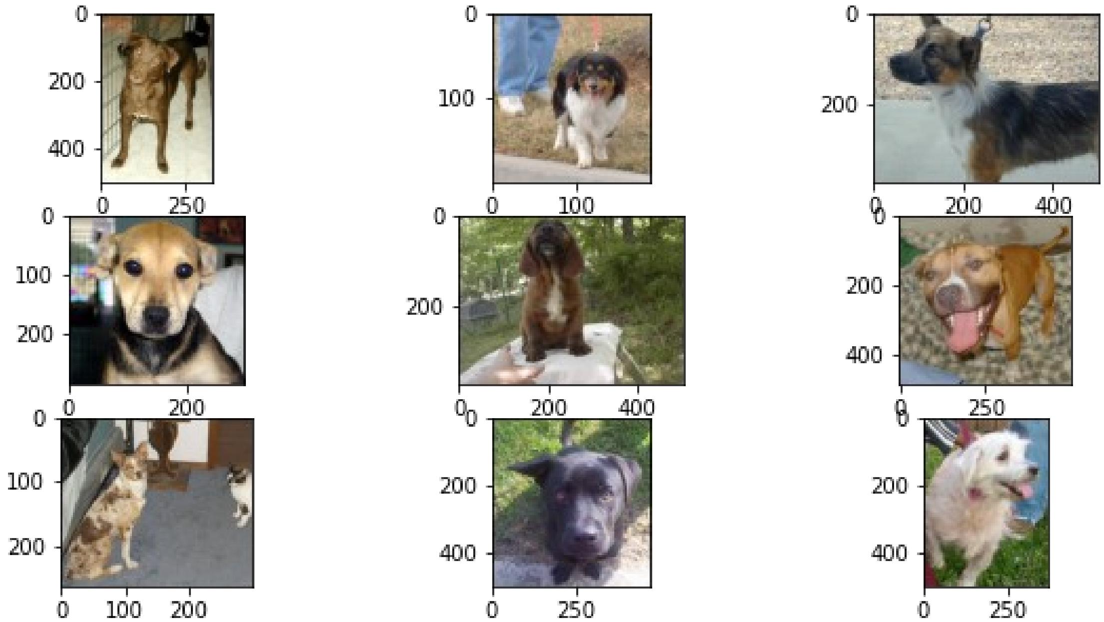
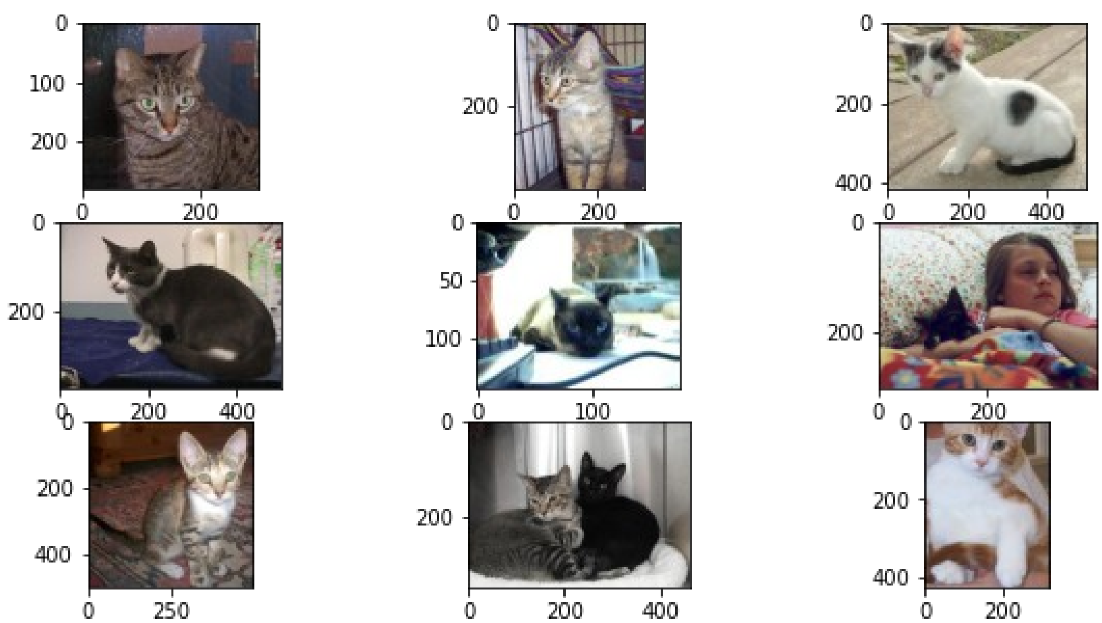
# Pet Classification Model Using CNN.

```
# import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.image import imread
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Dropout,
BatchNormalization, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import load_img,
ImageDataGenerator
import os
import random
from tabulate import tabulate
import warnings
warnings.filterwarnings('ignore')

# !unzip data.zip

# show images (train dogs)
plt.figure(figsize=(10, 5))
for i in range(9):
    j= i+1
    plt.subplot(3,3,j)
    img = imread('data/train/dogs/' + str(j) + '.jpg')
    plt.imshow(img)
```



```
# show images (train cats)
plt.figure(figsize=(10, 5))
for i in range(9):
    j= i+1
    plt.subplot(3,3,j)
    img = imread('data/train/cats/' + str(j) + '.jpg')

    plt.imshow(img)
```



```
# Prepare images for modelling
```

```
datagen2 = ImageDataGenerator(rescale = 1./255,shear_range =
0.2,zoom_range = 0.2,
                     validation_split = 0.2,
                     horizontal_flip = True,
                     featurewise_center=True,
                     featurewise_std_normalization=True,
                     rotation_range=90,
                     width_shift_range=0.1,
                     height_shift_range=0.1)

# trainset

training_set = datagen2.flow_from_directory('data/train',
classes=['dogs', 'cats'])
```

```
Found 40 images belonging to 2 classes.
```

```
validation_set = datagen2.flow_from_directory('data/test',
classes=['dogs', 'cats'])
```

```
Found 20 images belonging to 2 classes.
```

```
training_set.num_classes
```

```
2
```

```
training_set.image_shape
```

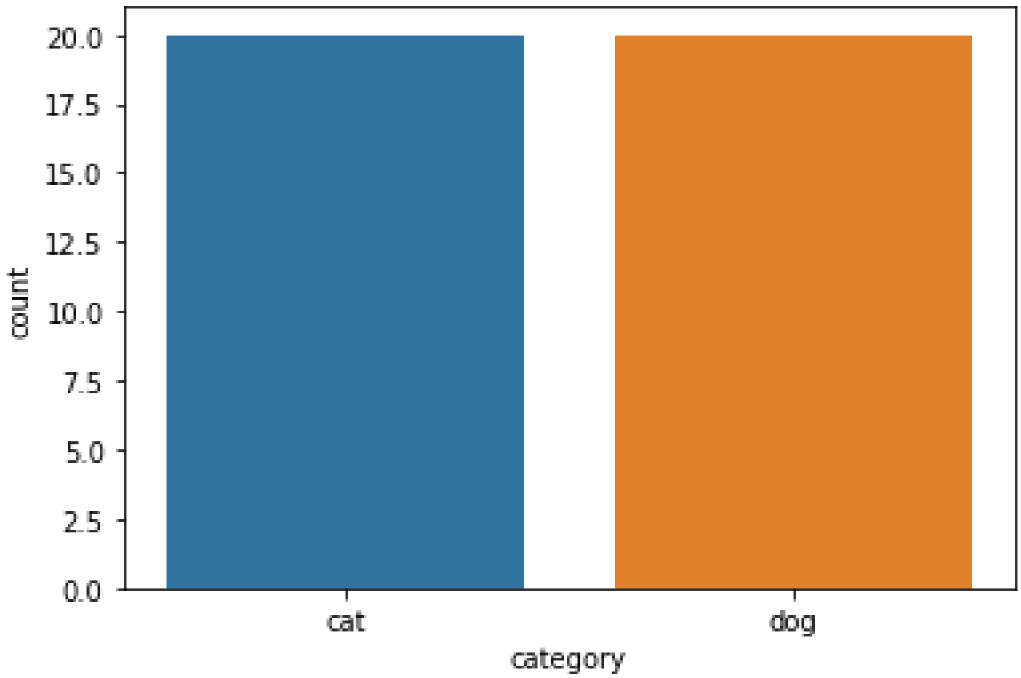
```
(256, 256, 3)
```

```
# list file names from cats and dogs folders in train
fn_cats = os.listdir('data/train/cats')
fn_dogs = os.listdir('data/train/dogs')

# categorize 0 for cats and 1 for dogs in train
categories = []

for image in fn_cats:
    category = image.split('.')[0]
    categories.append('cat')

for image in fn_dogs:
    category = image.split('.')[0]
    categories.append('dog')

df = pd.DataFrame({'filename': fn_cats+fn_dogs,
                  'category': categories})

df.head()
```

```
  filename category
0   10.jpg      cat
1    1.jpg      cat
```

```
2   18.jpg      cat
3    9.jpg      cat
4   14.jpg      cat
```

```
sns.countplot(df['category'])
plt.show()
```

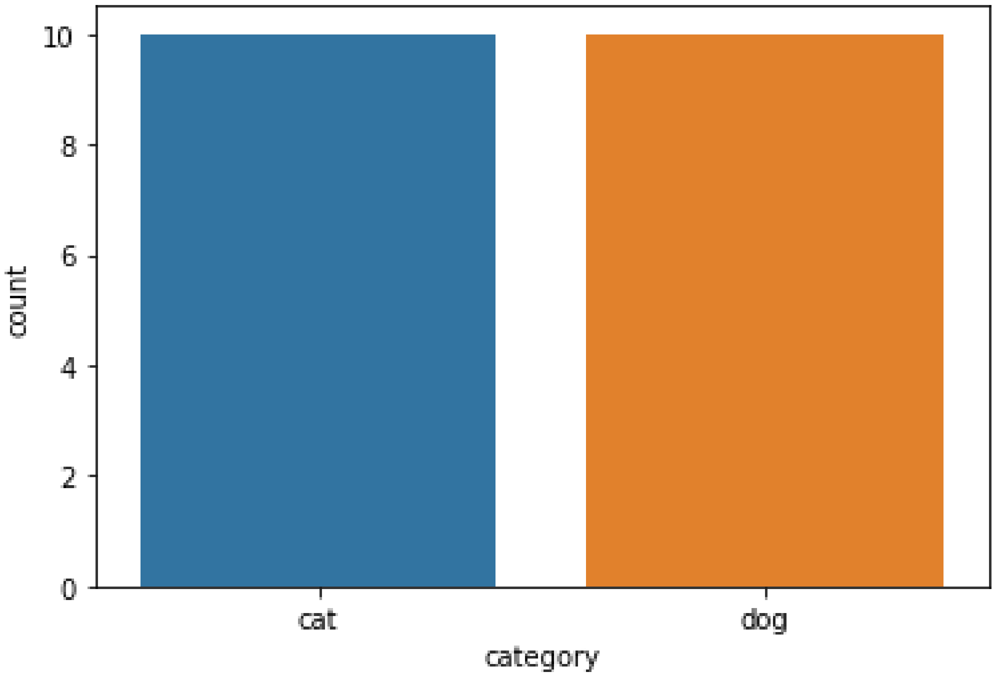



```
# list file names from cats and dogs folders in test
fn_test_cats = os.listdir('data/test/cats')
fn_test_dogs = os.listdir('data/test/dogs')

# categorize 0 for cats and 1 for dogs in train
categories = []

for image in fn_test_cats:
    category = image.split('.')[0]
    categories.append('cat')

for image in fn_test_dogs:
    category = image.split('.')[0]
    categories.append('dog')

df_test = pd.DataFrame({'filename': fn_test_cats+fn_test_dogs,
                  'category': categories})

df_test.head()
```

```
  filename category
0  108.jpg      cat
1  104.jpg      cat
```

```
2  102.jpg      cat
3  106.jpg      cat
4  101.jpg      cat
```

```
sns.countplot(df_test['category'])
plt.show()
```




```
def createCNNModel():
    model = Sequential()
    model.add(Conv2D(name='conv_layer1', filters= 32, kernel_size=5,
                  activation='relu',
                  padding='valid',
                  input_shape=[256, 256, 3]))
    model.add(BatchNormalization())
    model.add(MaxPool2D(name='max_pool_layer1', pool_size= 2))
    model.add(Conv2D(name='conv_layer2', filters= 64, kernel_size=5,
                  activation='relu',
                  padding='valid'))
    model.add(BatchNormalization())
    model.add(MaxPool2D(name='max_pool_layer2', pool_size= 2))
    model.add(Flatten(name='flatten_layer'))
    model.add(Dense(units = 32, name='dense_layer1', activation =
'relu'))
    model.add(Dropout(0.4, name='dropout'))
    model.add(BatchNormalization())
    model.add(Dense(units= 2, name='dense_output', activation=
'softmax'))
    print(model.summary())
    return model
```

```python
def compileAndTrainModel(model):
    model.compile(optimizer='adam',
             loss = 'categorical_crossentropy',
             metrics = ['accuracy'])
    return model

callback = EarlyStopping(monitor='val_loss', patience=5)

def fitAndEvaluateModel(model, epoch):
    history = model.fit(training_set,
          validation_data = validation_set,
          epochs=epoch,
          callbacks=[callback])

    result = model.evaluate(validation_set)
    history = pd.DataFrame(history.history)
    return (history, model, result)

head = ['Loss', 'Accuracy']

data = []

history, model1, result =
fitAndEvaluateModel(compileAndTrainModel(createCNNModel()), 100)
```

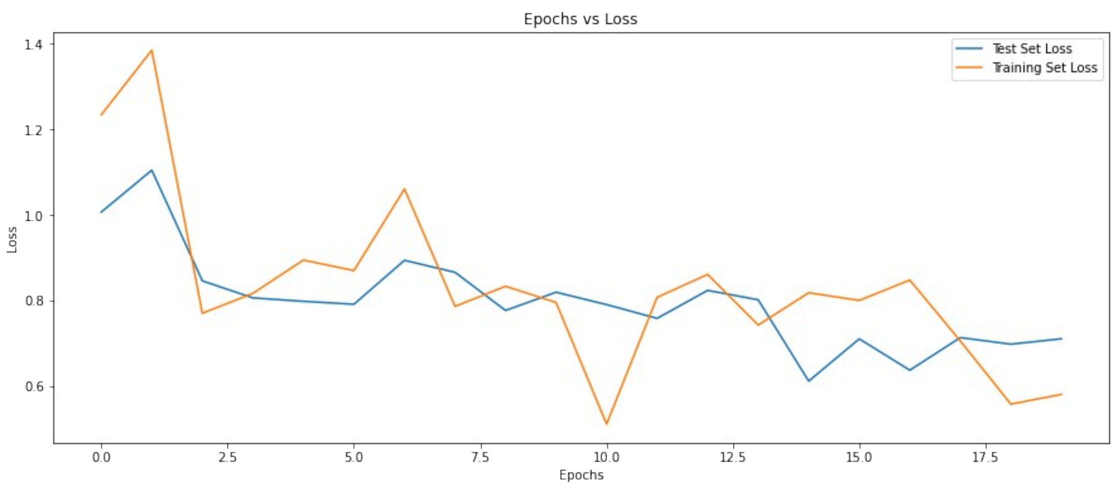
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv_layer1 (Conv2D)        (None, 252, 252, 32)      2432      
                                                                 
 batch_normalization (BatchN  (None, 252, 252, 32)     128       
 ormalization)                                                   
                                                                 
 max_pool_layer1 (MaxPooling  (None, 126, 126, 32)     0         
 2D)                                                             
                                                                 
 conv_layer2 (Conv2D)        (None, 122, 122, 64)      51264     
                                                                 
 batch_normalization_1 (Batc  (None, 122, 122, 64)     256       
 hNormalization)                                                 
                                                                 
 max_pool_layer2 (MaxPooling  (None, 61, 61, 64)       0         
 2D)                                                             
                                                                 
 flatten_layer (Flatten)     (None, 238144)            0         
                                                                 
 dense_layer1 (Dense)        (None, 32)                7620640   
                                                                 
 dropout (Dropout)           (None, 32)                0         
```

```
 batch_normalization_2 (Batc  (None, 32)              128
 hNormalization)

 dense_output (Dense)        (None, 2)                66

=================================================================
Total params: 7,674,914
Trainable params: 7,674,658
Non-trainable params: 256
_________________________________________________________________
None
Epoch 1/100
2/2 [==============================] - 14s 7s/step - loss: 1.2341 -
accuracy: 0.4750 - val_loss: 1.0066 - val_accuracy: 0.5000
Epoch 2/100
2/2 [==============================] - 8s 2s/step - loss: 1.3852 -
accuracy: 0.3500 - val_loss: 1.1047 - val_accuracy: 0.5500
Epoch 3/100
2/2 [==============================] - 8s 2s/step - loss: 0.7700 -
accuracy: 0.6000 - val_loss: 0.8458 - val_accuracy: 0.6000
Epoch 4/100
2/2 [==============================] - 8s 6s/step - loss: 0.8164 -
accuracy: 0.5250 - val_loss: 0.8059 - val_accuracy: 0.6000
Epoch 5/100
2/2 [==============================] - 8s 6s/step - loss: 0.8944 -
accuracy: 0.5750 - val_loss: 0.7979 - val_accuracy: 0.5000
Epoch 6/100
2/2 [==============================] - 8s 2s/step - loss: 0.8699 -
accuracy: 0.5250 - val_loss: 0.7910 - val_accuracy: 0.6500
Epoch 7/100
2/2 [==============================] - 8s 2s/step - loss: 1.0607 -
accuracy: 0.5000 - val_loss: 0.8938 - val_accuracy: 0.4500
Epoch 8/100
2/2 [==============================] - 8s 2s/step - loss: 0.7861 -
accuracy: 0.6000 - val_loss: 0.8658 - val_accuracy: 0.5000
Epoch 9/100
2/2 [==============================] - 8s 2s/step - loss: 0.8331 -
accuracy: 0.6750 - val_loss: 0.7766 - val_accuracy: 0.5000
Epoch 10/100
2/2 [==============================] - 8s 6s/step - loss: 0.7952 -
accuracy: 0.6500 - val_loss: 0.8192 - val_accuracy: 0.5000
Epoch 11/100
2/2 [==============================] - 8s 6s/step - loss: 0.5107 -
accuracy: 0.7500 - val_loss: 0.7902 - val_accuracy: 0.5500
Epoch 12/100
2/2 [==============================] - 8s 6s/step - loss: 0.8072 -
accuracy: 0.5750 - val_loss: 0.7580 - val_accuracy: 0.6000
Epoch 13/100
2/2 [==============================] - 8s 2s/step - loss: 0.8608 -
accuracy: 0.5250 - val_loss: 0.8235 - val_accuracy: 0.6000
```

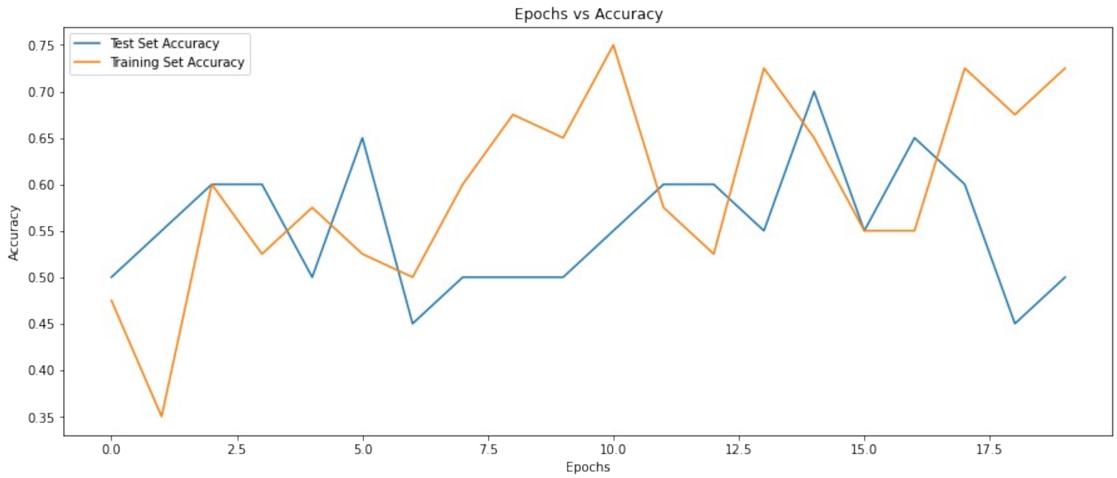
```
Epoch 14/100
2/2 [==============================] - 8s 6s/step - loss: 0.7424 -
accuracy: 0.7250 - val_loss: 0.8015 - val_accuracy: 0.5500
Epoch 15/100
2/2 [==============================] - 8s 2s/step - loss: 0.8179 -
accuracy: 0.6500 - val_loss: 0.6113 - val_accuracy: 0.7000
Epoch 16/100
2/2 [==============================] - 8s 2s/step - loss: 0.8000 -
accuracy: 0.5500 - val_loss: 0.7101 - val_accuracy: 0.5500
Epoch 17/100
2/2 [==============================] - 8s 2s/step - loss: 0.8476 -
accuracy: 0.5500 - val_loss: 0.6368 - val_accuracy: 0.6500
Epoch 18/100
2/2 [==============================] - 8s 6s/step - loss: 0.7052 -
accuracy: 0.7250 - val_loss: 0.7130 - val_accuracy: 0.6000
Epoch 19/100
2/2 [==============================] - 8s 2s/step - loss: 0.5574 -
accuracy: 0.6750 - val_loss: 0.6977 - val_accuracy: 0.4500
Epoch 20/100
2/2 [==============================] - 8s 2s/step - loss: 0.5802 -
accuracy: 0.7250 - val_loss: 0.7103 - val_accuracy: 0.5000
1/1 [==============================] - 1s 1s/step - loss: 0.6969 -
accuracy: 0.6000
```

```
data.append(result)
```

```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 2], label='Test Set Loss')
plt.plot(history.iloc[:, 0], label='Training Set Loss')
plt.title('Epochs vs Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 3], label='Test Set Accuracy')
plt.plot(history.iloc[:, 1], label='Training Set Accuracy')
plt.title('Epochs vs Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



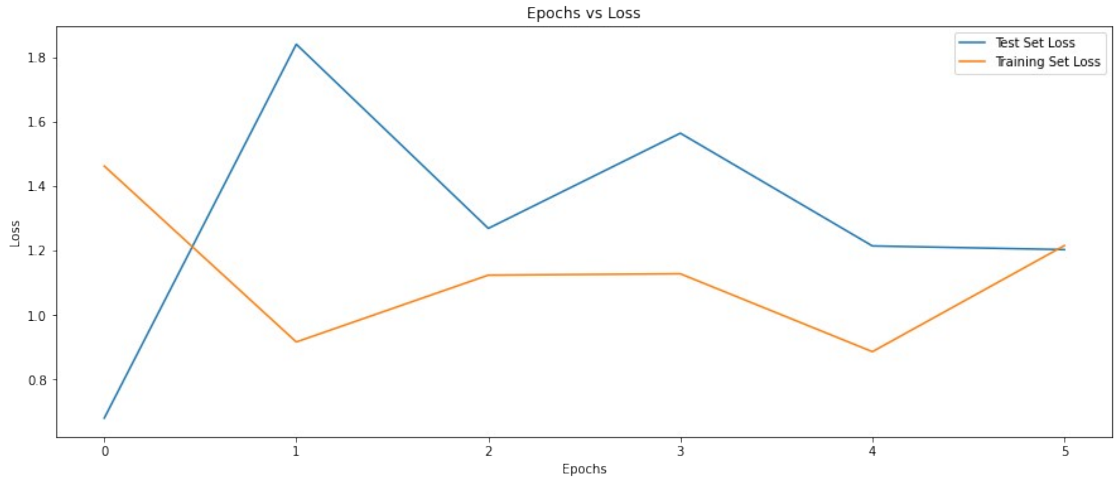

```
history, model2, result =
fitAndEvaluateModel(compileAndTrainModel(createCNNModel()), 200)
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_layer1 (Conv2D)        (None, 252, 252, 32)      2432

 batch_normalization_3 (Batc  (None, 252, 252, 32)     128
 hNormalization)

 max_pool_layer1 (MaxPooling  (None, 126, 126, 32)     0
 2D)

 conv_layer2 (Conv2D)        (None, 122, 122, 64)      51264

 batch_normalization_4 (Batc  (None, 122, 122, 64)     256
 hNormalization)

 max_pool_layer2 (MaxPooling  (None, 61, 61, 64)       0
 2D)

 flatten_layer (Flatten)     (None, 238144)            0

 dense_layer1 (Dense)        (None, 32)                7620640
```

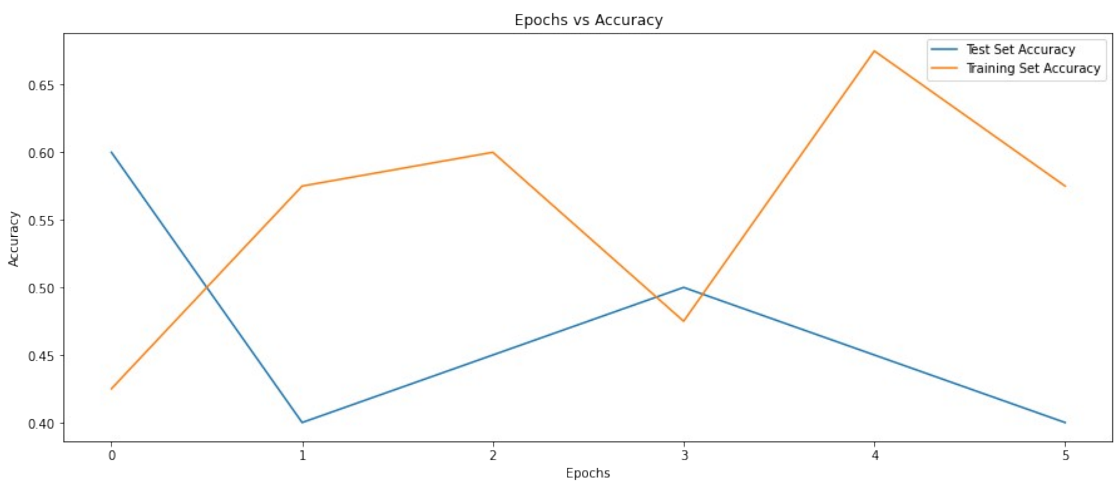
```
 dropout (Dropout)           (None, 32)                0

 batch_normalization_5 (Batc  (None, 32)               128
 hNormalization)

 dense_output (Dense)        (None, 2)                 66

=================================================================
Total params: 7,674,914
Trainable params: 7,674,658
Non-trainable params: 256
_________________________________________________________________
None
Epoch 1/200
2/2 [==============================] - 9s 3s/step - loss: 1.4617 -
accuracy: 0.4250 - val_loss: 0.6787 - val_accuracy: 0.6000
Epoch 2/200
2/2 [==============================] - 8s 6s/step - loss: 0.9153 -
accuracy: 0.5750 - val_loss: 1.8401 - val_accuracy: 0.4000
Epoch 3/200
2/2 [==============================] - 8s 2s/step - loss: 1.1229 -
accuracy: 0.6000 - val_loss: 1.2681 - val_accuracy: 0.4500
Epoch 4/200
2/2 [==============================] - 8s 2s/step - loss: 1.1271 -
accuracy: 0.4750 - val_loss: 1.5638 - val_accuracy: 0.5000
Epoch 5/200
2/2 [==============================] - 8s 2s/step - loss: 0.8852 -
accuracy: 0.6750 - val_loss: 1.2136 - val_accuracy: 0.4500
Epoch 6/200
2/2 [==============================] - 8s 2s/step - loss: 1.2147 -
accuracy: 0.5750 - val_loss: 1.2022 - val_accuracy: 0.4000
1/1 [==============================] - 1s 1s/step - loss: 1.1061 -
accuracy: 0.4500
```

```
data.append(result)
```

```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 2], label='Test Set Loss')
plt.plot(history.iloc[:, 0], label='Training Set Loss')
plt.title('Epochs vs Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 3], label='Test Set Accuracy')
plt.plot(history.iloc[:, 1], label='Training Set Accuracy')
plt.title('Epochs vs Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
history, model3, result =
fitAndEvaluateModel(compileAndTrainModel(createCNNModel()), 300)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv_layer1 (Conv2D)        (None, 252, 252, 32)      2432

 batch_normalization_6 (Batc  (None, 252, 252, 32)     128
 hNormalization)
```

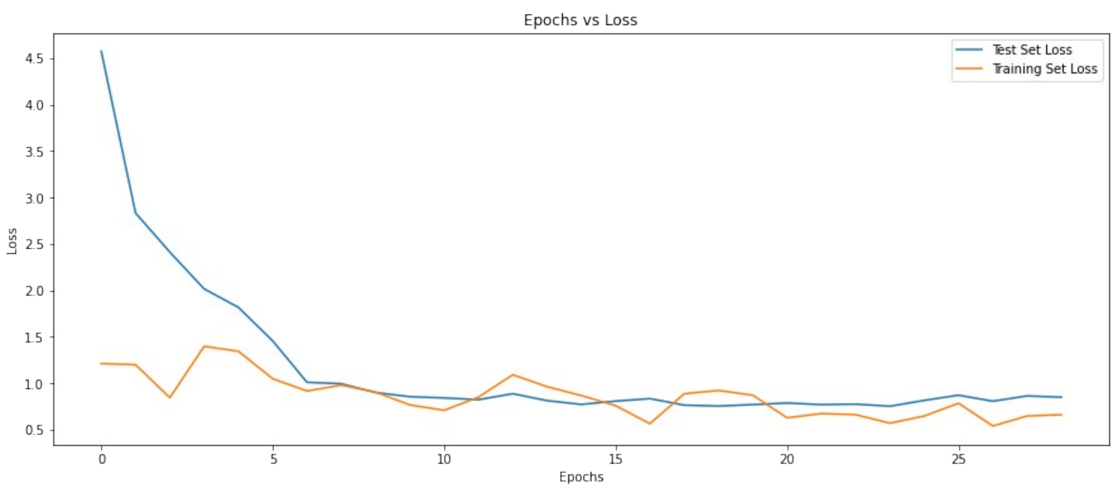
```
 max_pool_layer1 (MaxPooling  (None, 126, 126, 32)     0
 2D)

 conv_layer2 (Conv2D)        (None, 122, 122, 64)      51264

 batch_normalization_7 (Batc  (None, 122, 122, 64)     256
 hNormalization)

 max_pool_layer2 (MaxPooling  (None, 61, 61, 64)       0
 2D)

 flatten_layer (Flatten)     (None, 238144)            0

 dense_layer1 (Dense)        (None, 32)                7620640

 dropout (Dropout)           (None, 32)                0

 batch_normalization_8 (Batc  (None, 32)               128
 hNormalization)

 dense_output (Dense)        (None, 2)                 66

=================================================================
Total params: 7,674,914
Trainable params: 7,674,658
Non-trainable params: 256
_________________________________________________________________
None
Epoch 1/300
2/2 [==============================] - 9s 3s/step - loss: 1.2129 -
accuracy: 0.4750 - val_loss: 4.5735 - val_accuracy: 0.5000
Epoch 2/300
2/2 [==============================] - 8s 6s/step - loss: 1.1996 -
accuracy: 0.5250 - val_loss: 2.8331 - val_accuracy: 0.5000
Epoch 3/300
2/2 [==============================] - 8s 6s/step - loss: 0.8447 -
accuracy: 0.6500 - val_loss: 2.4120 - val_accuracy: 0.5000
Epoch 4/300
2/2 [==============================] - 8s 2s/step - loss: 1.3972 -
accuracy: 0.5000 - val_loss: 2.0162 - val_accuracy: 0.4500
Epoch 5/300
2/2 [==============================] - 8s 6s/step - loss: 1.3446 -
accuracy: 0.3750 - val_loss: 1.8170 - val_accuracy: 0.3500
Epoch 6/300
2/2 [==============================] - 8s 6s/step - loss: 1.0489 -
accuracy: 0.5500 - val_loss: 1.4559 - val_accuracy: 0.5000
Epoch 7/300
2/2 [==============================] - 8s 6s/step - loss: 0.9168 -
accuracy: 0.5250 - val_loss: 1.0106 - val_accuracy: 0.4000
Epoch 8/300
```

```
2/2 [==============================] - 8s 2s/step - loss: 0.9800 -
accuracy: 0.5250 - val_loss: 0.9949 - val_accuracy: 0.4500
Epoch 9/300
2/2 [==============================] - 8s 2s/step - loss: 0.9052 -
accuracy: 0.6750 - val_loss: 0.9003 - val_accuracy: 0.3000
Epoch 10/300
2/2 [==============================] - 8s 6s/step - loss: 0.7667 -
accuracy: 0.6250 - val_loss: 0.8555 - val_accuracy: 0.5000
Epoch 11/300
2/2 [==============================] - 8s 2s/step - loss: 0.7093 -
accuracy: 0.6500 - val_loss: 0.8422 - val_accuracy: 0.4500
Epoch 12/300
2/2 [==============================] - 8s 6s/step - loss: 0.8503 -
accuracy: 0.5250 - val_loss: 0.8236 - val_accuracy: 0.5500
Epoch 13/300
2/2 [==============================] - 8s 2s/step - loss: 1.0920 -
accuracy: 0.5000 - val_loss: 0.8879 - val_accuracy: 0.5000
Epoch 14/300
2/2 [==============================] - 8s 2s/step - loss: 0.9632 -
accuracy: 0.6000 - val_loss: 0.8132 - val_accuracy: 0.5000
Epoch 15/300
2/2 [==============================] - 8s 6s/step - loss: 0.8678 -
accuracy: 0.5250 - val_loss: 0.7725 - val_accuracy: 0.5500
Epoch 16/300
2/2 [==============================] - 8s 2s/step - loss: 0.7590 -
accuracy: 0.6250 - val_loss: 0.8080 - val_accuracy: 0.5500
Epoch 17/300
2/2 [==============================] - 8s 6s/step - loss: 0.5648 -
accuracy: 0.6500 - val_loss: 0.8351 - val_accuracy: 0.4500
Epoch 18/300
2/2 [==============================] - 8s 6s/step - loss: 0.8875 -
accuracy: 0.5500 - val_loss: 0.7647 - val_accuracy: 0.6000
Epoch 19/300
2/2 [==============================] - 8s 2s/step - loss: 0.9224 -
accuracy: 0.5000 - val_loss: 0.7545 - val_accuracy: 0.6000
Epoch 20/300
2/2 [==============================] - 8s 6s/step - loss: 0.8715 -
accuracy: 0.5750 - val_loss: 0.7705 - val_accuracy: 0.5000
Epoch 21/300
2/2 [==============================] - 8s 6s/step - loss: 0.6283 -
accuracy: 0.6500 - val_loss: 0.7873 - val_accuracy: 0.5000
Epoch 22/300
2/2 [==============================] - 8s 2s/step - loss: 0.6741 -
accuracy: 0.5750 - val_loss: 0.7706 - val_accuracy: 0.5000
Epoch 23/300
2/2 [==============================] - 8s 2s/step - loss: 0.6621 -
accuracy: 0.6500 - val_loss: 0.7756 - val_accuracy: 0.5000
Epoch 24/300
2/2 [==============================] - 8s 6s/step - loss: 0.5706 -
accuracy: 0.7500 - val_loss: 0.7527 - val_accuracy: 0.5000
```

```
Epoch 25/300
2/2 [==============================] - 8s 2s/step - loss: 0.6473 -
accuracy: 0.7250 - val_loss: 0.8159 - val_accuracy: 0.4000
Epoch 26/300
2/2 [==============================] - 8s 2s/step - loss: 0.7852 -
accuracy: 0.6250 - val_loss: 0.8718 - val_accuracy: 0.3500
Epoch 27/300
2/2 [==============================] - 8s 2s/step - loss: 0.5401 -
accuracy: 0.7250 - val_loss: 0.8070 - val_accuracy: 0.5000
Epoch 28/300
2/2 [==============================] - 8s 6s/step - loss: 0.6477 -
accuracy: 0.6750 - val_loss: 0.8640 - val_accuracy: 0.4000
Epoch 29/300
2/2 [==============================] - 8s 6s/step - loss: 0.6613 -
accuracy: 0.6500 - val_loss: 0.8502 - val_accuracy: 0.3500
1/1 [==============================] - 1s 1s/step - loss: 0.7279 -
accuracy: 0.5500
```

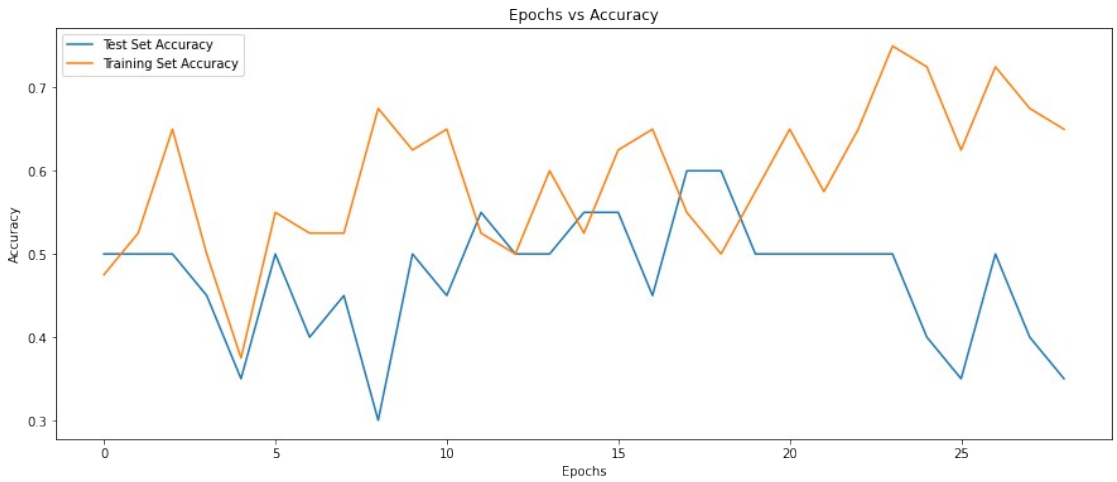
```
data.append(result)
```

```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 2], label='Test Set Loss')
plt.plot(history.iloc[:, 0], label='Training Set Loss')
plt.title('Epochs vs Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
plt.figure(figsize = (15,6))
plt.plot(history.iloc[:, 3], label='Test Set Accuracy')
plt.plot(history.iloc[:, 1], label='Training Set Accuracy')
plt.title('Epochs vs Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
```

```
plt.legend()
plt.show()
```




```
# Report

print(tabulate(data, headers=head, tablefmt="grid"))

+----------+------------+
|     Loss |   Accuracy |
+==========+============+
| 0.696867 |       0.6  |
+----------+------------+
| 1.10612  |       0.45 |
+----------+------------+
| 0.727904 |       0.55 |
+----------+------------+
```

Model 2 gave poor performance in terms of loss and accuracy. So we can go for model 1.

```
# Predict the test images with optimal model

pred = model1.predict(validation_set)
df_test['pred_category']  = np.argmax(pred, axis=1)
df_test['pred_category'] = df_test['pred_category'].replace({0:'cat',
1: 'dog'})
df_test.head(10)

  filename category pred_category
0  108.jpg      cat           dog
1  104.jpg      cat           dog
2  102.jpg      cat           dog
3  106.jpg      cat           cat
4  101.jpg      cat           dog
5  110.jpg      cat           dog
6  109.jpg      cat           dog
7  105.jpg      cat           cat
```

| | | | |
|---|---|---|---|
| 8 | 103.jpg | cat | dog |
| 9 | 107.jpg | cat | cat |