FLIP ROBO

IMAGE SCRAPING AND CLASSIFICATION PROJECT

Submitted by:

Shubham Dewangan

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 Mr. Shubham Yadav for his constant guidance and support.

Some of the reference sources are as follows:

- Internet
- Coding Ninjas
- Medium.com
- Analytics Vidhya
- StackOverflow

TABLE OF CONTENTS

ACKNOWLEDGMENT i
INTRODUCTION 1
BUSINESS PROBLEM FRAMING 1
CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM 1
REVIEW OF LITERATURE 1
MOTIVATION FOR THE PROBLEM UNDERTAKEN 2
ANALYTICAL PROBLEM FRAMING 2
MATHEMATICAL/ ANALYTICAL MODELING OF THE PROBLEM 2
DATA SOURCES AND THEIR FORMATS 3
DATA PREPROCESSING DONE 3
DATA INPUTS- LOGIC- OUTPUT RELATIONSHIPS 3
HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED 4
MODEL/S DEVELOPMENT AND EVALUATION 5
IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS) 5
TESTING OF IDENTIFIED APPROACHES (ALGORITHMS) 5
RUN AND EVALUATE SELECTED MODELS 6
KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDE CONSIDERATION 8
VISUALIZATION 8
KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION 17
CONCLUSION 17
KEY FINDINGS AND CONCLUSIONS OF THE STUDY 17
LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE 17
LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK 17

INTRODUCTION

BUSINESS PROBLEM FRAMING

Classification is a systematic arrangement in groups and categories based on its features. Image classification came into existence for decreasing the gap between the computer vision and human vision by training the computer with the data. The image classification is achieved by differentiating the image into the prescribed category based on the content of the vision. We use Deep Learning to accomplish the task of image Classification.

In this project we have to classify whether is image is of a jeans, a trouser or a saree. We could we such model either for automatic segmentation of items using IOT and techniques for Industry 4.0

CONCEPTUAL BACKGROUND OF THE DOMAIN PROBLEM

Deep learning (DL) is a sub field to the machine learning, capable of learning through its own method of computing. A deep learning model is introduced to persistently break down information with a homogeneous structure like how a human would make determinations. To accomplish this, deep learning utilizes a layered structure of several algorithms expressed as an artificial neural system (ANN). The architecture of an ANN is simulated with the help of the biological neural network of the human brain. This makes the deep learning most capable than the standard machine learning models.

In deep neural networks every node decides its basic inputs by itself and sends it to the next tier on behalf of the previous tier. We train the data in the networks by giving an input image and conveying the network about its output. Neural networks are expressed in terms of number of layers involved for producing the inputs and outputs and the depth of the neural network.

REVIEW OF LITERATURE

Recently, image classification is growing and becoming a trend among technology developers especially with the growth of data in different parts of industry such as ecommerce, automotive, healthcare, and gaming. The most obvious example of this technology is applied to Facebook. Facebook now can detect up to 98% accuracy in order to identify your face with only a few tagged images and classified it into your Facebook's album. The technology itself almost beats the ability of human in image classification or recognition.

One of the dominant approaches for this technology is deep learning. Deep learning falls under the category of Artificial Intelligence where it can act or think like a human. Normally, the system itself will be set with hundreds or maybe thousands of input data in order to make the 'training' session to be more efficient and fast. It starts by giving some sort of 'training' with all the input data (Faux & Luthon, 2012).

Image classification has become a major challenge in machine vision and has a long history with it. The challenge includes a broad intra-class range of images caused by colour, size, environmental conditions and shape. It is required big data of labelled training images and to prepare this big data, it consumes a lot of time and cost as for the training purpose only In this project we will be using a transfer learning state of the art model for getting the best results that is VGG 16. Which was a winner in the ImageNet Competition?

MOTIVATION FOR THE PROBLEM UNDERTAKEN

The problem was undertaken in order to classify images efficiently using best deep learning algorithms and data augmentation techniques.

ANALYTICAL PROBLEM FRAMING

MATHEMATICAL/ ANALYTICAL MODELING OF THE PROBLEM

We have scraped images of all 3 classes that are men's jeans, men's trouser and sarees for women from the internet and we have built our model by training it on this data. We have used transfer learning to get state of the art results for our model.

DATA SOURCES AND THEIR FORMATS

DATA COLLECTION PHASE:

Data has been scraped from **amazon.in** using a python script which with selenium. All the data is in the **.jpg image format**.

We have over 376 images per class.

MODEL BUILDING PHASE:

After the data collection and preparation is done, you need to build an image classification model that will classify between these 3 categories mentioned above. You can play around with optimizers and learning rates for improving your model's performance.

DATA PREPROCESSING DONE

- We have first labelled the image data.
- We have manually removed irrelevant images that were downloading via the script.
- We have removed the duplicate images using a script.
- We have also performed multiple data augmentation techniques in order to train the model better or multiple angles and orientation of the same image.

DATA INPUTS- LOGIC- OUTPUT RELATIONSHIPS

We have given raw yet cleaned images to the machine learning model and the output produced is a label of the image on which the model is predicted

HARDWARE AND SOFTWARE REQUIREMENTS AND TOOLS USED

HARDWARE:

Device specifications Copy

Device name	LAPTOP-N0SDNE9F
Processor	AMD Ryzen 5 4600H with Radeon Graphics 3.00 GHz
Installed RAM	8.00 GB (7.37 GB usable)
Device ID	FD942C3B-FA5D-4994-AD8C-62A017AA85FA
Product ID	00331-10000-00001-AA287
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications Copy

Edition	Windows 11 Pro Insider Preview
Version	Dev
Installed on	16-09-2021
OS build	22458.1000
Serial number	PF20GFKB
Experience	Windows Feature Experience Pack 1000.22458.1000.0

Microsoft Services Agreement
Microsoft Software Licence Terms

SOFTWARE:

- Jupyter Notebook (Anaconda 3) – Python 3.8.5

LIBRARIES:

- Numpy
- Pandas
- Tensorflow
- MAtplotlib
- Keras.savemodel

MODEL/S DEVELOPMENT AND EVALUATION

IDENTIFICATION OF POSSIBLE PROBLEM-SOLVING APPROACHES (METHODS)

- We will be first scraping the data from amazon and then cleaning the data to be further used for training the model.
- Then splitting the data into train and test set to model evaluation.
- We would perform multiple data augmentation techniques on the images for better generalization of our model.
- We could check and identity an optimal batch size and number of epoch to Train the model using trial and error method also keeping the epoch loss for both training set and validation in mind.

TESTING OF IDENTIFIED APPROACHES (ALGORITHMS)

In [3]:
```
# Let's try to print some of the scrapped images from each category
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

train_jeans=r'Clothes/Train/Jeans_Images'
train_saree=r'Clothes/Train/Sarees_Images'
train_trouser=r'Clothes/Train/Trousers_Images'


Cloth_train=[train_jeans, train_saree, train_trouser]
for dirs in Cloth_train:
    k=listdir(dirs)
    for i in k[:3]:
        img=mpimg.imread('{}/{}'.format(dirs,i))
        plt.imshow(img)
        plt.axis('off')
        plt.show()
```

RUN AND EVALUATE SELECTED MODELS

After collecting the data we next do training of the data. For that firstly, we created a main folder called “Clothes” in current working directory inside which I further created two folders called “Train” and “Test”.

In Train folder we have kept 250 images from each clothing category and remaining 126 images we have kept in Test folder for each category. Hence, we got 750 images for training and 378 for testing.

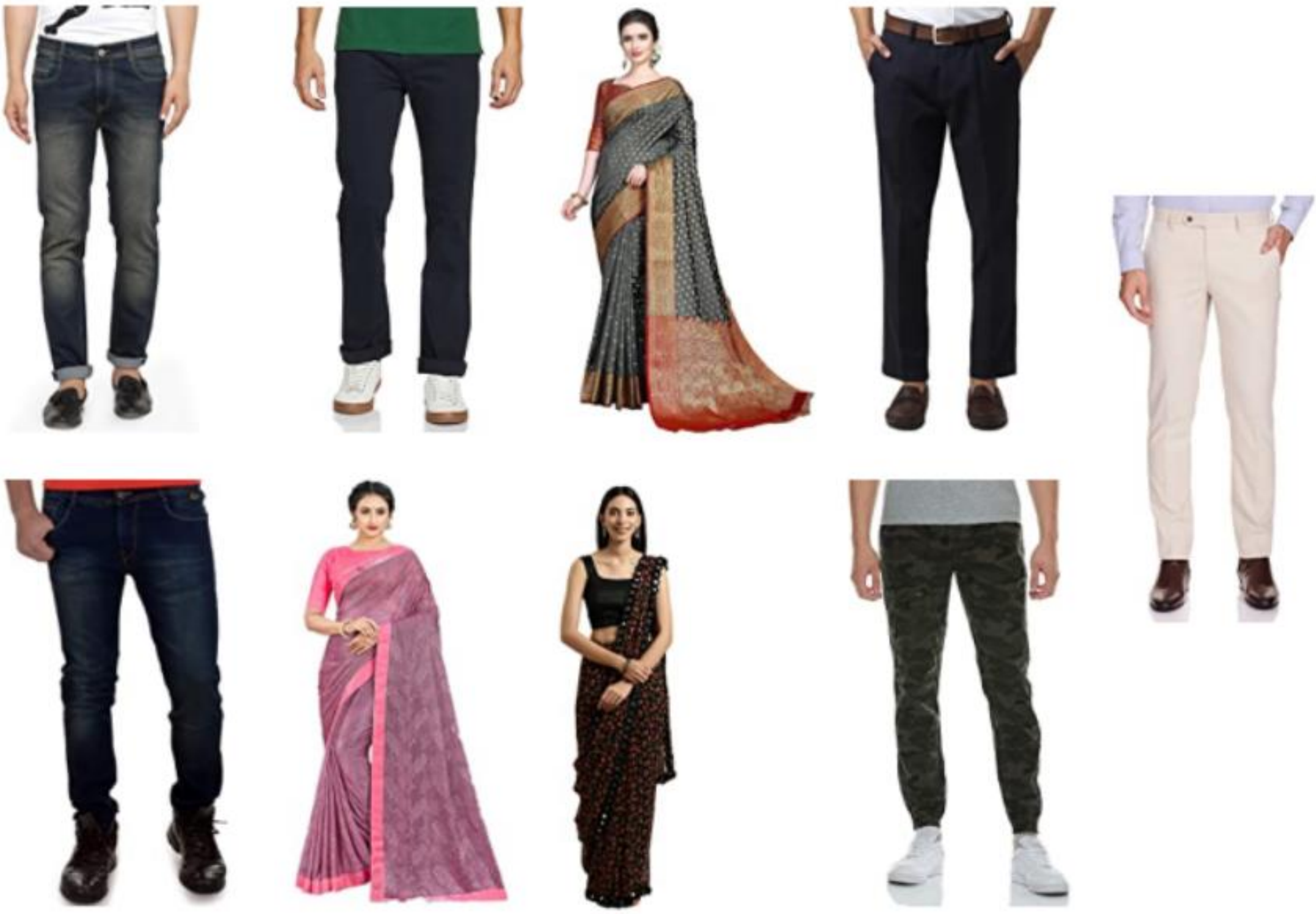
```
In [1]: import os
        from os import listdir
```

```
In [2]: train_data=r'Clothes/Train'
        test_data=r'Clothes/Test'
```

```
In [3]: # Let's try to print some of the scrapped images from each category
        import matplotlib.image as mpimg
        import matplotlib.pyplot as plt

        train_jeans=r'Clothes/Train/Jeans_Images'
        train_saree=r'Clothes/Train/Sarees_Images'
        train_trouser=r'Clothes/Train/Trousers_Images'


        Cloth_train=[train_jeans, train_saree, train_trouser]
        for dirs in Cloth_train:
            k=listdir(dirs)
            for i in k[:3]:
                img=mpimg.imread('{}/{}'.format(dirs,i))
                plt.imshow(img)
                plt.axis('off')
                plt.show()
```



```
In [6]: print("Count of Training Images")
        print("No.of Images of Sarees in train dataset -> ",len(os.listdir(r'Clothes/Train/Sarees_Images')))
        print("No.of Images of Jeans in train dataset -> ",len(os.listdir(r'Clothes/Train/Jeans_Images')))
        print("No.of Images of Trousers in train dataset ->",len(os.listdir(r'Clothes/Train/Trousers_Images')))
        "\n"

        print("Count of Test Images")
        print("No.of Images of Sarees in test dataset-> ",len(os.listdir(r'Clothes/Test/Sarees_Images')))
        print("No.of Images of Jeans in test dataset ->",len(os.listdir(r'Clothes/Test/Jeans_Images')))
        print("No.of Images of Trousers in test dataset-> ",len(os.listdir(r'Clothes/Test/Trousers_Images')))
```

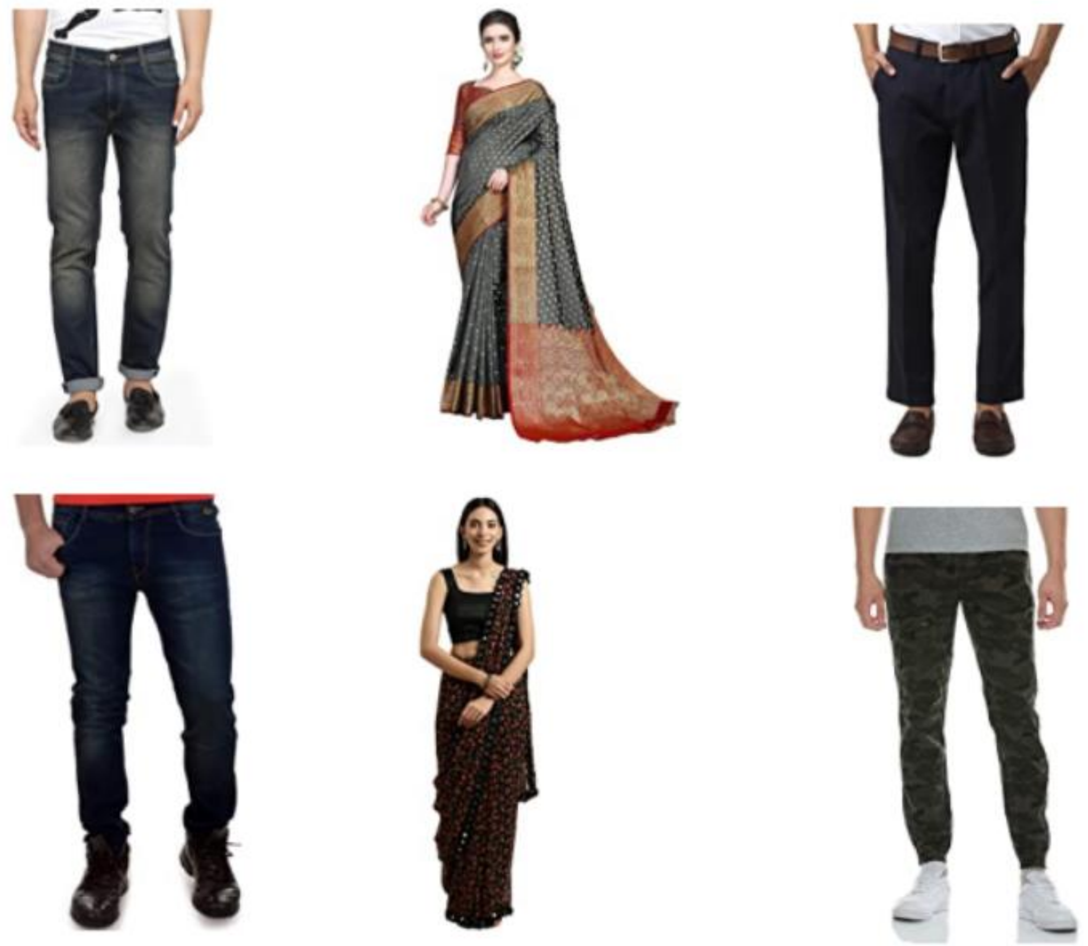
```
Count of Training Images
No.of Images of Sarees in train dataset ->  250
No.of Images of Jeans in train dataset ->  250
No.of Images of Trousers in train dataset -> 250
Count of Test Images
No.of Images of Sarees in test dataset->  126
No.of Images of Jeans in test dataset -> 126
No.of Images of Trousers in test dataset->  126
```

Next, we defined dimensions of images and other parameters also. Then, for data augmentation we defined training and testing set.

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

We have considered the model accuracy and loss for both the training and validation data.

VISUALIZATION



FINAL MODEL

```
In [9]: # Data Augmentation on Training Images

        Train_datagen=ImageDataGenerator(rescale=1./255,
                                          zoom_range=0.2,
                                          rotation_range=30,
                                          horizontal_flip=True)
        Training_set=Train_datagen.flow_from_directory(train_data,
                                                        target_size=(img_width,img_height),
                                                        batch_size=batch_size,
                                                        class_mode='categorical')

        # Test Data Generator
        Test_datagen=ImageDataGenerator(rescale=1./255)
        Test_set=Test_datagen.flow_from_directory(test_data,
                                                   target_size=(img_width,img_height),
                                                   batch_size=batch_size,
                                                   class_mode='categorical')

        Found 750 images belonging to 3 classes.
        Found 378 images belonging to 3 classes.
```

In [10]:
```
# Creating the model
model=Sequential()

# First convolution layer
model.add(Conv2D(32,(3,3),input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Second convolution layer
model.add(Conv2D(32,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Third convolution layer
model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# Fourth convolution layer
model.add(Conv2D(64,(3,3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))


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='adam',metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 126, 126, 32)      896       
_________________________________________________________________
activation (Activation)      (None, 126, 126, 32)      0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 63, 63, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        9248      
_________________________________________________________________
activation_1 (Activation)    (None, 61, 61, 32)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 30, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 64)        18496     
_________________________________________________________________
activation_2 (Activation)    (None, 28, 28, 64)        0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 64)        0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        36928     
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 64)        0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0         
_________________________________________________________________
dense (Dense)                (None, 128)               295040    
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0         
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 387       
_________________________________________________________________
activation_5 (Activation)    (None, 3)                 0         
=================================================================
Total params: 360,995
Trainable params: 360,995
Non-trainable params: 0
_________________________________________________________________
None
```

In [12]:
```
# Fitting the Training Data
history = model.fit(
    Training_set,
    epochs=epoch,
    validation_data=Test_set,
    validation_steps=test_samples//batch_size,
    steps_per_epoch=train_samples//batch_size,
    callbacks=[ES,MC])
```

```
Epoch 1/100
20/20 [==============================] - 7s 318ms/step - loss: 1.1634 - accuracy: 0.3250 - val_loss: 1.0863 - val_accuracy: 0.5
648

Epoch 00001: val_accuracy improved from -inf to 0.56481, saving model to best.h5
Epoch 2/100
20/20 [==============================] - 5s 243ms/step - loss: 1.0150 - accuracy: 0.4658 - val_loss: 0.9229 - val_accuracy: 0.6
667

Epoch 00002: val_accuracy improved from 0.56481 to 0.66667, saving model to best.h5
Epoch 3/100
20/20 [==============================] - 4s 205ms/step - loss: 0.7390 - accuracy: 0.6208 - val_loss: 0.7042 - val_accuracy: 0.7
685

Epoch 00003: val_accuracy improved from 0.66667 to 0.76852, saving model to best.h5
Epoch 4/100
20/20 [==============================] - 4s 183ms/step - loss: 0.6753 - accuracy: 0.5897 - val_loss: 0.6856 - val_accuracy: 0.6
759

Epoch 00004: val_accuracy did not improve from 0.76852
Epoch 5/100
20/20 [==============================] - 4s 175ms/step - loss: 0.6104 - accuracy: 0.6292 - val_loss: 0.5043 - val_accuracy: 0.7
037

Epoch 00005: val_accuracy did not improve from 0.76852
Epoch 6/100
20/20 [==============================] - 3s 164ms/step - loss: 0.4883 - accuracy: 0.7458 - val_loss: 0.5594 - val_accuracy: 0.6
852

Epoch 00006: val_accuracy did not improve from 0.76852
Epoch 7/100
20/20 [==============================] - 3s 159ms/step - loss: 0.5876 - accuracy: 0.6958 - val_loss: 0.5740 - val_accuracy: 0.7
870

Epoch 00007: val_accuracy improved from 0.76852 to 0.78704, saving model to best.h5
Epoch 8/100
20/20 [==============================] - 3s 153ms/step - loss: 0.4887 - accuracy: 0.7458 - val_loss: 0.4281 - val_accuracy: 0.8
148

Epoch 00008: val_accuracy improved from 0.78704 to 0.81481, saving model to best.h5
Epoch 9/100
20/20 [==============================] - 3s 150ms/step - loss: 0.5279 - accuracy: 0.7094 - val_loss: 0.5301 - val_accuracy: 0.7
407
```

```
Epoch 00009: val_accuracy did not improve from 0.81481
Epoch 10/100
20/20 [==============================] - 3s 151ms/step - loss: 0.5061 - accuracy: 0.7333 - val_loss: 0.5017 - val_accuracy: 0.7
778

Epoch 00010: val_accuracy did not improve from 0.81481
Epoch 11/100
20/20 [==============================] - 3s 150ms/step - loss: 0.4637 - accuracy: 0.7564 - val_loss: 0.4859 - val_accuracy: 0.8
148

Epoch 00011: val_accuracy did not improve from 0.81481
Epoch 12/100
20/20 [==============================] - 3s 159ms/step - loss: 0.5117 - accuracy: 0.7292 - val_loss: 0.5018 - val_accuracy: 0.7
593

Epoch 00012: val_accuracy did not improve from 0.81481
Epoch 13/100
20/20 [==============================] - 3s 151ms/step - loss: 0.4261 - accuracy: 0.7708 - val_loss: 0.4354 - val_accuracy: 0.7
778

Epoch 00013: val_accuracy did not improve from 0.81481
Epoch 14/100
20/20 [==============================] - 3s 145ms/step - loss: 0.4713 - accuracy: 0.7393 - val_loss: 0.4124 - val_accuracy: 0.7
778

Epoch 00014: val_accuracy did not improve from 0.81481
Epoch 15/100
20/20 [==============================] - 3s 148ms/step - loss: 0.4132 - accuracy: 0.7708 - val_loss: 0.4210 - val_accuracy: 0.8
241

Epoch 00015: val_accuracy improved from 0.81481 to 0.82407, saving model to best.h5
Epoch 16/100
20/20 [==============================] - 3s 147ms/step - loss: 0.4374 - accuracy: 0.7750 - val_loss: 0.3824 - val_accuracy: 0.8
241

Epoch 00016: val_accuracy did not improve from 0.82407
Epoch 17/100
20/20 [==============================] - 3s 149ms/step - loss: 0.3998 - accuracy: 0.8248 - val_loss: 0.3882 - val_accuracy: 0.8
333
```

```
Epoch 00017: val_accuracy improved from 0.82407 to 0.83333, saving model to best.h5
Epoch 18/100
20/20 [==============================] - 3s 148ms/step - loss: 0.4043 - accuracy: 0.8167 - val_loss: 0.4286 - val_accuracy: 0.8
056

Epoch 00018: val_accuracy did not improve from 0.83333
Epoch 19/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3823 - accuracy: 0.8542 - val_loss: 0.4104 - val_accuracy: 0.8
056

Epoch 00019: val_accuracy did not improve from 0.83333
Epoch 20/100
20/20 [==============================] - 3s 151ms/step - loss: 0.4160 - accuracy: 0.8120 - val_loss: 0.3653 - val_accuracy: 0.8
333

Epoch 00020: val_accuracy did not improve from 0.83333
Epoch 21/100
20/20 [==============================] - 3s 158ms/step - loss: 0.4151 - accuracy: 0.8042 - val_loss: 0.3617 - val_accuracy: 0.7
963

Epoch 00021: val_accuracy did not improve from 0.83333
Epoch 22/100
20/20 [==============================] - 3s 149ms/step - loss: 0.4284 - accuracy: 0.8083 - val_loss: 0.3414 - val_accuracy: 0.8
426

Epoch 00022: val_accuracy improved from 0.83333 to 0.84259, saving model to best.h5
Epoch 23/100
20/20 [==============================] - 3s 150ms/step - loss: 0.4224 - accuracy: 0.7917 - val_loss: 0.4326 - val_accuracy: 0.7
685

Epoch 00023: val_accuracy did not improve from 0.84259
Epoch 24/100
20/20 [==============================] - 3s 144ms/step - loss: 0.4685 - accuracy: 0.7906 - val_loss: 0.4298 - val_accuracy: 0.8
241

Epoch 00024: val_accuracy did not improve from 0.84259
Epoch 25/100
20/20 [==============================] - 3s 144ms/step - loss: 0.3173 - accuracy: 0.8761 - val_loss: 0.3369 - val_accuracy: 0.8
519

Epoch 00025: val_accuracy improved from 0.84259 to 0.85185, saving model to best.h5
Epoch 26/100
20/20 [==============================] - 3s 147ms/step - loss: 0.4260 - accuracy: 0.7750 - val_loss: 0.3436 - val_accuracy: 0.8
519

Epoch 00026: val_accuracy did not improve from 0.85185
Epoch 27/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3003 - accuracy: 0.8750 - val_loss: 0.3424 - val_accuracy: 0.8
241

Epoch 00027: val_accuracy did not improve from 0.85185
Epoch 28/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3994 - accuracy: 0.8034 - val_loss: 0.3203 - val_accuracy: 0.8
611
```

```
Epoch 00028: val_accuracy improved from 0.85185 to 0.86111, saving model to best.h5
Epoch 29/100
20/20 [==============================] - 3s 149ms/step - loss: 0.3684 - accuracy: 0.8375 - val_loss: 0.4096 - val_accuracy: 0.8
611

Epoch 00029: val_accuracy did not improve from 0.86111
Epoch 30/100
20/20 [==============================] - 3s 144ms/step - loss: 0.4239 - accuracy: 0.8120 - val_loss: 0.3562 - val_accuracy: 0.7
963

Epoch 00030: val_accuracy did not improve from 0.86111
Epoch 31/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3416 - accuracy: 0.8462 - val_loss: 0.3834 - val_accuracy: 0.7
778

Epoch 00031: val_accuracy did not improve from 0.86111
Epoch 32/100
20/20 [==============================] - 3s 145ms/step - loss: 0.4526 - accuracy: 0.8205 - val_loss: 0.4387 - val_accuracy: 0.7
593

Epoch 00032: val_accuracy did not improve from 0.86111
Epoch 33/100
20/20 [==============================] - 3s 147ms/step - loss: 0.4118 - accuracy: 0.8083 - val_loss: 0.4212 - val_accuracy: 0.7
778

Epoch 00033: val_accuracy did not improve from 0.86111
Epoch 34/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3174 - accuracy: 0.8500 - val_loss: 0.3387 - val_accuracy: 0.8
148

Epoch 00034: val_accuracy did not improve from 0.86111
Epoch 35/100
20/20 [==============================] - 3s 155ms/step - loss: 0.3790 - accuracy: 0.8292 - val_loss: 0.3084 - val_accuracy: 0.8
426

Epoch 00035: val_accuracy did not improve from 0.86111
Epoch 36/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3438 - accuracy: 0.8417 - val_loss: 0.3108 - val_accuracy: 0.8
796
```

```
Epoch 00036: val_accuracy improved from 0.86111 to 0.87963, saving model to best.h5
Epoch 37/100
20/20 [==============================] - 3s 151ms/step - loss: 0.2976 - accuracy: 0.8500 - val_loss: 0.3775 - val_accuracy: 0.7
963

Epoch 00037: val_accuracy did not improve from 0.87963
Epoch 38/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3045 - accuracy: 0.8462 - val_loss: 0.3372 - val_accuracy: 0.8
519

Epoch 00038: val_accuracy did not improve from 0.87963
Epoch 39/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3524 - accuracy: 0.8333 - val_loss: 0.2886 - val_accuracy: 0.8
796

Epoch 00039: val_accuracy did not improve from 0.87963
Epoch 40/100
20/20 [==============================] - 3s 149ms/step - loss: 0.3026 - accuracy: 0.8419 - val_loss: 0.3037 - val_accuracy: 0.8
333

Epoch 00040: val_accuracy did not improve from 0.87963
Epoch 41/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3094 - accuracy: 0.8583 - val_loss: 0.3183 - val_accuracy: 0.8
426

Epoch 00041: val_accuracy did not improve from 0.87963
Epoch 42/100
20/20 [==============================] - 3s 144ms/step - loss: 0.2887 - accuracy: 0.8547 - val_loss: 0.3349 - val_accuracy: 0.8
426

Epoch 00042: val_accuracy did not improve from 0.87963
Epoch 43/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2708 - accuracy: 0.8917 - val_loss: 0.2985 - val_accuracy: 0.8
519

Epoch 00043: val_accuracy did not improve from 0.87963
Epoch 44/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2939 - accuracy: 0.8667 - val_loss: 0.3355 - val_accuracy: 0.8
519

Epoch 00044: val_accuracy did not improve from 0.87963
Epoch 45/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3611 - accuracy: 0.8417 - val_loss: 0.3608 - val_accuracy: 0.7
963

Epoch 00045: val_accuracy did not improve from 0.87963
Epoch 46/100
20/20 [==============================] - 3s 144ms/step - loss: 0.3596 - accuracy: 0.8162 - val_loss: 0.2978 - val_accuracy: 0.8
889

Epoch 00046: val_accuracy improved from 0.87963 to 0.88889, saving model to best.h5
Epoch 47/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3126 - accuracy: 0.8458 - val_loss: 0.2511 - val_accuracy: 0.8
796


Epoch 00047: val_accuracy did not improve from 0.88889
Epoch 48/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3175 - accuracy: 0.8625 - val_loss: 0.3382 - val_accuracy: 0.8
333

Epoch 00048: val_accuracy did not improve from 0.88889
Epoch 49/100
20/20 [==============================] - 3s 144ms/step - loss: 0.3399 - accuracy: 0.8462 - val_loss: 0.3598 - val_accuracy: 0.8
333

Epoch 00049: val_accuracy did not improve from 0.88889
Epoch 50/100
20/20 [==============================] - 3s 149ms/step - loss: 0.2484 - accuracy: 0.8958 - val_loss: 0.3006 - val_accuracy: 0.8
611

Epoch 00050: val_accuracy did not improve from 0.88889
Epoch 51/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2696 - accuracy: 0.8958 - val_loss: 0.3114 - val_accuracy: 0.8
241

Epoch 00051: val_accuracy did not improve from 0.88889
Epoch 52/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3177 - accuracy: 0.8333 - val_loss: 0.3214 - val_accuracy: 0.8
426

Epoch 00052: val_accuracy did not improve from 0.88889
Epoch 53/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3145 - accuracy: 0.8542 - val_loss: 0.2466 - val_accuracy: 0.9
167

Epoch 00053: val_accuracy improved from 0.88889 to 0.91667, saving model to best.h5
Epoch 54/100
20/20 [==============================] - 3s 146ms/step - loss: 0.2733 - accuracy: 0.8625 - val_loss: 0.3438 - val_accuracy: 0.8
519

Epoch 00054: val_accuracy did not improve from 0.91667
Epoch 55/100
20/20 [==============================] - 3s 144ms/step - loss: 0.2685 - accuracy: 0.8675 - val_loss: 0.2987 - val_accuracy: 0.8
241
```

```
Epoch 00055: val_accuracy did not improve from 0.91667
Epoch 56/100
20/20 [==============================] - 3s 161ms/step - loss: 0.2519 - accuracy: 0.8750 - val_loss: 0.3902 - val_accuracy: 0.8
426

Epoch 00056: val_accuracy did not improve from 0.91667
Epoch 57/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3179 - accuracy: 0.8583 - val_loss: 0.3076 - val_accuracy: 0.8
241

Epoch 00057: val_accuracy did not improve from 0.91667
Epoch 58/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2855 - accuracy: 0.8625 - val_loss: 0.3052 - val_accuracy: 0.8
704

Epoch 00058: val_accuracy did not improve from 0.91667
Epoch 59/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3446 - accuracy: 0.8375 - val_loss: 0.4225 - val_accuracy: 0.8
333

Epoch 00059: val_accuracy did not improve from 0.91667
Epoch 60/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3632 - accuracy: 0.8292 - val_loss: 0.3932 - val_accuracy: 0.7
963

Epoch 00060: val_accuracy did not improve from 0.91667
Epoch 61/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3457 - accuracy: 0.8504 - val_loss: 0.3091 - val_accuracy: 0.8
333

Epoch 00061: val_accuracy did not improve from 0.91667
Epoch 62/100
20/20 [==============================] - 3s 145ms/step - loss: 0.2950 - accuracy: 0.8590 - val_loss: 0.3057 - val_accuracy: 0.8
889

Epoch 00062: val_accuracy did not improve from 0.91667
Epoch 63/100
20/20 [==============================] - 3s 150ms/step - loss: 0.2788 - accuracy: 0.8792 - val_loss: 0.3805 - val_accuracy: 0.8
333

Epoch 00063: val_accuracy did not improve from 0.91667
Epoch 64/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3322 - accuracy: 0.8333 - val_loss: 0.4158 - val_accuracy: 0.8
333

Epoch 00064: val_accuracy did not improve from 0.91667
Epoch 65/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3064 - accuracy: 0.8458 - val_loss: 0.3225 - val_accuracy: 0.8
611

Epoch 00065: val_accuracy did not improve from 0.91667
Epoch 66/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3727 - accuracy: 0.8250 - val_loss: 0.3576 - val_accuracy: 0.8
241


Epoch 00066: val_accuracy did not improve from 0.91667
Epoch 67/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3222 - accuracy: 0.8417 - val_loss: 0.2558 - val_accuracy: 0.8
889

Epoch 00067: val_accuracy did not improve from 0.91667
Epoch 68/100
20/20 [==============================] - 3s 147ms/step - loss: 0.3138 - accuracy: 0.8542 - val_loss: 0.3412 - val_accuracy: 0.8
426

Epoch 00068: val_accuracy did not improve from 0.91667
Epoch 69/100
20/20 [==============================] - 3s 158ms/step - loss: 0.2290 - accuracy: 0.8708 - val_loss: 0.3201 - val_accuracy: 0.8
611

Epoch 00069: val_accuracy did not improve from 0.91667
Epoch 70/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3611 - accuracy: 0.8376 - val_loss: 0.2918 - val_accuracy: 0.8
611

Epoch 00070: val_accuracy did not improve from 0.91667
Epoch 71/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2440 - accuracy: 0.8846 - val_loss: 0.2272 - val_accuracy: 0.9
167

Epoch 00071: val_accuracy did not improve from 0.91667
Epoch 72/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2630 - accuracy: 0.8761 - val_loss: 0.2778 - val_accuracy: 0.8
426

Epoch 00072: val_accuracy did not improve from 0.91667
Epoch 73/100
20/20 [==============================] - 3s 166ms/step - loss: 0.2433 - accuracy: 0.9000 - val_loss: 0.3104 - val_accuracy: 0.8
889

Epoch 00073: val_accuracy did not improve from 0.91667
Epoch 74/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2900 - accuracy: 0.8718 - val_loss: 0.4042 - val_accuracy: 0.8
148
```

```
Epoch 00074: val_accuracy did not improve from 0.91667
Epoch 75/100
20/20 [==============================] - 3s 149ms/step - loss: 0.2632 - accuracy: 0.8625 - val_loss: 0.2708 - val_accuracy: 0.8
796

Epoch 00075: val_accuracy did not improve from 0.91667
Epoch 76/100
20/20 [==============================] - 3s 153ms/step - loss: 0.2182 - accuracy: 0.8974 - val_loss: 0.3251 - val_accuracy: 0.8
611

Epoch 00076: val_accuracy did not improve from 0.91667
Epoch 77/100
20/20 [==============================] - 3s 149ms/step - loss: 0.3346 - accuracy: 0.8542 - val_loss: 0.3020 - val_accuracy: 0.8
981

Epoch 00077: val_accuracy did not improve from 0.91667
Epoch 78/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2568 - accuracy: 0.8750 - val_loss: 0.2957 - val_accuracy: 0.8
796

Epoch 00078: val_accuracy did not improve from 0.91667
Epoch 79/100
20/20 [==============================] - 3s 152ms/step - loss: 0.2655 - accuracy: 0.8708 - val_loss: 0.2462 - val_accuracy: 0.8
889

Epoch 00079: val_accuracy did not improve from 0.91667
Epoch 80/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3897 - accuracy: 0.8419 - val_loss: 0.3334 - val_accuracy: 0.8
796

Epoch 00080: val_accuracy did not improve from 0.91667
Epoch 81/100

20/20 [==============================] - 3s 144ms/step - loss: 0.4057 - accuracy: 0.8120 - val_loss: 0.4114 - val_accuracy: 0.7
870

Epoch 00081: val_accuracy did not improve from 0.91667
Epoch 82/100
20/20 [==============================] - 3s 148ms/step - loss: 0.3201 - accuracy: 0.8417 - val_loss: 0.3249 - val_accuracy: 0.8
611

Epoch 00082: val_accuracy did not improve from 0.91667
Epoch 83/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2748 - accuracy: 0.8875 - val_loss: 0.2720 - val_accuracy: 0.8
981

Epoch 00083: val_accuracy did not improve from 0.91667
Epoch 84/100
20/20 [==============================] - 3s 149ms/step - loss: 0.2395 - accuracy: 0.8958 - val_loss: 0.2477 - val_accuracy: 0.8
796

Epoch 00084: val_accuracy did not improve from 0.91667
Epoch 85/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2355 - accuracy: 0.8833 - val_loss: 0.1624 - val_accuracy: 0.9
259


Epoch 00085: val_accuracy improved from 0.91667 to 0.92593, saving model to best.h5
Epoch 86/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2665 - accuracy: 0.9000 - val_loss: 0.3178 - val_accuracy: 0.8
611

Epoch 00086: val_accuracy did not improve from 0.92593
Epoch 87/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2811 - accuracy: 0.8750 - val_loss: 0.2993 - val_accuracy: 0.8
426

Epoch 00087: val_accuracy did not improve from 0.92593
Epoch 88/100
20/20 [==============================] - 3s 150ms/step - loss: 0.3478 - accuracy: 0.8292 - val_loss: 0.2998 - val_accuracy: 0.8
519

Epoch 00088: val_accuracy did not improve from 0.92593
Epoch 89/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2586 - accuracy: 0.8958 - val_loss: 0.2915 - val_accuracy: 0.8
519

Epoch 00089: val_accuracy did not improve from 0.92593
Epoch 90/100
20/20 [==============================] - 3s 144ms/step - loss: 0.2673 - accuracy: 0.9060 - val_loss: 0.2261 - val_accuracy: 0.8
704

Epoch 00090: val_accuracy did not improve from 0.92593
Epoch 91/100
20/20 [==============================] - 3s 146ms/step - loss: 0.3361 - accuracy: 0.8500 - val_loss: 0.3601 - val_accuracy: 0.8
333

Epoch 00091: val_accuracy did not improve from 0.92593
Epoch 92/100
20/20 [==============================] - 3s 147ms/step - loss: 0.2981 - accuracy: 0.8708 - val_loss: 0.3063 - val_accuracy: 0.8
611

Epoch 00092: val_accuracy did not improve from 0.92593
Epoch 93/100
20/20 [==============================] - 3s 150ms/step - loss: 0.2810 - accuracy: 0.8632 - val_loss: 0.3251 - val_accuracy: 0.8
333
```

```
Epoch 00093: val_accuracy did not improve from 0.92593
Epoch 94/100
20/20 [==============================] - 3s 153ms/step - loss: 0.2562 - accuracy: 0.8625 - val_loss: 0.3169 - val_accuracy: 0.8
796

Epoch 00094: val_accuracy did not improve from 0.92593
Epoch 95/100
20/20 [==============================] - 3s 144ms/step - loss: 0.2560 - accuracy: 0.8974 - val_loss: 0.2159 - val_accuracy: 0.8
889

Epoch 00095: val_accuracy did not improve from 0.92593
Epoch 96/100
20/20 [==============================] - 3s 145ms/step - loss: 0.3654 - accuracy: 0.8333 - val_loss: 0.2766 - val_accuracy: 0.8
796

Epoch 00096: val_accuracy did not improve from 0.92593
Epoch 97/100
20/20 [==============================] - 3s 146ms/step - loss: 0.2460 - accuracy: 0.8750 - val_loss: 0.2248 - val_accuracy: 0.8
519

Epoch 00097: val_accuracy did not improve from 0.92593
Epoch 98/100
20/20 [==============================] - 3s 145ms/step - loss: 0.2771 - accuracy: 0.8675 - val_loss: 0.2175 - val_accuracy: 0.9
167

Epoch 00098: val_accuracy did not improve from 0.92593
Epoch 99/100
20/20 [==============================] - 3s 148ms/step - loss: 0.2586 - accuracy: 0.8718 - val_loss: 0.3953 - val_accuracy: 0.7
778

Epoch 00099: val_accuracy did not improve from 0.92593
Epoch 100/100
20/20 [==============================] - 3s 144ms/step - loss: 0.2832 - accuracy: 0.8718 - val_loss: 0.3408 - val_accuracy: 0.8
333

Epoch 00100: val_accuracy did not improve from 0.92593
```

In [14]:

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

Out[14]:

	loss	accuracy	val_loss	val_accuracy
0	1.163439	0.325000	1.086341	0.564815
1	1.015001	0.465812	0.922880	0.666667
2	0.738990	0.620833	0.704198	0.768519
3	0.675312	0.589744	0.685559	0.675926
4	0.610409	0.629167	0.504257	0.703704
...	...	...	...	...
95	0.365366	0.833333	0.276568	0.879630
96	0.246001	0.875000	0.224810	0.851852
97	0.277133	0.867521	0.217455	0.916667
98	0.258594	0.871795	0.395345	0.777778
99	0.283184	0.871795	0.340813	0.833333

100 rows × 4 columns

KEY METRICS FOR SUCCESS IN SOLVING PROBLEM UNDER CONSIDERATION

- When it comes to the evaluation of a data science model's performance, sometimes accuracy may not be the best indicator.
- Some problems that we are solving in real life might have a very imbalanced class and using accuracy might not give us enough confidence to understand the algorithm's performance.
- In the Rating Prediction problem that we are trying to solve, the data is balanced. So accuracy score nearly tells the right predictions. So the problem of overfitting in this problem is nearly not to occur. So here, we are using an accuracy score to find a better model.

CONCLUSION

KEY FINDINGS AND CONCLUSIONS OF THE STUDY

Our Model was able to classify the three clothing items distinctly. Since in all three categories there were some extra/unnecessary items other than the main items hence, it could have been removed and we could have got better result. Moreover, training data could have been increased.

LEARNING OUTCOMES OF THE STUDY IN RESPECT OF DATA SCIENCE

The larger the data the better the model could predict. Multi class predictions are somewhat relatively harder to train in comparison to a Binary class prediction. Data Augmentation is necessary where we have small datasets of images.

LIMITATIONS OF THIS WORK AND SCOPE FOR FUTURE WORK

One could always provide more data for training the model in order to get better results. We could use to model for apparel segmentation at Supermarkets and shopping stores.