ASSIGNMENT 1B

Joshua Paterson, Kaiyun Yu
N10193197, n9889663

Contribution Table

Person	Contribution
N10193197 – Joshua Paterson	Problem 1
N9889663 - Kaiyun Yu	Problem 2

Assignment 1B

Problem 1

CNN Model Design choices

For the first two parts (with and without data augmentation) of problem 1 used a convolution neural network made from scratch but the design is heavily influenced from the models used in the lectures and practicals. The layers of this model can be seen below in figure 1. The design of this model can be separated into 2 parts being made up of being your classic feedforward neural network and convolution/pooling parts. The convolution/pooling parts of the model are implemented directly after the first input layers (being of shape 32,32,3 meaning it is a 32,32-pixel image with 3 layers representing colour) and is repeated 3 times. The purpose of this layer is to compress the data down to a more manageable size and filter key features. Each of these 3 parts are made of the structure dot pointed below:

- 2x convolutional layers
- Normalisation layers
- Dropout layer
- Pooling layers

Each of these layers start with 2 convolutional layers with the purpose of extracting and filtering information from any sample. For the first pair of convolutional layers, it is design to produce 32 layers of dimensionality and this value is doubled per convolution/pooling parts. This is done in the hope that over each layer more combinations or patterns will be captured. A kernel size of (3,3) was chosen because the image itself is quite small being (32,32) so a large kernel size was not required.

After the 2x convolutional layers a layer dedicated to normalising the outputs of then is used. This layer is used to return values to a common scale after the data has been changed by the previous layers.

After a dropout layer is placed. This layer is used as removing some data at random may help to prevent overfitting to the training data.

The last part of each convolution/pooling parts is a pooling layer. Each of these layers have a pooling size of 2x2 each time the layer is used the dimension of the image is reduced by half. This is used to summarise features and reduce noise.

The final quarter of the model is made up of a feedforward section. To allow this, a layer is used to flatten the results of the previous convolution/pooling parts. This part of the model is made up of 5 layers where all but last layer uses the ‘relu’ activation function as it is a linear function and as I have no clue what the model will find a linear activation seemed like a reasonable place to start. The last layer has no activation function. The last layer was given no activation function as with trial and

error it was found that without one the model would produce better results. The first 3 layers are of 128 neurons wide. Through trial and error, it was found that increasing the number of 128-layers would produce better results however as explained latter 3 seemed like a good balance between training time and accuracy. The next layer is a layer containing 64 neuron and is used to reduce the number of neurons gradually before entering the final layer which is made up of 10 neurons as there are 10 classes.

```
Model: "cnn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 32, 32, 32)        9248
_________________________________________________________________
batch_normalization_9 (Batch (None, 32, 32, 32)        128
_________________________________________________________________
spatial_dropout2d_9 (Spatial (None, 32, 32, 32)        0
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
batch_normalization_10 (Batc (None, 16, 16, 64)        256
_________________________________________________________________
spatial_dropout2d_10 (Spatia (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_11 (Batc (None, 8, 8, 128)         512
_________________________________________________________________
spatial_dropout2d_11 (Spatia (None, 8, 8, 128)         0
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               262272
_________________________________________________________________
dense_16 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_17 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_18 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_19 (Dense)             (None, 10)                650
=================================================================
Total params: 592,106
Trainable params: 591,658
Non-trainable params: 448
_________________________________________________________________
```

Figure 1: CNN model layers

Loss Function

The loss function to be used for the models was the sparse_categorical_crossentropy loss function. This function was chosen as each sample will only belong to 1 of the classes where in similar functions samples may belong to multiple classes. This function was also chosen as labels came in the form of integers.

Dataset

The dataset for this problem consisted of 1000 labelled images for training the 10000 for testing. It should be noted as seen in figures 3 to 8 that both training and testing datasets do not have the same amount of training and testing data for all possible labels. The data seems to follow the trend that samples labelled with 2 will have 80% the number of samples that samples labelled with 1 will have this will continue as samples labelled with 3 will have 80% of samples when compared to samples labelled with 2. This is significant as this bias in training data could be translated into the models.

Data Augmentations

With the data augmentations used only a few parameters were chosen to change/alter images in the training space. This is because in each of the samples used for training and testing the number was always clearly in the centre of the image and other numbers could be present in a sample therefore the location of the number could not be changed to much. For simplicity 4 parameters were chosen to alter being zoom, height and small changes to rotation and shear_range as seen in table 1 below. Each of these parameters was chosen as they do not change the position (being the centre of image) of the labelled number except for height_shift_range. Height_shift_range was used as there was no images found in the training set with numbers above or below the labelled image as numbers are written left to right or horizontally.

Table 1: Data Augmentation used on Data

Data Augmentation	**Range**
rotation_rang	10 degrees
zoom_range	80 to 120%
height_shift_range	10%
shear_range	15%

Computational Constraints

Due to computational constraints the model will be limited to a small model with less then 30 layers. Creating a large model would produce huge training times that would be infeasible to train due to time constraints on the project. However, a high level of accuracy has been achieved with the model detailed above as seen under the comparison heading with the model able to achieve above 80% accuracy on the testing set. It should be noted that with a more complex model (more layers) a higher level of accuracy should be able to be achieved.

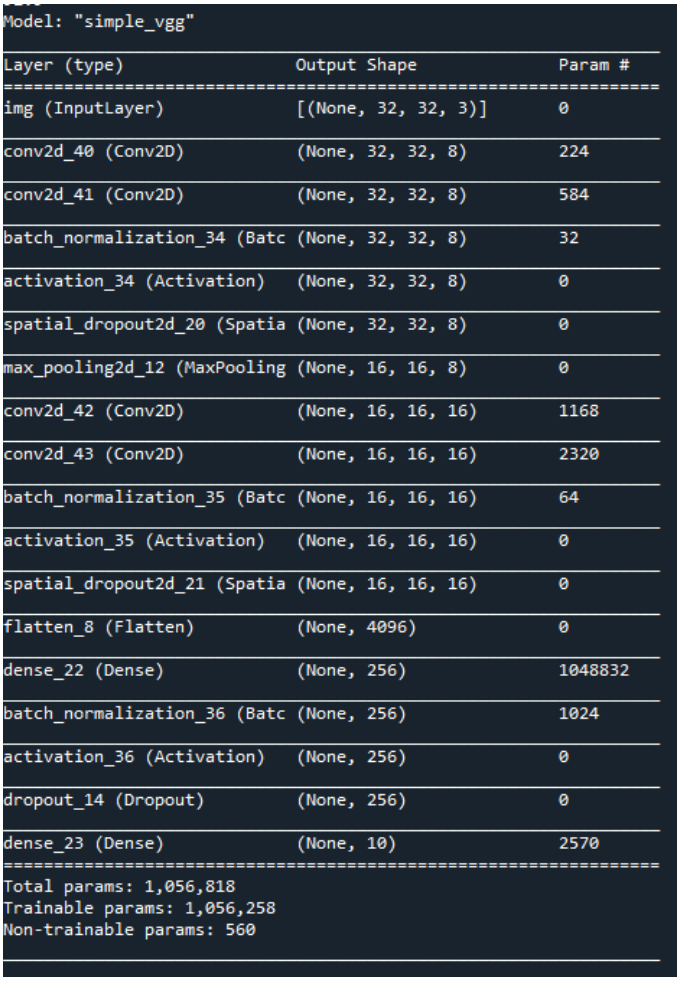
Model description

Four models were used on the training and testing data provided. These being the CNN model detailed above both with and without data augmentations done the training data and a model developed for the CIFAR Dataset with and without data augmentations applied to the training samples.

When trained with no data argumentations a batch size of 40 and 50 epochs was chosen to train the data on the model. These numbers where selected by using trial and error by comparing both testing and training dataset accuracy however a batch size of 40 seemed reasonable as there where only a small amount of training data to use and 50 epochs was chosen as there is only a small amount of data a high value would expose the data too much to the model and could have caused overfitting.

With the model trained on the dataset with data augmentation applied to it the number of epochs was greatly increase to 250 as the data could produce more training data and was more likely to avoid overfitting because of this.

The CIFAR model is named 'vgg_2stage_CIFAR_small.h5 and was chosen because the input image shape and number of outputted classes where the same as the SHVN dataset. This allowed the model to be used for fine tuning on the SHVN dataset with no modification needed on the model itself. This model was also tested with and without data augmentations. The same batch size and epochs were used for training being 40 for batch size and 250 epochs when trained with data augmentations and 40 for batch size and 50 epochs when trained without data augmentations. This was for the same reasons as the purpose build CNN model detailed above.



```
Model: "simple_vgg"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
img (InputLayer)             [(None, 32, 32, 3)]       0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 32, 32, 8)         224
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 32, 32, 8)         584
_________________________________________________________________
batch_normalization_34 (Batc (None, 32, 32, 8)         32
_________________________________________________________________
activation_34 (Activation)   (None, 32, 32, 8)         0
_________________________________________________________________
spatial_dropout2d_20 (Spatia (None, 32, 32, 8)         0
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_42 (Conv2D)           (None, 16, 16, 16)        1168
_________________________________________________________________
conv2d_43 (Conv2D)           (None, 16, 16, 16)        2320
_________________________________________________________________
batch_normalization_35 (Batc (None, 16, 16, 16)        64
_________________________________________________________________
activation_35 (Activation)   (None, 16, 16, 16)        0
_________________________________________________________________
spatial_dropout2d_21 (Spatia (None, 16, 16, 16)        0
_________________________________________________________________
flatten_8 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_22 (Dense)             (None, 256)               1048832
_________________________________________________________________
batch_normalization_36 (Batc (None, 256)               1024
_________________________________________________________________
activation_36 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_14 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_23 (Dense)             (None, 10)                2570
=================================================================
Total params: 1,056,818
Trainable params: 1,056,258
Non-trainable params: 560
_________________________________________________________________
```

Figure 2: CIFAR model Layers

As seen in figure 2 the CIFAR model is very similar to the model create for this problem above seen under the heading CNN Model Design choices. The difference these models have are that the CIFAR model has 2 less pooling layers and 1 less convolutional pair. It also has an activation layer added between the normalisation and dropout layers however the greatest difference between these models is that where the model created for this project is designed to increase the number of dimensions (filters) further through the layers where the CIFAR layers decrease the number of dimensions the further into the network. It should also be noted that after the layers are flattened in the CIFAR model the widths (number of neurons per layer) of the new model is twice as large as the

one created for this project and makes no attempt to decrease the layers gradually when close to the output layer.

CNN with no Data Augmentation Results

The results for the CNN model with no data augmentation used can be seen in figures 3 for the training data and figure 4 for the testing data. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified being 100% on the training images and 82.66% on the testing dataset.

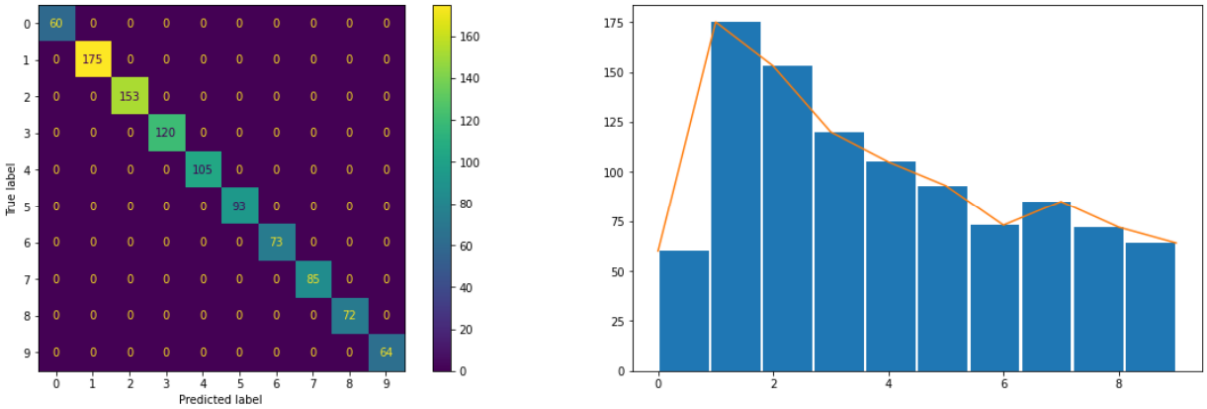


Figure 3: CNN with no Data Augmentation Training Dataset

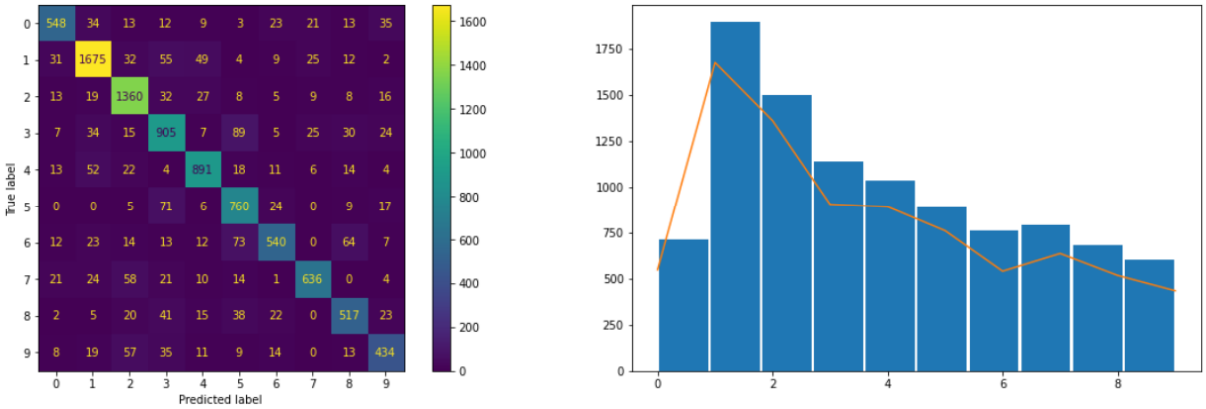


Figure 4: CNN with no Data Augmentation Testing Dataset

CNN Data Augmentation Results

The results for the CNN model with data augmentation used can be seen in figures 5 for the training data and figure 6 for the testing data. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified being 100% on the training images and 83.75% on the testing dataset.

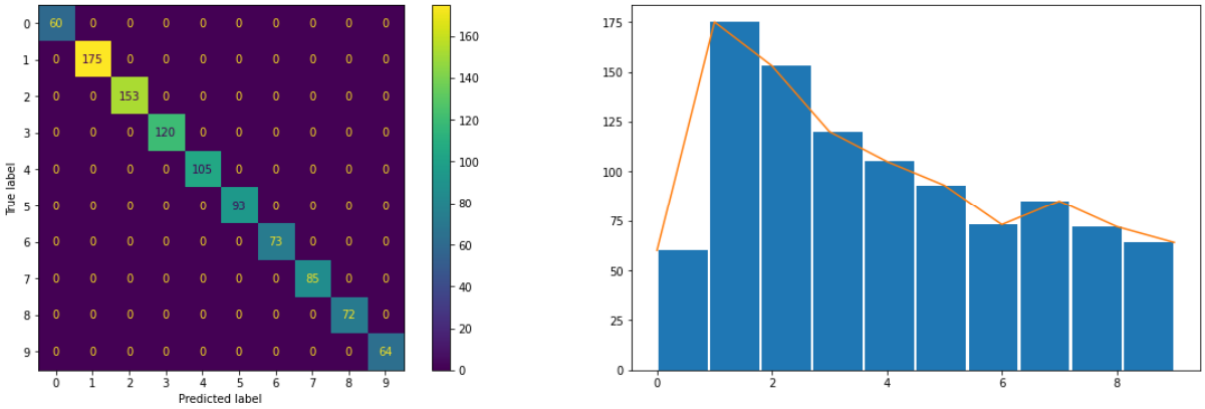


Figure 5: CNN with Data Augmentation Training Dataset

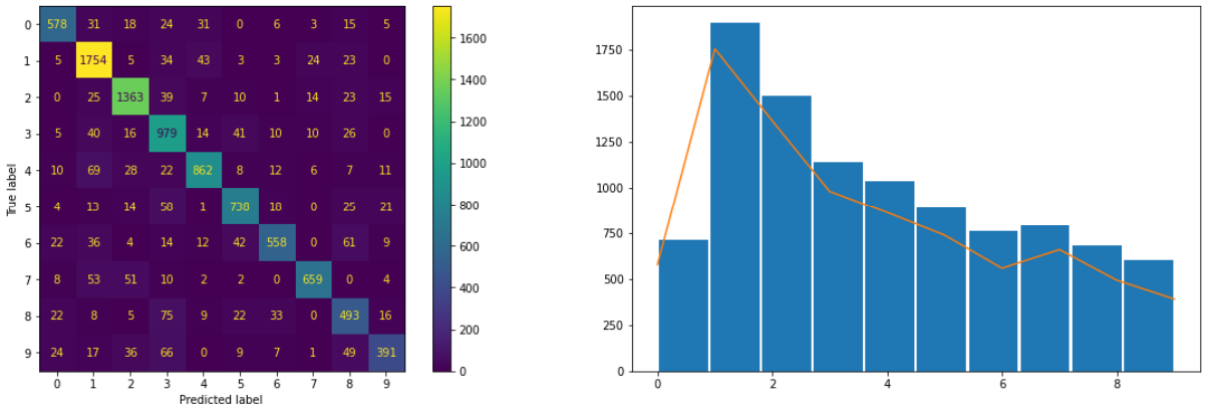


Figure 6: CNN with Data Augmentation Testing Dataset

CIFAR with Data Augmentation

The results for the CIFAR model with data augmentation used can be seen in figures 6 for the training data and figure 7 for the testing data. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified being 97% on the training images and 80.53% on the testing dataset.

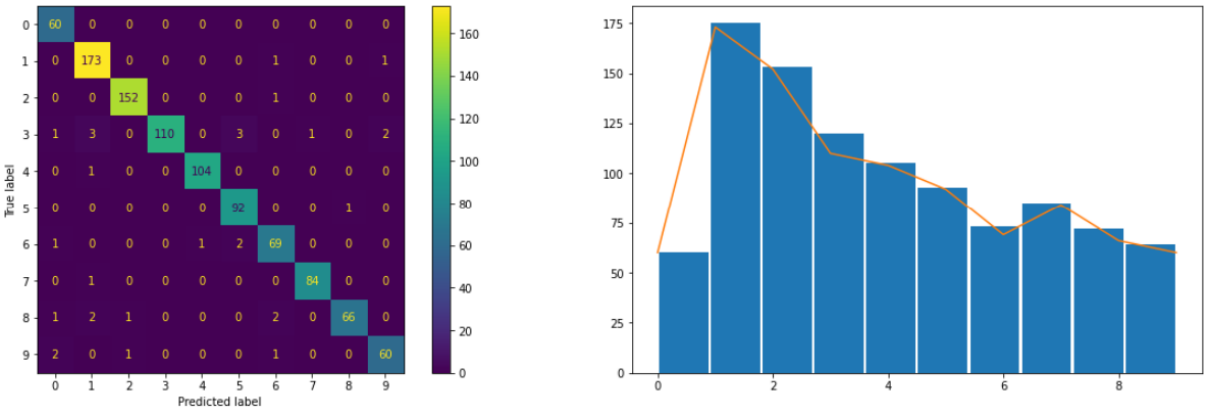


Figure 7: CIFAR with Data Augmentation Training Dataset

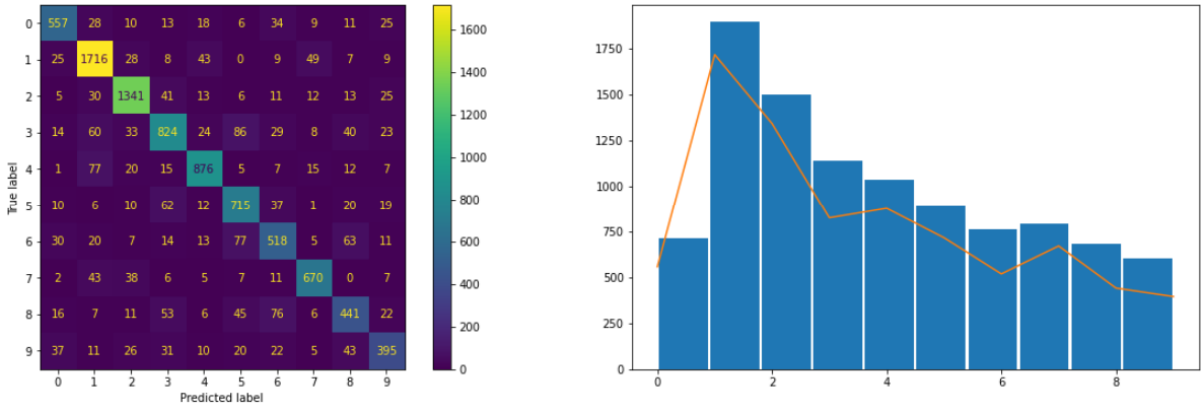


Figure 8: CIFAR with Data Augmentation Testing Dataset

CIFAR without Data Augmentation

The results for the CIFAR model without data augmentation used can be seen in figures 9 for the training data and figure 10 for the testing data. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified. The overall accuracy of the model can be seen below under model comparison on both the training and testing data given in table 2 which shows the percentage of correct samples classified being 62.8% on the training images and 46.7% on the testing dataset. Where unseen from the previous models the bias in the dataset has been shown within this model as the labels with more training data being samples labelled 1 and 2 have a better ratio of correctly predicted samples this can be seen in figures 9 and 10 left most plot as samples labelled 1 and 2 are the only labels to have a higher accuracy then around 50% in both training and testing data spaces.

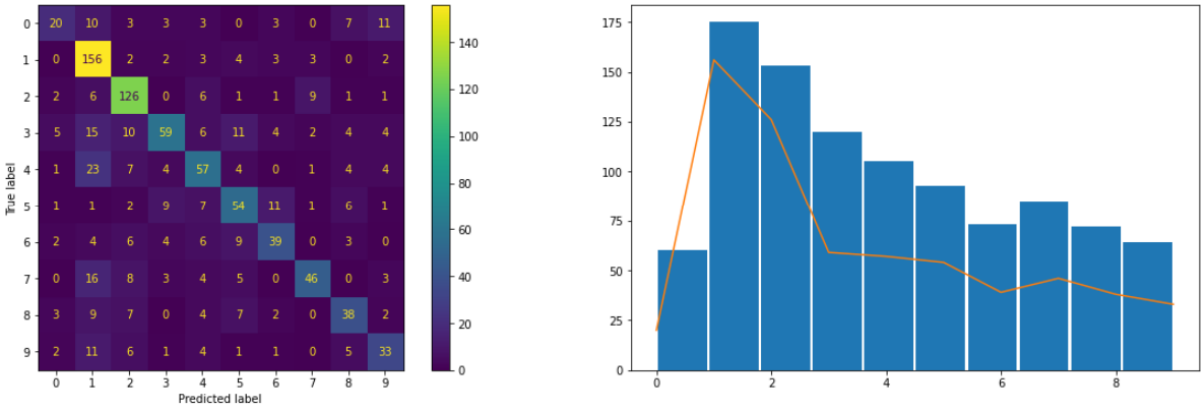


Figure 9: CIFAR without Data Augmentation Training Dataset

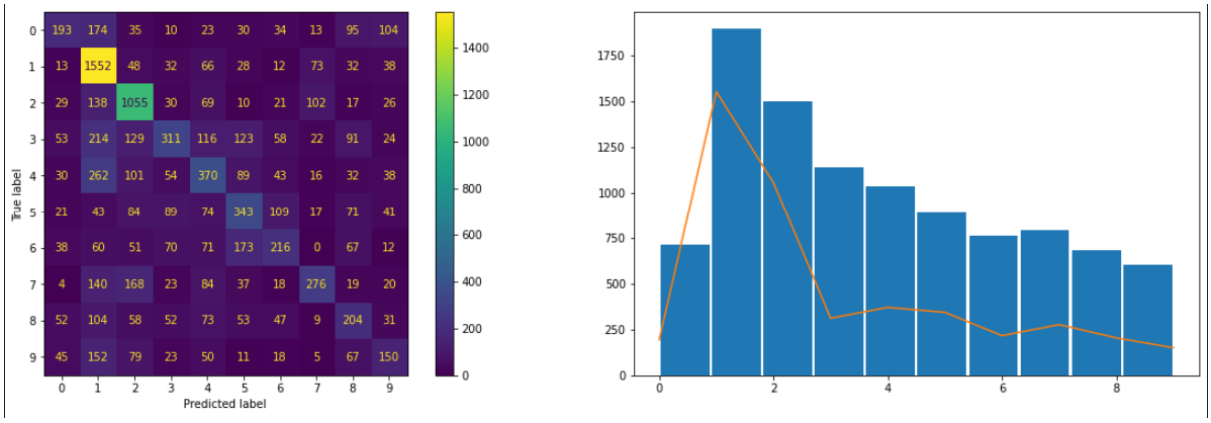


Figure 10: CIFAR without Data Augmentation Testing Dataset

Comparison of models

As seen in table 2 below all the tested models performed to a similar accuracy of around 80% on the testing samples and near perfect on the training samples except for the CIFAR model trained without data augmentations with an accuracy of 63% on the training samples and 47% on the testing samples.

Both model types being the purpose-built CNN model and CIFAR model where able to produce better results with data augmentations applied to the training data. It should be noted that the accuracy increase with the CNN model was small being of around 1.09% increase in accuracy on the testing samples where the CIFAR model was able to increase its accuracy by 33% on the testing samples. This was likely because the CNN model performed far better without data augmentations then the CIFAR showing there is demising returns the more accurate the model without data augmentations.

Both the purpose-built CNN models where able to produce the best result on this problem both being able to produce 100% accuracy on the data used for training and both being able to produce better results than the CIFAR models on the testing dataset where at worst there was a 2% increase in accuracy.

The CIFAR model was not able to produce very accurate result when data augmentations where not applied when used for fine tuning. This was likely due to the limit amount of data that the model could use for training (1000 samples). Due to both the increase in data due to fine tuning and the increased number of epoch used when training, the model was able to produce a accuracy on both training and testing that was comparable to the CNN models being around 2% less accurate on the testing set and 3 less accurate on the training set.

In conclusion all but the CIFAR model trained without data argumentations where able to achieve accuracy levels of around 80% on the testing set however this is still quite a poor performing model as it is still wrong 1 in 5 times. This low level of accuracy is likely due to both the small amount of data for training available as well as the self-imposed restrictions to model size of 30 layers. It is likely that a better result could be achieve with more data and a more complex model.

Table 2: Accuracy of Models on Datasets

Models	Training data Accuracy (%)	Testing data Accuracy (%)
CNN with no data augmentation	100.0	82.66
CNN with data augmentation	100.0	83.75
CIFAR_small model with data augmentation	97.0	80.53
CIFAR_small model without data augmentation	62.8	46.7

Problem 2

PCA and data extraction

Data is splinted to training set and testing set already. Data is saved as 5933 jpg images with the shape of 128x64. Pil.Image function has saved all Data in a 4D numpy array which size as (5933, 128,64,3) that keeps original size and information of training set and the data needs to be reshaped and resize in further.

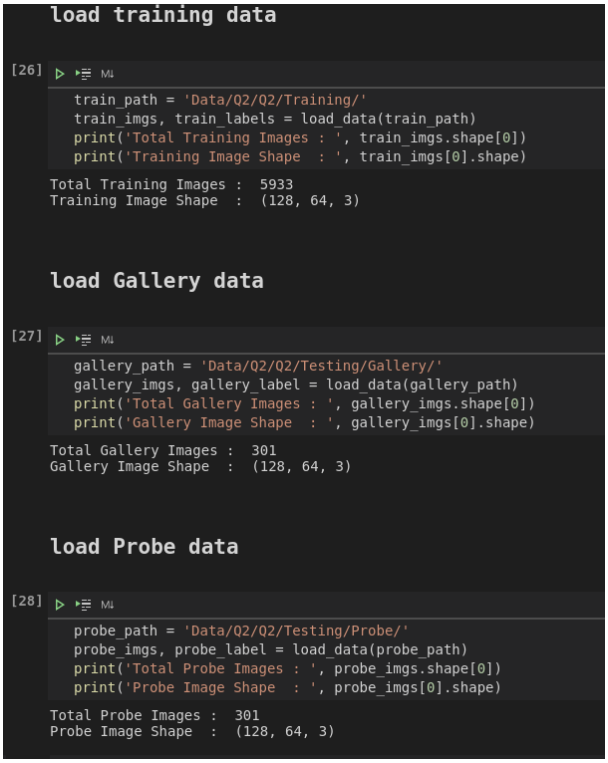


Figure 11:Data loading

PCA is more suitable with unsupervised machine leaning, PCA is mainly to find a better projection method from the angle of the covariance of the feature to select the direction of the sample point projection with the largest variance while LDA considers the classification label information more and seeks to size the data point distances between different categories after projection and minimize the data point distances of the same category. Which means choosing the direction with the best classification performance.

Reshape the data to 4 rows, 5933 columns matrix and use PCA to dimensions the data for extracting feature information and calculated Euclidean Distance which is (301,301).

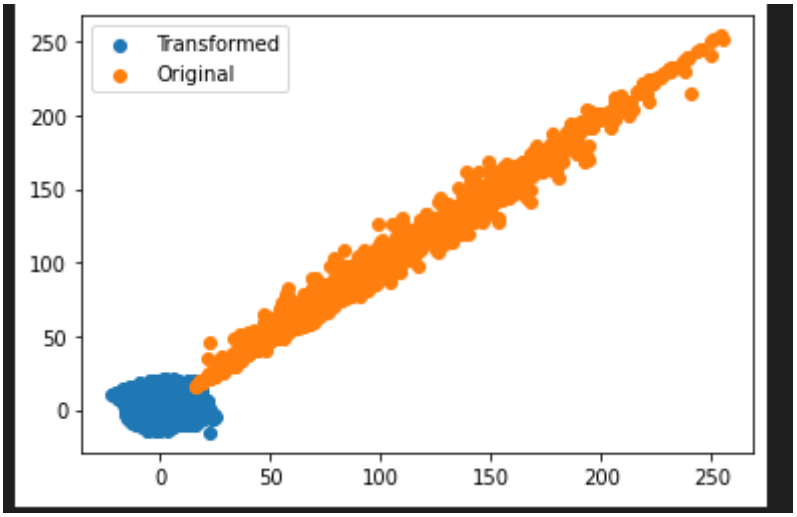


Figure 12: Data transformation

Hyperparameter selection

K as the only hyperparameter in the KNN algorithm, the selection of the K value will have an intuitive and important impact on the prediction results of the final algorithm. Because of complex and large size of Data k = 50 is chosen, too small value of k will easily cause overfit.

KNN method non deep learning method,

Use CMC Curve to get Top1, Top5 and Top10 Accuracy.

The CMC curve comprehensively reflects the performance of the classifier. Its evaluation indicators have the same meaning as the top1 err and top5 err evaluation indicators commonly used in deep learning. The difference is that the Rank on the abscissa represents the correct rate rather than the error rate.

Top1 accuracy is 19%, top 5 accuracy is 23% and top10 accuracy is 27%.
And use plot to show the CMC Curve.

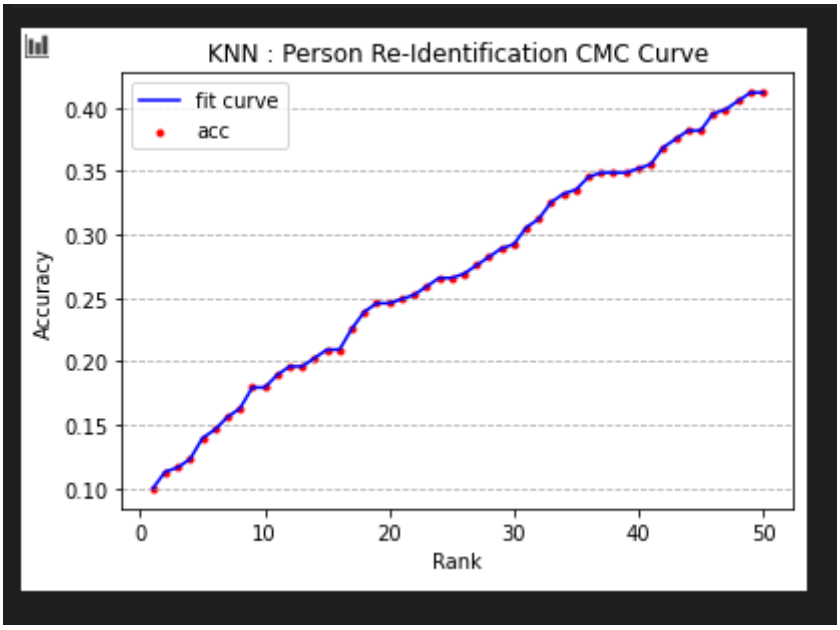


Figure 13: KNN Person Re-Identification CMC Curve

CNN deep learning method

Resnet compare with another CNN deep learning methods, Balduzzi found that even if the mode of the gradient stabilizes within the normal range after BN, the correlation of the gradient decays continuously as the number of layers increases. It has been proven that ResNet can effectively reduce the attenuation of this correlation. Resnet provide higher accuracy with more layers compare with “network”. (David Balduzzi, 2017)

```
model = resnet_v1(input_shape=(128, 64, 3), depth=20, num_classes=3080)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
model.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(?, 128, 64, 3)	0	
conv2d_21 (Conv2D)	(None, 128, 64, 16)	448	input_2[0][0]
batch_normalization_19 (BatchNo	(None, 128, 64, 16)	64	conv2d_21[0][0]
activation_19 (Activation)	(None, 128, 64, 16)	0	batch_normalization_19[0][0]
conv2d_22 (Conv2D)	(None, 128, 64, 16)	2320	activation_19[0][0]
batch_normalization_20 (BatchNo	(None, 128, 64, 16)	64	conv2d_22[0][0]
activation_20 (Activation)	(None, 128, 64, 16)	0	batch_normalization_20[0][0]
conv2d_23 (Conv2D)	(None, 128, 64, 16)	2320	activation_20[0][0]
batch_normalization_21 (BatchNo	(None, 128, 64, 16)	64	conv2d_23[0][0]
add_9 (Add)	(None, 128, 64, 16)	0	activation_19[0][0] batch_normalization_21[0][0]
activation_21 (Activation)	(None, 128, 64, 16)	0	add_9[0][0]
conv2d_24 (Conv2D)	(None, 128, 64, 16)	2320	activation_21[0][0]
batch_normalization_22 (BatchNo	(None, 128, 64, 16)	64	conv2d_24[0][0]
activation_22 (Activation)	(None, 128, 64, 16)	0	batch_normalization_22[0][0]
conv2d_25 (Conv2D)	(None, 128, 64, 16)	2320	activation_22[0][0]
batch_normalization_23 (BatchNo	(None, 128, 64, 16)	64	conv2d_25[0][0]
add_10 (Add)	(None, 128, 64, 16)	0	activation_21[0][0] batch_normalization_23[0][0]
activation_23 (Activation)	(None, 128, 64, 16)	0	add_10[0][0]
conv2d_26 (Conv2D)	(None, 128, 64, 16)	2320	activation_23[0][0]
batch_normalization_24 (BatchNo	(None, 128, 64, 16)	64	conv2d_26[0][0]
activation_24 (Activation)	(None, 128, 64, 16)	0	batch_normalization_24[0][0]
conv2d_27 (Conv2D)	(None, 128, 64, 16)	2320	activation_24[0][0]
batch_normalization_25 (BatchNo	(None, 128, 64, 16)	64	conv2d_27[0][0]
add_11 (Add)	(None, 128, 64, 16)	0	activation_23[0][0] batch_normalization_25[0][0]
activation_25 (Activation)	(None, 128, 64, 16)	0	add_11[0][0]
conv2d_28 (Conv2D)	(None, 64, 32, 32)	4640	activation_25[0][0]
batch_normalization_26 (BatchNo	(None, 64, 32, 32)	128	conv2d_28[0][0]
activation_26 (Activation)	(None, 64, 32, 32)	0	batch_normalization_26[0][0]
conv2d_29 (Conv2D)	(None, 64, 32, 32)	9248	activation_26[0][0]
conv2d_30 (Conv2D)	(None, 64, 32, 32)	544	activation_25[0][0]
batch_normalization_27 (BatchNo	(None, 64, 32, 32)	128	conv2d_29[0][0]
add_12 (Add)	(None, 64, 32, 32)	0	conv2d_30[0][0] batch_normalization_27[0][0]
activation_27 (Activation)	(None, 64, 32, 32)	0	add_12[0][0]
conv2d_31 (Conv2D)	(None, 64, 32, 32)	9248	activation_27[0][0]
batch_normalization_28 (BatchNo	(None, 64, 32, 32)	128	conv2d_31[0][0]
activation_28 (Activation)	(None, 64, 32, 32)	0	batch_normalization_28[0][0]
conv2d_32 (Conv2D)	(None, 64, 32, 32)	9248	activation_28[0][0]
batch_normalization_29 (BatchNo	(None, 64, 32, 32)	128	conv2d_32[0][0]
add_13 (Add)	(None, 64, 32, 32)	0	activation_27[0][0] batch_normalization_29[0][0]
activation_29 (Activation)	(None, 64, 32, 32)	0	add_13[0][0]
conv2d_33 (Conv2D)	(None, 64, 32, 32)	9248	activation_29[0][0]
batch_normalization_30 (BatchNo	(None, 64, 32, 32)	128	conv2d_33[0][0]
activation_30 (Activation)	(None, 64, 32, 32)	0	batch_normalization_30[0][0]
conv2d_34 (Conv2D)	(None, 64, 32, 32)	9248	activation_30[0][0]
batch_normalization_31 (BatchNo	(None, 64, 32, 32)	128	conv2d_34[0][0]
add_14 (Add)	(None, 64, 32, 32)	0	activation_29[0][0] batch_normalization_31[0][0]
activation_31 (Activation)	(None, 64, 32, 32)	0	add_14[0][0]
conv2d_36 (Conv2D)	(None, 32, 16, 64)	18496	activation_31[0][0]
batch_normalization_32 (BatchNo	(None, 32, 16, 64)	256	conv2d_35[0][0]
activation_32 (Activation)	(None, 32, 16, 64)	0	batch_normalization_32[0][0]
conv2d_36 (Conv2D)	(None, 32, 16, 64)	36928	activation_32[0][0]
conv2d_37 (Conv2D)	(None, 32, 16, 64)	2112	activation_31[0][0]
batch_normalization_33 (BatchNo	(None, 32, 16, 64)	256	conv2d_36[0][0]
add_15 (Add)	(None, 32, 16, 64)	0	conv2d_37[0][0] batch_normalization_33[0][0]
activation_33 (Activation)	(None, 32, 16, 64)	0	add_15[0][0]
conv2d_39 (Conv2D)	(None, 32, 16, 64)	36928	activation_33[0][0]
batch_normalization_34 (BatchNo	(None, 32, 16, 64)	256	conv2d_38[0][0]
activation_34 (Activation)	(None, 32, 16, 64)	0	batch_normalization_34[0][0]
conv2d_39 (Conv2D)	(None, 32, 16, 64)	36928	activation_34[0][0]
batch_normalization_35 (BatchNo	(None, 32, 16, 64)	256	conv2d_39[0][0]
add_16 (Add)	(None, 32, 16, 64)	0	activation_33[0][0] batch_normalization_35[0][0]
activation_35 (Activation)	(None, 32, 16, 64)	0	add_16[0][0]
conv2d_40 (Conv2D)	(None, 32, 16, 64)	36928	activation_35[0][0]
batch_normalization_36 (BatchNo	(None, 32, 16, 64)	256	conv2d_40[0][0]
activation_36 (Activation)	(None, 32, 16, 64)	0	batch_normalization_36[0][0]
conv2d_41 (Conv2D)	(None, 32, 16, 64)	36928	activation_36[0][0]
batch_normalization_37 (BatchNo	(None, 32, 16, 64)	256	conv2d_41[0][0]
add_17 (Add)	(None, 32, 16, 64)	0	activation_35[0][0] batch_normalization_37[0][0]
activation_37 (Activation)	(None, 32, 16, 64)	0	add_17[0][0]
average_pooling2d_1 (AveragePoo	(None, 4, 2, 64)	0	activation_37[0][0]
flatten_1 (Flatten)	(None, 512)	0	average_pooling2d_1[0][0]
dense_1 (Dense)	(None, 200)	102600	flatten_1[0][0]

```
Total params: 427,692
Trainable params: 426,316
Non-trainable params: 1,376
```

Figure 14: CNN deep learning method layers printout

10 layers of Resnet achieve Top-1 accuracy is 31%, Top-5 accuracy is 52%, and Top-10 accuracy is 62%, 20 layers of Resnet is appliable, Top-1 accuracy is 46%, Top-5 accuracy has achieved 54% and Top-10 accuracy achieve 68%. The outcome has significant results, which matched in the most cases. Thus, compare with other CNN learning method, Resnet achieve significant results when number of layers has increased, other CNN learning method accuracy decrease when layers numbers too high.

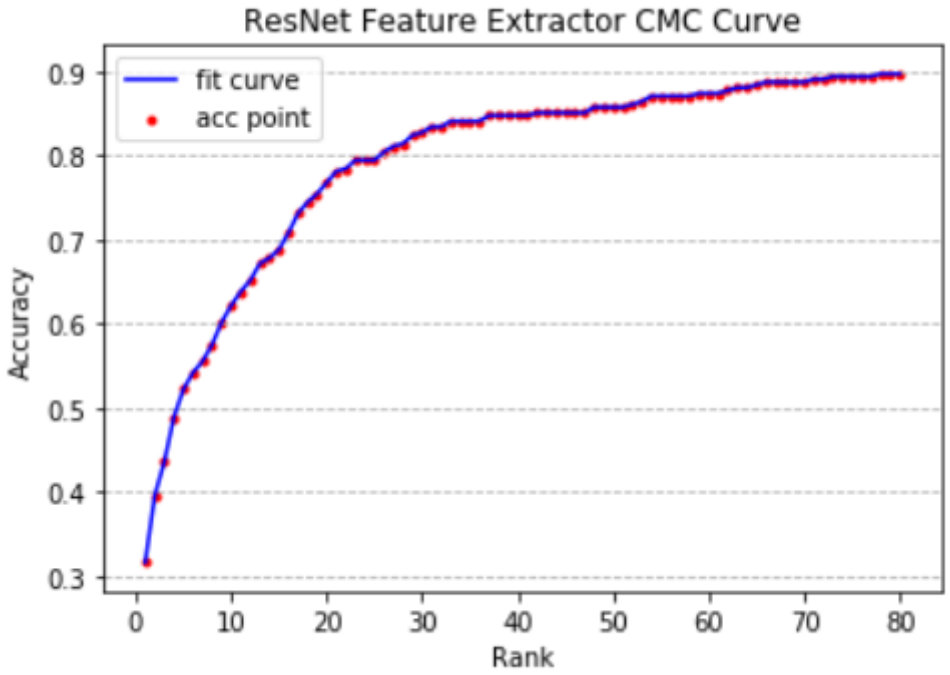


Figure 15: ResNet Feature Extractor CMC Curve

Comparison of deep and non-deep methods

Firstly, Deep learning method needs a lot of matrix calculation, thus deep learning method takes much more times compare with non-deep learning method also more relies on hardware performance.

Secondly, the accuracy of non-deep learning depends on the accuracy of feature processing. An excellent feature processor can significantly improve the accuracy of non-deep learning. However, making a feature processor relies on a lot of professional knowledge. When the data complexity increases, and the variables increase the production of feature processors will become very complicated. But deep learning attempts to obtain high-level features directly from data. This is the main difference between deep learning and traditional machine learning algorithms. Based on this, deep learning reduces the work of designing feature extractors for each problem.

Finally, when the amount of data is small, the performance of deep learning is not good, because deep learning algorithms require a large amount of data to understand the patterns contained therein. Thus, due to different problem-solving methods, deep learning and non-deep learning are suitable for different fields.

References

David Balduzzi, M. F.-D. (2017). The Shattered Gradients Problem: If resnets are the answer, then what is the question? *Proceedings of the 34th International Conference on Machine Learning* (pp. 70:342-35). Sydney, NSW, Australia: Proceedings of Machine Learning Research. Retrieved from http://proceedings.mlr.press/v70/balduzzi17b.html

Appendix Code

```
ne # -*- coding: utf-8 -*-
"""
Created on Wed Apr 28 17:46:50 2021

@author: User
"""

# -*- coding: utf-8 -*-
"""
Created on Thu Apr 15 13:11:21 2021

@author: User
"""

import scipy.io
import matplotlib.pyplot as plt

import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import numpy as np
import numpy
import tensorflow.keras as keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to_categorical
import seaborn as sns


if __name__=="__main__":

    #import data
    test_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_test.mat')
    train_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_train.mat')

    # Load images and labels
    test_Y = np.array(test_data['test_Y'])
    test_X = np.array(test_data['test_X'])  /255.0

    train_Y = np.array(train_data['train_Y'])
    train_X = np.array(train_data['train_X']) /255.0

    # Check the shape of the data
    print(test_X.shape)
    print(train_X.shape)

    # Fix the axes of the images
    test_X = np.moveaxis(test_X, -1, 0)
    train_X = np.moveaxis(train_X, -1, 0)

    print(test_X.shape)
    print(train_X.shape)

    # Plot a random image and its label
```

```
plt.imshow(train_X[350])
plt.show()
print(train_Y[350])

#reshape train Y to vector format
print(test_Y)

#replace 10 to 0s in ys
train_Y = np.where(train_Y==10, 0, train_Y)
test_Y = np.where(test_Y==10, 0, test_Y)

a = 5

print(test_Y[a])
print(test_Y[a+1])

def unique(list1):
    x = np.array(list1)
    print(np.unique(x))

print("unique")

unique(test_Y)


def build_model(num_classes, output_activation=None):#week 4 lec/week 5 prac
    # our model, input in an image shape
    inputs = keras.Input(shape=(32, 32, 3,))

    # run pairs of conv layers, all 3s3 kernels
    x = keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same',
    activation=None)(inputs)
    x = keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)

    # rinse and repeat with 2D convs, batch norm, dropout and max pool
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)

    # final conv2d, batch norm and spatial dropout
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)
```

```
    # flatten layer
    x = keras.layers.Flatten()(x)
    # we'll use a couple of dense layers here, mainly so that we can show what another
    dropout layer looks like
    # in the middle
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    # the output
    outputs = keras.layers.Dense(num_classes, activation=output_activation)(x)

    # build the model, and print a summary
    model_cnn = keras.Model(inputs=inputs, outputs=outputs, name='cnn_model')

    return model_cnn

model = build_model(10)
model.summary()

model.compile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          optimizer=keras.optimizers.Adam(),
          metrics=['accuracy'])

#for i in range(1):

model.fit(train_X,train_Y,batch_size = 40,epochs=50)

predictions = model.predict(test_X)

a = 5

print(predictions[a])
print(test_Y[a])
print(predictions[a+1])
print(test_Y[a+1])
print(predictions[a+2])
print(test_Y[a+2])


def eval_model(model, x_test, y_test):
        test_scores = model.evaluate(x_test, y_test, verbose=2)
        print('Test loss:', test_scores[0])
        print('Test accuracy:', test_scores[1])

        pred = model.predict(x_test);
        indexes = tf.argmax(pred, axis=1)

        cm = confusion_matrix(y_test, indexes)
        fig = plt.figure(figsize=[20, 6])
        ax = fig.add_subplot(1, 2, 1)
        c = ConfusionMatrixDisplay(cm, display_labels=range(len(numpy.unique(y_test))))
        c.plot(ax = ax)

        ax = fig.add_subplot(1, 2, 2)
        ax.hist(y_test, bins=len(numpy.diagonal(cm)), rwidth=0.95)
        ax.plot(numpy.diagonal(cm))
```

```
eval_model(model, train_X, train_Y)
eval_model(model, test_X, test_Y)

pred = model.predict(test_X);
indexes = tf.argmax(pred, axis=1)
count = 0
print("_check_")
print(test_Y[9])
print(indexes[9].numpy())
print("____test")
for i in range(10000):
    if test_Y[i] == indexes[i].numpy():
        count = count + 1
print(count)
print(len(indexes))
print((count/len(indexes))*100)
print("____train")
pred = model.predict(train_X);
indexes = tf.argmax(pred, axis=1)
count = 0
for i in range(1000):
    if train_Y[i] == indexes[i].numpy():
        count = count + 1
print(count)
print(len(indexes))
print((count/len(indexes))*100)
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 28 17:46:50 2021

@author: User
"""

# -*- coding: utf-8 -*-
"""
Created on Thu Apr 15 13:11:21 2021

@author: User
"""

import scipy.io
import matplotlib.pyplot as plt

import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import numpy as np
import numpy
import tensorflow.keras as keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to_categorical
import seaborn as sns

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img

if __name__=="__main__":

    #import data
    test_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_test.mat')
    train_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_train.mat')

    # Load images and labels
    test_Y = np.array(test_data['test_Y'])
    test_X = np.array(test_data['test_X'])  /255.0

    train_Y = np.array(train_data['train_Y'])
    train_X = np.array(train_data['train_X']) /255.0

    # Check the shape of the data
    print(test_X.shape)
    print(train_X.shape)

    # Fix the axes of the images
    test_X = np.moveaxis(test_X, -1, 0)
    train_X = np.moveaxis(train_X, -1, 0)

    print(test_X.shape)
    print(train_X.shape)
```

```
# Plot a random image and its label

plt.imshow(train_X[350])
plt.show()
print(train_Y[350])

#reshape train Y to vector format
print(test_Y)

#replace 10 to 0s in ys
train_Y = np.where(train_Y==10, 0, train_Y)
test_Y = np.where(test_Y==10, 0, test_Y)

a = 5

print(test_Y[a])
print(test_Y[a+1])

def unique(list1):
    x = np.array(list1)
    print(np.unique(x))

print("unique")

unique(test_Y)




def build_model(num_classes, output_activation=None):#week 4 lec/week 5 prac
    # our model, input in an image shape
    inputs = keras.Input(shape=(32, 32, 3,))

    # run pairs of conv layers, all 3s3 kernels
    x = keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same',
    activation=None)(inputs)
    x = keras.layers.Conv2D(filters=32, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)

    # rinse and repeat with 2D convs, batch norm, dropout and max pool
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.Conv2D(filters=64, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)

    # final conv2d, batch norm and spatial dropout
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.Conv2D(filters=128, kernel_size=(3,3), padding='same',
    activation=None)(x)
    x = keras.layers.BatchNormalization()(x)
```

```
    x = keras.layers.SpatialDropout2D(0.2)(x)
    x = keras.layers.MaxPool2D(pool_size=(2, 2))(x)

    # flatten layer
    x = keras.layers.Flatten()(x)
    # we'll use a couple of dense layers here, mainly so that we can show what another
    dropout layer looks like
    # in the middle
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(128, activation='relu')(x)
    x = keras.layers.Dense(64, activation='relu')(x)
    # the output
    outputs = keras.layers.Dense(num_classes, activation=output_activation)(x)

    # build the model, and print a summary
    model_cnn = keras.Model(inputs=inputs, outputs=outputs, name='cnn_model')

    return model_cnn

model = build_model(10)
model.summary()

model.compile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          optimizer=keras.optimizers.Adam(),
          metrics=['accuracy'])
#Data augmentations

datagen = ImageDataGenerator(
                        rotation_range=10,
                        zoom_range=[0.9, 1.1],
                        height_shift_range=0.10,
                        shear_range=0.15,
                        #channel_shift_range=100,
                        #brightness_range=(0.3,0.9)
                        )



#model.fit(train_X,train_Y,batch_size = 32,epochs=50)
his = model.fit_generator(datagen.flow(train_X, train_Y, batch_size=40),epochs=250)
print("--------------")
#print(his.history)
print("--------------")

predictions = model.predict(test_X)

a = 5

print(predictions[a])
print(test_Y[a])
print(predictions[a+1])
print(test_Y[a+1])
print(predictions[a+2])
print(test_Y[a+2])
```

```
    def eval_model(model, x_test, y_test):
        test_scores = model.evaluate(x_test, y_test, verbose=2)
        print('Test loss:', test_scores[0])
        print('Test accuracy:', test_scores[1])

        pred = model.predict(x_test);
        indexes = tf.argmax(pred, axis=1)

        cm = confusion_matrix(y_test, indexes)
        fig = plt.figure(figsize=[20, 6])
        ax = fig.add_subplot(1, 2, 1)
        c = ConfusionMatrixDisplay(cm, display_labels=range(len(numpy.unique(y_test))))
        c.plot(ax = ax)

        ax = fig.add_subplot(1, 2, 2)
        ax.hist(y_test, bins=len(numpy.diagonal(cm)), rwidth=0.95)
        ax.plot(numpy.diagonal(cm))

    eval_model(model, train_X, train_Y)
    eval_model(model, test_X, test_Y)

    pred = model.predict(test_X);
    indexes = tf.argmax(pred, axis=1)
    count = 0
    print("_check_")
    print(test_Y[9])
    print(indexes[9].numpy())
    print("____test")
    for i in range(10000):
        if test_Y[i] == indexes[i].numpy():
            count = count + 1
    print(count)
    print(len(indexes))
    print((count/len(indexes))*100)
    print("____train")
    pred = model.predict(train_X);
    indexes = tf.argmax(pred, axis=1)
    count = 0
    for i in range(1000):
        if train_Y[i] == indexes[i].numpy():
            count = count + 1
    print(count)
    print(len(indexes))
    print((count/len(indexes))*100)

    print("count elements")
    print(np.count_nonzero(test_Y == 0))
    print(np.count_nonzero(test_Y == 1))
    print(np.count_nonzero(test_Y == 2))
    print(np.count_nonzero(test_Y == 3))
    print(np.count_nonzero(test_Y == 4))
    print(np.count_nonzero(test_Y == 5))
    print(np.count_nonzero(test_Y == 6))
    print(np.count_nonzero(test_Y == 7))
    print(np.count_nonzero(test_Y == 8))
    print(np.count_nonzero(test_Y == 9))
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat May  1 12:35:52 2021

@author: User
"""

import scipy.io
import matplotlib.pyplot as plt

import tensorflow as tf
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

import numpy as np
import numpy
import tensorflow.keras as keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
from sklearn.metrics import confusion_matrix
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.utils import to_categorical
import seaborn as sns

from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array, load_img
from sklearn.model_selection import train_test_split


if __name__=="__main__":

    #import data
    test_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_test.mat')
    train_data = scipy.io.loadmat('CAB420_Assessment_1B_Data\Data\Q1\q1_train.mat')

    # Load images and labels
    test_Y = np.array(test_data['test_Y'])
    test_X = np.array(test_data['test_X'])  /255.0

    train_Y = np.array(train_data['train_Y'])
    train_X = np.array(train_data['train_X']) /255.0

    # Check the shape of the data
    print(test_X.shape)
    print(train_X.shape)

    # Fix the axes of the images
    test_X = np.moveaxis(test_X, -1, 0)
    train_X = np.moveaxis(train_X, -1, 0)

    print(test_X.shape)
    print(train_X.shape)

    # Plot a random image and its label

    plt.imshow(train_X[350])
    plt.show()
    print(train_Y[350])
```

```
#reshape train Y to vector format
print(test_Y)

#replace 10 to 0s in ys
train_Y = np.where(train_Y==10, 0, train_Y)
test_Y = np.where(test_Y==10, 0, test_Y)

a = 5

print(test_Y[a])
print(test_Y[a+1])

def unique(list1):
    x = np.array(list1)
    print(np.unique(x))

print("unique")

unique(test_Y)

datagen = ImageDataGenerator(
                        rotation_range=10,
                        zoom_range=[0.9, 1.1],
                        height_shift_range=0.10,
                        shear_range=0.15,
                        #channel_shift_range=100,
                        #brightness_range=(0.1, 0.9)
                        )

model = keras.models.load_model('vgg_2stage_CIFAR_small.h5')
model.summary()

model.compile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          optimizer=keras.optimizers.SGD(),#(lr=1e-4, momentum=0.9),
          metrics=['accuracy'])
#model.fit_generator(datagen.flow(train_X, train_Y, batch_size=40),epochs=250)
model.fit(train_X,train_Y,batch_size = 40,epochs=50)
#model.fit(train_X, train_Y,
#      batch_size=128,
#      epochs=10,
#      validation_data=(test_X, test_Y))



def eval_model(model, x_test, y_test):
   test_scores = model.evaluate(x_test, y_test, verbose=2)
   print('Test loss:', test_scores[0])
   print('Test accuracy:', test_scores[1])

   pred = model.predict(x_test);
   indexes = tf.argmax(pred, axis=1)

   cm = confusion_matrix(y_test, indexes)
   fig = plt.figure(figsize=[20, 6])
   ax = fig.add_subplot(1, 2, 1)
   c = ConfusionMatrixDisplay(cm, display_labels=range(len(numpy.unique(y_test))))
```

```
    c.plot(ax = ax)

    ax = fig.add_subplot(1, 2, 2)
    ax.hist(y_test, bins=len(numpy.diagonal(cm)), rwidth=0.95)
    ax.plot(numpy.diagonal(cm))

eval_model(model, train_X, train_Y)
eval_model(model, test_X, test_Y)

pred = model.predict(test_X);
indexes = tf.argmax(pred, axis=1)
count = 0
print("_check_")
print(test_Y[9])
print(indexes[9].numpy())
print("____test")
for i in range(10000):
    if test_Y[i] == indexes[i].numpy():
        count = count + 1
print(count)
print(len(indexes))
print((count/len(indexes))*100)
print("____train")
pred = model.predict(train_X);
indexes = tf.argmax(pred, axis=1)
count = 0
for i in range(1000):
    if train_Y[i] == indexes[i].numpy():
        count = count + 1
print(count)
print(len(indexes))
print((count/len(indexes))*100)
```