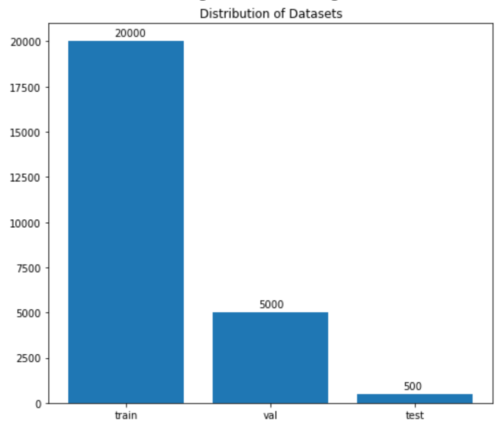
Nanyang Technological University

EE4483 Mini Project (Option 2)

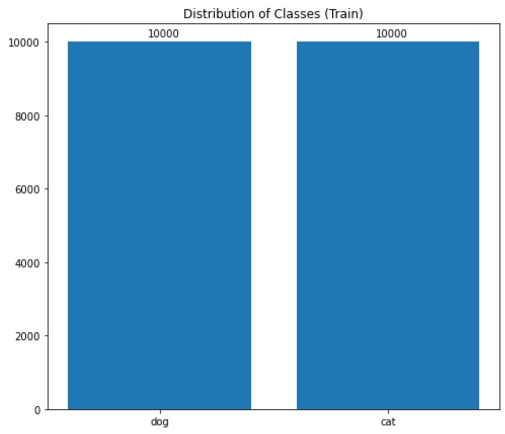
Koo Jie Hui U1920958F

a) State the amount of image data that you used to form your training set and testing set. Describe the data pre-processing procedures and image augmentations if any

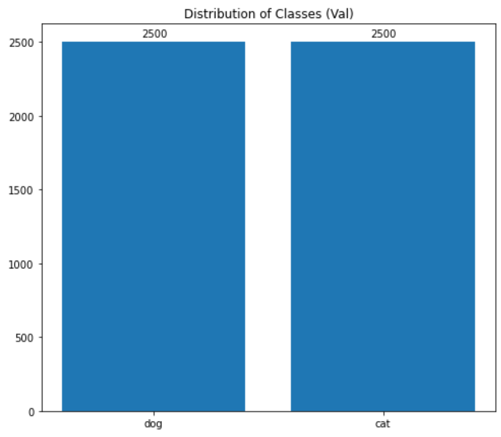
Distribution of Training, Testing, and Validation Set



Distribution of classes of Training Set



Distribution of classes of Validation Set



Pre-processing

Images are resized to 64x64 pixels, with aspect ratio not preserved. This may potentially affect model performance. However, this is still done for the following reasons.

1) Images are resized to a smaller size, 64x64, to reduce computational complexity.
2) Model architecture is designed, with the convolution and fully connected layers defined at the start. Therefore, a uniform image size is required for all images to be able to be input into the same Convolution layer.
3) If aspect ratio is to be preserved, padding with a value, i.e. zero, will have to be done. However, this will increase model complexity as the model would have to learn to ignore the padded areas. Therefore, images are simply resized to 64x64 regardless of aspect ratio.
4) As all images are resized to 64x64 regardless of aspect ratio, all images will be distorted to a roughly similar amount

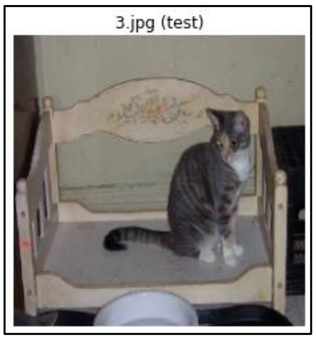


Figure 1: 64x64 Image

Normalisation of images are also done, with each pixel value normalised to a value to between 0 to 1. This is achieved by dividing each pixel value by 255. The normalisation is done for training, validation and testing data.

Next, shuffling of training data is done so as to remove biases. Biases includes examples such as feeding all cat photos first, before feeding all dog photos.

Lastly, data augmentation is done for model 4. Data augmentation includes random rotation, random translation and random contrast. This increases the variance of the training set to combat overfitting, as it has a regularisation effect. Moreover, it encourages the model to be invariant to rotations, translations and contrast.

b) Select or build at least one machine learning model (e.g., CNN + linear layer, etc.) to construct your classifier. Clearly describe the model you use, including a figure of model architecture, the input and output dimensions, structure of the model, loss function(s), training strategy, etc. Include your code and instructions on how to run the code. If non-deterministic method is used, ensure reproducibility of your results by averaging the results over multiple runs, cross-validation, fixing random seed, etc.

Link to code here:

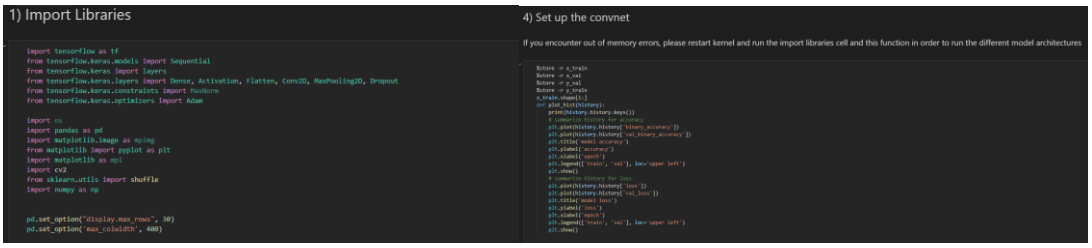
https://entuedu-my.sharepoint.com/:u:/g/personal/jkoo007_e_ntu_edu_sg/EQF3a7FKgBBGhWAwUdxcealB5wTov-Vmh171PK1uA1B3LA?e=grsSDo

The following 4 models were built on an iterative basis. Based on the results and behaviour of the preceding models, model 4 is the final model that is used to obtain submission.csv.

It is to be noted that further works include the use of Keras dataset library to import training data, instead of concatenating training data into a list, to reduce memory usage.

When running the code, if there are out of memory errors when attempting to generate the outputs of the different models, please do the following.

1) restart the jupyter kernel
2) run the import libraries cell (1) and cell (4)



1) Import Libraries

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense, Activation, Flatten, Conv2D, MaxPooling2D, Dropout
from tensorflow.keras.constraints import MaxNorm
from tensorflow.keras.optimizers import Adam

import os
import pandas as pd
import matplotlib.image as mpimg
from matplotlib import pyplot as plt
import matplotlib as mpl
import cv2
from sklearn.utils import shuffle
import numpy as np


pd.set_option("display.max_rows", 30)
pd.set_option('max_colwidth', 400)
```

4) Set up the convnet

If you encounter out of memory errors, please restart kernel and run the import libraries cell and this function in order to run the different model architectures

```
%store -r x_train
%store -r x_val
%store -r y_val
%store -r y_train
x_train.shape[1:]
def plot_hist(history):
    print(history.history.keys())
    # summarize history for accuracy
    plt.plot(history.history['binary_accuracy'])
    plt.plot(history.history['val_binary_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    plt.show()
```

3) Run the cell for the model you would like to obtain the output for
4) For cifar10_Machine_Learning.ipynb, if there are out of memory errors please restart the IDE.

Model 1

Model Description

Type	Convolutional Neural Network (CNN)
Input Dimensions	(64, 64, 3), image width = 64, image height = 64, RGB
Output Dimensions	(1), a single float value between 0 to 1. Since it is binary classification, only 1 output node is needed to obtain a result. A 2 output node model will work as well, but with 1 lesser node, there would be lesser parameters and therefore for each iteration, the computation required is lesser. Therefore, one output node is used.
Output classes	0 = cat, 1 = dog
Accuracy calculation:	Binary accuracy, comparison between output class and rounded output of sigmoid function
Loss function	Binary cross entropy

Training Strategy

Number of Epochs	50
Learning Rate	0.0001
Batch Size	32

Architecture

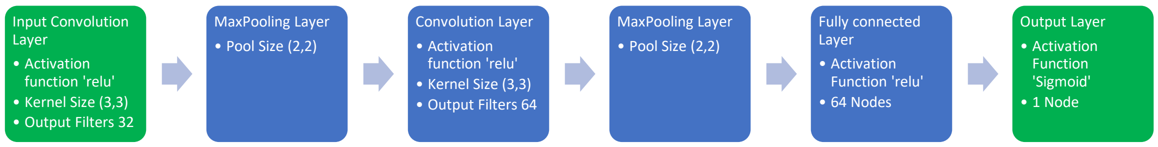
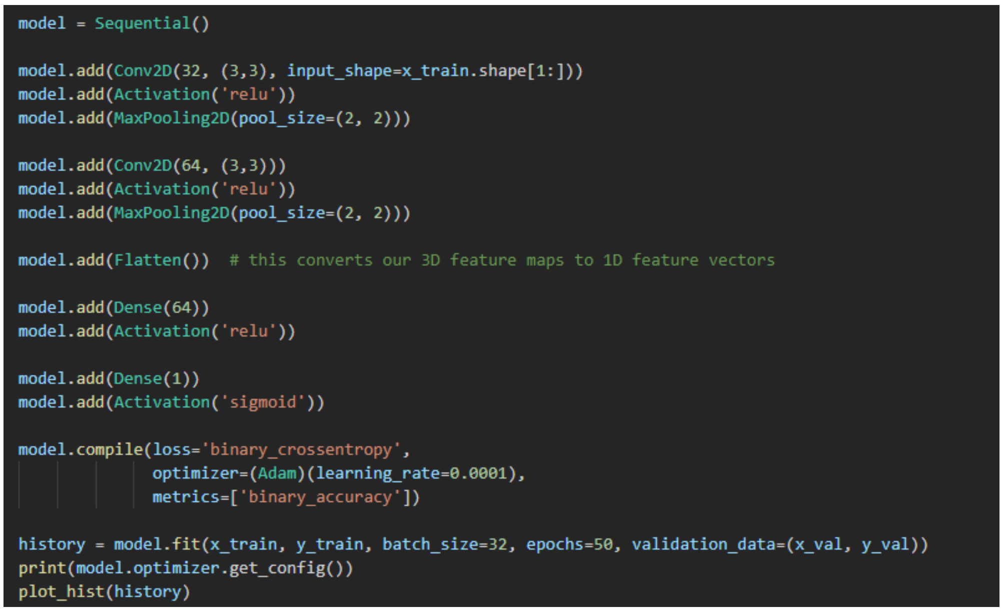


Figure 2: Architecture of Model 1



```
model = Sequential()

model.add(Conv2D(32, (3,3), input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=(Adam)(learning_rate=0.0001),
              metrics=['binary_accuracy'])

history = model.fit(x_train, y_train, batch_size=32, epochs=50, validation_data=(x_val, y_val))
print(model.optimizer.get_config())
plot_hist(history)
```

Figure 3: Code of Model 1

Results

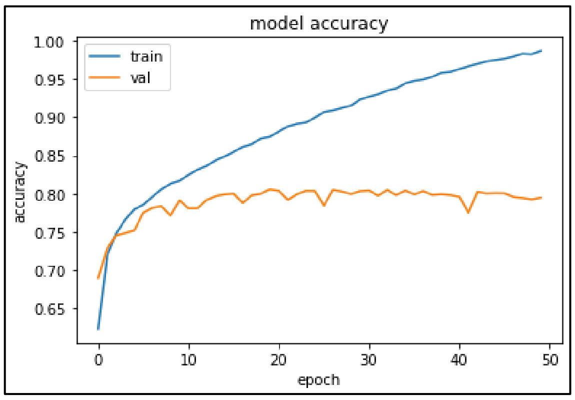


Figure 4: Model 1 Accuracy Learning Curve

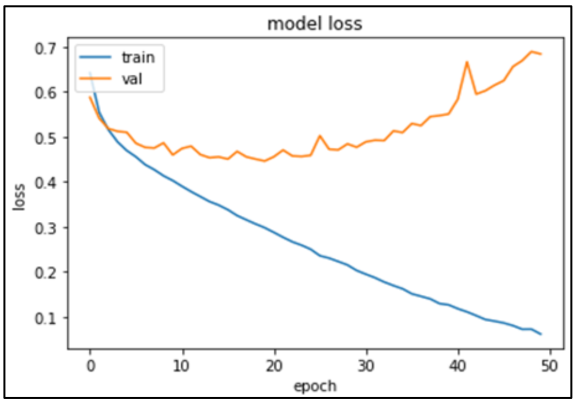


Figure 5: Model 1 Loss Learning Curve

This model serves as a starting point for the next few machine learning models.

It can be seen from figure 5 that there is significant overfitting. This is because the model is continually learning from the training set well but fails to generalize to the validation data. This is evidenced by the continually falling training loss but 'u' shape increasing validation loss as the number of epochs occur.

It is to be noted that the validation accuracy converges, but trends downwards slightly, while the validation loss increases significantly. One reason could be that the bad predictions are continually worsening as the number of epochs increases, but due to thresholding, the accuracy remains relatively stable. For example, a dog prediction of 0.7 worsening to 0.6, is still classed as a dog. This behaviour will be further amplified as binary cross-entropy loss function is used, which penalises bad predictions far more strongly than good predictions.

Model 2

Model Description

Type	Convolutional Neural Network (CNN)
Input Dimensions	(64, 64, 3), image width = 64, image height = 64, RGB
Output Dimensions	(1), a single float value between 0 to 1
Output classes	0 = cat, 1 = dog
Accuracy calculation:	Binary accuracy, comparison between output class and rounded output of sigmoid function
Loss function	Binary cross entropy

Training Strategy

Number of Epochs	40 Reduced number of epochs as it is expected that overfitting will occur at a lower number of epochs.
Learning Rate	0.0001
Batch Size	32
Additional Notes	To examine the effect of additional complexity, another fully connected layer is added.

Architecture

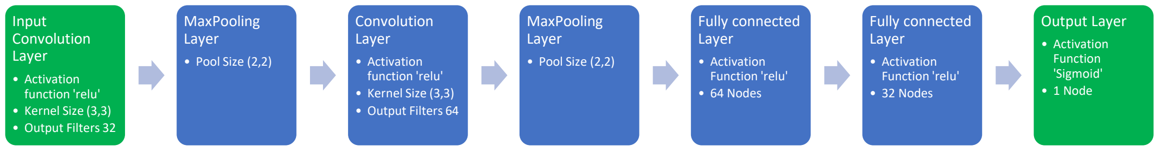
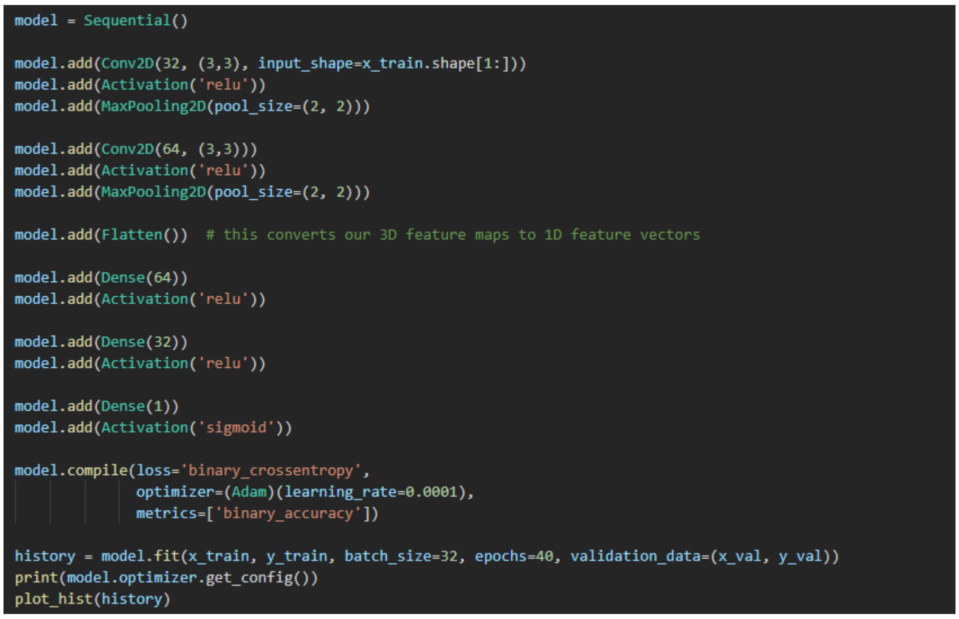


Figure 6: Architecture of Model 2



```
model = Sequential()

model.add(Conv2D(32, (3,3), input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))
model.add(Activation('relu'))

model.add(Dense(32))
model.add(Activation('relu'))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=(Adam)(learning_rate=0.0001),
              metrics=['binary_accuracy'])

history = model.fit(x_train, y_train, batch_size=32, epochs=40, validation_data=(x_val, y_val))
print(model.optimizer.get_config())
plot_hist(history)
```

Figure 7: Code of Model 2

Results

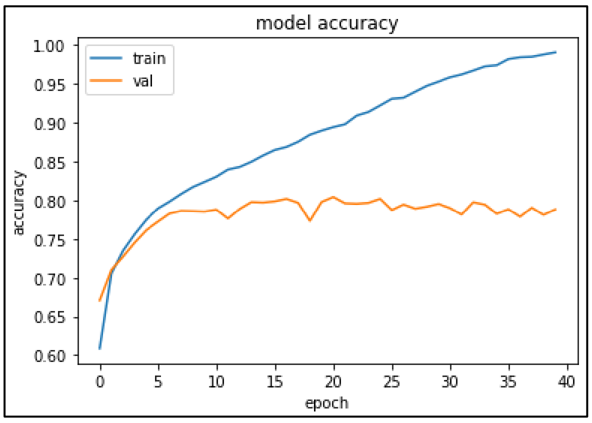


Figure 8: Model 2 Accuracy Learning Curve

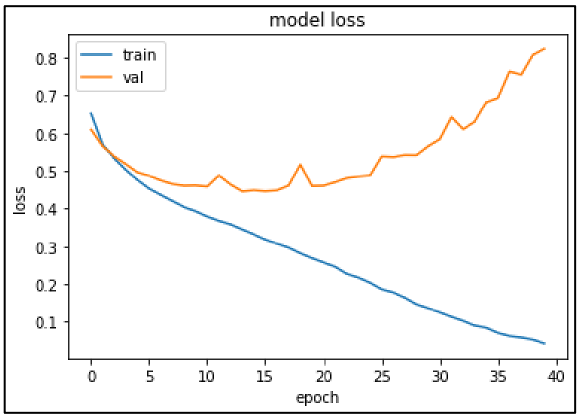


Figure 9: Model 2 Loss Learning Curve

Model 2 is the same as model 1, but with the addition of a fully connected layer right before the output layer.

We can observe that the validation loss of model 2 has increased at a faster rate as compared to model 1. Model 2 validation loss reached 0.7 at around epoch 35, while model 1 reached 0.7 at around epoch 50.

With the added complexity from the additional fully connected layer, model 2 adapted even more to the training data than model 1 causing worse performance. It failed to figure out general relationships, thus performing worse on the validation set.

Model 3

Model Description

Type	Convolutional Neural Network (CNN)
Input Dimensions	(64, 64, 3), image width = 64, image height = 64, RGB
Output Dimensions	(1), a single float value between 0 to 1
Output classes	0 = cat, 1 = dog
Accuracy calculation:	Binary accuracy, comparison between output class and rounded output of sigmoid function
Loss function	Binary cross entropy

Training Strategy

Number of Epochs	150 Increased number of epochs as it is expected that with some regularisation, it will take the model longer to learn from training data
Learning Rate	0.0001
Batch Size	32
Additional Notes	Dropouts are added to reduce model complexity and reduce overfitting. However, an additional convolution layer is added to extract more features. Even though more complexity is added, dropouts should combat overfitting.

Architecture

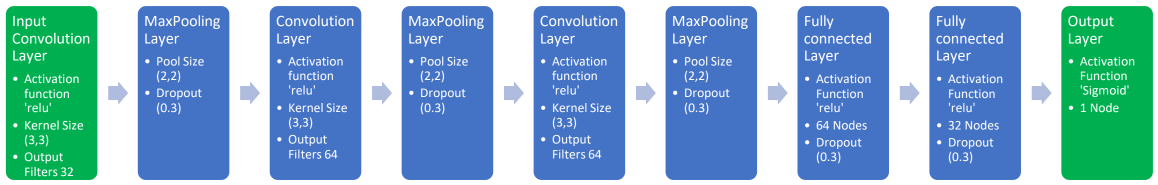
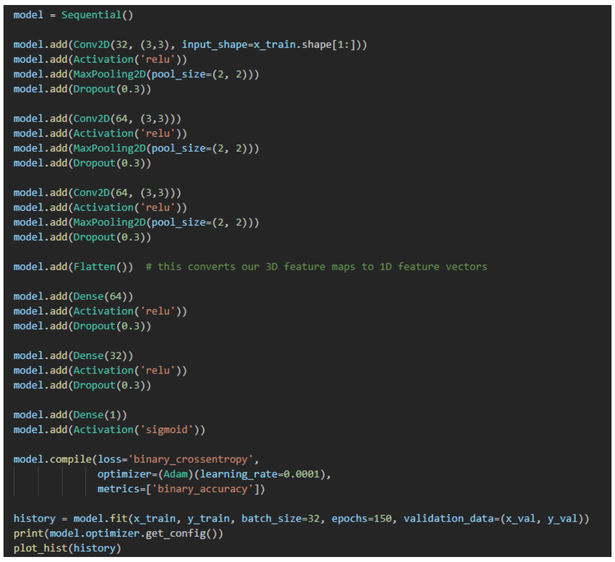


Figure 10: Architecture of Model 3



```python
model = Sequential()

model.add(Conv2D(32, (3,3), input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=(Adam)(learning_rate=0.0001),
              metrics=['binary_accuracy'])

history = model.fit(x_train, y_train, batch_size=32, epochs=150, validation_data=(x_val, y_val))
print(model.optimizer.get_config())
plot_hist(history)
```

Figure 11: Code of Model 3

Results

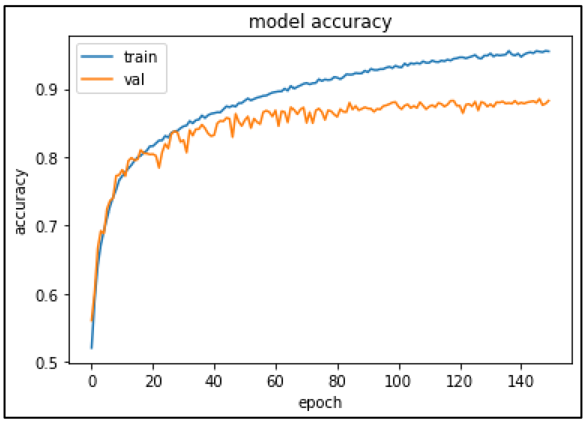


Figure 12: Model 3 Accuracy Learning Curve

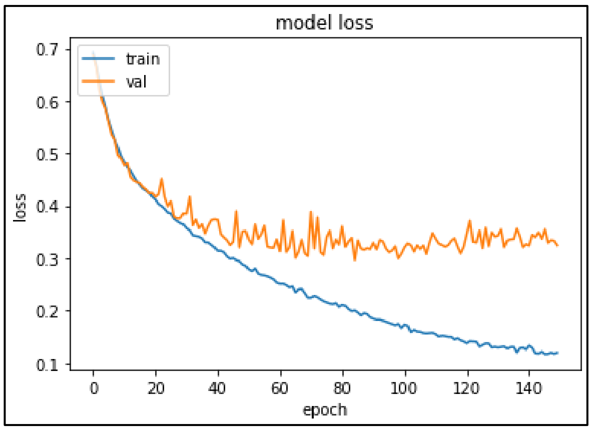


Figure 13: Model 3 Loss Learning Curve

We can see that despite the added complexity with the additional convolutional layer, with dropouts after every layer, overfitting has been reduced substantially. Accuracy has also increased, reaching more than 80% by epoch 40, while validation loss has largely converged. However, we still experience overfitting, and therefore it may be wise to apply additional regularisation techniques.

Model 4

Model Description

Type	Convolutional Neural Network (CNN)
Input Dimensions	(64, 64, 3), image width = 64, image height = 64, RGB
Output Dimensions	(1), a single float value between 0 to 1
Output classes	0 = cat, 1 = dog
Accuracy calculation:	Binary accuracy, comparison between output class and rounded output of sigmoid function
Loss function	Binary cross entropy

Training Strategy

Number of Epochs	300 Epochs were used, as it is expected that with data augmentation, the model will take additional epochs to learn from the training data.
Learning Rate	0.00012 Increased learning rate is applied to encourage convergence at an earlier epoch.
Batch Size	32
Additional Notes	Data augmentation was added to expand the number of training samples. This allows the model to train on a bigger variety of pictures with differing locations/orientation/contrast of cats/dogs. This will have a bigger regularisation effect on the model.

Architecture

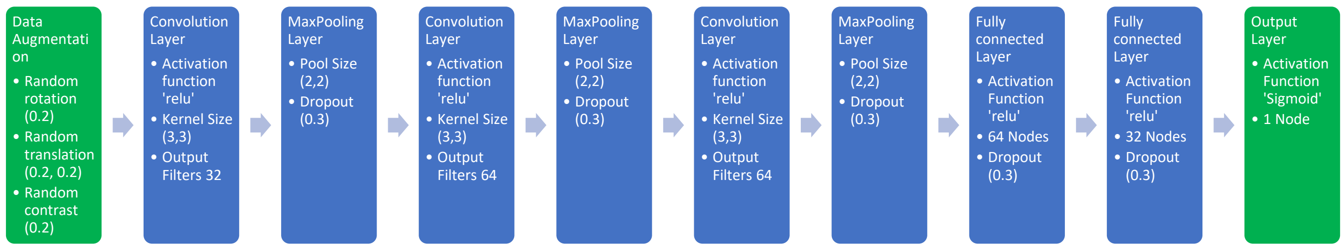
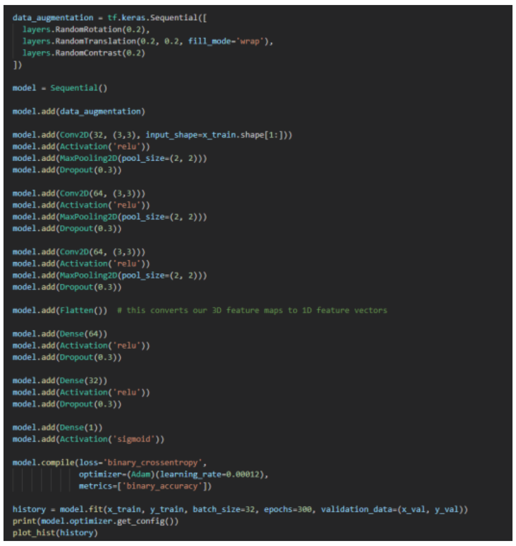


Figure 14: Architecture of Model 4



```
data_augmentation = tf.keras.Sequential([
  layers.RandomRotation(0.2),
  layers.RandomTranslation(0.2, 0.2, fill_mode='wrap'),
  layers.RandomContrast(0.2)
])

model = Sequential()

model.add(data_augmentation)

model.add(Conv2D(32, (3,3), input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=(Adam)(learning_rate=0.00012),
              metrics=['binary_accuracy'])

history = model.fit(x_train, y_train, batch_size=32, epochs=300, validation_data=(x_val, y_val))
print(model.optimizer.get_config())
plot_hist(history)
```

Figure 15: Code of Model 4

Results

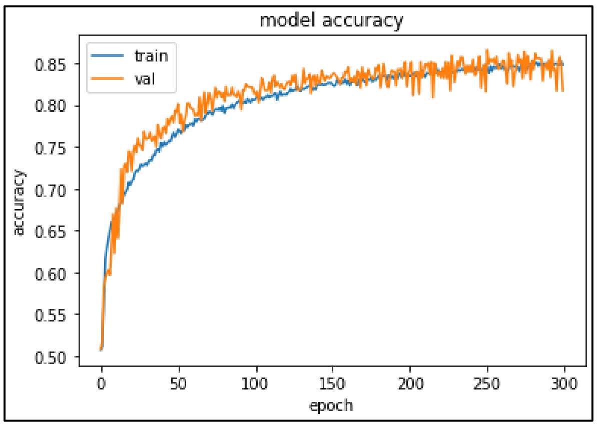


Figure 16: Model 4 Accuracy Learning Curve

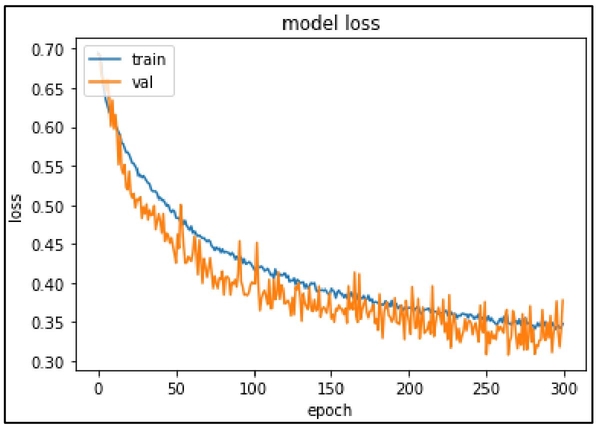


Figure 17: Model 4 Loss Learning Curve

We can observe significantly reduced overfitting in the model. Even at 300 epochs, we do not see as much overfitting. With an increased learning rate of 0.00012, we see a saturation of validation accuracy rate at around 80%. Learning rate is increased so as to achieve convergence at a lower epoch. Early stopping is also done, as the accuracy rate does not seem to change as much at epoch 150 to epoch 300.

It can also be observed that at lower epochs, the validation loss is lower than the training loss. This is because with data augmentation, there is a larger variety of training images, thus the model finds it more difficult to learn and adapt to the training data. However, as more epochs pass, the model gradually is still able to learn from the training data.

c) Discuss how you consider and determine the parameters (e.g., learning rate, etc.) / settings of your model as well as your reasons of doing so. (10% marks)

The training parameters of model 4 were largely derived from the preceding models. Initially, in model 1, we can see that there is significant overfitting. This is due to the model adapting very well to the training data and thus failing to generalise to validation data. This observation is further proven by model 2, where adding more complexity has increased the overfitting problem.

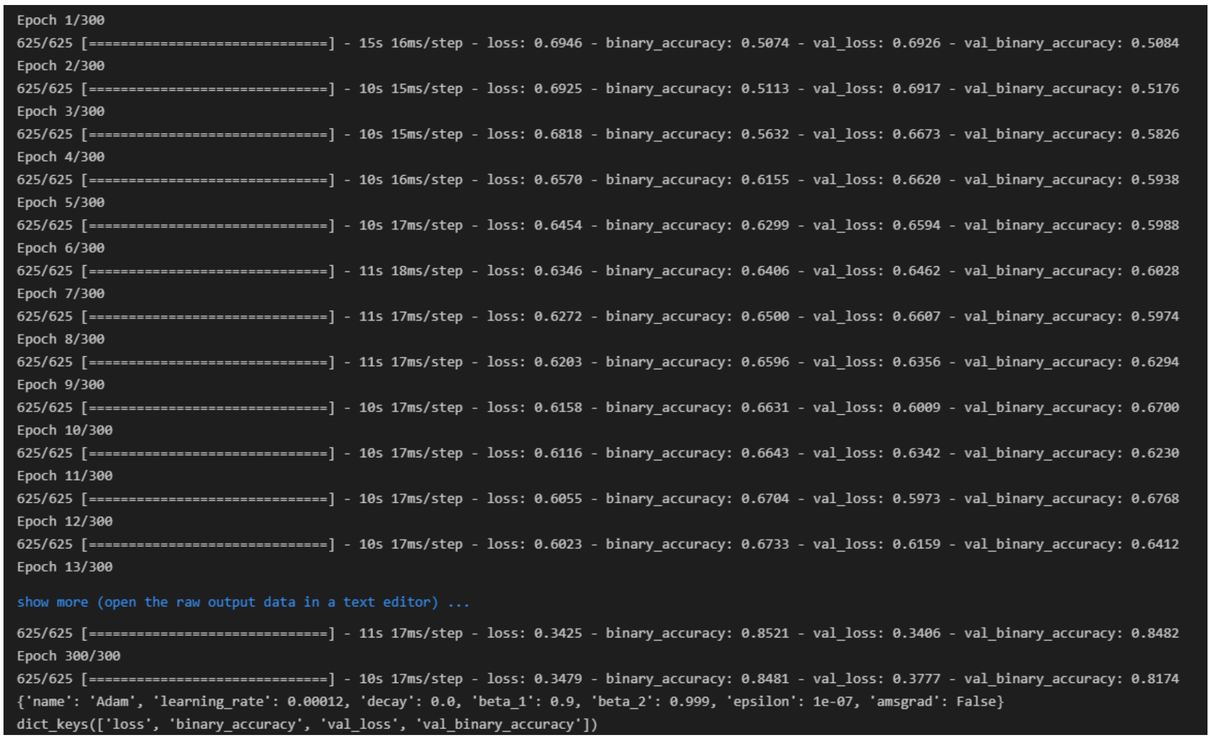
By adding dropouts in model 3, we can see a drastic reduction in overfitting, while the addition of data augmentation has further reduced the overfitting.

Lastly, for model 4, with training data augmentation, it is more difficult for the model to adapt to the training data. Therefore, we can see that initially, validation loss is lower than the training loss. Learning rate is also slightly increased to 0.00012 to encourage convergence more quickly.

Importantly, maintaining the balance between bias and variance is crucial. This is kept in mind when deciding the parameters and deciding what complexities to add or remove across models 1 to 4. If the model is too simple, the model may have a higher probability to suffer from high bias and low variance. This can lead to high error rates on both train and test data sets. However, if the model is too complex, the model may have a higher probability to suffer from low bias and high variance. This means that the model adapts too well to the training data as well as the noise in the training data, leading to low error on training data but high error on test data. As the number of convolution or fully connected layers increase, the complexity of the model increases.

d) Report the classification accuracy on validation set. Apply the classifier(s) built to the test set. Submit the "submission.csv" with the results you obtained. (20% marks)

Accuracy of the validation set is 81.74%.



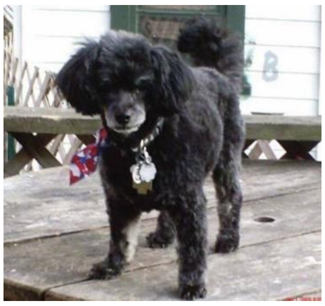
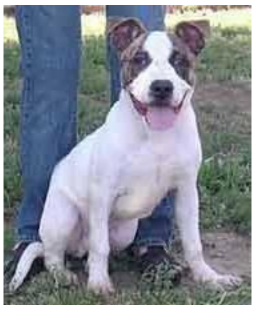
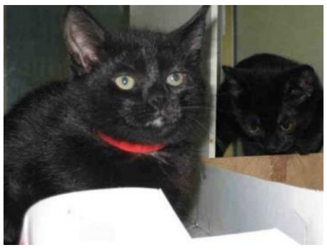
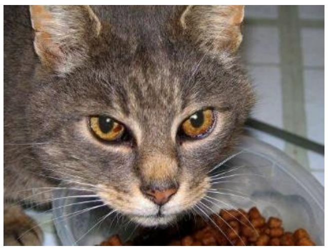
```
Epoch 1/300
625/625 [==============================] - 15s 16ms/step - loss: 0.6946 - binary_accuracy: 0.5074 - val_loss: 0.6926 - val_binary_accuracy: 0.5084
Epoch 2/300
625/625 [==============================] - 10s 15ms/step - loss: 0.6925 - binary_accuracy: 0.5113 - val_loss: 0.6917 - val_binary_accuracy: 0.5176
Epoch 3/300
625/625 [==============================] - 10s 15ms/step - loss: 0.6818 - binary_accuracy: 0.5632 - val_loss: 0.6673 - val_binary_accuracy: 0.5826
Epoch 4/300
625/625 [==============================] - 10s 16ms/step - loss: 0.6570 - binary_accuracy: 0.6155 - val_loss: 0.6620 - val_binary_accuracy: 0.5938
Epoch 5/300
625/625 [==============================] - 10s 17ms/step - loss: 0.6454 - binary_accuracy: 0.6299 - val_loss: 0.6594 - val_binary_accuracy: 0.5988
Epoch 6/300
625/625 [==============================] - 11s 18ms/step - loss: 0.6346 - binary_accuracy: 0.6406 - val_loss: 0.6462 - val_binary_accuracy: 0.6028
Epoch 7/300
625/625 [==============================] - 11s 17ms/step - loss: 0.6272 - binary_accuracy: 0.6500 - val_loss: 0.6607 - val_binary_accuracy: 0.5974
Epoch 8/300
625/625 [==============================] - 11s 17ms/step - loss: 0.6203 - binary_accuracy: 0.6596 - val_loss: 0.6356 - val_binary_accuracy: 0.6294
Epoch 9/300
625/625 [==============================] - 10s 17ms/step - loss: 0.6158 - binary_accuracy: 0.6631 - val_loss: 0.6009 - val_binary_accuracy: 0.6700
Epoch 10/300
625/625 [==============================] - 10s 17ms/step - loss: 0.6116 - binary_accuracy: 0.6643 - val_loss: 0.6342 - val_binary_accuracy: 0.6230
Epoch 11/300
625/625 [==============================] - 10s 17ms/step - loss: 0.6055 - binary_accuracy: 0.6704 - val_loss: 0.5973 - val_binary_accuracy: 0.6768
Epoch 12/300
625/625 [==============================] - 10s 17ms/step - loss: 0.6023 - binary_accuracy: 0.6733 - val_loss: 0.6159 - val_binary_accuracy: 0.6412
Epoch 13/300

show more (open the raw output data in a text editor) ...

625/625 [==============================] - 11s 17ms/step - loss: 0.3425 - binary_accuracy: 0.8521 - val_loss: 0.3406 - val_binary_accuracy: 0.8482
Epoch 300/300
625/625 [==============================] - 10s 17ms/step - loss: 0.3479 - binary_accuracy: 0.8481 - val_loss: 0.3777 - val_binary_accuracy: 0.8174
{'name': 'Adam', 'learning_rate': 0.00012, 'decay': 0.0, 'beta_1': 0.9, 'beta_2': 0.999, 'epsilon': 1e-07, 'amsgrad': False}
dict_keys(['loss', 'binary_accuracy', 'val_loss', 'val_binary_accuracy'])
```

Figure 14: Epochs and their validation accuracies for model 4

e) Analyse some correctly and incorrectly classified (if any) samples in the test set. Select 1-2 cases to discuss the strength and weakness of the model. (10% marks)

Correctly classified

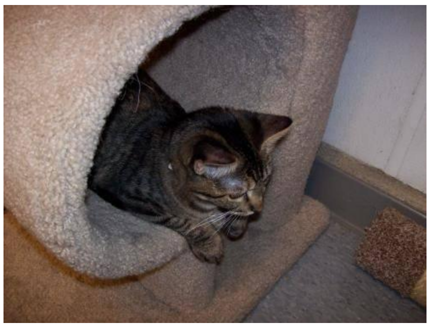
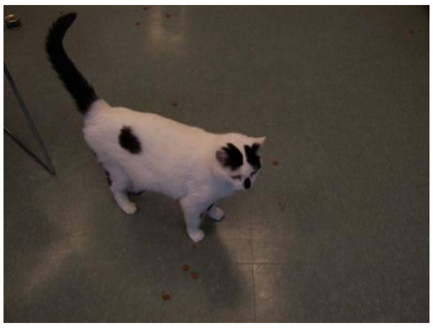
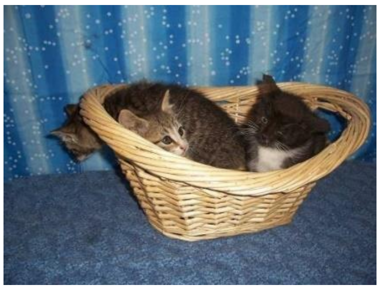
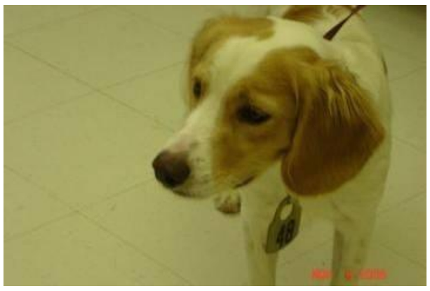
We can see for this picture that the subject stands out clearly from the background. This means that the pixel intensities of the subject are significantly different from the intensities of the background allowing the model to better extract the relevant edges of the subject. Moreover, the subject is not obscured, with features prominent.

Incorrectly classified

We can see that there are quite a few incorrectly classified pictures with a top down perspective of the subject. Data augmentation with vertical and horizontal flip as well as random perspective transform could improve the accuracy.

Another case would be where the body of the subject is obscured. Coupled with a background that is similar in colour to the subject, performance could be impacted. Lastly, performance can also be impacted by the number of subjects in the picture. These subjects may obscure each other's body.

f) Discuss how different choice of models and data processing may affect the project in terms of accuracy on validation set. (10% marks)

Please refer to section b and c

g) Apply and improve your classification algorithm to a multi-category image classification problem for Cifar-10 dataset (https://www.cs.toronto.edu/~kriz/cifar.html). Describe details about the dataset, classification problem that you are solving, and how your algorithm can tackle this problem. Report your results for the testing set of Cifar-10. (20% marks)

Model 5

Model Description

Type	Convolutional Neural Network (CNN)
Input Dimensions	(32, 32, 3), image width = 32, image height = 32, RGB
Output Dimensions	(10), 10 nodes. When testing, a list is obtained. The values in the list are theresholded, and the Index with a thresholded 1 indicates the class. Index 0 to 9 are mapped to the following classes. Airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck
Output classes	10 classes
Accuracy calculation:	Comparison between output class and rounded output of sigmoid function
Loss function	Categorical cross entropy

Training Strategy

Number of Epochs	300 Epochs were used, similar to model 4
Learning Rate	0.0001
Batch Size	32
Additional Notes	Data augmentation was removed as the training set has 50 thousand samples, which should be enough in terms of training data variety.

Splitting of training data into training and validation dataset

This is done to obtain the parameters for the model without letting the model see the test set. Ideally, the number of classes for training and validation set should be the same for each class. The random train test split gives roughly the same number of data samples for each class.

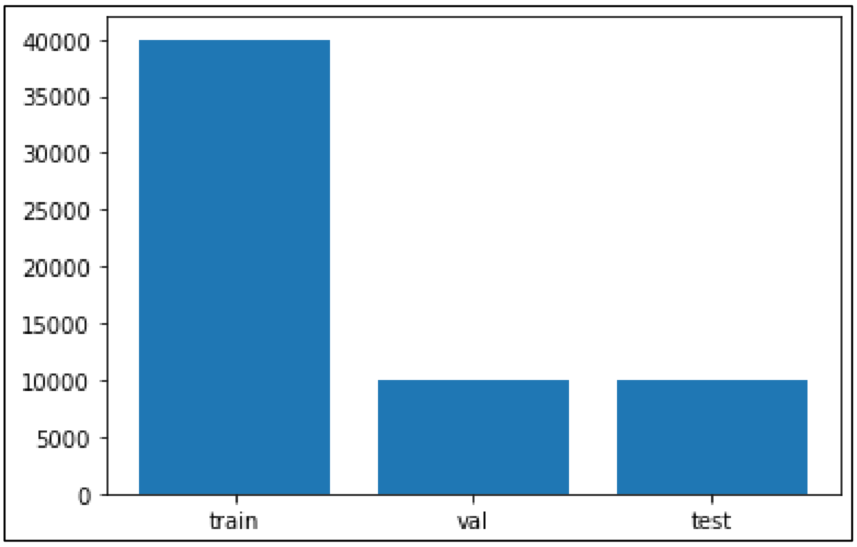


Figure 15: Distribution of Train, Validation and Test data samples

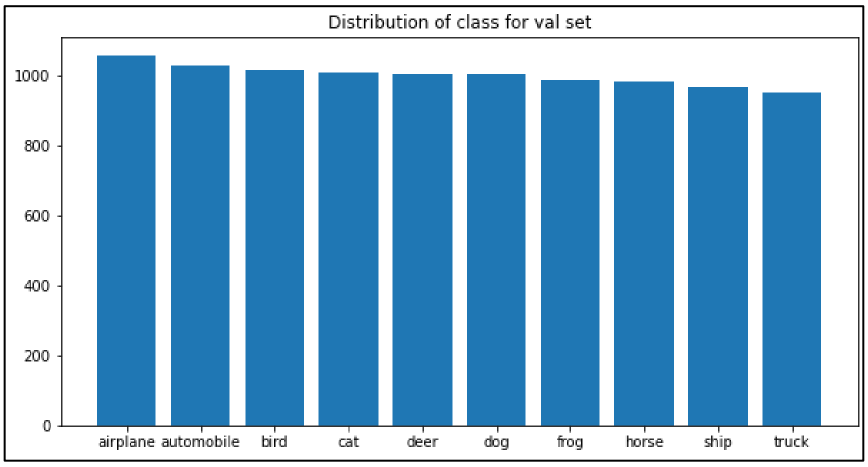


Figure 16: Distribution of classes for validation set

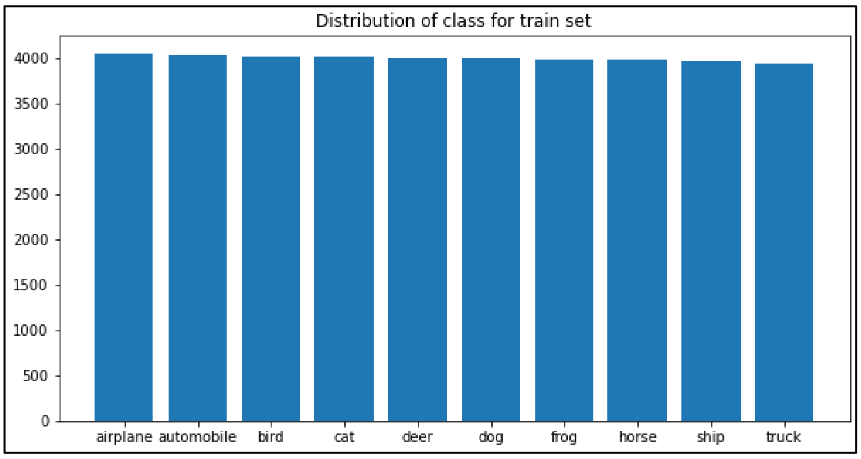


Figure 17: Distribution of classes for training set

Architecture

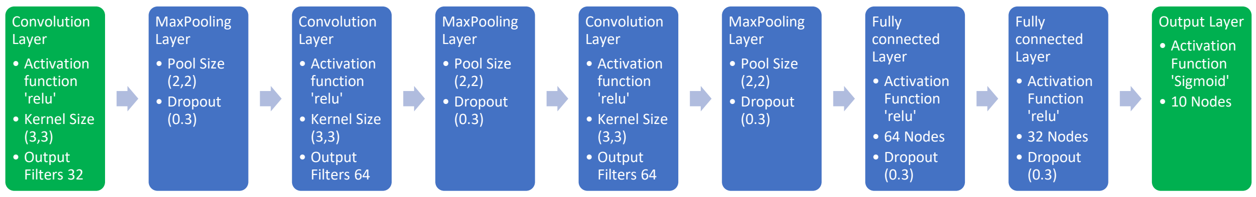
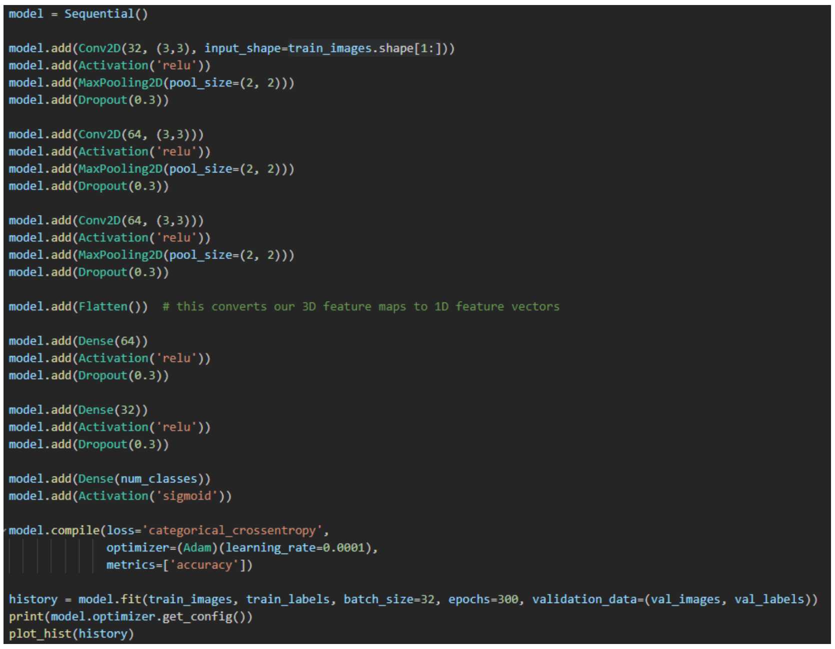


Figure 18: Architecture of Model 5



```
model = Sequential()

model.add(Conv2D(32, (3,3), input_shape=train_images.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Conv2D(64, (3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(32))
model.add(Activation('relu'))
model.add(Dropout(0.3))

model.add(Dense(num_classes))
model.add(Activation('sigmoid'))

model.compile(loss='categorical_crossentropy',
              optimizer=(Adam)(learning_rate=0.0001),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, batch_size=32, epochs=300, validation_data=(val_images, val_labels))
print(model.optimizer.get_config())
plot_hist(history)
```

Figure 19: Code of Model 5

Results

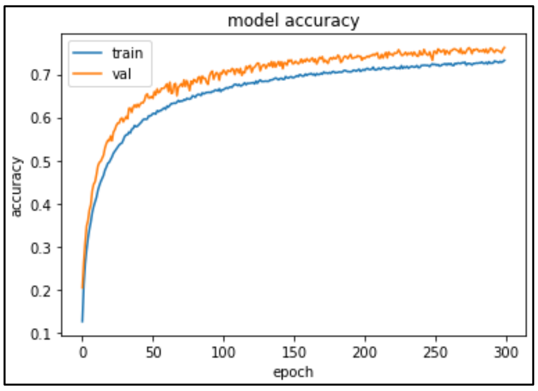


Figure 20: Model 5 Accuracy Learning Curve

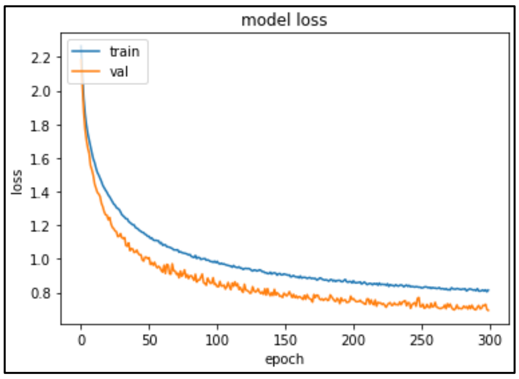
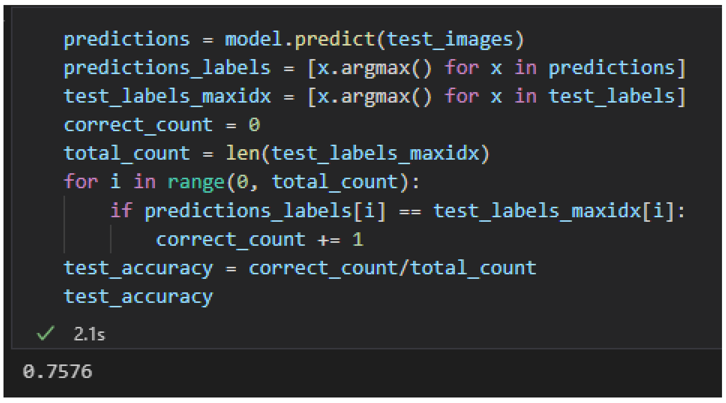


Figure 21: Model 5 Loss Learning Curve

We can observe that the validation loss is lower than the training loss. This could be because there are significantly more training data samples than the validation data. Thus, there is a larger variety of training data samples as compared to validation data. In other words, the training dataset is more "difficult" than the validation data.

When model is fit onto the test set, the accuracy garnered was about 75.76 %.



```
predictions = model.predict(test_images)
predictions_labels = [x.argmax() for x in predictions]
test_labels_maxidx = [x.argmax() for x in test_labels]
correct_count = 0
total_count = len(test_labels_maxidx)
for i in range(0, total_count):
    if predictions_labels[i] == test_labels_maxidx[i]:
        correct_count += 1
test_accuracy = correct_count/total_count
test_accuracy
```

✓ 2.1s

```
0.7576
```