_Applied_Machine_Learning

May 2022,

```
Project 3 ON APPLIED MACHINE LEARNING  CAT VS DOG IMAGE CLASSIFICATION
```

The Dogs vs. Cats dataset is a computer vision dataset that involves classifying photos as either containing a dog or cat. In this project,three models were developed and evaluated using the cat and dog dataset provided on canvas. Three Machine Learning models were implemented in this project which are Support Vector Classifier,RESNET and MOBILENETV2.

For the training data, I applied some transformations. This allows us to augment data which means that our model would be able to generalize better and prevent overfitting. The transformations are: Resizing: Brightness alteration: Horizontal flip: Rotation: the image can rotate to either side by a ° angle etc.

[1]:
```
from os import listdir from numpy import
asarray from numpy import save from
keras.preprocessing.image import load_img
from keras.preprocessing.image import
img_to_array import sys
from matplotlib import pyplot from keras.utils
import to_categorical from keras.models import
Sequential from keras.layers import Conv2D from
keras.layers import MaxPooling2D from
keras.layers import Dense from keras.layers
import Flatten,BatchNormalization from
keras.optimizers import SGD from
keras.preprocessing.image import
ImageDataGenerator from PIL import Image import
keras

from sklearn.model_selection import
train_test_split from sklearn.metrics import
classification_report from
keras.layers.convolutional import Conv2D from
keras.layers.convolutional import
MaxPooling2D from keras.layers.core import
Activation
```

```python
from keras.layers.core import Flatten
from keras.layers.core import Dropout
from keras.layers.core import Dense
from keras import backend as K
from keras.optimizers import RMSprop, Adam
from keras.regularizers import l2
from keras.utils import np_utils
from imutils import build_montages
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import cv2
import os
from sklearn.metrics import classification_report, log_loss, accuracy_score,
→roc_curve, auc
import matplotlib as mpl
from IPython.display import display
%matplotlib inline
import pandas as pd
import numpy as np
from PIL import Image
from resizeimage import resizeimage
from skimage.feature import hog
from skimage.color import rgb2grey
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from torchvision.datasets import ImageFolder
```

#MODEL 1 USING SUPPORT VECTOR CLASSIFIER#####

[2]:

```python
train_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/train"
test_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/test"
```

[3]:

```python
# importing images using torchvision dataset =
ImageFolder("C:/Users/soar/Desktop/dataset_dogs_vs_cats/train")

# creating labels dataframe imgs, labels =
zip(*dataset.imgs) imgs = list(imgs) labels =
list(labels) labels_df = pd.DataFrame({'image':
imgs, 'label':labels}) labels_df
```

[3]:

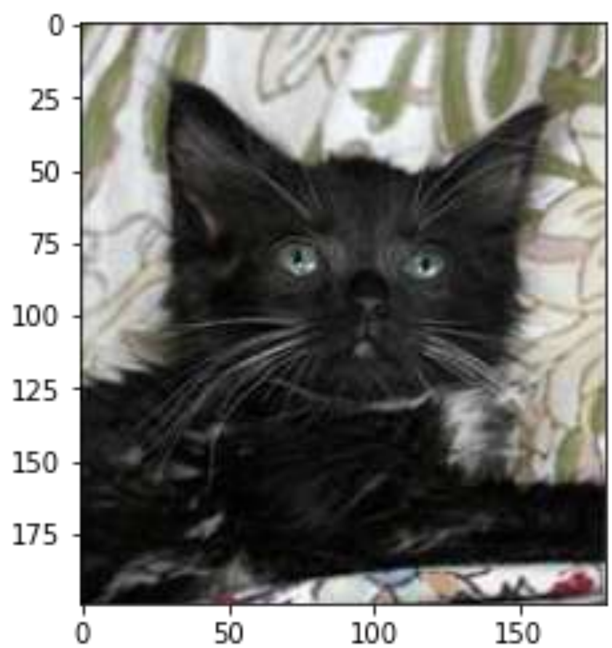
	image	label
0	C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…	0
1	C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…	0

```
2    C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    0
3    C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    0
4    C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    0
 ..                                                …    …
769  C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    1
770  C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    1
771  C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    1
772  C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    1
773  C:/Users/soar/Desktop/dataset_dogs_vs_cats/tra…    1

[774 rows x 2 columns]
```

[4]:

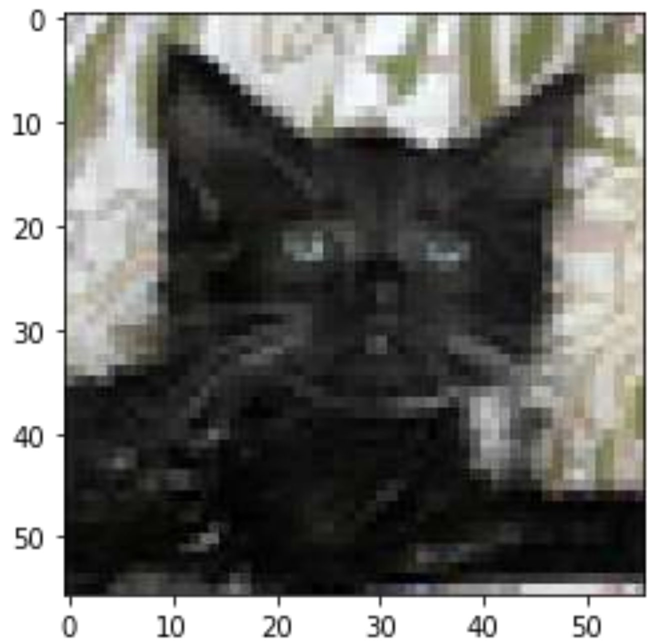
```
def get_image(path):
    img = Image.open(path)
    return np.array(img)

# showing a dog image
dog_row = labels_df[labels_df.label == 1].reset_index().image[23]
plt.imshow(get_image(dog_row))

# showing a cat image
cat_row = labels_df[labels_df.label == 0].reset_index().image[79]
plt.imshow(get_image(cat_row))
```

[4]: <matplotlib.image.AxesImage at 0x1a38eb793d0>



Resizing the image to our desired resolution.

[5]:
```
# image preprocessing
img = Image.open(cat_row)
img = resizeimage.resize_cover(img, [56, 56])#reshaping
plt.imshow(np.array(img), cmap='gray')
plt.show()
```



Creating image features and flattening it into a single row. Algorithms require data to be in a format where rows correspond to images and columns correspond to features. This means that all the information for a given image needs to be contained in a single row.

[6]:
```
def create_features(path):
    img = Image.open(path)
    img = resizeimage.resize_cover(img, [56, 56])
    img_arr = np.array(img)
    # flatten three channel color image
    color_features = img_arr.flatten()
    # convert image to greyscale
    grey_image = rgb2grey(img_arr)
    flat_features = np.hstack((color_features))
    return flat_features
dog_features = create_features(dog_row)
```

Looping over images Creating features for each image and then stacking the flattened features arrays into a big matrix that we can pass into our model. In the resulting features matrix, rows correspond to images and columns to features.

[7]:
```python
def create_feature_matrix(label_df):
    features_list = []

    for img_path in labels_df.image:
        # get features for image
        img_features = create_features(img_path)
        features_list.append(img_features)

    feature_matrix = np.array(features_list)
    return feature_matrix

feature_matrix = create_feature_matrix(labels_df)
```

Scale feature matrix. Many machine learning methods are built to work best with data that has a mean of 0 and unit variance. The features are scaled using the StandardScaler provided by scikit-learn

[8]:
```python
# get shape of feature matrix
print('Feature matrix shape is: ', feature_matrix.shape)

# define standard scaler
ss = StandardScaler()
# run this on our feature matrix
imgs_stand = ss.fit_transform(feature_matrix)
```

```
Feature matrix shape is:  (774, 9408)
```

Split into train and test sets Now we convert our data into train and test sets. We'll use 80% of images as our training data and test our model on the remaining 20%.

[9]:
```python
X_train, X_test, y_train, y_test = train_test_split(imgs_stand,
                                                    labels_df.label.values,
                                                    test_size=.2,
                                                    random_state=1234123)

# look at the distrubution of labels in the train set
pd.Series(y_train).value_counts()
```

[9]:
```
1    319
0    300
dtype: int64
```

Train model Now let's finally build our model! We'll do this using an SVM classifier with a linear kernel.

[10]:
```python
# define support vector classifier
svm = SVC(kernel='linear',probability=True, random_state=42)

# fit model
svm.fit(X_train, y_train)
```

[10]:
```
SVC(kernel='linear', probability=True, random_state=42)
```

Score model. Use our trained model to generate predictions for our data.

[11]:
```
# generate predictions
y_pred = svm.predict(X_test)

# calculate accuracy
accuracy = accuracy_score(y_pred, y_test)
print('Model accuracy is: ', accuracy)
```

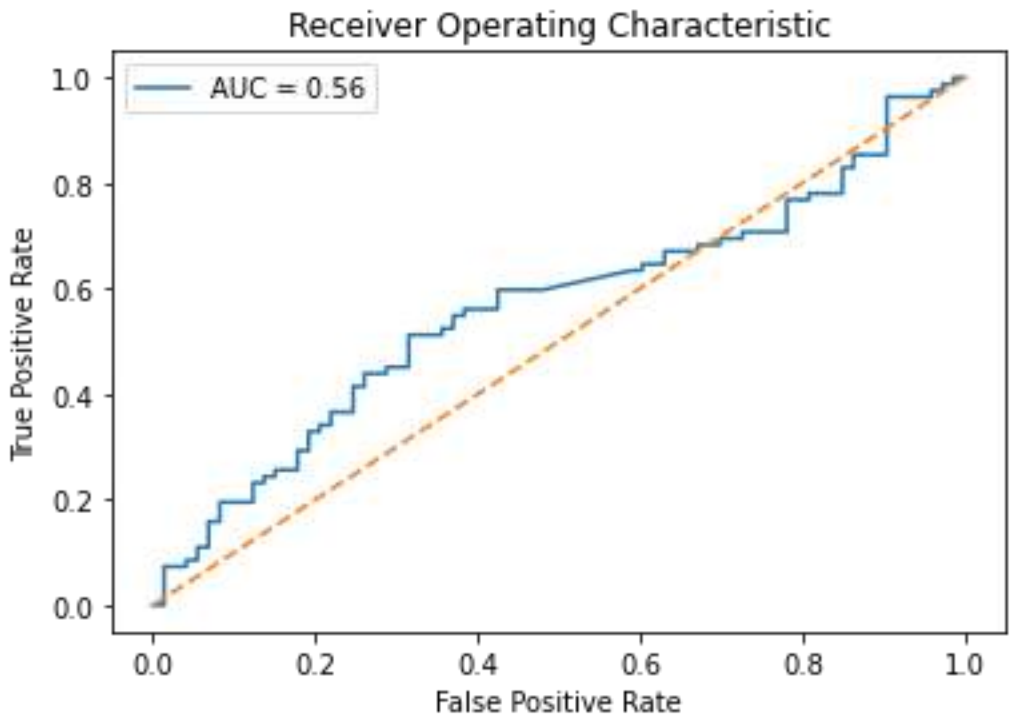
```
Model accuracy is:  0.5935483870967742
```

ROC curve + AUC

We'll use svm.predict_proba to get the probability that each class that is the true label. Using the default settings, probabilities of 0.5 or above are assigned a class label of 1.0 and those below are assigned a 0.0. However, this threshold can be adjusted. The (ROC curve) plots the false positive rate and true positive rate at different thresholds. ROC curves are judged visually by how close they are to the upper lefthand corner.

[12]:
```
# predict probabilities for X_test using predict_proba
probabilities = svm.predict_proba(X_test)

# select the probabilities for label 1.0
y_proba = probabilities[:, 1]

# calculate false positive rate and true positive rate at different thresholds
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, 
 ↪y_proba, pos_label=1)

# calculate AUC
roc_auc = auc(false_positive_rate, true_positive_rate)

plt.title('Receiver Operating Characteristic')
# plot the false positive rate on the x axis and the true positive rate on the 
 ↪y axis
roc_plot = plt.plot(false_positive_rate,
                    true_positive_rate,
                    label='AUC = {:0.2f}'.format(roc_auc))

plt.legend(loc=0)
plt.plot([0,1], [0,1], ls='--')
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate');
```



Visualizing Predictions Let's take random images from the test set and predict and display them individually.

[13]:
```
from random import randint

test = ImageFolder("C:/Users/soar/Desktop/dataset_dogs_vs_cats/test")
imgs, labels = zip(*test.imgs)
imgs = list(imgs)
labels = list(labels)
```

[14]:
```
random_ix = randint(0, len(imgs))
label = {0: 'Cat', 1:'Dog'}
rand_img = imgs[random_ix]
# create features of the image
test_features = create_features(rand_img)
# predict
prediction = svm.predict([test_features])
print("Prediction: " + label[prediction[0]])
print("Actual: " + label[labels[random_ix]])
# display image
display(Image.open(rand_img))
```

```
Prediction: Cat
Actual: Cat
```

USINGSECONDMODELRESNET RESNET enhances its network feature extraction ability through cross layer feature fusion, and network performance gradually improves with the deepening of network. For our implementation, the ResNet is a series of convolutional blocks that encapsulate: a convolutional layer, normalization of the data, a nonlinear activation function (RELU) and in some steps a max pooling layer. The ResNet model is implemented by skipping connections on two to three layers containing ReLU and batch normalization among the architectures 11. He et al. showed that the ResNet model performs better in image classification than other models, indicating that the image features were extracted well by ResNet The residual block on ResNet is defined as follows 11: =(,+)

where x is input layer; y is output layer; and F function is represented by the residual map. Residual block on ResNet can be accomplished if the input data dimensions are identical to the output data dimensions. In addition, each ResNet block consists of two layers (for ResNet-18 and ResNet-34 networks) or three layers (for ResNet-50 and ResNet-101 networks). The two initial layers of the ResNet architecture resemble GoogleNet by doing convolution 7 × 7 and max-pooling with size 3 × 3 with stride number 227. In this study, we used ResNet- 18 and ResNet-50 models.The weights of ResNet are initialized using Stochastics Gradient Descent's (SGD) with standard momentum parameters.

[15]:
```
train_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/train"
test_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/test"
```

#######SECOND MODEL USING RESNET50#########

[16]:
```
import tensorflow as tf
```

[17]:
```
img_size=224
```

[18]:
```
#Normalising Image
def normalize_data(img):

    #Normalize for ResNet50
    return tf.keras.applications.resnet50.preprocess_input(img)
```

[19]:
```
#Splitting to tain and test datagen
train_datagen=tf.keras.preprocessing.image.ImageDataGenerator(rotation_range=20,
                                                                              ␣
 ↪width_shift_range=20
                                                                          ␣
 ↪,height_shift_range=20,
                                                                          horizontal_flip=True,
                                                                          ␣
 ↪preprocessing_function=normalize_data)
test_datagen=tf.keras.preprocessing.image.
 ↪ImageDataGenerator(preprocessing_function=normalize_data)
```

[20]:
```
#Training (from dataframe)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(img_size,
                                                                 img_size),
                                                    batch_size=64,
                                                    shuffle=True,
                                                    seed=12345,
                                                    class_mode='categorical')
```

```
Found 774 images belonging to 2 classes.
```

[21]:
```
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(img_size, img_size),
    batch_size=64,
    shuffle=False,
    class_mode='categorical')
```

```
Found 228 images belonging to 2 classes.
```

[22]:
```
            tf.keras.backend.clear_session()            model              =
tf.keras.applications.ResNet50(include_top=False,       #classification
layer␣ ↪not included for imagenet
                                        input_shape=(img_size,img_size,3),
                                        weights='imagenet')
```

[23]:
```
#Set pre-trained model layers to not trainable
for layer in model.layers:
    layer.trainable = False
```

[24]:
```
#get Output layer of Pre-trained model
x1 = model.output

#Global average pool to reduce number of features and Flatten the output
x2 = tf.keras.layers.GlobalAveragePooling2D()(x1)
```

[25]:
```
#Add output layer
prediction = tf.keras.layers.Dense(2,activation='softmax')(x2)
```

[26]:
```
#Using Keras Model class
final_model = tf.keras.models.Model(inputs=model.input, #Pre-trained model␣
 ↪input as input layer
                                    outputs=prediction) #Output layer added
```

[27]:
```
#Compile the model
final_model.compile(optimizer='adam', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])
```

[28]:
```
Resnet= final_model.fit(train_generator,
                epochs=5,
                steps_per_epoch= 8005//64,
                validation_data=test_generator,
                validation_steps = 2023//64,
                callbacks=[])
```

```
Epoch 1/5
 13/125 [==>…] - ETA: 14:31 - loss: 0.6101 - accuracy:
0.6517WARNING:tensorflow:Your input ran out of data; interrupting
training. Make sure that your dataset or generator can generate at
least `steps_per_epoch * epochs` batches (in this case, 625 batches).
You may need to use the repeat() function when building your dataset.
WARNING:tensorflow:Your input ran out of data; interrupting training.
Make sure that your dataset or generator can generate at least
`steps_per_epoch * epochs` batches (in this case, 31 batches). You may
need to use the repeat() function when building your dataset.
125/125 [==============================] - 139s 1s/step - loss: 0.4506
accuracy: 0.7716 - val_loss: 0.1865 - val_accuracy: 0.9386
```

[29]:
```
 score = final_model.evaluate(test_generator,
verbose=0) print("Loss: " + str(score[0]))
print("Accuracy: " + str(score[1]*100) + "%")
```

```
Loss: 0.1864604651927948
Accuracy: 93.85964870452881%
```

[30]:
```python
load_img("C:/Users/soar/Desktop/test/42.jpg",target_size=(224,224))
```

[30]:

[31]:
```python
image=load_img("C:/Users/soar/Desktop/test/42.jpg",target_size=(224,224))

image=img_to_array(image)
image=image/255.0
prediction_image=np.array(image)
prediction_image= np.expand_dims(image, axis=0)
```

[32]:
```python
reverse_mapping={0:"cat",1:"dog"}

def mapper(value):
    return reverse_mapping[value]

prediction=final_model.predict(prediction_image)
value=np.argmax(prediction)
move_name=mapper(value)

#print(prediction)
#print(value)
print("Prediction is {}.".format(move_name))
```

```
Prediction is dog.
```

[33]:
```python
#THIRD MODEL USING MOBILENETV2
```

MobileNetV2 is very similar to the original MobileNet, except that it uses inverted residual blocks with bottlenecking features. It has a drastically lower parameter count than the original MobileNet.

MobileNets support any input size greater than 32 x 32, with larger image sizes offering better performance. This function returns a Keras image classification model, optionally loaded with weights pre-trained on ImageNet. Arguments

input_shape: Optional shape tuple, which specifies if you would like to use a model with an input image resolution that is not (224, 224, 3). It has exactly 3 inputs channels (150, 150, 3). alpha: Float between 0 and 1. controls the width of the network. This is known as the width multiplier in the MobileNetV2 paper, but the name is kept for consistency with applications.MobileNetV1 model in Keras. If alpha < 1.0, proportionally decreases the number of filters in each layer. If alpha > 1.0, proportionally increases the number of filters in each layer. If alpha = 1, default number of filters from the paper are used at each layer. include_top: Boolean, whether to include the fully-connected layer at the top of the network. Defaults to True. weights: String, one of None (random initialization), `imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded. input_tensor: Optional Keras tensor (i.e. output of layers.Input()) to use as image input for the model. pooling: String, optional pooling mode for feature extraction which includes_top is False. None means that the output of the model will be the 4D tensor output of the last convolutional block. avg means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor. max means that global max pooling will be applied. classes: Integer, optional number of classes to classify images into, only to be specified if include_top is True, and if no weights argument is specified. classifier_activation: A str or callable. The activation function to use on the ``top'' layer. Ignored unless include_top=True. Set classifier_activation=None to return the logits of the ``top'' layer. When loading pretrained weights, classifier_activation can only be None or ``softmax''.

[34]:
```
#Image directory
train_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/train"
test_dir="C:/Users/soar/Desktop/dataset_dogs_vs_cats/test"
```

[35]:
```
#Loading the image,removing DS_Store
dataset=[]
mapping={"dogs":0,"cats":1}
count=0

for file in os.listdir(train_dir):
    path=os.path.join(train_dir,file)
    for im in os.listdir(path):
        if im != '_DS_Store':
            image=load_img(os.path.join(path,im), grayscale=False,␣
↪color_mode='rgb', target_size=(150,150))
            image=img_to_array(image)
            image=image/255.0
            dataset.append([image,count])
    count+=1
```

[36]:
```
testset=[]
mapping={"dogs":0,"cats":1}
count=0

for file in os.listdir(test_dir):
    path=os.path.join(test_dir,file)
    for im in os.listdir(path):
        if im != '_DS_Store':
            image=load_img(os.path.join(path,im), grayscale=False,␣
↪color_mode='rgb', target_size=(150,150))
            image=img_to_array(image)
            image=image/255.0
            testset.append([image,count])
    count+=1
```

[37]:
```
data,labels0=zip(*dataset)
test,tlabels0=zip(*testset)
```

[38]:
```
labels1=to_categorical(labels0)
data=np.array(data)
labels=np.array(labels1)
print("Data Shape:{}\nLabels shape: {}".format(data.shape,labels.shape))
```

```
Data Shape:(774, 150, 150, 3)
Labels shape: (774, 2)
```

[39]:
```
     tlabels1=to_categorical(tlabels0)      test=np.array(test)
tlabels=np.array(tlabels1) print("Test Shape:{}\nTLabels shape:
{}".format(test.shape,tlabels.shape))
```

```
Test Shape:(228, 150, 150, 3)
TLabels shape: (228, 2)
```

[40]:
```
#Splitting the dataset
trainx,testx,trainy,testy=train_test_split(data,labels,test_size=0.
 ↪2,random_state=44)
```

[41]:
```
print(trainx.shape)
print(testx.shape)
print(trainy.shape)
print(testy.shape)
```

```
(619, 150, 150, 3)
(155, 150, 150, 3)
(619, 2)
(155, 2)
```

[42]:
```
datagen =␣
 ↪ImageDataGenerator(horizontal_flip=True,vertical_flip=True,rotation_range=20,zo
 om_range=0.
 ↪2, width_shift_range=0.2,height_shift_range=0.2,shear_range=0.
 ↪1,fill_mode="nearest")
```

[43]:
```
pretrained_model = tf.keras.applications.
 ↪MobileNetV2(input_shape=(150,150,3),alpha=1.
 ↪0,include_top=False,weights='imagenet',pooling='avg')
pretrained_model.trainable = False
```

```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows`
is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224)
will be loaded as the default.
```

[44]:
```
inputs = pretrained_model.input x = tf.keras.layers.Dense(128,
activation='relu')(pretrained_model.output) outputs =
tf.keras.layers.Dense(2, activation='softmax')(x) model =
tf.keras.Model(inputs=inputs, outputs=outputs)
```

[45]:
```
model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[46]:
```
his=model.fit(datagen.
 ↪flow(trainx,trainy,batch_size=32),validation_data=(testx,testy),epochs=30)#Fitt
 ing␣
```

```
↪the model

Epoch 1/30
20/20 [==============================] - 20s 813ms/step - loss: 0.9107
-
accuracy: 0.6468 - val_loss: 0.1823 - val_accuracy: 0.9290
Epoch 2/30
20/20 [==============================] - 16s 815ms/step - loss: 0.3104
accuracy: 0.8575 - val_loss: 0.1375 - val_accuracy: 0.9484
Epoch 3/30
20/20 [==============================] - 16s 788ms/step - loss: 0.2596
accuracy: 0.8942 - val_loss: 0.1416 - val_accuracy: 0.9355
Epoch 4/30
20/20 [==============================] - 16s 789ms/step - loss: 0.2626
accuracy: 0.8979 - val_loss: 0.1411 - val_accuracy: 0.9419
Epoch 5/30
20/20 [==============================] - 16s 790ms/step - loss: 0.2472
accuracy: 0.8855 - val_loss: 0.1039 - val_accuracy: 0.9548
Epoch 6/30
20/20 [==============================] - 15s 758ms/step - loss: 0.2226
-
accuracy: 0.8857 - val_loss: 0.1032 - val_accuracy:
0.9548 Epoch 7/30
20/20 [==============================] - 15s 736ms/step - loss: 0.2009
-
accuracy: 0.9165 - val_loss: 0.1291 - val_accuracy: 0.9355
Epoch 8/30
20/20 [==============================] - 15s 746ms/step - loss: 0.1948
-
accuracy: 0.9133 - val_loss: 0.0919 - val_accuracy: 0.9484
Epoch 9/30
20/20 [==============================] - 15s 755ms/step - loss: 0.1661
-
accuracy: 0.9217 - val_loss: 0.1095 - val_accuracy: 0.9355
Epoch 10/30
20/20 [==============================] - 15s 743ms/step - loss: 0.1739
-
accuracy: 0.9160 - val_loss: 0.0941 - val_accuracy: 0.9548
Epoch 11/30
20/20 [==============================] - 15s 743ms/step - loss: 0.1864
-
accuracy: 0.9161 - val_loss: 0.1216 - val_accuracy: 0.9355
Epoch 12/30
20/20 [==============================] - 15s 740ms/step - loss: 0.1708
accuracy: 0.9241 - val_loss: 0.1265 - val_accuracy: 0.9419
Epoch 13/30
```

```
20/20 [==============================] - 15s 729ms/step - loss: 0.1743
-
accuracy: 0.9183 - val_loss: 0.1593 - val_accuracy: 0.9355
Epoch 14/30
20/20 [==============================] - 15s 744ms/step - loss: 0.2035
-
accuracy: 0.8964 - val_loss: 0.1039 - val_accuracy: 0.9419
Epoch 15/30
20/20 [==============================] - 15s 743ms/step - loss: 0.1546
-
accuracy: 0.9414 - val_loss: 0.1220 - val_accuracy: 0.9548
Epoch 16/30
20/20 [==============================] - 16s 797ms/step - loss: 0.1533
-
accuracy: 0.9320 - val_loss: 0.1076 - val_accuracy: 0.9484
Epoch 17/30
20/20 [==============================] - 15s 740ms/step - loss: 0.2542
-
accuracy: 0.9074 - val_loss: 0.0971 - val_accuracy: 0.9484
Epoch 18/30
20/20 [==============================] - 15s 761ms/step - loss: 0.1701
accuracy: 0.9357 - val_loss: 0.2030 - val_accuracy: 0.9226
Epoch 19/30
20/20 [==============================] - 15s 744ms/step - loss: 0.2004
accuracy: 0.9094 - val_loss: 0.1017 - val_accuracy: 0.9419
Epoch 20/30
20/20 [==============================] - 15s 747ms/step - loss: 0.1650
accuracy: 0.9385 - val_loss: 0.1045 - val_accuracy: 0.9484
Epoch 21/30
20/20 [==============================] - 15s 751ms/step - loss: 0.1533
accuracy: 0.9430 - val_loss: 0.1087 - val_accuracy: 0.9419
Epoch 22/30
20/20 [==============================] - 15s 749ms/step - loss: 0.0880
-
accuracy: 0.9761 - val_loss: 0.1186 - val_accuracy:
0.9484 Epoch 23/30
20/20 [==============================] - 15s 748ms/step - loss: 0.1463
-
accuracy: 0.9440 - val_loss: 0.1423 - val_accuracy: 0.9419
Epoch 24/30
20/20 [==============================] - 15s 743ms/step - loss: 0.1202
-
accuracy: 0.9563 - val_loss: 0.1301 - val_accuracy: 0.9484
Epoch 25/30
20/20 [==============================] - 15s 736ms/step - loss: 0.0866
-
```

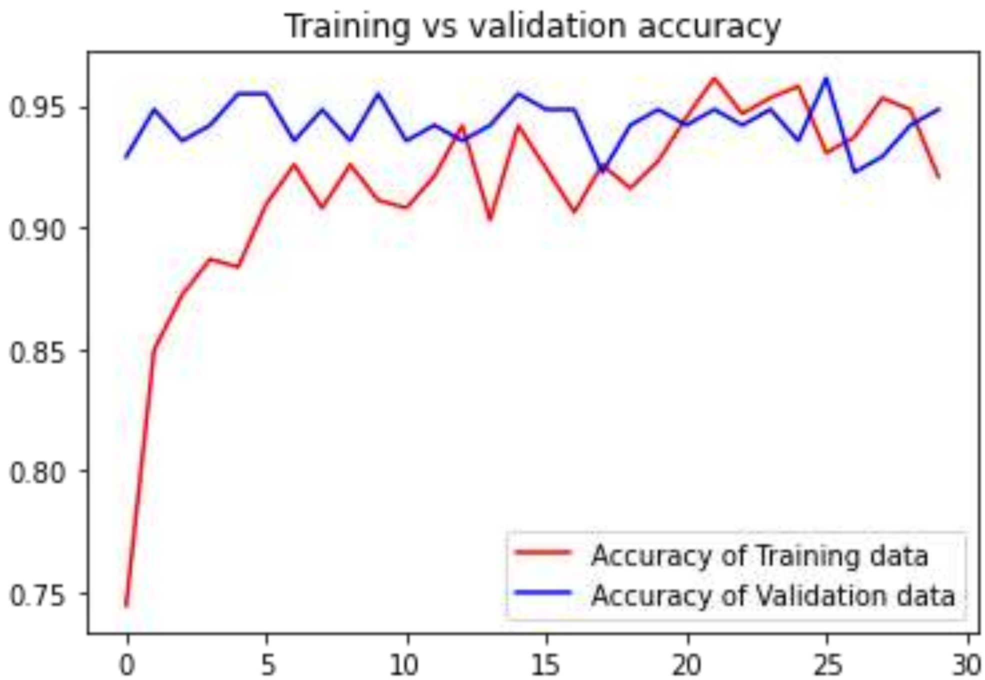
```
accuracy: 0.9702 - val_loss: 0.1458 - val_accuracy: 0.9355
Epoch 26/30
20/20 [==============================] - 15s 741ms/step - loss: 0.1709
-
accuracy: 0.9298 - val_loss: 0.1008 - val_accuracy: 0.9613
Epoch 27/30
20/20 [==============================] - 15s 748ms/step - loss: 0.1699
-
accuracy: 0.9226 - val_loss: 0.1506 - val_accuracy: 0.9226
Epoch 28/30
20/20 [==============================] - 15s 744ms/step - loss: 0.1199
accuracy: 0.9582 - val_loss: 0.1779 - val_accuracy: 0.9290
Epoch 29/30
20/20 [==============================] - 15s 736ms/step - loss: 0.1283
-
accuracy: 0.9448 - val_loss: 0.1621 - val_accuracy: 0.9419
Epoch 30/30
20/20 [==============================] - 15s 764ms/step - loss: 0.1610
accuracy: 0.9211 - val_loss: 0.1160 - val_accuracy: 0.9484
```

[47]:
```
   y_pred=model.predict(testx)#Model
accuracy
pred=np.argmax(y_pred,axis=1)  ground
=             np.argmax(testy,axis=1)
print(classification_report(ground,p
red))
```

```
              precision    recall  f1-score   support

           0       0.97      0.92      0.95        76
           1       0.93      0.97      0.95        79

    accuracy                           0.95       155
   macro avg       0.95      0.95      0.95       155
weighted avg       0.95      0.95      0.95       155
```

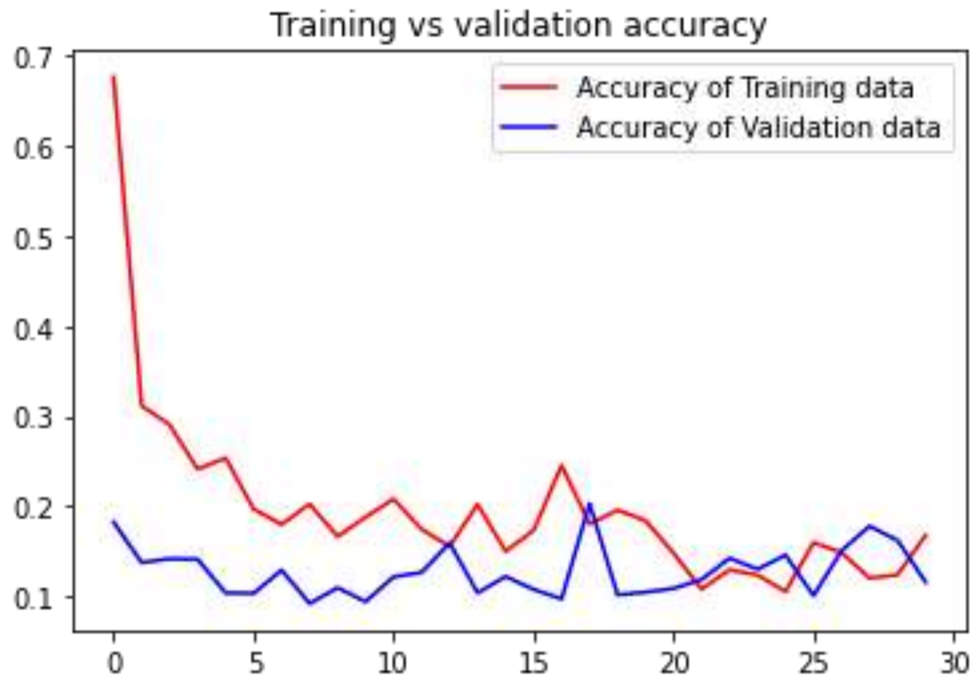
[48]:
```
get_acc = his.history['accuracy']
value_acc =
his.history['val_accuracy'] get_loss
= his.history['loss']
validation_loss =
his.history['val_loss']

epochs = range(len(get_acc)) plt.plot(epochs, get_acc, 'r',
label='Accuracy of Training data') plt.plot(epochs,
value_acc, 'b', label='Accuracy of Validation data')
plt.title('Training vs validation accuracy')
```

```
plt.legend(loc=0)
plt.figure()
plt.show()
```




```
<Figure size 432x288 with 0 Axes>
```

```
[49]: epochs = range(len(get_loss)) plt.plot(epochs, get_loss, 'r',
label='Accuracy of Training data') plt.plot(epochs,
validation_loss, 'b', label='Accuracy of Validation data')
plt.title('Training vs validation accuracy')
plt.legend(loc=0)
plt.figure()
plt.show()
```



<Figure size 432x288 with 0 Axes> [50]:

[50]:
```
load_img("C:/Users/soar/Desktop/test/42.jpg",target_size=(150,150))
```

[51]:
```
image=load_img("C:/Users/soar/Desktop/test/42.jpg",target_size=(150,150))

image=img_to_array(image)
image=image/255.0
prediction_image=np.array(image)
prediction_image= np.expand_dims(image, axis=0)
```

[52]:
```
reverse_mapping={0:"cat",1:"dog"}

def mapper(value):
    return reverse_mapping[value]

prediction=model.predict(prediction_image)
value=np.argmax(prediction)
move_name=mapper(value)

#print(prediction)
#print(value)
print("Prediction is {}.".format(move_name))
```

Prediction is dog.

[53]:
```
load_img("C:/Users/soar/Desktop/test/45.jpg",target_size=(150,150))
```

[53]:

[54]:
```
image=load_img("C:/Users/soar/Desktop/test/45.jpg",target_size=(150,150))

image=img_to_array(image)
image=image/255.0
prediction_image=np.array(image)
prediction_image= np.expand_dims(image, axis=0)
```

[55]:
```
reverse_mapping={0:"cat",1:"dog"}

def mapper(value):
    return reverse_mapping[value]

prediction=model.predict(prediction_image)
value=np.argmax(prediction)
move_name=mapper(value)
```

```
#print(prediction)
#print(value)
print("Prediction is {}.".format(move_name))
```

```
Prediction is cat.
```

[56]:
```
print(test.shape)
prediction2=model.predict(test)
print(prediction2.shape)

PRED=[]
for item in prediction2:
    value2=np.argmax(item)
    PRED+=[value2]
```

```
(228, 150, 150, 3)
(228, 2)
```

[57]:
```
ANS=tlabels0
```

[58]:
```
accuracy=accuracy_score(ANS,PRED)
print(accuracy)
```

```
0.9429824561403509
```

Conclusion

First model using Support Vector Classifier had an accuracy of 59% though when it comes to prediction it prediction some correctly and few wrongly. Second model using RESNET achieved an accuracy of 92% with good predictions Third model MOBILENETV2 which is the best model amongst the three achieved an accuracy of 94% with good predictions too.