DEEP LEARNING

(CS6005)

MINI PROJECT

On: Computer vision using transfer learning

RESNET-50

(with dog-vs-cat dataset)

- Aparna S S
2018103008
CSE – ‘P’ batch

Date- 12/05/2021

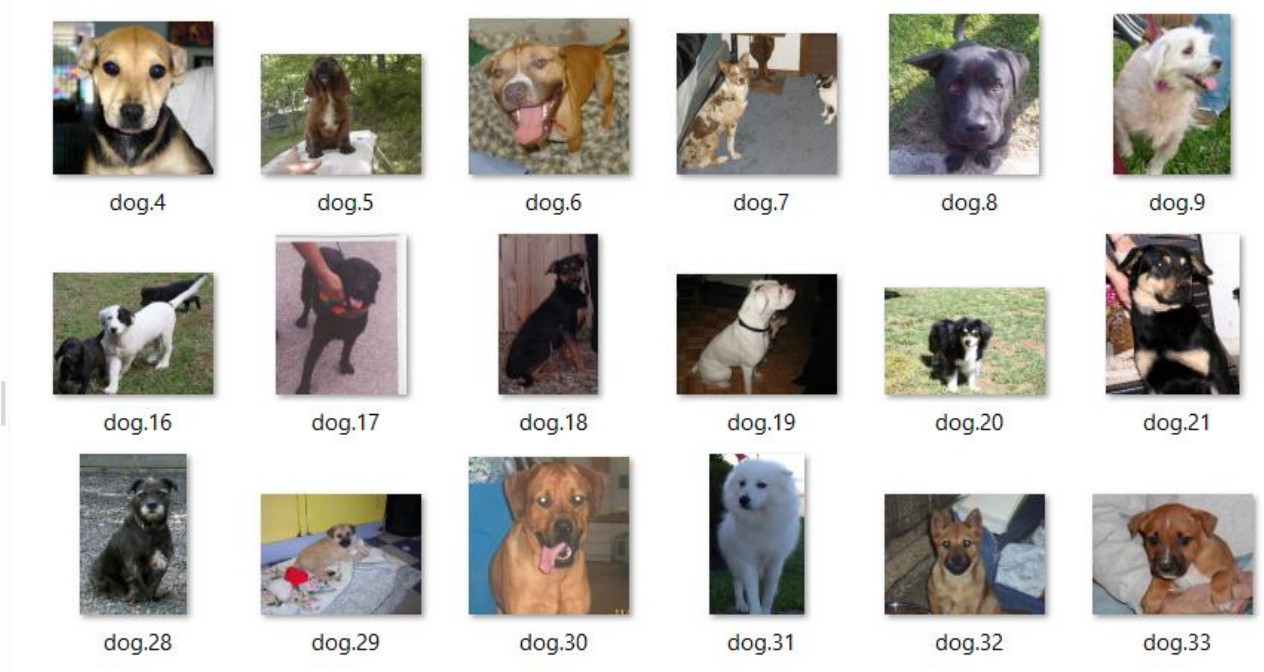
Problem Statement:

One of the problems where deep learning excels is image classification. The goal in image classification is to classify a specific picture according to a set of possible categories. A classic example of image classification is the identification of cats and dogs in a set of pictures (i.e. dogs-vs-cats dataset).
From a deep learning perspective, the image classification problem can be solved through **transfer learning**. Actually, several state-of-the-art results in image classification are based on transfer learning solutions.
This project shows how to implement a transfer learning solution for image classification problems. The implementation proposed in this article is based on Keras, which uses the programming language Python. Following this implementation, you will be able to solve any image classification problem quickly and easily.
There are several transfer learning models such as **VGG-16, ResNet-50, EfficientNet, Inception V3** etc. The model used for image classification of dog-vs-cat dataset in this project is **ResNet-50**.

Dataset details:

The dataset used for image classification is the **dogs-vs-cats** dataset.

The dataset contains 25000 images for training set with 12500 images for cats and 12500 images for dogs. The test dataset contains 12500 images. The images in training set are stored in the format “type of animal.image number.jpg”. As both cats and dogs are stored together, we need to first separate the dog and cat images to get the label as pre-processing and then feed to the model for supervised learning.



url- https://www.kaggle.com/c/dogs-vs-cats/data

Modules:

1. Loading the dataset:

- ✓ The dataset is too huge for training. Hence, the dataset is filtered with 1000 cat and 1000 dog images in train and 100 images in test.

- ✓ The first few train images i.e from x_train are plotted using the **pyplot** package in the **matplotlib** library. These images are subplotted to describe the layout of the figure. The layout is organized in rows and columns, which are represented by the first and second argument. The third argument represents the index of the current plot. Here, the rows is 330, column is 1 and the index of the current plot is i(which is a loop variable used for displaying the number of images(which is 9)). This is done using **subplot** function. The plotted images are then displayed using the **imshow** function.

- ✓ The cat and dog images are separated and stored in a pandas dataframe. From this, the test_split is taken as 0.3 i.e. 30% for testing and 70% for training.

2. Data augmentation:

- ✓ As the model learns well when the number of images is increased, the images are augmented. This augmentation is done using the **ImageDataGenerator** class in the **Keras** module which generates batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely. The image data is generated by transforming the actual training images by rotation, crop, shifts, shear, zoom, flip, reflection, normalization etc. These augmented images are hence stored in the **datagen** variable.

- ✓ The testing data is augmented by rescaling the above datagen.

- ✓ Since images are stored as file and not as arrays, we need to extract using flow_from_dataframe method with the name of the dataframe as train_df and test_df respectively and the batch size of 256.

- ✓ The generated images (of train split) are visualized using the **pyplot** function (as mentioned above).

3. ResNet-50 Model:

✓ Building the model-

- Since, this is a pretrained model, we need to tune the model to our usage by adding parameters. For this, we have flattened the output layer to 1 dimension and added a densely connected layer of 256 neurons with activation function as relu. Further a dropout of 0.5 is added and the final layer with sigmoid activation is added.
- This model is further tuned by freezing a part of the model and not freezing some parts of the model.
  The layers that are not frozen learn new weights and are trained again with a very low learning rate.
  layers.trainable = False, implies the layer is frozen (0-140 layers are frozen)
  layers.trainable = True implies the layer is ready to be trained(140 – the last layers are trained).

✓ Training the model-

A learning rate schedule class is defined which changes the learning rate based on the number of epochs. The learning rate is set to 0.0001 initially. This is done in order to best fit the model as a very low lr can make the model underfit and high lr can make the model overfit. By changing the learning rates, the model learns variantly and hence is best fitted. The **number of epochs** in the basic model and tuned model are **25** and **10** respectively and the **batch_size** of **2** is incorporated in

the tuned model. Early stopping is used to stop the training when the loss maintains the same range in the patience level of epochs and the **patience** level is set to **5**.
The optimizers shape and mould your model into its most accurate possible form by futzing with the weights. The **optimizer** used is the **Adam** which restricts the oscillations in the vertical direction and hence increase the learning rate and our algorithm could take larger steps in the horizontal direction converging faster.
The model is fit and complied with the above-mentioned parameters.

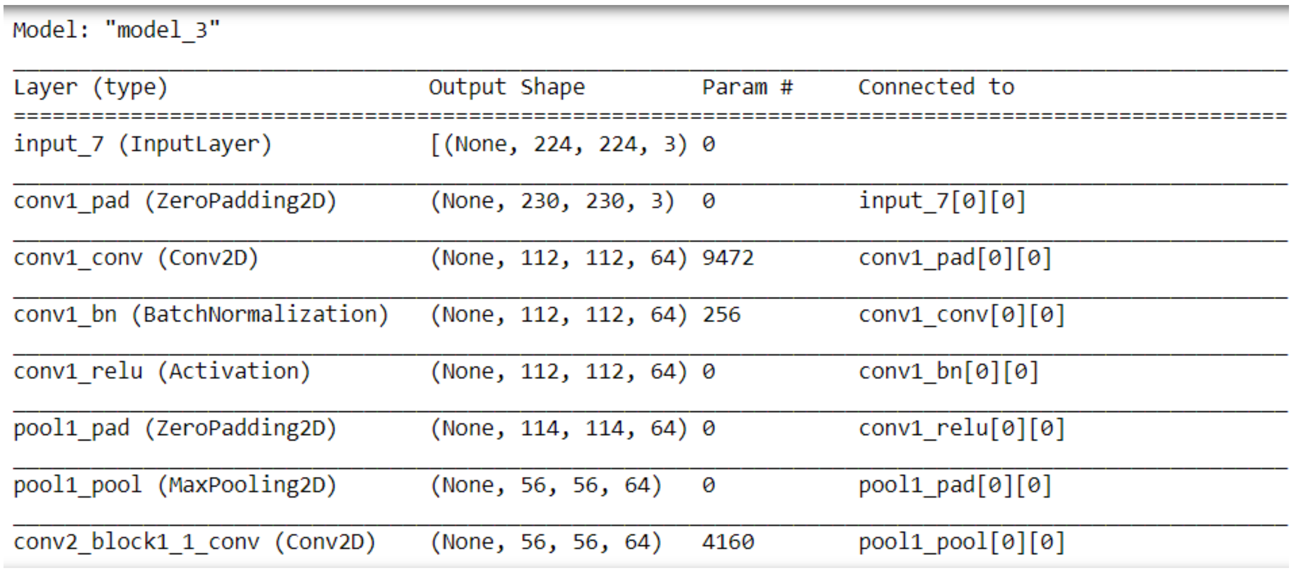
- ✓ **Performance metrics-**

To check how the model works for the test split i.e. x_test and y_test the metrics are calculated. The **metric** used here are **Accuracy** and **Confusion matrix**.

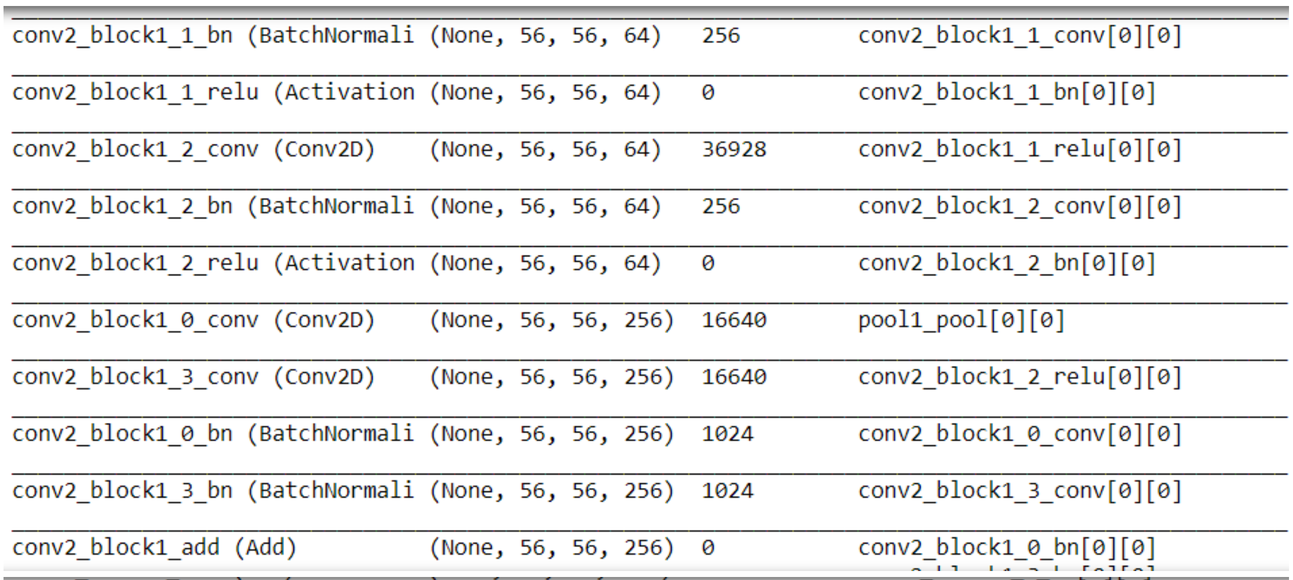
Model summary:

The model used is **ResNet-50 model** and the model summary is:

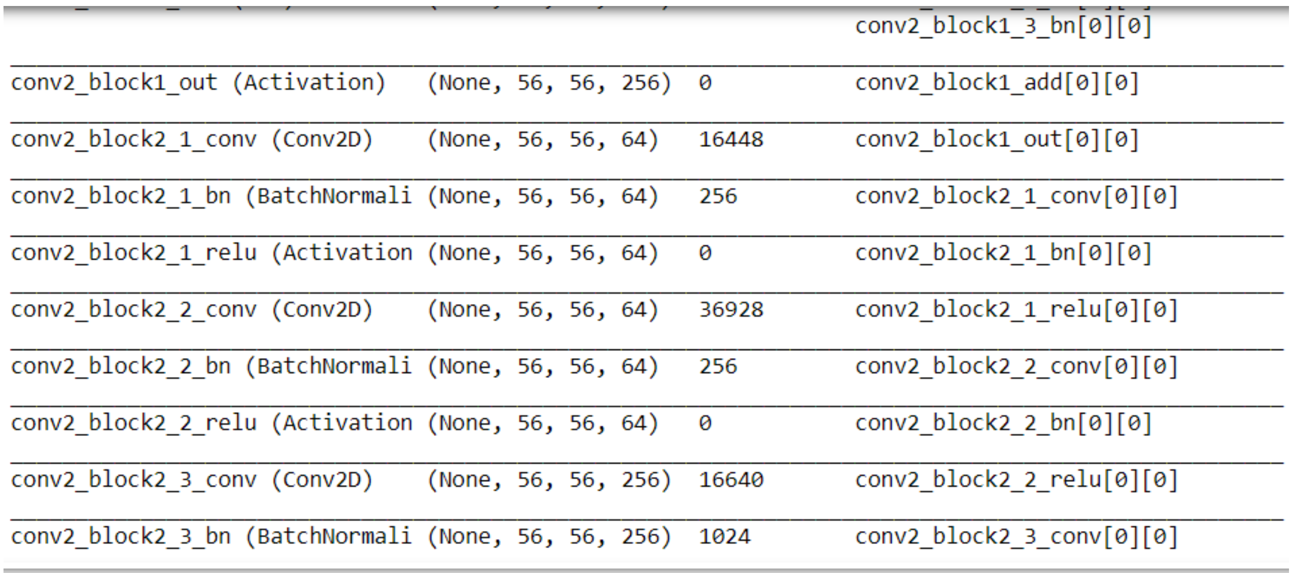
1. Basic model



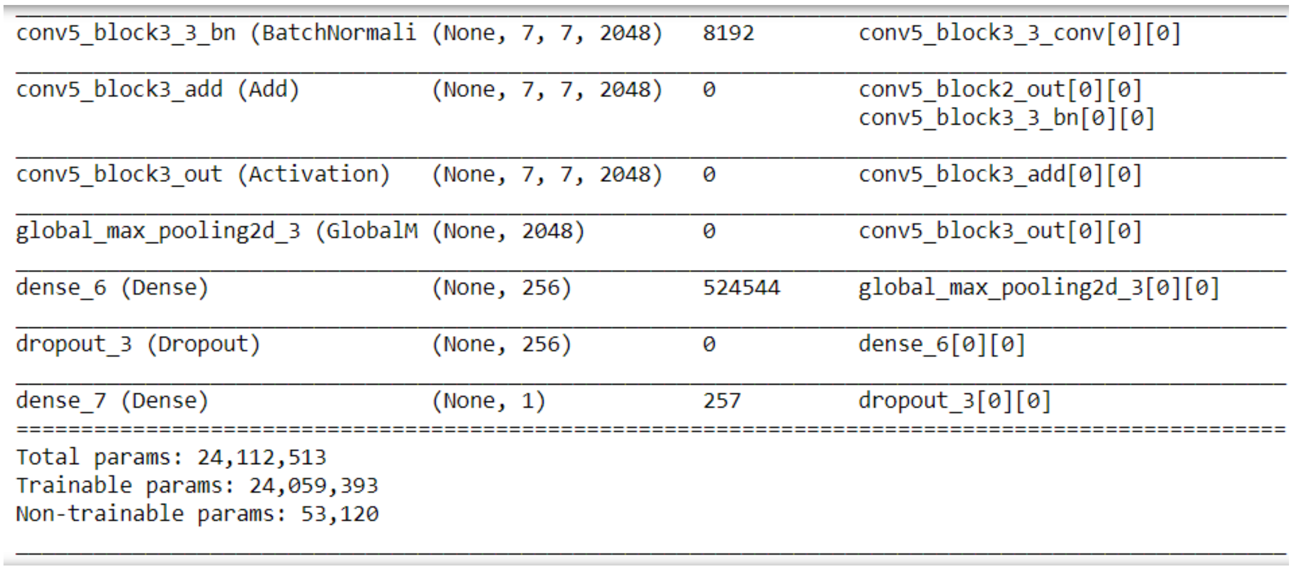
```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_7 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_7[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160        pool1_pool[0][0]
```



```
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0           conv2_block1_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block1_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0           conv2_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640       pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_3_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_add (Add)          (None, 56, 56, 256)  0           conv2_block1_0_bn[0][0]
```

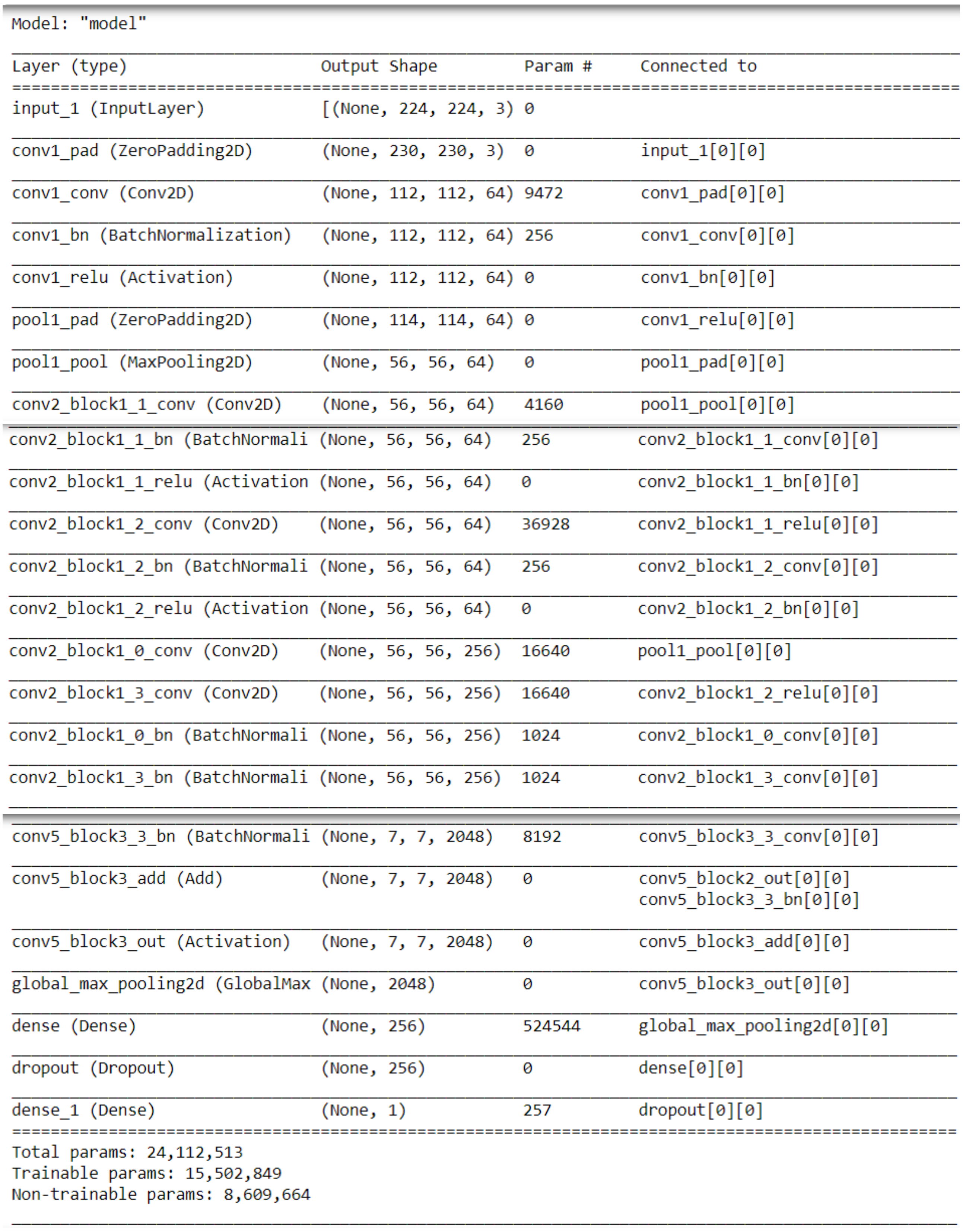


```
                                                                 conv2_block1_3_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_out (Activation)   (None, 56, 56, 256)  0           conv2_block1_add[0][0]
__________________________________________________________________________________________________
conv2_block2_1_conv (Conv2D)    (None, 56, 56, 64)   16448       conv2_block1_out[0][0]
__________________________________________________________________________________________________
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_1_relu (Activation (None, 56, 56, 64)   0           conv2_block2_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block2_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block2_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block2_2_relu (Activation (None, 56, 56, 64)   0           conv2_block2_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block2_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block2_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block2_3_conv[0][0]
```



```
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_conv[0][0]
__________________________________________________________________________________________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
__________________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
__________________________________________________________________________________________________
global_max_pooling2d_3 (GlobalM (None, 2048)         0           conv5_block3_out[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 256)          524544      global_max_pooling2d_3[0][0]
__________________________________________________________________________________________________
dropout_3 (Dropout)             (None, 256)          0           dense_6[0][0]
__________________________________________________________________________________________________
dense_7 (Dense)                 (None, 1)            257         dropout_3[0][0]
==================================================================================================
Total params: 24,112,513
Trainable params: 24,059,393
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

2. Fine-tuned model



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160        pool1_pool[0][0]
```

```
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0           conv2_block1_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block1_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_2_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_2_relu (Activation (None, 56, 56, 64)   0           conv2_block1_2_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_0_conv (Conv2D)    (None, 56, 56, 256)  16640       pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_3_conv (Conv2D)    (None, 56, 56, 256)  16640       conv2_block1_2_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_0_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_0_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256)  1024        conv2_block1_3_conv[0][0]
__________________________________________________________________________________________________
```

```
conv5_block3_3_bn (BatchNormali (None, 7, 7, 2048)   8192        conv5_block3_3_conv[0][0]
__________________________________________________________________________________________________
conv5_block3_add (Add)          (None, 7, 7, 2048)   0           conv5_block2_out[0][0]
                                                                 conv5_block3_3_bn[0][0]
__________________________________________________________________________________________________
conv5_block3_out (Activation)   (None, 7, 7, 2048)   0           conv5_block3_add[0][0]
__________________________________________________________________________________________________
global_max_pooling2d (GlobalMax (None, 2048)         0           conv5_block3_out[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      global_max_pooling2d[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 256)          0           dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 1)            257         dropout[0][0]
==================================================================================================
Total params: 24,112,513
Trainable params: 15,502,849
Non-trainable params: 8,609,664
__________________________________________________________________________________________________
```

Coding Snapshots:

Importing Packages

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os
from glob import glob
import cv2
import random
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
import seaborn as sns
%matplotlib inline
```

```
from keras.models import Sequential
from keras import layers, optimizers
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense, GlobalMaxPooling2D
from tensorflow.keras.callbacks import EarlyStopping
from keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
from keras.applications.vgg16 import VGG16

import matplotlib.pyplot as plt
from PIL import Image
from glob import glob
import tensorflow as tf
```

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler , StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

```
from keras.models import Sequential, load_model, Model
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, LearningRateScheduler
from keras.utils import to_categorical
```

Plotting the images

```
from matplotlib import pyplot
from matplotlib.image import imread
folder = 'dogs-vs-cats/train/'
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    filename = folder + 'dog.' + str(i) + '.jpg'

    image = imread(filename)
    pyplot.imshow(image)
pyplot.show()
```

```
from matplotlib import pyplot
from matplotlib.image import imread
folder = 'dogs-vs-cats/train/'
for i in range(9):
    pyplot.subplot(330 + 1 + i)
    filename = folder + 'cat+ str(i) + '.jpg'

    image = imread(filename)
    pyplot.imshow(image)
pyplot.show()
```

```
label = []
for file in listdir(folder):
    category = file.split('.')
    if category[2] != 'jpg':
      print(category[2])
    if category[0] == "cat":
        label.append(0)
    else:
        label.append(1)
```

```
df = pd.DataFrame({ 'filename': listdir(folder), 'label': label })
df['label'].value_counts().plot.bar()
```

Split train and validation set

```
train_df, val_df = train_test_split(df, test_size=0.3)
train_df = train_df.reset_index()
val_df = val_df.reset_index()
```

```
train_df = train_df.drop(['index'], axis = 1)
val_df = val_df.drop(['index'], axis = 1)
print(train_df.shape, val_df.shape)
```

```
train_df['label'] = train_df['label'].astype('str')
val_df['label'] = val_df['label'].astype('str')

print(train_df.head())
print(val_df.head())
print(train_df.info(), val_df.info())
```

Data Augmentation

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=True,
        featurewise_std_normalization=True,
        samplewise_std_normalization=True,
        zca_whitening=False,
        rotation_range=0.7,
        zoom_range = 0.2,
        width_shift_range=0.30,
        height_shift_range=0.25,
        horizontal_flip=False,
        vertical_flip=True,
        rescale=1./255,
)
```

Training Generator

```
train_generator = datagen.flow_from_dataframe(
                dataframe = train_df,
                directory = "dogs-vs-cats/train/",
                x_col = "filename",
                y_col = "label",
                batch_size = 256,
                seed = 1,
                shuffle = True,
                class_mode = "binary",
                target_size=(256, 256)
)
```

```
plt.figure(figsize=(10, 10))
for i in range(0, 9):
    plt.subplot(3, 3, i+1)
    for X_img, Y_img in train_generator:
        image = X_img[0]
        plt.imshow(image)
        break
plt.show()
```

Validation Generator

```
validation_datagen = ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_dataframe(
                dataframe = val_df,
                directory = "dogs-vs-cats/train/",
                x_col = "filename",
                y_col = "label",
                batch_size = 256,
                class_mode = 'binary',
                target_size = (256, 256),
)
```

Test Generator

```
test = sorted(os.listdir("dogs-vs-cats/test1/"))
test_df = pd.DataFrame({ 'filename': test })
test_df.to_csv('prediction.csv')
```

```
test_gen = ImageDataGenerator(rescale=1./255)

test_generator = test_gen.flow_from_dataframe(
              dataframe = test_df,
              directory = "dogs-vs-cats/test1/",
              x_col = "filename",
              y_col = None,
              class_mode = None,
              target_size = (256, 256),
              shuffle = True
)
```

RESNET50-basic model

```
from keras.applications import ResNet50

input_shape = (image_size, image_size, 3)

pre_trained_model = ResNet50(input_shape=input_shape, include_top=False, weights="imagenet")
# pre_trained_model.summary()

last_layer = pre_trained_model.get_layer('conv5_block3_out')
last_output = last_layer.output

# Flatten the output layer to 1 dimension
x = GlobalMaxPooling2D()(last_output)
# Add a fully connected layer with 256 hidden units and ReLU activation
x = Dense(256, activation='relu')(x)
# Add a dropout rate of 0.5
x = Dropout(0.5)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

resnet_model = Model(pre_trained_model.input, x)

resnet_model.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(lr=1e-4),
              metrics=['accuracy'])

resnet_model.summary()
```

```
early_stopping = EarlyStopping(monitor='loss', patience=5, verbose=1)
history = resnet_model.fit_generator((train_generator), epochs=25,
                              validation_data = (validation_generator), verbose=1,
                              callbacks = [reduce_lr, early_stopping])
```

```
epoch_loss = history.history['loss']
epoch_val_loss = history.history['val_loss']
epoch_acc = history.history['accuracy']
epoch_val_acc = history.history['val_accuracy']

plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
plt.plot(range(0,len(epoch_loss)), epoch_loss, 'b-', linewidth=2, label='Train Loss')
plt.plot(range(0,len(epoch_val_loss)), epoch_val_loss, 'r-', linewidth=2, label='Val Loss')
plt.title('Evolution of LOSS on train & validation datasets over epochs')
plt.legend(loc='best')

plt.subplot(1,2,2)
plt.plot(range(0,len(epoch_acc)), epoch_acc, 'b-', linewidth=2, label='Train Accuracy')
plt.plot(range(0,len(epoch_val_acc)), epoch_val_acc, 'r-', linewidth=2,label='Val Accuracy')
plt.title('Evolution of ACCURACY on train & validation datasets over epochs')
plt.legend(loc='best')

plt.show()
```

```
loss, accuracy = model.evaluate_generator(validation_generator, workers=12)
print("Validation: accuracy = %f  ;  loss = %f " % (accuracy, loss))
```

```
predict = model.predict_generator(test_generator)
threshold = 0.5
Y_test_pred = np.where(predict > 0.5, 1,0)
```

```
test_res = pd.read_csv('.prediction-test/.csv')
Y_test_actual = test_res['label']
Y_test_actual = np.asarray(Y_test_actual)
```

```
# compute the confusion matrix
Y_test_actual = Y_test_actual.astype('str')
Y_test_pred = Y_test_pred.astype('str')
confusion_mtx = confusion_matrix(Y_test_actual, Y_test_pred)

f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="Blues",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

RESNET50 Fine tuning

```
from keras.applications import ResNet50

image_size = 224
input_shape = (image_size, image_size, 3)

resnet_pretrained_model = ResNet50(input_shape=input_shape, include_top=False, weights="imagenet")

for layer in resnet_pretrained_model.layers[:140]:
    layer.trainable = False

for layer in resnet_pretrained_model.layers[140:]:
    layer.trainable = True
```

```
last_layer = resnet_pretrained_model.get_layer('conv5_block3_out')
last_output = last_layer.output

# Flatten the output layer to 1 dimension
x = GlobalMaxPooling2D()(last_output)
# Add a fully connected layer with 256 hidden units and ReLU activation
x = Dense(256, activation='relu')(x)
# Add a dropout rate of 0.5
x = Dropout(0.5)(x)
# Add a final sigmoid layer for classification
x = layers.Dense(1, activation='sigmoid')(x)

resnet_model_fine_tuned = Model(resnet_pretrained_model.input, x)

resnet_model_fine_tuned.compile(loss='binary_crossentropy',
              optimizer=optimizers.Adam(lr=1e-4),
              metrics=['accuracy'])

resnet_model_fine_tuned.summary()
```

```
early_stopping = EarlyStopping(monitor='loss', patience=5, verbose=1)
history = resnet_model_fine_tuned.fit_generator((train_generator), epochs=10,batch_size=2
                              validation_data = (validation_generator), verbose=1)
```

```
epoch_loss = hist.history['loss']
epoch_val_loss = hist.history['val_loss']
epoch_acc = hist.history['accuracy']
epoch_val_acc = hist.history['val_accuracy']

plt.figure(figsize=(20,6))
plt.subplot(1,2,1)
plt.plot(range(0,len(epoch_loss)), epoch_loss, 'b-', linewidth=2, label='Train Loss')
plt.plot(range(0,len(epoch_val_loss)), epoch_val_loss, 'r-', linewidth=2, label='Val Loss')
plt.title('Evolution of LOSS on train & validation datasets over epochs')
plt.legend(loc='best')

plt.subplot(1,2,2)
plt.plot(range(0,len(epoch_acc)), epoch_acc, 'b-', linewidth=2, label='Train Accuracy')
plt.plot(range(0,len(epoch_val_acc)), epoch_val_acc, 'r-', linewidth=2,label='Val Accuracy')
plt.title('Evolution of ACCURACY on train & validation datasets over epochs')
plt.legend(loc='best')

plt.show()
```

```
loss, accuracy = model.evaluate_generator(validation_generator, workers=12)
print("Validation: accuracy = %f  ;  loss = %f " % (accuracy, loss))
```

```
predict = model.predict_generator(test_generator)
threshold = 0.5
Y_test_pred = np.where(predict > 0.5, 1,0)
```

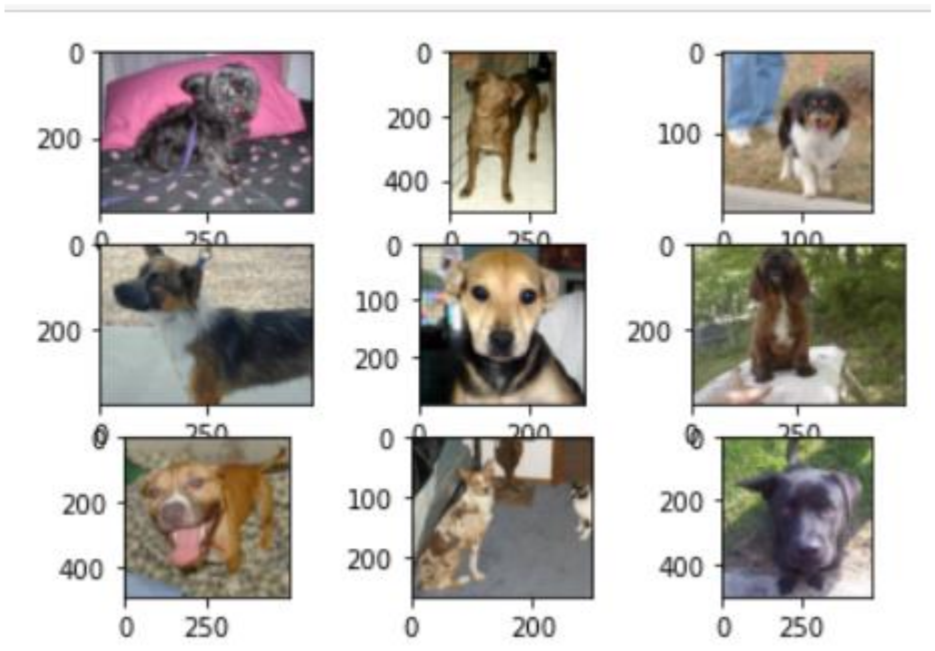
```
test_res = pd.read_csv('.prediction-test/.csv')
Y_test_actual = test_res['label']
Y_test_actual = np.asarray(Y_test_actual)
```

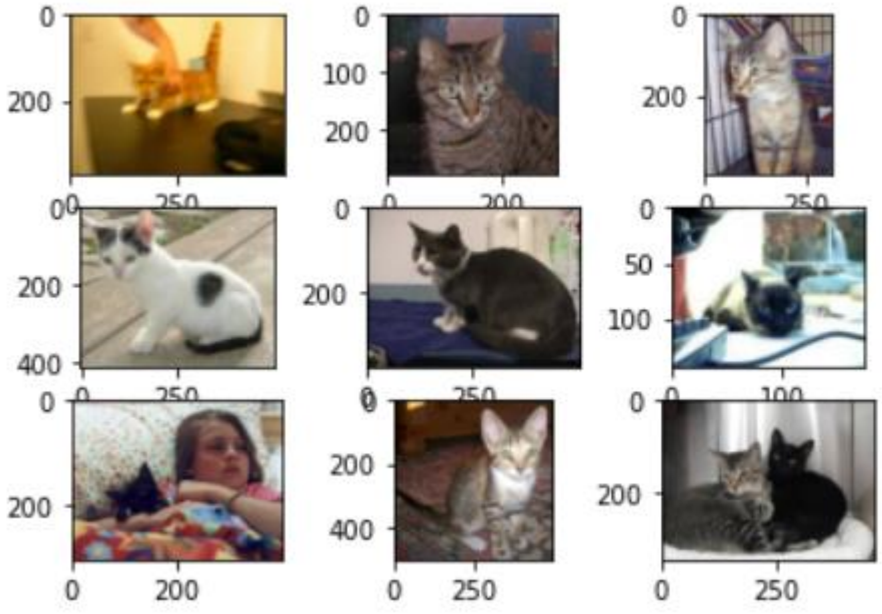
```
# compute the confusion matrix
Y_test_actual = Y_test_actual.astype('str')
Y_test_pred = Y_test_pred.astype('str')
confusion_mtx = confusion_matrix(Y_test_actual, Y_test_pred)

f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True, linewidths=0.01,cmap="Blues",linecolor="gray", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

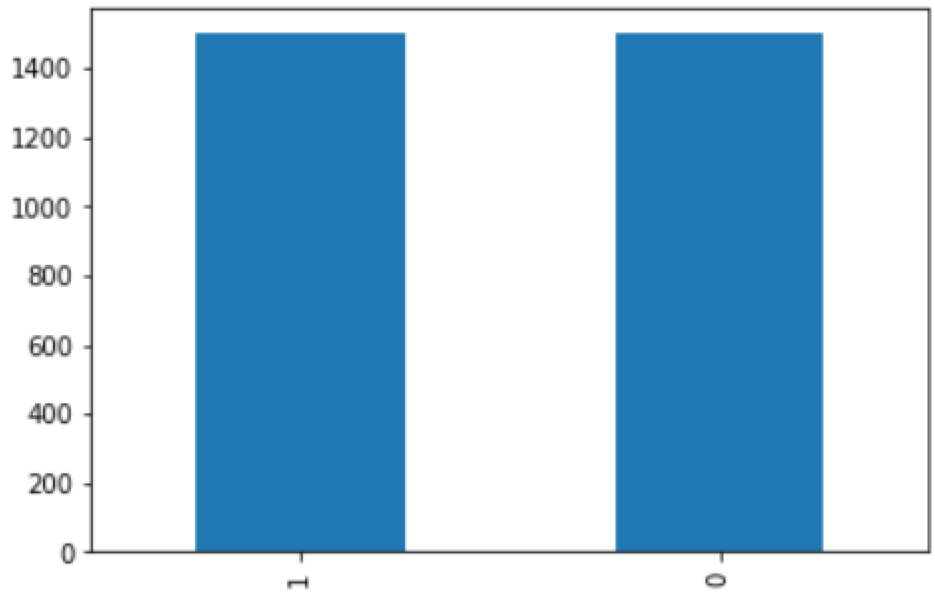
Results:

Glimpse of dataset images (a plot of first few images)-

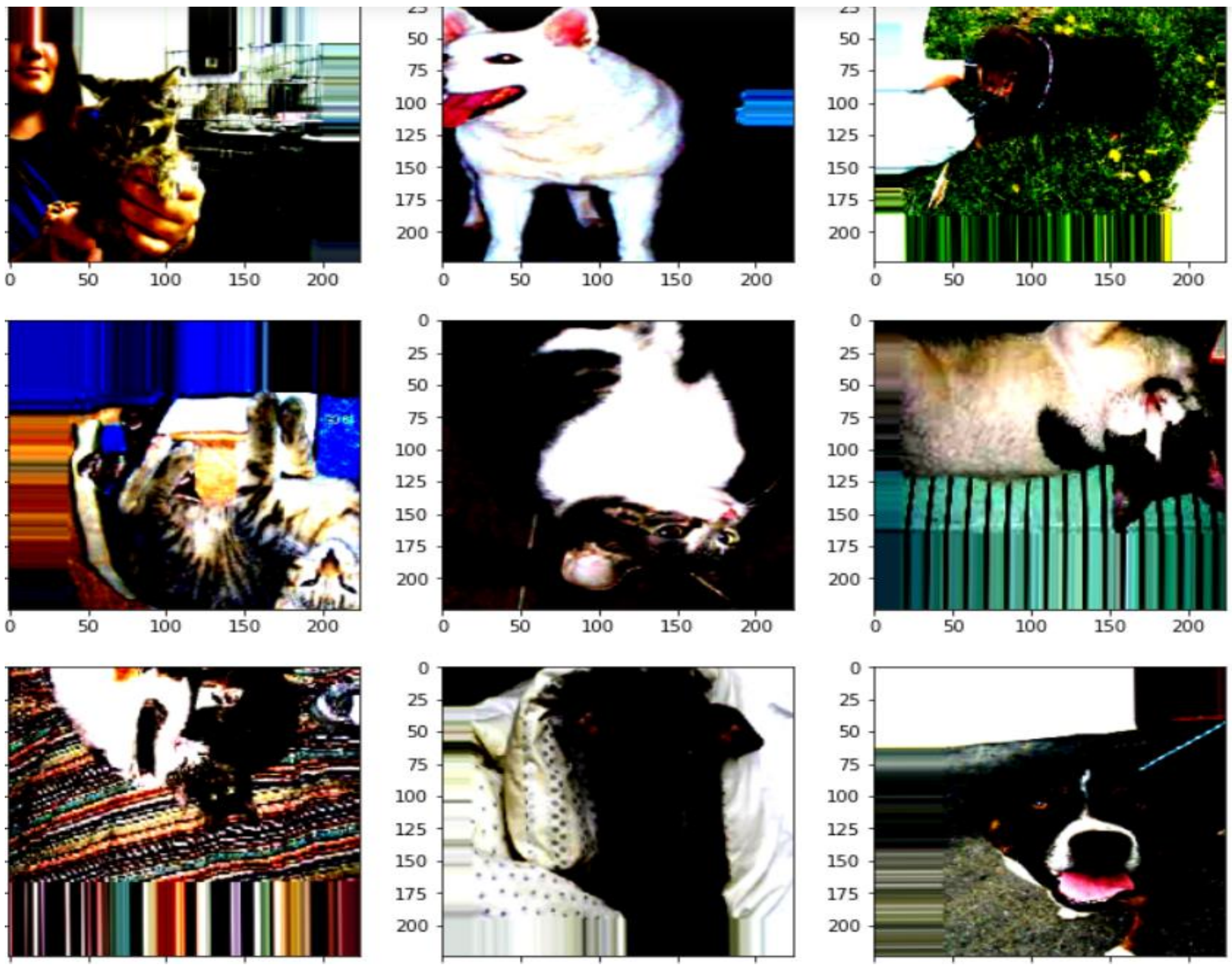




Dataframe of the separated images-

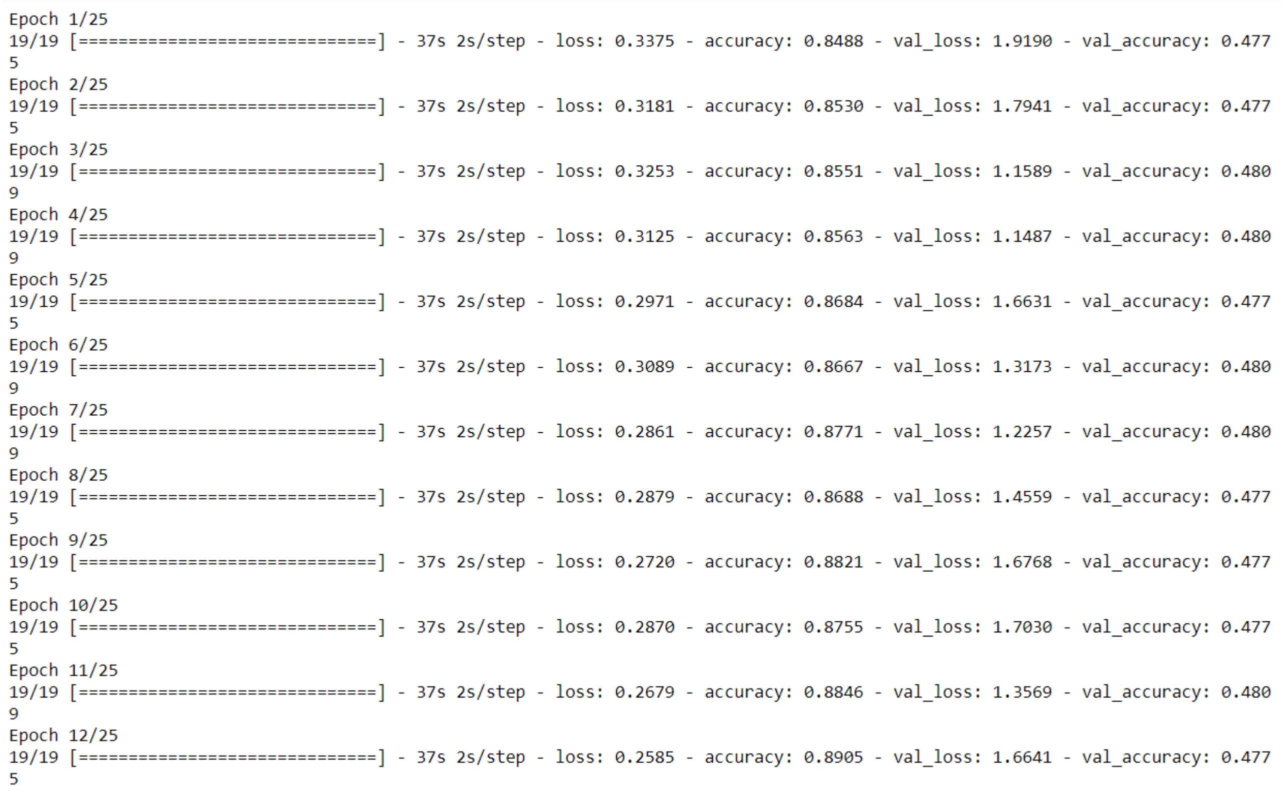


Glimpse of augmented images (a plot of few augmented images)-

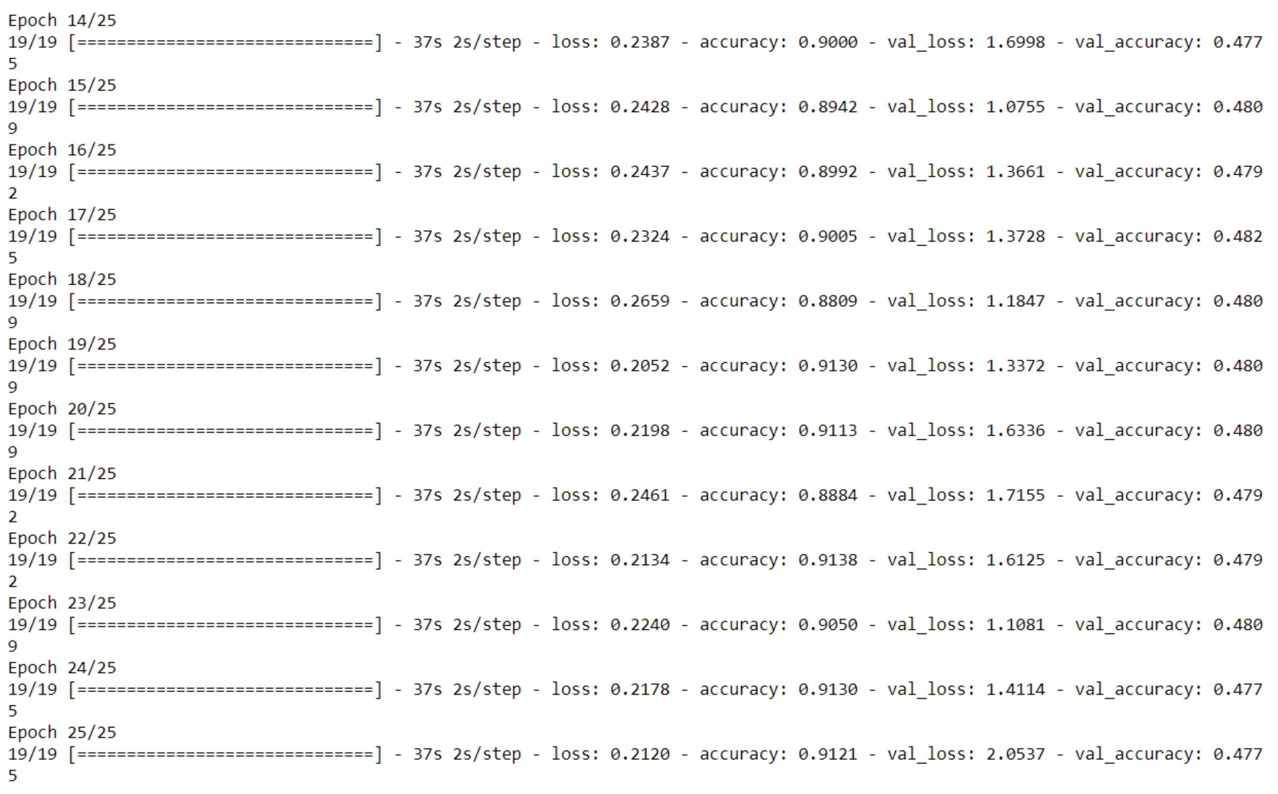


Model fit (the model is fitted with the given parameters)-

Basic model:



```
Epoch 1/25
19/19 [==============================] - 37s 2s/step - loss: 0.3375 - accuracy: 0.8488 - val_loss: 1.9190 - val_accuracy: 0.477
5
Epoch 2/25
19/19 [==============================] - 37s 2s/step - loss: 0.3181 - accuracy: 0.8530 - val_loss: 1.7941 - val_accuracy: 0.477
5
Epoch 3/25
19/19 [==============================] - 37s 2s/step - loss: 0.3253 - accuracy: 0.8551 - val_loss: 1.1589 - val_accuracy: 0.480
9
Epoch 4/25
19/19 [==============================] - 37s 2s/step - loss: 0.3125 - accuracy: 0.8563 - val_loss: 1.1487 - val_accuracy: 0.480
9
Epoch 5/25
19/19 [==============================] - 37s 2s/step - loss: 0.2971 - accuracy: 0.8684 - val_loss: 1.6631 - val_accuracy: 0.477
5
Epoch 6/25
19/19 [==============================] - 37s 2s/step - loss: 0.3089 - accuracy: 0.8667 - val_loss: 1.3173 - val_accuracy: 0.480
9
Epoch 7/25
19/19 [==============================] - 37s 2s/step - loss: 0.2861 - accuracy: 0.8771 - val_loss: 1.2257 - val_accuracy: 0.480
9
Epoch 8/25
19/19 [==============================] - 37s 2s/step - loss: 0.2879 - accuracy: 0.8688 - val_loss: 1.4559 - val_accuracy: 0.477
5
Epoch 9/25
19/19 [==============================] - 37s 2s/step - loss: 0.2720 - accuracy: 0.8821 - val_loss: 1.6768 - val_accuracy: 0.477
5
Epoch 10/25
19/19 [==============================] - 37s 2s/step - loss: 0.2870 - accuracy: 0.8755 - val_loss: 1.7030 - val_accuracy: 0.477
5
Epoch 11/25
19/19 [==============================] - 37s 2s/step - loss: 0.2679 - accuracy: 0.8846 - val_loss: 1.3569 - val_accuracy: 0.480
9
Epoch 12/25
19/19 [==============================] - 37s 2s/step - loss: 0.2585 - accuracy: 0.8905 - val_loss: 1.6641 - val_accuracy: 0.477
5
```



```
Epoch 14/25
19/19 [==============================] - 37s 2s/step - loss: 0.2387 - accuracy: 0.9000 - val_loss: 1.6998 - val_accuracy: 0.477
5
Epoch 15/25
19/19 [==============================] - 37s 2s/step - loss: 0.2428 - accuracy: 0.8942 - val_loss: 1.0755 - val_accuracy: 0.480
9
Epoch 16/25
19/19 [==============================] - 37s 2s/step - loss: 0.2437 - accuracy: 0.8992 - val_loss: 1.3661 - val_accuracy: 0.479
2
Epoch 17/25
19/19 [==============================] - 37s 2s/step - loss: 0.2324 - accuracy: 0.9005 - val_loss: 1.3728 - val_accuracy: 0.482
5
Epoch 18/25
19/19 [==============================] - 37s 2s/step - loss: 0.2659 - accuracy: 0.8809 - val_loss: 1.1847 - val_accuracy: 0.480
9
Epoch 19/25
19/19 [==============================] - 37s 2s/step - loss: 0.2052 - accuracy: 0.9130 - val_loss: 1.3372 - val_accuracy: 0.480
9
Epoch 20/25
19/19 [==============================] - 37s 2s/step - loss: 0.2198 - accuracy: 0.9113 - val_loss: 1.6336 - val_accuracy: 0.480
9
Epoch 21/25
19/19 [==============================] - 37s 2s/step - loss: 0.2461 - accuracy: 0.8884 - val_loss: 1.7155 - val_accuracy: 0.479
2
Epoch 22/25
19/19 [==============================] - 37s 2s/step - loss: 0.2134 - accuracy: 0.9138 - val_loss: 1.6125 - val_accuracy: 0.479
2
Epoch 23/25
19/19 [==============================] - 37s 2s/step - loss: 0.2240 - accuracy: 0.9050 - val_loss: 1.1081 - val_accuracy: 0.480
9
Epoch 24/25
19/19 [==============================] - 37s 2s/step - loss: 0.2178 - accuracy: 0.9130 - val_loss: 1.4114 - val_accuracy: 0.477
5
Epoch 25/25
19/19 [==============================] - 37s 2s/step - loss: 0.2120 - accuracy: 0.9121 - val_loss: 2.0537 - val_accuracy: 0.477
5
```

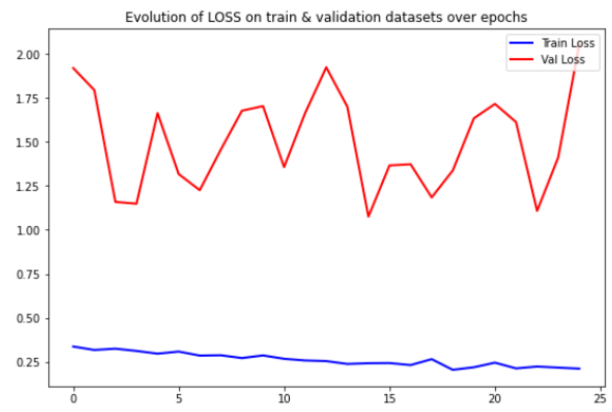
Tuned model:

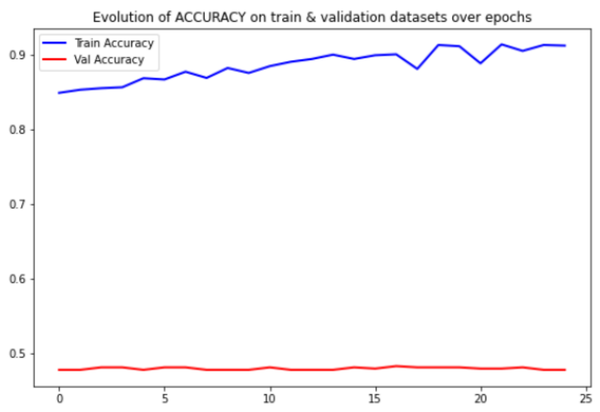
```
Epoch 1/10
500/500 [==============================] - 381s 754ms/step - loss: 2.5697 - accuracy: 0.2527 - val_loss: 1.6369 - val_accuracy:
0.4218
Epoch 2/10
500/500 [==============================] - 405s 809ms/step - loss: 1.8923 - accuracy: 0.3570 - val_loss: 1.5607 - val_accuracy:
0.4499
Epoch 3/10
500/500 [==============================] - 406s 812ms/step - loss: 1.7055 - accuracy: 0.4109 - val_loss: 1.7097 - val_accuracy:
0.4470
Epoch 4/10
500/500 [==============================] - 391s 783ms/step - loss: 1.5876 - accuracy: 0.4536 - val_loss: 1.4324 - val_accuracy:
0.5243
Epoch 5/10
500/500 [==============================] - 385s 769ms/step - loss: 1.4987 - accuracy: 0.4843 - val_loss: 1.3974 - val_accuracy:
0.5294
Epoch 6/10
500/500 [==============================] - 385s 770ms/step - loss: 1.4456 - accuracy: 0.5039 - val_loss: 1.3002 - val_accuracy:
0.5715
Epoch 7/10
500/500 [==============================] - 386s 772ms/step - loss: 1.3899 - accuracy: 0.5255 - val_loss: 1.3727 - val_accuracy:
0.5516
Epoch 8/10
500/500 [==============================] - 387s 774ms/step - loss: 1.3425 - accuracy: 0.5429 - val_loss: 1.3069 - val_accuracy:
0.5825
Epoch 9/10
500/500 [==============================] - 386s 772ms/step - loss: 1.3081 - accuracy: 0.5595 - val_loss: 1.2573 - val_accuracy:
0.5793
Epoch 10/10
500/500 [==============================] - 392s 784ms/step - loss: 1.2873 - accuracy: 0.5669 - val_loss: 1.2077 - val_accuracy:
0.6082
```

Performance metrics -

Accuracy and loss plot:

For basic model-

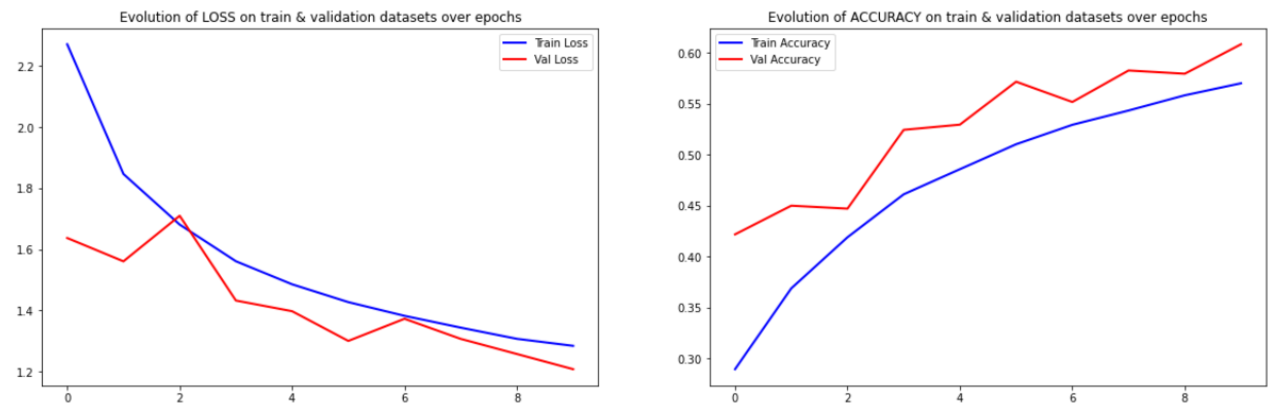




Accuracy and loss

```
Validation: accuracy = 0.477537  ;  loss = 2.053684
```

For tuned model-

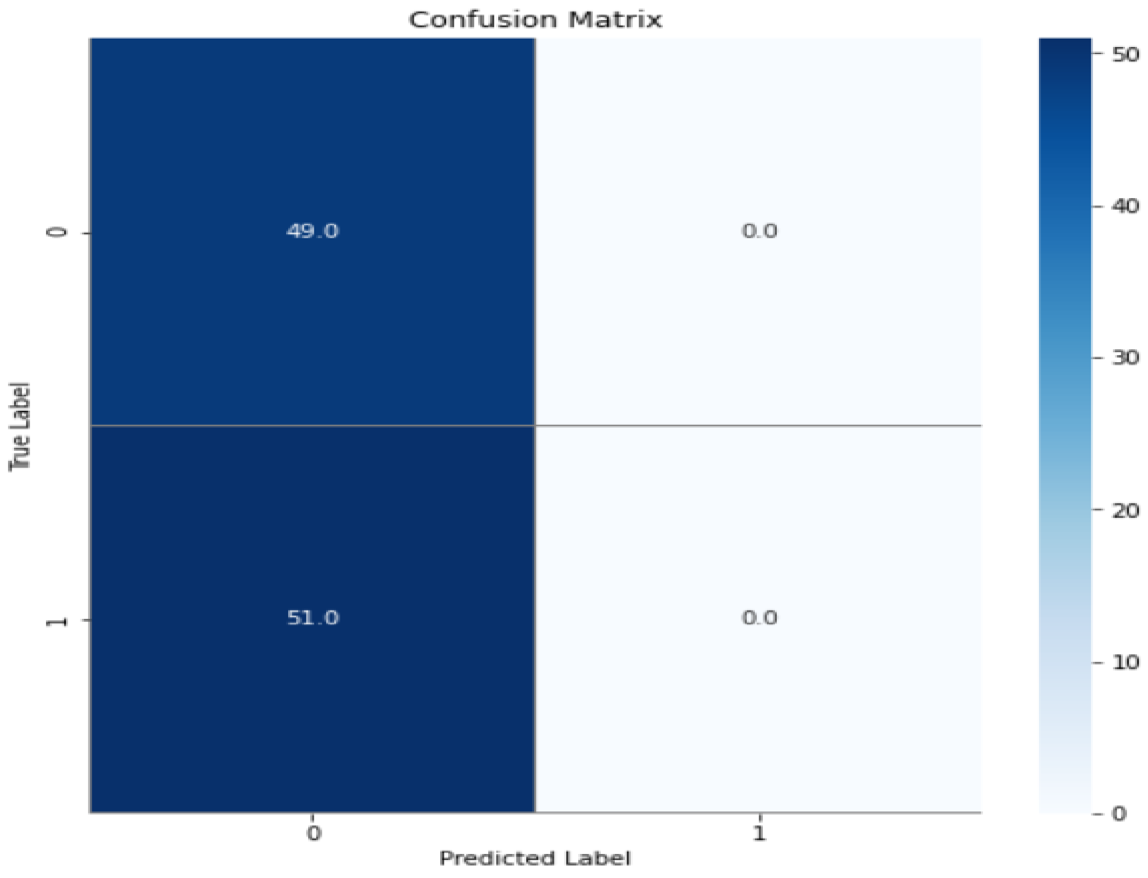


Accuracy and loss

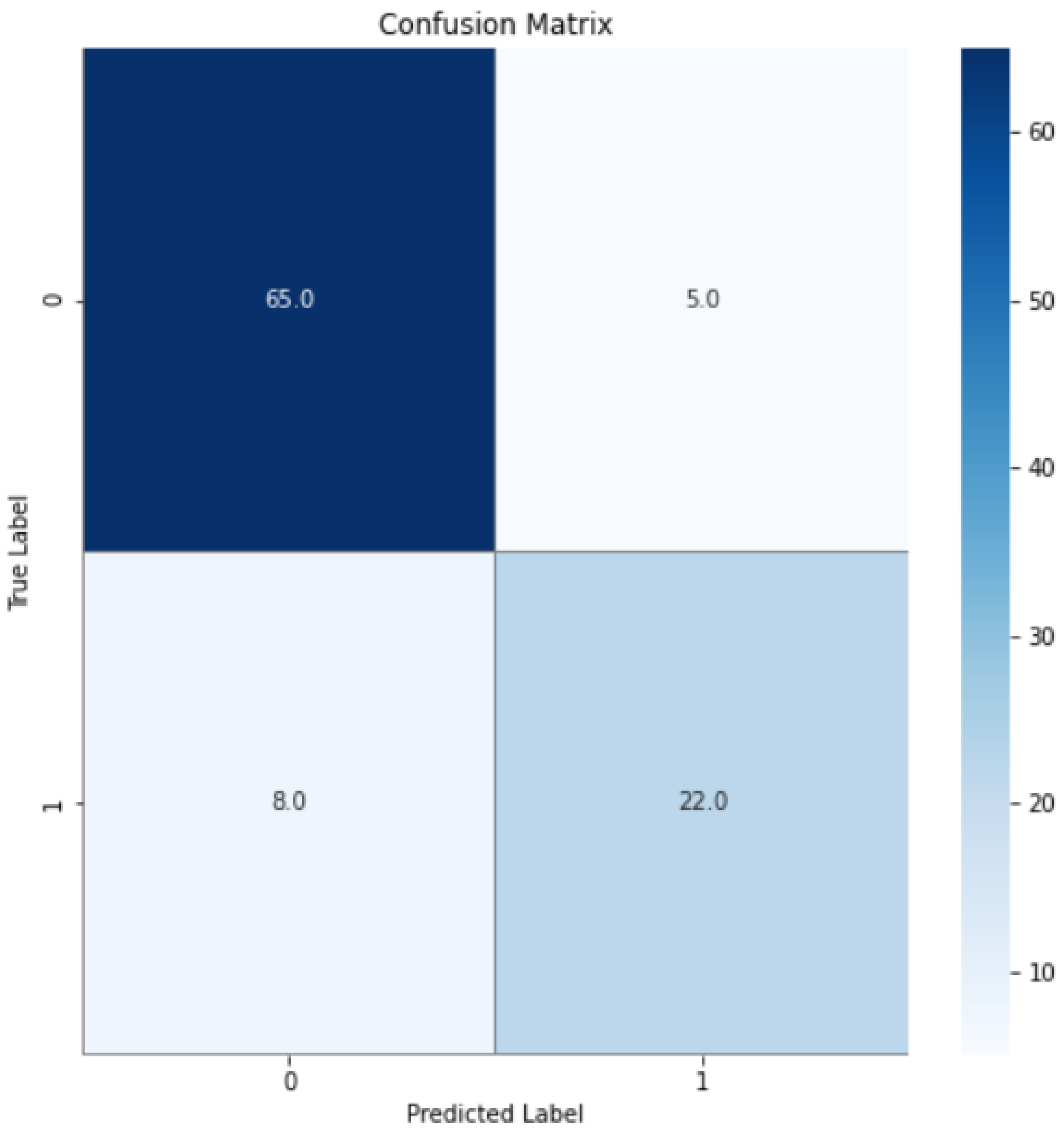
```
Validation: accuracy = 75.665302  ;  loss = 77.887653
```

Confusion matrix:

For basic model-



For tuned model-



Conclusion:

Hence the image classification of dog-vs-cats dataset is successfully implemented using ResNet-50 model. This can also be carried out with other transfer learning techniques for varied number of epochs. The main issue with the increasing number of epochs and the dataset size is the computational time. If that is not a problem then the classification can be done more efficiently with better accuracies.