirely your own work.

2. The classifier must be capable of running on a standard, consumer laptop and be CPU driven - Met

3. The inference will also run on the laptop, so size ($\leq$ 1 million parameters), and inference time of the model is important ( $\geq$. 20ms inference time on a single CPU thread) - Met

4. You may use any preferred framework, e.g. PyTorch, TensorFlow, Keras, MXNet, Caffe, Theano, etc. - Tensorflow

5. Use of a pre-trained model or transfer learning is not allowed for this task; the model must be entirely trained by you. - Met

6. You may base the model architecture of your solution on an open-source model, but please reference which model architecture you are using, and document any changes you make on top of the open-source model. - Met

7. You may use ChatGPT to help you with this project, but we would suggest that you do a traditional literature review before consulting ChatGPT, as, for this particular problem, it will limit your thinking and result in a subpar solution - Not Used

8. If you use ChatGPT, please include a transcript of your conversation with the Large Language Model as a text file with your test submission. We want to see how you prompt the AI and how you use it to expand your thinking. - Not Used

9. Structure your solution in a manner that demonstrates how you would approach a problem in a professional context. Show us how you structure your project, code, and other resources - Met