IMAGE SCRAPING AND CLASSIFICATION PROJECT

Submitted by:

SHUBHAM SHUKLA

ACKNOWLEDGMENT

I would like to thank Flip Robo Technologies for providing me with the opportunity to work on this project from which I have learned a lot. I am also grateful to my SME, Mr. Shubham Yadav for his constant guidance. Some of the reference sources are as follows:

- Stack Overflow
- Medium.com
- scikit-learn.org
- Python official documentation

INTRODUCTION

BUSINESS PROBLEM FRAMING

Image classification is a supervised learning problem: define a set of target classes (objects to identify in images), and train a model to recognize them using labeled example photos. Early computer vision models relied on raw pixel data as the input to the model. The position of the object, background behind the object, ambient lighting, camera angle, and camera focus all can produce fluctuation in raw pixel data; these differences are significant enough that they cannot be corrected for by taking weighted averages of pixel RGB values.

The advancements in the field of autonomous driving also serve as a great example of the use of image classification in the real-world. For example, we can build an image classification model that recognizes various objects, such as other vehicles, pedestrians, traffic lights, and signposts on the road.

The idea behind this project is to build a deep learning-based Image Classification model on images that will be scraped from e-commerce portal. This is done to make the model more and more robust.

CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM

REVIEW OF LITERATURE

Classification between objects is a fairly easy task for us, but it has proved to be a complex one for machines and therefore image classification has been an important task within the field of computer vision.
Image classification refers to the labeling of images into one of a number of predefined classes.

There are potentially *n* number of classes in which a given image can be classified. Manually checking and classifying images could be a tedious task especially when they are massive in number (say 10,000) and therefore it will be very useful if we could automate this entire process using computer vision.

MOTIVATION FOR THE PROBLEM UNDERTAKEN

Image classification is the primary domain, in which deep neural networks play the most important role of medical image analysis. The image classification accepts the given input

images and produces output classification for identifying whether the disease is present or not.

Image classification is a complex process that may be affected by many factors. Because classification results are the basis for many environmental and socioeconomic applications, scientists and practitioners have made great efforts in developing advanced classification approaches and techniques for improving classification accuracy. Image classification is used in a lot in basic fields like medicine, education and security. Correct classification has vital importance, especially in medicine. Therefore, improved methods are needed in this field. The proposed deep CNNs are an often-used architecture for deep learning and have been widely used in computer vision and audio recognition.

ANALYTICAL PROBLEM FRAMING

DATA SOURCES AND THEIR FORMATS

Data for the project is being scrapped from amazon.in using python web scrapping libraries such as selenium, beautifulsoup, etc. More than 200 images of saree, jeans and trousers are scrapped for the project and saved into individual folders.

```
#function to make directory
def make_directory(dirname):
    current_path=os.getcwd()
    path=os.path.join(current_path, dirname)
    if not os.path.exists(path):
        os.makedirs(path)

#function to scrape images
def  scrap_images_url(driver):


    s=driver.find_elements_by_xpath("//div[@class='a-section aok-relative s-image-tall-aspect']//img")
    print(len(s))
    product_data={}

    product_data['image_urls']=[]

    for image in s:
        source=image.get_attribute('src')
        product_data["image_urls"].append(source)
    print("R S Data")

    return product_data

#function to sace images in the directory
def save_images(data, dirname, page):
    for index,link in enumerate(data['image_urls']):
        response=requests.get(link)
        with open("{0}/img _{0}{1}{2}.jpeg".format(dirname,page,index),"wb") as file:
            file.write(response.content)
```

```
driver= webdriver.Chrome(r"chromedriver")
currentpageurl=driver.get('https://www.amazon.in/s?k=sarees&ref=nb_sb_noss')
DIRNAME="Sarees"
make_directory(DIRNAME)
start_page=1
total_pages=5
for page in range(start_page,total_pages):
    time.sleep(2)
    try:
        prod_details=scrap_images_url(driver=driver)
        print("Scrapping page {0} of {1} pages".format(page,total_pages))

        # Downloading the images
        save_images(data=prod_details,dirname=DIRNAME,page=page)
        print("Scrapping of page{0}Done!!".format(page))

        # Moving to the next page
        print("Moving to the next page")
        try:

            driver.find_element_by_xpath("//a[@class='s-pagination-item s-pagination-next s-pagination-button s-paginati

        except:
            driver.find_element_by_xpath("//li[@class='a-last']//a").click()

    except StaleElementReferenceException as Exception:
        print("We are facing an exception")
        print("The page value at the time out exception is {}".format(exception_page))

        # Moving to the next page
        print("Moving to the next page")
```

```
        # Moving to the next page
        print("Moving to the next page")
        try:
            driver.find_element_by_xpath("//a[@class='s-pagination-item s-pagination-next s-pagination-button s-paginati

        except:
            driver.find_element_by_xpath("//li[@class='a-last']//a").click()

        print("the new page is {}".format(new_page))
```

```
68
R S Data
Scrapping page 1 of 5 pages
Scrapping of page1Done!!
Moving to the next page
60
R S Data
Scrapping page 2 of 5 pages
Scrapping of page2Done!!
Moving to the next page
60
R S Data
Scrapping page 3 of 5 pages
Scrapping of page3Done!!
Moving to the next page
60
R S Data
Scrapping page 4 of 5 pages
Scrapping of page4Done!!
Moving to the next page
```

```
#scraping Jeans(men)
```

DATA PREPROCESSING DONE

Data Pre-processing

```
43]:
# Validation
Data_gen=ImageDataGenerator(
    # used to rescale the pixel values from [0, 255] to [0, 1] interval
    rescale=1./255)
validation_generator=Data_gen.flow_from_directory(validation_data_dir,
                                                  target_size=(img_width,img_height),
                                                  batch_size=32,
                                                  class_mode='categorical',
                                                 shuffle=False)

# Training

train_generator=train_generator_augmented.flow_from_directory(train_data_dir,
                                                              target_size=(img_width,img_height),
                                                              batch_size=batch_size,
                                                              class_mode='categorical')


Found 120 images belonging to 3 classes.
Found 646 images belonging to 3 classes.
```

```
44]:
# checking class indices
train_generator.class_indices

Out[44]: {'Jeans': 0, 'Saree': 1, 'Trousers': 2}
```

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

HARDWARE:

HP Pavilion X360

SOFTWARE:

Jupyter Notebook (Anaconda 3) – Python 3.9, TensorFlow-

2.5.0Microsoft Office 365 Package

LIBRARIES USED:

```
import pandas as pd
import numpy as np
import matplotlib.cm as cm
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout, Activation, Conv2D, MaxPooling2D,BatchNormalization
from tensorflow.keras import optimizers
import os
from os import listdir
import shutil
import random
import scipy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from matplotlib.image import imread
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing import image
```

```
#Importing required libraries
from selenium import webdriver
from selenium.common.exceptions import StaleElementReferenceException
import shutil
import os
import pandas as pd
import requests
import time
```

MODEL/S DEVELOPMENT AND EVALUATION

IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS)

Data Augmentation

```
# Creating our data generator for our training data
train_generator_augmented=ImageDataGenerator(
                            rotation_range=30, # rotate the image 20 degrees
                            width_shift_range=0.10, # Shift the pic width by a max of 5%
                            height_shift_range=0.10, # Shift the pic height by a max of 5%
                            rescale=1./255, # Rescale the image by normalzing it.
                            shear_range=0.2, # Shear means cutting away part of the image (max 20%)
                            zoom_range=0.2, # Zoom in by 20% max
                            horizontal_flip=True, # Allo horizontal flipping
                            fill_mode='nearest' # Fill in missing pixels with the nearest filled value
                           )
```

EarlyStopping and ModelCheckpoint

```
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=30)
mc = ModelCheckpoint('best_model.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

Training

```
history = model.fit(
    train_generator,
    epochs=epoch,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples//batch_size,
    steps_per_epoch=nb_train_samples//batch_size,
    callbacks=[es, mc]

)
```

RUN AND EVALUATE SELECTED MODELS

Training our model

```
input_shape=(128,128,3)
img_width=128
img_height=128

batch_size=12
epoch=100

train_data_dir='./clothes/train'
validation_data_dir='./clothes/test'

nb_train_samples=167
nb_validation_samples=40

model=Sequential()

# This is the first convolution

model.add(Conv2D(32,(3,3),padding='same',input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# This is the Second convolution

model.add(Conv2D(32,(3,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# This is the third convolution

model.add(Conv2D(64,(3,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# This is the fourth convolution

model.add(Conv2D(64,(3,3),padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Flatten the results to feed into a DNN

```

```
model.add(Dropout(0.25))

# Flatten the results to feed into a DNN


model.add(Flatten())
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(3))
model.add(Activation('softmax'))
print(model.summary())


model.compile(loss='categorical_crossentropy',optimizer = RMSprop(learning_rate = 0.001),metrics=['accuracy'])
```

```
Model: "sequential_3"
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 128, 128, 32)	896
activation_18 (Activation)	(None, 128, 128, 32)	0
max_pooling2d_12 (MaxPooling	(None, 64, 64, 32)	0
dropout_15 (Dropout)	(None, 64, 64, 32)	0
conv2d_13 (Conv2D)	(None, 64, 64, 32)	9248
activation_19 (Activation)	(None, 64, 64, 32)	0
max_pooling2d_13 (MaxPooling	(None, 32, 32, 32)	0
dropout_16 (Dropout)	(None, 32, 32, 32)	0
conv2d_14 (Conv2D)	(None, 32, 32, 64)	18496
activation_20 (Activation)	(None, 32, 32, 64)	0
max_pooling2d_14 (MaxPooling	(None, 16, 16, 64)	0
dropout_17 (Dropout)	(None, 16, 16, 64)	0

```
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 16, 16, 64)        36928
_________________________________________________________________
activation_21 (Activation)   (None, 16, 16, 64)        0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
dropout_18 (Dropout)         (None, 8, 8, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 4096)              0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               524416
_________________________________________________________________
activation_22 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 387
_________________________________________________________________
activation_23 (Activation)   (None, 3)                 0
=================================================================
Total params: 590,371
Trainable params: 590,371
Non-trainable params: 0
_________________________________________________________________
None
```

```python
history = model.fit(
    train_generator,
    epochs=epoch,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples//batch_size,
    steps_per_epoch=nb_train_samples//batch_size,
    callbacks=[es, mc]

)
```

```
Epoch 1/100
13/13 [==============================] - 8s 494ms/step - loss: 1.3677 - accuracy: 0.3782 - val_loss: 1.0896 - val_accuracy:
0.4167

Epoch 00001: val_accuracy improved from -inf to 0.41667, saving model to best_model.h5
Epoch 2/100
13/13 [==============================] - 5s 357ms/step - loss: 1.0943 - accuracy: 0.4359 - val_loss: 1.1232 - val_accuracy:
0.1667

Epoch 00002: val_accuracy did not improve from 0.41667
Epoch 3/100
13/13 [==============================] - 4s 328ms/step - loss: 1.0647 - accuracy: 0.4615 - val_loss: 1.0600 - val_accuracy:
0.5833
```

```
Epoch 00003: val_accuracy improved from 0.41667 to 0.58333, saving model to best_model.h5
Epoch 4/100
13/13 [==============================] - 5s 399ms/step - loss: 0.9700 - accuracy: 0.5256 - val_loss: 0.8891 - val_accuracy:
0.7812

Epoch 00004: val_accuracy improved from 0.58333 to 0.78125, saving model to best_model.h5
Epoch 5/100
13/13 [==============================] - 4s 322ms/step - loss: 0.7876 - accuracy: 0.6346 - val_loss: 0.8478 - val_accuracy:
0.6250

Epoch 00005: val_accuracy did not improve from 0.78125
Epoch 6/100
13/13 [==============================] - 4s 315ms/step - loss: 0.6919 - accuracy: 0.6346 - val_loss: 0.6534 - val_accuracy:
0.5938

Epoch 00006: val_accuracy did not improve from 0.78125
Epoch 7/100
13/13 [==============================] - 4s 313ms/step - loss: 0.6705 - accuracy: 0.6538 - val_loss: 0.7104 - val_accuracy:
0.5833

Epoch 00007: val_accuracy did not improve from 0.78125
Epoch 8/100
13/13 [==============================] - 4s 309ms/step - loss: 0.6820 - accuracy: 0.5962 - val_loss: 0.4977 - val_accuracy:
0.8229

Epoch 00008: val_accuracy improved from 0.78125 to 0.82292, saving model to best_model.h5
Epoch 9/100
13/13 [==============================] - 4s 286ms/step - loss: 0.6737 - accuracy: 0.6603 - val_loss: 0.6271 - val_accuracy:
0.5833
Epoch 00009: val_accuracy did not improve from 0.82292
Epoch 10/100
13/13 [==============================] - 4s 285ms/step - loss: 0.5245 - accuracy: 0.6987 - val_loss: 0.5468 - val_accuracy:
0.5417

Epoch 00010: val_accuracy did not improve from 0.82292
Epoch 11/100
13/13 [==============================] - 4s 283ms/step - loss: 0.7341 - accuracy: 0.5897 - val_loss: 0.5143 - val_accuracy:
0.8854

Epoch 00011: val_accuracy improved from 0.82292 to 0.88542, saving model to best_model.h5
Epoch 12/100
13/13 [==============================] - 4s 287ms/step - loss: 0.5377 - accuracy: 0.7273 - val_loss: 0.4142 - val_accuracy:
0.8750

Epoch 00012: val_accuracy did not improve from 0.88542
Epoch 13/100
13/13 [==============================] - 4s 284ms/step - loss: 0.5638 - accuracy: 0.7372 - val_loss: 0.4055 - val_accuracy:
0.8958

Epoch 00013: val_accuracy improved from 0.88542 to 0.89583, saving model to best_model.h5
Epoch 14/100
13/13 [==============================] - 4s 286ms/step - loss: 0.5358 - accuracy: 0.7179 - val_loss: 0.3624 - val_accuracy:
0.9375

Epoch 00014: val_accuracy improved from 0.89583 to 0.93750, saving model to best_model.h5
Epoch 15/100
13/13 [==============================] - 4s 281ms/step - loss: 0.4607 - accuracy: 0.7821 - val_loss: 0.6013 - val_accuracy:
0.7188
```

```
Epoch 00077: val_accuracy did not improve from 0.96875
Epoch 78/100
13/13 [==============================] - 4s 282ms/step - loss: 0.5875 - accuracy: 0.8141 - val_loss: 0.2634 - val_accuracy:
0.9062

Epoch 00078: val_accuracy did not improve from 0.96875
Epoch 79/100
13/13 [==============================] - 4s 278ms/step - loss: 0.2512 - accuracy: 0.9026 - val_loss: 0.2259 - val_accuracy:
0.8958

Epoch 00079: val_accuracy did not improve from 0.96875
Epoch 80/100
13/13 [==============================] - 4s 286ms/step - loss: 0.3215 - accuracy: 0.8654 - val_loss: 0.2491 - val_accuracy:
0.8958

Epoch 00080: val_accuracy did not improve from 0.96875
Epoch 81/100
13/13 [==============================] - 4s 282ms/step - loss: 0.3655 - accuracy: 0.8462 - val_loss: 0.2304 - val_accuracy:
0.9167


Epoch 00081: val_accuracy did not improve from 0.96875
Epoch 82/100
13/13 [==============================] - 4s 281ms/step - loss: 0.4779 - accuracy: 0.8269 - val_loss: 0.2891 - val_accuracy:
0.8854

Epoch 00082: val_accuracy did not improve from 0.96875
Epoch 00082: early stopping
```

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

```
losses = pd.DataFrame(model.history.history)
losses
```

48]:

	loss	accuracy	val_loss	val_accuracy
0	1.367717	0.378205	1.089632	0.416667
1	1.094335	0.435897	1.123187	0.166667
2	1.064728	0.461538	1.060041	0.583333
3	0.970046	0.525641	0.889132	0.781250
4	0.787574	0.634615	0.847837	0.625000
...	...	...	...	...
77	0.587525	0.814103	0.263357	0.906250
78	0.251201	0.902597	0.225921	0.895833
79	0.321486	0.865385	0.249084	0.895833
80	0.365542	0.846154	0.230393	0.916667
81	0.477936	0.826923	0.289129	0.885417

82 rows × 4 columns

VISUALIZATIONS

```
#Lets see first two images of each  saree_dir_train, jean_dir_train, trouser_dir_train dataset
Dir=[saree_dir_train, jean_dir_train, trouser_dir_train]

import matplotlib.image as mpimg
for di in Dir:
    k=listdir(di)
    for i in k[:2]:
        img=mpimg.imread('{}/{}'.format(di,i))
        plt.imshow(img)
        plt.axis('off')
        plt.show()
```

d2459faa4f5e007c2335a4cd392749c9bda933181...

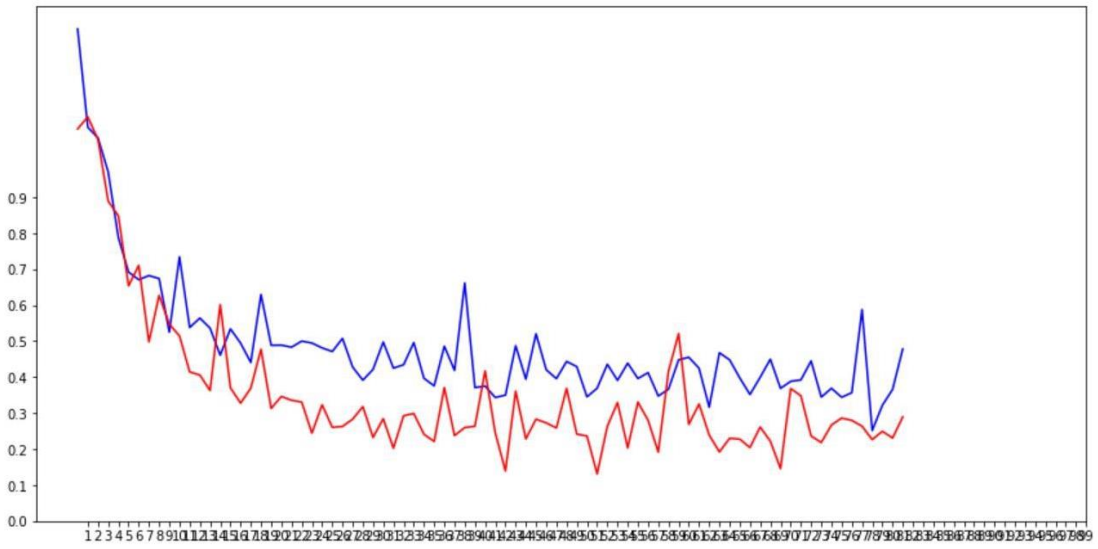
INTERPRETATION OF THE RESULTS

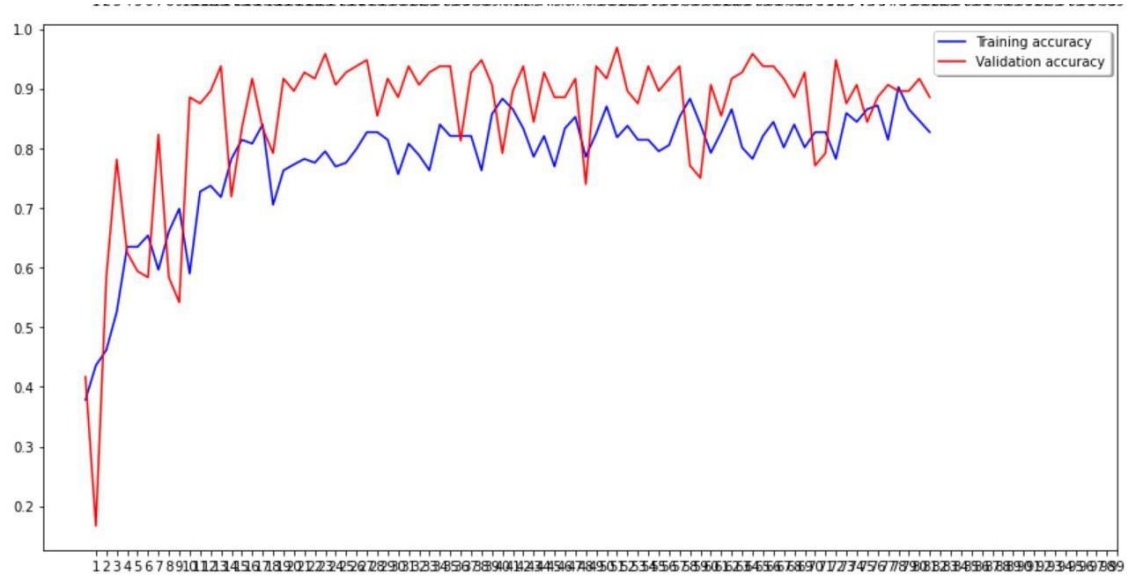
Plotting model accuracy and loss

```
#Virtualize Training
import matplotlib.pyplot as plt
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
ax1.plot(history.history['loss'], color='b', label="Training loss")
ax1.plot(history.history['val_loss'], color='r', label="validation loss")
ax1.set_xticks(np.arange(1, epoch, 1))
ax1.set_yticks(np.arange(0, 1, 0.1))

ax2.plot(history.history['accuracy'], color='b', label="Training accuracy")
ax2.plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
ax2.set_xticks(np.arange(1, epoch, 1))

legend = plt.legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```






```
#lets evaluate our model
model.evaluate(validation_generator)
```

```
4/4 [==============================] - 1s 133ms/step - loss: 0.3017 - accuracy: 0.9083
```

```
0]: [0.30165496468544006, 0.9083333611488342]
```

```
# Confusion Matrix and Classification Report
```

```
from sklearn.metrics import classification_report, confusion_matrix
class_labels = validation_generator.class_indices
class_labels = {v: k for k, v in class_labels.items()}


Y_pred = model.predict(validation_generator, nb_validation_samples)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(validation_generator.classes, y_pred))
print('Classification Report')
target_names = list(class_labels.values())
print(classification_report(validation_generator.classes, y_pred, target_names=target_names))
```

```
Confusion Matrix
[[31  0  9]
 [ 0 38  2]
 [ 0  0 40]]
Classification Report
              precision    recall  f1-score   support

       Jeans       1.00      0.78      0.87        40
       Saree       1.00      0.95      0.97        40
    Trousers       0.78      1.00      0.88        40

    accuracy                           0.91       120
   macro avg       0.93      0.91      0.91       120
weighted avg       0.93      0.91      0.91       120
```

CONCLUSION

Predicting the test images ¶

```
2]:  1 pred = model.predict(validation_generator)
     2
     3 y_classes = pred.argmax(axis=-1)
     4 print(y_classes)
     5 res = pd.DataFrame()
     6 res['ImageId'] = list(range(1,121))
     7 res['Predicted_Label'] = y_classes
     8 res['Actual_Label']=validation_generator.classes
     9 res.head(50)
    10 #res.to_csv("output.csv", index = False)
```

```
[2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 0 2 2 0 0 0 0 0 2 2 0 0
 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2]
```

	ImageId	Predicted_Label	Actual_Label
0	1	2	0
1	2	0	0
2	3	0	0
3	4	0	0
4	5	0	0
5	6	0	0
6	7	0	0
7	8	0	0
8	9	0	0
9	10	0	0
10	11	0	0
11	12	0	0
12	13	0	0
13	14	0	0
14	15	0	0
15	16	0	0
16	17	0	0
17	18	0	0
18	19	0	0
19	20	0	0
20	21	2	0
21	22	0	0
22	23	2	0
23	24	2	0
24	25	2	0
25	26	0	0
26	27	2	0
27	28	2	0
28	29	0	0
29	30	0	0
30	31	0	0
31	32	0	0
32	33	0	0
33	34	2	0
34	35	2	0
35	36	0	0
36	37	0	0
37	38	0	0
38	39	0	0
39	40	0	0
40	41	1	1
41	42	1	1
42	43	1	1
43	44	1	1
44	45	1	1
45	46	1	1
46	47	1	1
47	48	1	1

```python
#Testing our classifier


test_dire=[saree_dir_test,jean_dir_test,trouser_dir_test]

for test_dir in test_dire:
    for i in listdir(test_dir):
        print("Input Image is:",i)
        img= image.load_img('{}/{}'.format(test_dir,i))
        test_image = image.load_img('{}/{}'.format(test_dir,i),target_size=(128, 128))
        test_image = image.img_to_array(test_image)
        plt.imshow(img)
        plt.axis('off')
        plt.show()
        test_image = np.expand_dims(test_image, axis=0)
        result = saved_model.predict(test_image)
        print("Predicted Label is:",np.argmax(result, axis=1),"\n")
```

```
Input Image is: img _Sarees420.jpeg
```

```
Predicted Label is: [1]
```

```
Input Image is: img _Sarees421.jpeg
```

```
Predicted Label is: [1]

Input Image is: img _Sarees422.jpeg
```

```
Predicted Label is: [1]
```

Input Image is: img _Jeans331.jpeg

Predicted Label is: [0]

Input Image is: img _Jeans332.jpeg

Predicted Label is: [0]

Predicted Label is: [0]

Input Image is: img _Trousers47.jpeg

Predicted Label is: [2]

THANK YOU