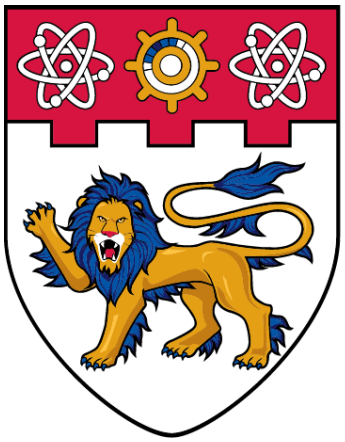


NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

CZ4042 Neural Networks and Deep Learning

Group Project E: Material Recognition

Pamela Sin Hui K1920110A

Teo Boon Shuan U1821709J

1. Introduction

Real-world materials have rich surface texture, geometry, lighting conditions, and clutter, which combine to make the problem difficult. The Materials in Context Database (MINC) was used, where 10 categories of fabric, foliage, glass, leather, metal, paper, plastic, stone, water, and wood were used, each category consisting of 2,500 images. For this project, we performed a series of experiments to determine which preprocessing to use, the best pre-trained featuriser and best set of hyperparameters, further experimenting with fine-tuning through unfreezing final layers of the featuriser.

Problem Statement:

The goal of this project is to train a convolutional neural network to classify color photographs of surfaces into one of 10 common material categories.

In this report, we adopted the following approach to the above problem statement:

We first perform image preprocessing, on two levels:

1. Basic
    a. Horizontal Flip - to obtain more data to train on
    b. Image Normalization - to rescale RGB pixel values from 0-255 to 0-1
2. Advanced
    a. CLAHE (Contrast-Limited Adaptive Histogram Equalization) - to differentiate boundaries and detect object of interest
    b. Otsu's Thresholding - to increase contrast and maintain texture information

We then experimented with the following models:

1. MobileNet
2. InceptionV3

2. Image Augmentation and Preprocessing

2.1 Image Dataset

2.2 Basic Preprocessing

Pre-processing is normally used in conjunction with CNNs (including MobileNet and InceptionV3) to improve the quality of data input. As an initial step, we performed two forms of initial pre-processing, namely: *horizontal flip* and *image normalization*.

2.2.1 Horizontal Flip

A horizontal flip is a data augmentation technique used to synthetically generate additional modified data. This generated data is meant to increase the amount of relevant data, which can directly improve the effectiveness of a CNN. There are two ways of applying horizontal flip: *offline augmentation* and *online augmentation*.

- Offline augmentation on the other hand, involves applying horizontal flip operations on all images in the dataset. This doubles the size of the dataset and can possibly be achieved using the opencv library to perform the augmentation and saving and adding the augmented images to the train dataset folder manually.
- Online augmentation involves applying horizontal flip operations in mini batches which are fed into a CNN. During each epoch, different image variants will be used for model training, and is

usually used for larger datasets to prevent the need for saving augmented images in a storage medium. This may be achieved using the ImageDataGenerator from the keras.preprocessing.images library, to augment the data and feed into the model on-the-fly.

In context to the MINC dataset, we opted for the use of *online augmentation*, where a horizontal flip is performed in mini batches that are fed into our MobileNet and InceptionV3 models. The reason is due to memory limitations of the provided Google Colab compute GPUs.

2.2.2 Image Normalization

Image normalization is a process that changes the range of pixel intensity values in any given image. These pixel intensity values are integers that range from 0 to 255. Unfortunately, large value inputs are not suitable for CNNs as these large values slow down the learning process of a CNN. As such, it is necessary to change the scale of the range to a smaller range, from 0 to 1 to preserve the learning speed of a CNN.

In context to the MINC dataset, the provided images all fit the profile of images with pixel intensity values that range from 0 to 255. As such, we applied image normalization on all the provided images before feeding into our MobileNet and InceptionV3 models for learning.

Using both pre-processing techniques, we managed to train a InceptionV3 model that achieves a score of 0.8594.

We used this score as the baseline metric to evaluate the use of other, more advanced pre-processing techniques. These advanced pre-processing techniques will be discussed in the following sections:

2.3 Advanced Preprocessing

2.3.1 Contrast-Limited Adaptive Histogram Equalization (CLAHE)

Overview

CLAHE is a variant of Adaptive Histogram Equalization (AHE) technique. AHE is an image pre-processing technique used to improve contrast in images. This is achieved through the computation of several histograms, which each correspond to a certain section of an image. These computed histograms are then used to redistribute the brightness values of an image, which improves local image contrast and enhances the definition of edges in an image.

Unfortunately, as there is a tendency for AHE to over-amplify the local image contrast in near-constant areas of the image, noise might be amplified in near-constant regions.

To address this problem of undesired noise amplification in near-constant regions, CLAHE was an alternative technique proposed. It works by limiting contrast amplification through the clipping of the histogram at a predefined value before computing the CDF. This limits the slope of the CDF and the transformation function, which limits undesired noise amplification.

Original Fabric Image

CLAHE Fabric Image

Rationale

The use of CLAHE was necessitated by the need to properly pre-process data for our MobileNet model. In context to the MINC Dataset, we wanted to improve the contrast in the provided images to improve the images fed into our MobileNet model so that defect features could be more accurately detected. An example of this is shown above:

```
--- Starting trial: run-0
{'num_units1': 96, 'num_units2': 64, 'dropout': 0.1, 'optimizer': 'adam'}
Epoch 1/4
782/782 [==============================] - 688s 876ms/step - loss: 2.5176 - accuracy: 0.0992 - top_k_categorical_ac
curacy: 0.4964
Epoch 2/4
782/782 [==============================] - 648s 829ms/step - loss: 2.3089 - accuracy: 0.0976 - top_k_categorical_ac
curacy: 0.4936
Epoch 3/4
782/782 [==============================] - 634s 810ms/step - loss: 2.3144 - accuracy: 0.0948 - top_k_categorical_ac
curacy: 0.4942
Epoch 4/4
782/782 [==============================] - 629s 803ms/step - loss: 2.3028 - accuracy: 0.0974 - top_k_categorical_ac
curacy: 0.4931
```

Accuracy of MobileNet using CLAHE

Outcome

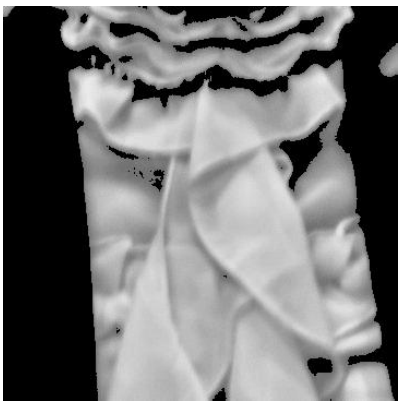
Unfortunately, with the use of CLAHE, we obtained a private score of 0.0974 which turned out to be surprisingly poor. We believe that the sole use of CLAHE may be insufficient before feeding the images to the model. By using a binary CLAHE threshold, as provided by opencv as cv2.THRESH_BINARY, it managed to define contours and contrasts in images to a very high degree. Subsequently, it loses a lot of information of what lies in each contour or boundary, as it was simply depicted as black-and-white. In the nature of detecting materials, this method alone fails to keep information about the texture and gradient of pixel values, which is crucial for its training.

2.3.2 Otsu’s Thresholding

Overview

Otsu's thresholding method proposes iterating through all the credible threshold values and enumerating a measure of spread for the pixel levels on each side of the threshold, i.e. the pixels that are in foreground or background. The intent is to decide the threshold value where the summation of foreground and background escalates is at its minimum.

Original Fabric Image



CLAHE Fabric Image

Rationale

The algorithm returns a single intensity threshold that separates pixels into two classes, foreground and background. As such, it provides a strong reason to attempt this preprocessing method, as we are convinced that after Otsu's method, the object of interest in images may be narrowed down due to differentiating it with its background.

Outcome

With the use of Otsu's thresholding, we obtained a score of 0.6982 which was rather promising. We believe that the performance may be lacking due to the fact that once the thresholding is applied - while this method may have tuned the contrast of the image successfully - the image loses an incredible amount of information in terms of its RGB pixel values as they would be converted into black-and-white images. However, it still underperforms as compared to our baseline MobileNet model of 0.8594, and hence we will not include it in our source code submission.

3. Literature review

3.1 MobileNetV2

The paper "MobileNetV2: Inverted Residuals and Linear Bottlenecks", states that the neural network architecture is specifically tailored for mobile and resource constrained environments, the MobileNetV2 network pushing the state of the art for mobile tailored computer vision models by significantly decreasing the number of operations and memory needed while retaining the same accuracy.

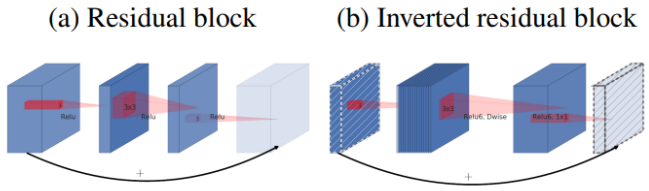
The paper additionally states that the main contribution is a novel layer module: the inverted residual with linear bottleneck. This module takes as an input a low-dimensional compressed representation which is first expanded to high dimension and filtered with a lightweight depthwise convolution. Features are subsequently projected back to a low-dimensional representation with a linear convolution. This convolutional module is particularly suitable for mobile designs, because it allows to significantly reduce the memory footprint needed during inference by never fully materializing large intermediate tensors. This reduces the need for main memory access in many embedded hardware designs that provide small amounts of very fast software controlled cache memory.

3.1.1 Depthwise Separable Convolutions

Depthwise Separable Convolutions are a key building block for many efficient neural network architectures, and it is used in MobileNetV2 (and MobileNetV1) as well. The basic idea is to replace a full convolutional operator with a factorized version that splits convolution into two separate layers. The first layer is called a depthwise convolution, it performs lightweight filtering by applying a single convolutional filter per input channel, hence not changing the depth. The second layer is a 1 × 1 pointwise convolution, which is responsible for building new features (increasing the number of channels) through computing linear combinations of the input channels. According to the paper, Depthwise Separable Convolutions are a drop-in replacement for standard convolutional layers and reduces computation compared to traditional layers by about a factor of k^2. Since MobileNetV2 uses 3 × 3 depthwise separable convolutions (k = 3), this reduces the computation cost by about a factor of 9, at only a small reduction in accuracy. Thus, in the standard convolution the image is transformed n times, whereas in separable convolution the image is transformed once through depthwise convolution and then elongated to n channels.

3.1.2 Inverted Residual and Linear Bottleneck Layer

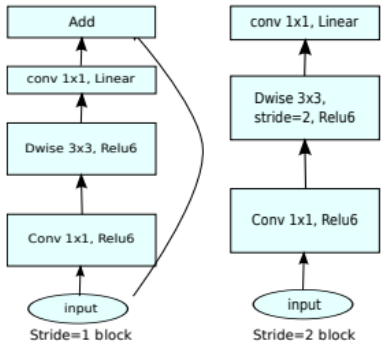
The premise of the inverted residual layer is that firstly, feature maps are able to be encoded in low-dimensional subspaces and secondly, non-linear activations result in information loss in spite of their ability to increase representational complexity.



The first layer takes in a low-dimensional tensor with k channels and performs a point-wise 1×1 convolution to expand the low-dimensional input feature map to a higher-dimensional space suited to non-linear activations, and then ReLU6 (a modification of the rectified linear unit where we limit the activation to a maximum size of 6 due to increased robustness when used with low-precision computation) is applied.

The second layer is the depthwise convolution using 3x3 kernels, followed by ReLU6 activation, achieving spatial filtering of the higher-dimensional tensor.

The third layer takes the spatially-filtered feature map and projects it back to a low-dimensional subspace using a 1×1 convolution but without any non-linearity. This projection results in a loss of information, so the activation function used is a linear activation. When the initial and final feature maps are of the same dimensions, a residual connection is added between these two low-dimensional feature maps to aid gradient flow during backpropagation.



We decided on the MobileNetV2 model for our project.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

3.2 InceptionV3

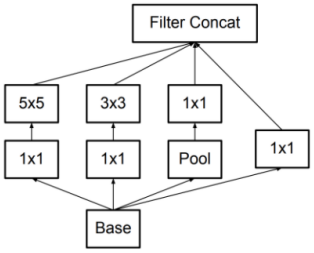
Inception v3 mainly focuses on burning less computational power by modifying the previous Inception architectures. This idea was proposed in the paper Rethinking the Inception Architecture for Computer Vision, published in 2015. It was co-authored by Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jonathon Shlens.

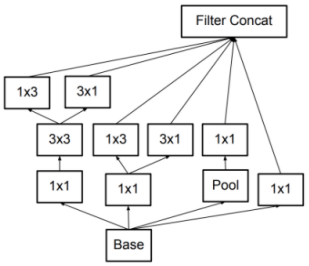
Inception v3 Architecture:

1. Factorized Convolutions: this helps to reduce the computational efficiency as it reduces the number of parameters involved in a network. It also keeps a check on the network efficiency.

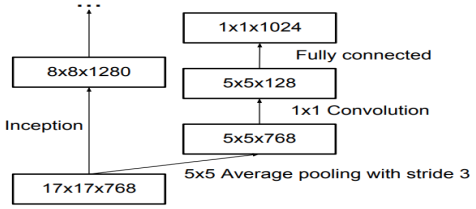
2. Smaller convolutions: replacing bigger convolutions with smaller convolutions definitely leads to faster training.

3. Asymmetric convolutions: A 3 × 3 convolution could be replaced by a 1 × 3 convolution followed by a 3 × 1 convolution. If a 3 × 3 convolution is replaced by a 2 × 2 convolution, the number of parameters would be slightly higher than the asymmetric convolution proposed.

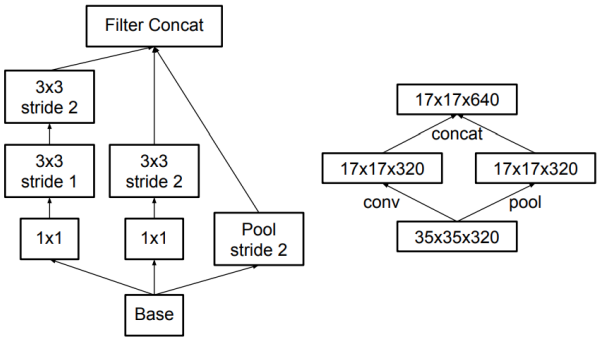




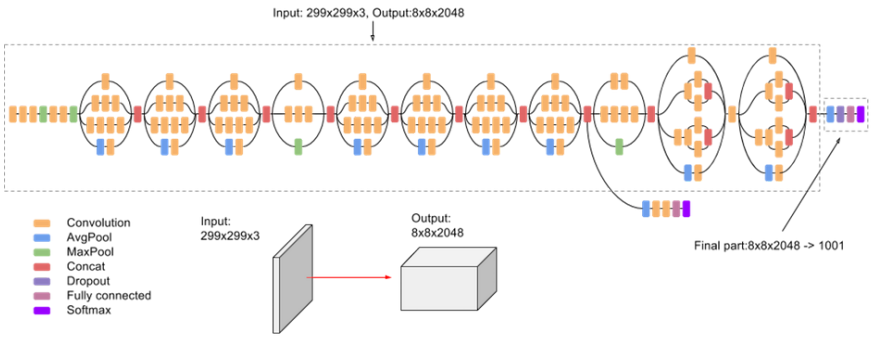
4. Auxiliary classifier: an auxiliary classifier is a small CNN inserted between layers during training, and the loss incurred is added to the main network loss. In GoogLeNet auxiliary classifiers were used for a deeper network, whereas in Inception v3 an auxiliary classifier acts as a regularizer.



5. Grid size reduction: Grid size reduction is usually done by pooling operations. However, to combat the bottlenecks of computational cost, a more efficient technique is proposed:



All the above concepts are consolidated into the final architecture.



According to the research, when augmented with an auxiliary classifier, factorization of convolutions, RMSProp, and Label Smoothing, Inception v3 can achieve the lowest error rates compared to its contemporaries.

Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v3-basic	23.4%	-	3.8
Inception-v3-rmsprop RMSProp	23.1%	6.3	3.8
Inception-v3-smooth Label Smoothing	22.8%	6.1	3.8
Inception-v3-fact Factorized 7×7	21.6%	5.8	4.8
Inception-v3 BN-auxiliary	**21.2%**	**5.6%**	4.8

4. Modelling

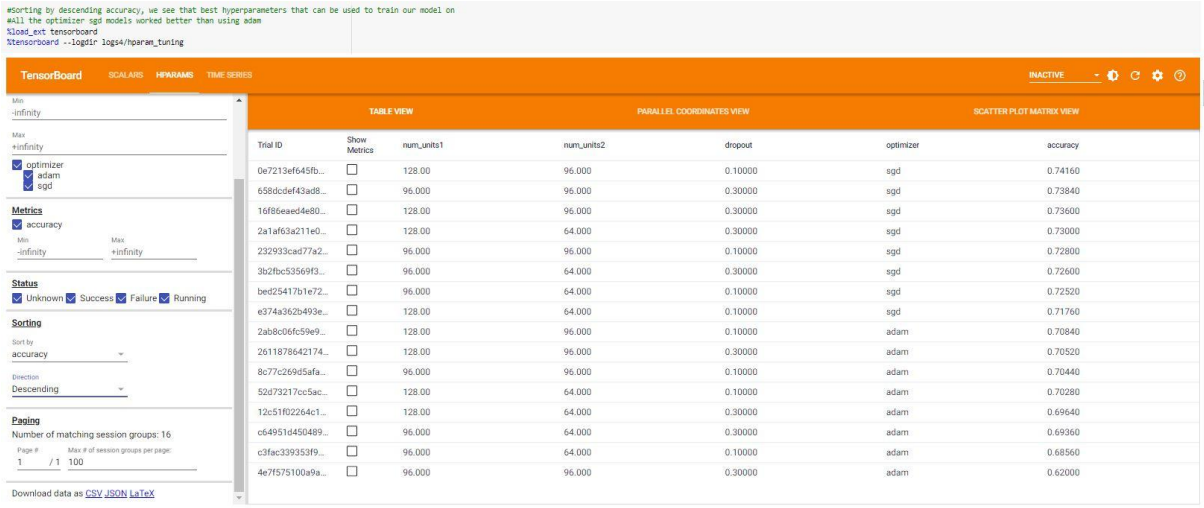
4.1 Experiments on use of featurisers and hyperparameter-tuning

4.1.1 MobileNetV2 featuriser

The MobileNetV2 pre-trained model was used as a starting-point featuriser, taking advantage of the computational and time resource required to develop this neural network model by transferring this knowledge (transfer learning) to the MINC image classification problem. After the MobileNetV2 featuriser, a flatten layer, a hidden layer, another hidden layer with dropout, and an output softmax layer was added. 16 experiments were performed, where the accuracy of all combinations of the below hyperparameters were recorded. Each experiment was run for 4 epochs.

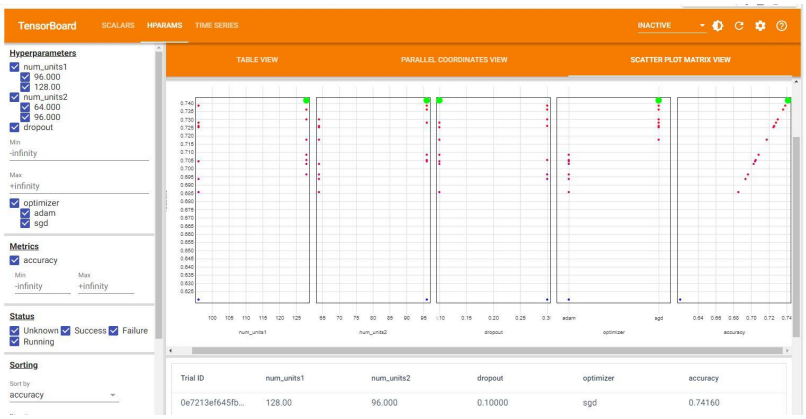
- Number of neurons in hidden layer: 96, 128
- Number of neurons in hidden layer with dropout: 64, 96
- Dropout probability: 0.1, 0.3
- Optimiser: Adam, SGD

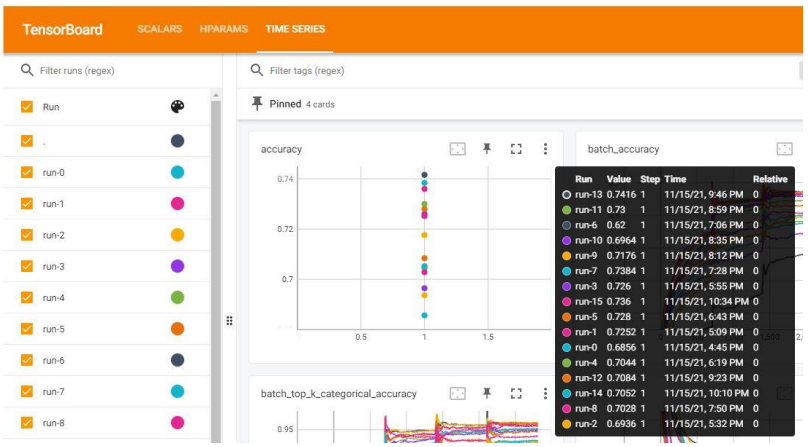
Thereafter, TensorBoard was used to provide visualization of the experiments.

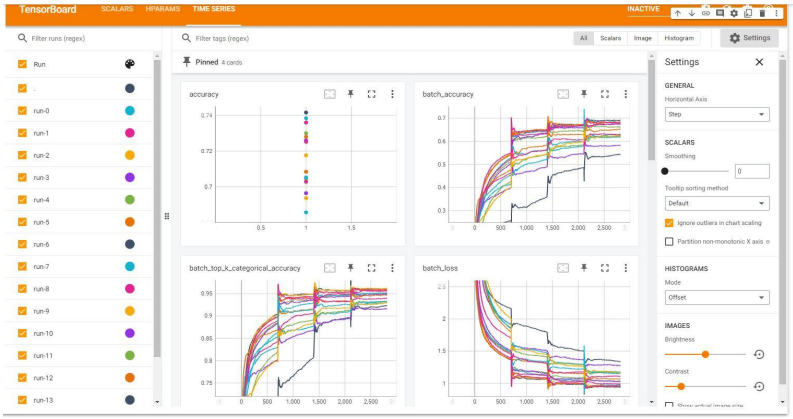


The above results are sorted according to decreasing order of accuracy. We note that across all the experiments, using SGD as optimiser outperformed using adam. In observing the trend of results, for

the other hyperparameters, there does not appear to be a clear consistency across the experiments which would definitely lead to a better accuracy.

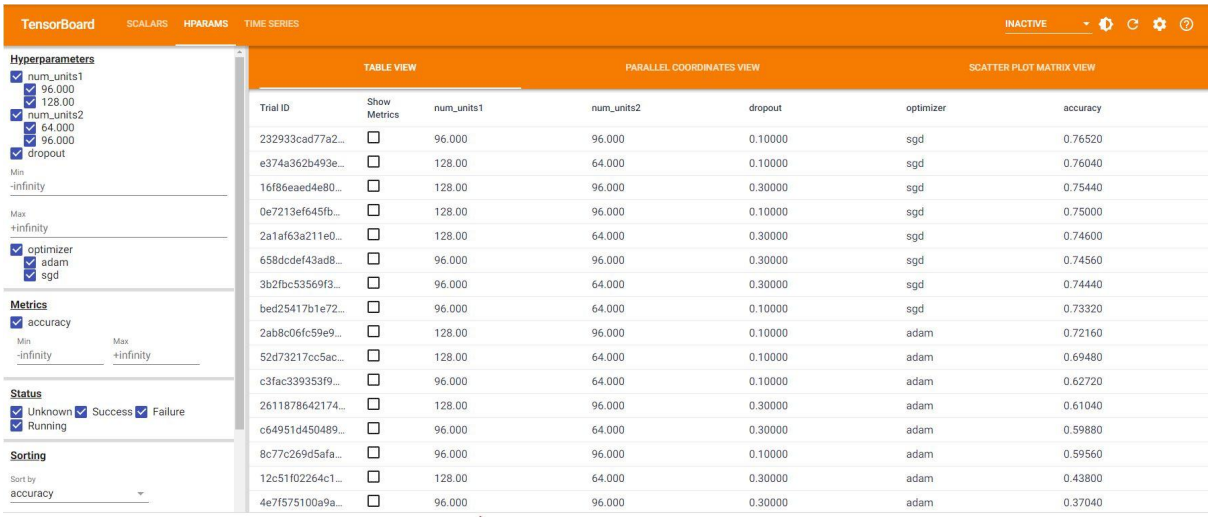






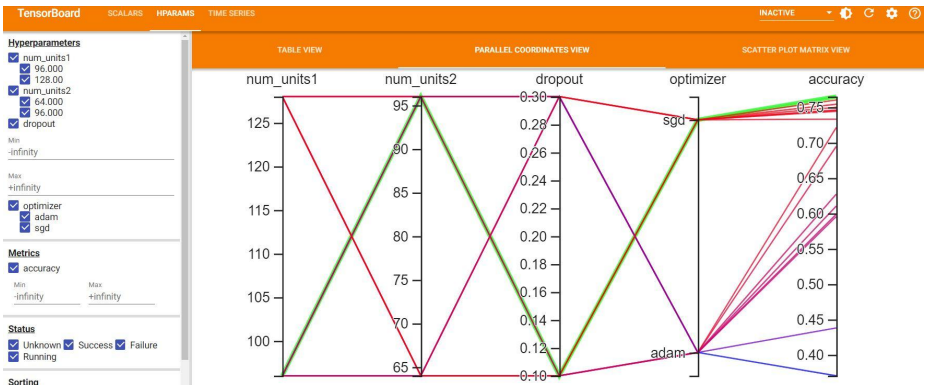
4.1.2 InceptionV3 featuriser

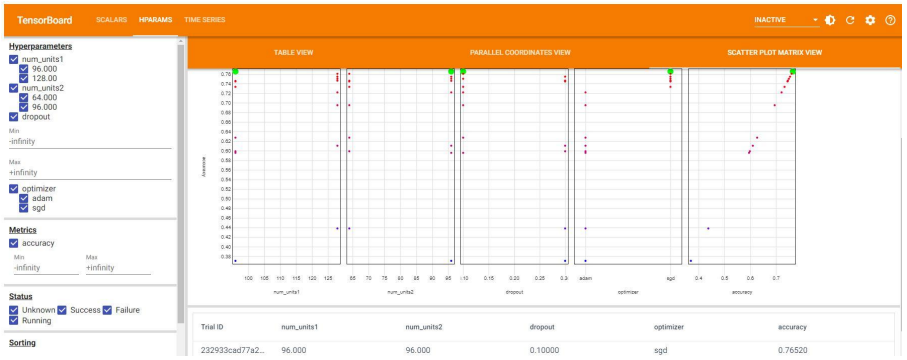
Another 16 experiments were performed using the combinations of hyperparameters documented in section 4.1.1, this time using InceptionV3 as the featuriser.

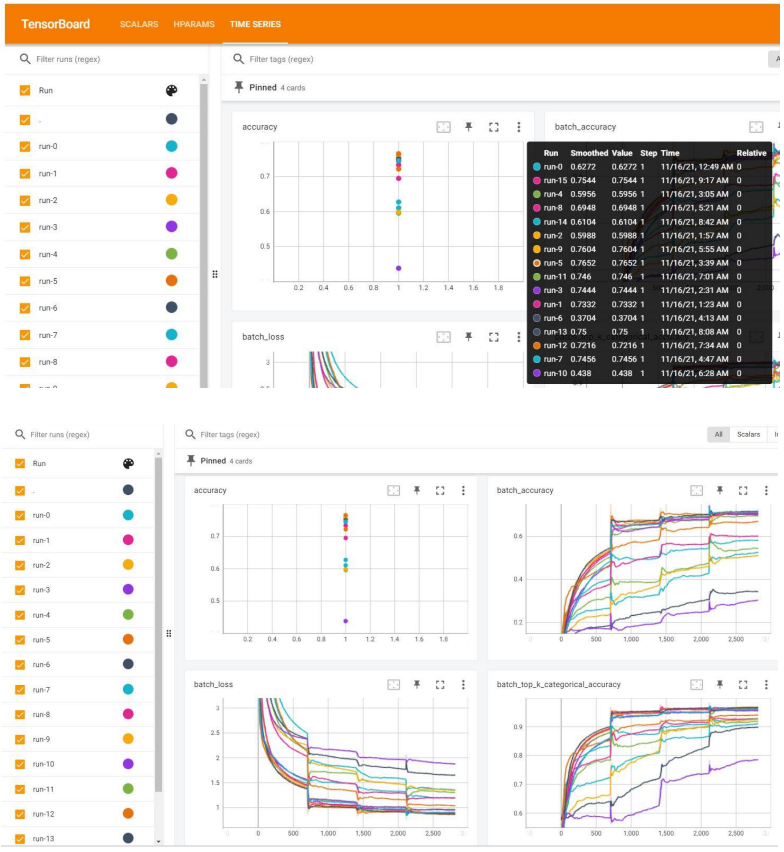


Trial ID	Show Metrics	num_units1	num_units2	dropout	optimizer	accuracy
232933cad77a2...	☐	96.000	96.000	0.10000	sgd	0.76520
e374a362b493e...	☐	128.00	64.000	0.10000	sgd	0.76040
16f86eaed4e80...	☐	128.00	96.000	0.30000	sgd	0.75440
0e7213ef645fb...	☐	128.00	96.000	0.10000	sgd	0.75000
2a1af63a211e0...	☐	128.00	64.000	0.30000	sgd	0.74600
658dcdef43ad8...	☐	96.000	96.000	0.30000	sgd	0.74560
3b2fbc53569f3...	☐	96.000	64.000	0.30000	sgd	0.74440
bed25417b1e72...	☐	96.000	64.000	0.10000	sgd	0.73320
2ab8c06fc59e9...	☐	128.00	96.000	0.10000	adam	0.72160
52d73217cc5ac...	☐	128.00	64.000	0.10000	adam	0.69480
c3fac339353f9...	☐	96.000	64.000	0.10000	adam	0.62720
2611878642174...	☐	128.00	96.000	0.30000	adam	0.61040
c64951d450489...	☐	96.000	64.000	0.30000	adam	0.59880
8c77c269d5afa...	☐	96.000	96.000	0.10000	adam	0.59560
12c51f02264c1...	☐	128.00	64.000	0.30000	adam	0.43800
4e7f575100a9a...	☐	96.000	96.000	0.30000	adam	0.37040

The above results are sorted according to decreasing order of accuracy. We note that across all the experiments, similar to the results obtained in section 4.1.1, using SGD as optimiser outperformed using adam. Similarly, for the other hyperparameters, there does not appear to be a clear consistency across the experiments which would definitely lead to a better accuracy.







From empirical results specific to the MINC dataset, the model that incorporates the InceptionV3 featuriser, 96 neurons in the hidden layer, 96 neurons in the hidden layer with dropout, a dropout probability of 0.1 and the SGD optimiser produced the best accuracy of 0.7652. This is versus the best hyperparameters using MobileNetv2 featuriser with 128 neurons in the hidden layer, 96 neurons in the hidden layer with dropout, a dropout probability of 0.1 and the SGD optimiser produced the best accuracy of 0.7416.

4.2 Model incorporating best hyperparameters

Based on the best set of hyperparameters obtained using the InceptionV3 featuriser, the model was run for 30 epochs based on a 90% train and 10% validation split.

The best validation accuracy obtained is 0.8281, and other validation accuracies range from around 0.7300 to that value. Given that the best validation accuracy is obtained at epoch 24, it appears to be beneficial to run the model through more epochs of training data.

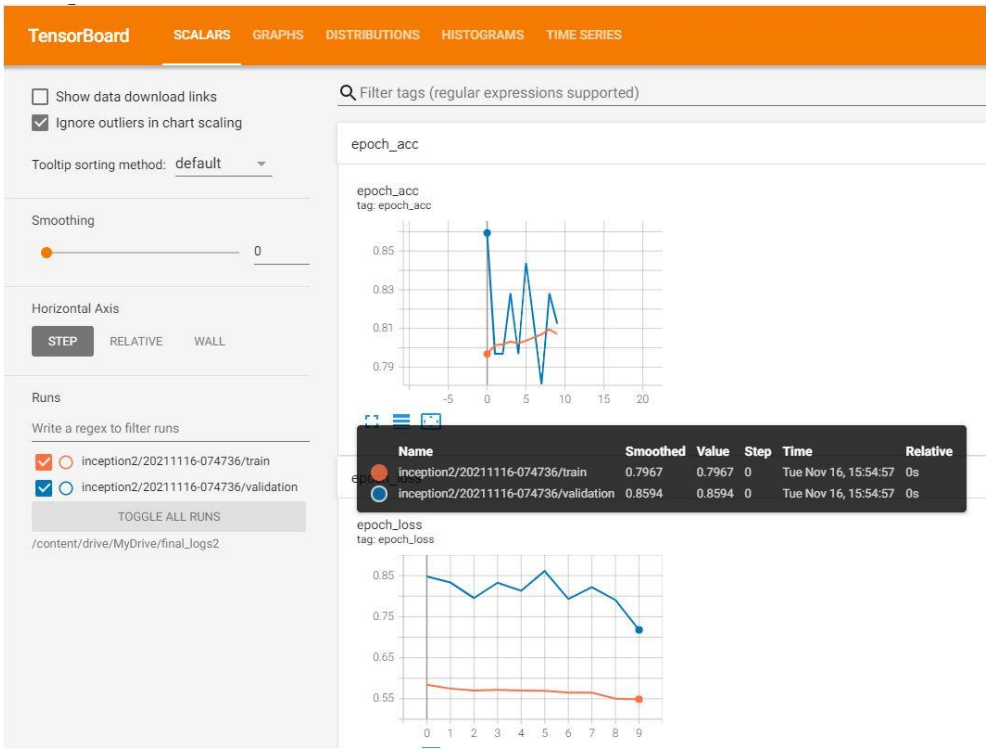
```
Found 2500 images belonging to 10 classes.
              precision    recall  f1-score   support

           0       0.82      0.68      0.75       250
           1       0.88      0.93      0.90       250
           2       0.69      0.82      0.75       250
           3       0.80      0.80      0.80       250
           4       0.71      0.71      0.71       250
           5       0.81      0.82      0.81       250
           6       0.70      0.74      0.72       250
           7       0.90      0.82      0.86       250
           8       0.92      0.87      0.89       250
           9       0.74      0.72      0.73       250

    accuracy                           0.79      2500
   macro avg       0.79      0.79      0.79      2500
weighted avg       0.79      0.79      0.79      2500
```

4.3 Fine-tuning by unfreezing final layers of featuriser

Thereafter, fine-tuning was done where the layers from “mixed7” in the InceptionV3 featuriser was unfrozen, and the model was run for another 10 epochs.



From the results, we see that the best validation accuracy of 0.8594 is obtained just after the first epoch. Thereafter, the validation accuracy jumps between 0.842 and 0.770. This could indicate that unfreezing the final layers may only provide a marginal increase in accuracy.

4.4 Summary of core results

Description of Model	Validation Accuracy
(i) Best of 16 experiments with MobileNetV2 featuriser (Hyperparameters: 128 neurons in hidden layer, 96 neurons in hidden layer with dropout, 0.10 dropout probability, SGD optimiser)	0.7416 (4 epochs)
(ii) Best of 16 experiments with InceptionV3 featuriser (Hyperparameters: 96 neurons in hidden layer, 96 neurons in hidden layer with dropout, 0.10 dropout probability, SGD optimiser)	0.7652 (4 epochs)
(iii) InceptionV3 featuriser with best hyperparameters	0.8281 (30 epochs)
(iv) Unfreeze certain layers in featuriser for fine-tuning	0.8594 (further 10 epochs)

5. Conclusion

In conclusion, we have tried experimenting with 2 different image preprocessing and augmentation methods: CLAHE and Otsu's thresholding on top of the Horizontal Flip, to differentiate the object of interest in each image from its background, though both methods of augmentation have failed to improve the accuracy of the MobileNet model. We realise there were a few actions we could have taken:

1. Denoise the images before performing the methods
2. Find methods to preserve RGB information of images while performing the methods

We would explore these steps in our future works, however due to the limitations of time we were unable to experiment them.

A possible reason for why using InceptionV3 as a featuriser outperforms MobileNetV2 is that the former uses standard convolution while the latter uses depthwise separable convolution. There are more parameters for InceptionV3 versus MobileNetV2, leading to a slight decrease in performance. The focus of this project had been on obtaining the best validation accuracy. Given the trade-off between accuracy and the computational time to output a prediction in real-world settings, a further extension of the project would be to log the computation time on test datasets and find a balance between the two objectives of maximising accuracy and minimising completion time.

Across all the experiments performed, models using SGD as optimiser outperformed using Adam. Despite the popularity of Adam, some research papers discuss that it can fail to converge to an optimal solution under specific settings. For example, the paper "Improving Generalization Performance by Switching from Adam to SGD" illustrates that adaptive optimization techniques such as Adam generalize poorly compared to SGD in certain settings. Other literature discuss empirical deep learning experiments where in certain settings, even when adaptive optimization methods achieve the same training loss or lower than non-adaptive methods, the test performance is worse.

In addition, we had arbitrarily decided to unfreeze InceptionV3 featurizer layers from mixed7 onwards. As such, more tests could be run to determine the results of gradually unfreezing layers in fine-tuning and to what degree this improves accuracy.

6. References

http://opensurfaces.cs.cornell.edu/publications/minc/

https://arxiv.org/pdf/1801.04381.pdf

https://arxiv.org/abs/1512.00567

https://paperswithcode.com/method/relu6

https://machinelearningmastery.com/transfer-learning-for-deep-learning/

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728

https://blog.paperspace.com/popular-deep-learning-architectures-resnet-inceptionv3-squeezenet/

https://medium.com/@luis_gonzales/a-look-at-mobilenetv2-inverted-residuals-and-linear-bottlenecks-d49f85c12423

https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34

https://arxiv.org/pdf/1712.07628.pdf

https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008

https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008