IMAGE CLASSIFICATION PROJECT

SUBMITTED BY:

NAINCY JOSHI

Problem statement

E-commerce are the biggest platform of data collection . Collecting and sorting the data from the biggest E-commerce site is a challenge. Product images provide a better first impression. According to a survey, more than **63 percent ** of consumers say that good product images are more important than product descriptions.

For an *e-commerce platform*, good quality product images are instrumental in convincing shoppers to buy. Product images can help shoppers to get a better virtual "feel" about the product and engage on a deeper level. Collection of over 200 product images under Apparel category. Two gender types women and men under Apparel.

Importing the dataset:

- The image classification dataset been scrapped using selenium through E-commerce website.
- The dataset has been uploaded on the google drive.
- We have mounted the google colab with the google drive so that we can connect the dataset through the drive.

In [25]:
```python
import warnings
warnings.filterwarnings("ignore")

import os
import glob
import shutil
```

In [26]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [1]:
```
!nvidia-smi
```

```
Sun Nov 28 11:27:11 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   66C    P8    36W / 149W |      0MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Importing the Library

- We have imported the important libraries and then stored the path of the train data and the test data.
- The train data path has been stored with the attribute train_path and the test data been stored on the test_path
- We will be using Inception V3 for image classification.

```python
# import the libraries
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3
# from keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.inception_v3 import preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img,img_to_array
from tensorflow.keras.models import Sequential
import numpy as np
from glob import glob
import tensorflow as tf
from IPython.display import Image, display
```

```python
# resize all the image to same size
Imaze_SIZE = [224,224]
train_path = "/content/drive/MyDrive/Hindu paper/Dataset/Train_data"
test_path = "/content/drive/MyDrive/Hindu paper/Dataset/Test_data"

# Import the Inception V3 library as shown below and ass preprocesing layer to the front of VGG
# Here we will be using imagenet weights

inception= InceptionV3(input_shape=Imaze_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
87924736/87910968 [==============================] - 1s 0us/step
```

Inception V3

1. Factorization into Smaller Convolutions

- 2. Spatial Factorization into Asymmetric Convolutions
- 3. Utility of Auxiliary Classifiers
- 4. Efficient Grid Size Reduction
- 5. The total loss used by the inception net during training.

total_loss = real_loss + 0.3 * aux_loss_1 + 0.3 * aux_loss_2

```
In [6]:  # Don't train the existing weights
         for layer in inception.layers:
           layer.trainable= False

In [7]:  # useful for g
         folders = glob('/content/drive/MyDrive/Hindu paper/Dataset/Train_data/*')
         folders

Out[7]: ['/content/drive/MyDrive/Hindu paper/Dataset/Train_data/Trousers',
         '/content/drive/MyDrive/Hindu paper/Dataset/Train_data/Saree',
         '/content/drive/MyDrive/Hindu paper/Dataset/Train_data/Jeans']

In [8]:  # our layers- we can add more if we want
         x = Flatten()(inception.output)

In [9]:  prediction = Dense(len(folders),activation='softmax')(x)

In [10]: # create a model Object
         model = Model(inputs=inception.input,outputs=prediction)

In [11]: # View the structure of the model
         model.summary()

        Model: "model"
        __________________________________________________________________________________________________
         Layer (type)                    Output Shape         Param #     Connected to
        ==================================================================================================
         input_1 (InputLayer)            [(None, 224, 224, 3  0           []
                                         )]

         conv2d (Conv2D)                 (None, 111, 111, 32  864         ['input_1[0][0]']
                                         )

         batch_normalization (BatchNorm  (None, 111, 111, 32  96          ['conv2d[0][0]']
```

The first layer in this network, tf.keras.layers.Flatten, transforms the format of the images from a two-dimensional array (of 28 by 28 pixels) to a one-dimensional array (of 28 * 28 = 784 pixels). Think of this layer as unstacking rows of pixels in the image and lining them up. This layer has no parameters to learn; it only reformats the data.

After the pixels are flattened, the network consists of a sequence of two tf.keras.layers.Dense layers. These are densely connected, or fully connected, neural layers. The first Dense layer has 128 nodes (or neurons). The second (and last) layer returns a logits array with length of 10. Each node contains a score that indicates the current image belongs to one of the 10 classes.

Pre-Processing

- **Softmax** extends this idea into a multi-class world. That is, Softmax assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities must add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would.
- Softmax might produce the following likelihoods of an image belonging to a particular class:
- Model . Summary() will provide the details of the project shape and imaze size and many more

Model Summary

```
In [11]: # View the structure of the model
         model.summary()

Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 conv2d (Conv2D)                (None, 111, 111, 32  864         ['input_1[0][0]']
                                )

 batch_normalization (BatchNorm  (None, 111, 111, 32  96          ['conv2d[0][0]']
 alization)                     )

 activation (Activation)        (None, 111, 111, 32  0           ['batch_normalization[0][0]']
                                )

 conv2d_1 (Conv2D)              (None, 109, 109, 32  9216        ['activation[0][0]']
                                )

 batch_normalization_1 (BatchNo  (None, 109, 109, 32  96          ['conv2d_1[0][0]']
 rmalization)                   )

 activation_1 (Activation)      (None, 109, 109, 32  0           ['batch_normalization_1[0][0]']
                                )

 conv2d_2 (Conv2D)              (None, 109, 109, 64  18432       ['activation_1[0][0]']
                                )

 batch_normalization_2 (BatchNo  (None, 109, 109, 64  192         ['conv2d_2[0][0]']
 rmalization)                   )

 activation_2 (Activation)      (None, 109, 109, 64  0           ['batch_normalization_2[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 54, 54, 64)   0           ['activation_2[0][0]']
```

Model Compile

```
In [12]: # tell the model what cost and optimization method to use
         model.compile(loss='categorical_crossentropy',
                       optimizer='adam',
                       metrics=['accuracy'])

In [13]:
```

For this project, we have choose

the tf.keras.optimizers.Adam optimizer

and tf.keras.losses.CategoricalCrossentropy loss function. To view training and validation accuracy for each training epoch, pass the metrics argument to Model.compile.

Loss function —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction.

Optimizer —This is how the model is updated based on the data it sees and its loss function.

Metrics —Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified.

Image Data Generator

In [13]:

```
# use the ImageDataGenerator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen= ImageDataGenerator(rescale=1./255,
                                  shear_range=0.2,
                                  zoom_range=0.2,
                                  horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [14]:

```
# make sure that we provide the same target size as initiated for the imaze size
training_set=train_datagen.flow_from_directory('/content/drive/MyDrive/Hindu paper/Dataset/Train_da
                                               target_size=(224,224),
                                               batch_size=32,
                                               class_mode='categorical')
```

```
Found 164 images belonging to 3 classes.
```

In [15]:

```
training_set.class_indices
```

Out[15]:

```
{'Jeans': 0, 'Saree': 1, 'Trousers': 2}
```

A data generator can also be used to specify the validation dataset and the test dataset. Often, a separate Image Data Generator instance is used that have the same pixel scaling configuration.

- As we can see that there is three classes in the data set. The classes are Jeans , saree and the Trouser.
- We have set the imaze size as 244*244.
- We took shear imaze and the zoom imaze as 20%.

Fitting the model

```
In [17]: #fit the model
         #Run the cell .It will take some time to execute
         r=model.fit_generator(training_set,
                               validation_data=test_set,
                               epochs=10,
                               steps_per_epoch=len(training_set),
                               validation_steps=len(test_set)
                               )
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning: `Model.fit_generator`
is deprecated and will be removed in a future version. Please use `Model.fit`, which supports gener
ators.
  import sys
Epoch 1/10
6/6 [==============================] - 54s 6s/step - loss: 3.7272 - accuracy: 0.4878 - val_loss: 1.
2872 - val_accuracy: 0.8158
Epoch 2/10
6/6 [==============================] - 3s 570ms/step - loss: 1.9289 - accuracy: 0.7683 - val_loss:
3.0203 - val_accuracy: 0.7368
Epoch 3/10
6/6 [==============================] - 3s 463ms/step - loss: 1.3629 - accuracy: 0.8293 - val_loss:
1.5753 - val_accuracy: 0.8158
Epoch 4/10
6/6 [==============================] - 3s 483ms/step - loss: 0.7134 - accuracy: 0.8963 - val_loss:
0.6317 - val_accuracy: 0.8684
```

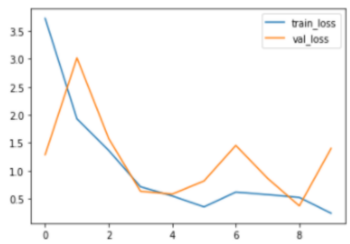
```
6/6 [==============================] - 3s 570ms/step - loss: 1.9289 - accuracy: 0.7683 - val_loss: 
3.0203 - val_accuracy: 0.7368
Epoch 3/10
6/6 [==============================] - 3s 463ms/step - loss: 1.3629 - accuracy: 0.8293 - val_loss: 
1.5753 - val_accuracy: 0.8158
Epoch 4/10
6/6 [==============================] - 3s 483ms/step - loss: 0.7134 - accuracy: 0.8963 - val_loss: 
0.6317 - val_accuracy: 0.8684
Epoch 5/10
6/6 [==============================] - 3s 480ms/step - loss: 0.5480 - accuracy: 0.8720 - val_loss: 
0.5812 - val_accuracy: 0.8684
Epoch 6/10
6/6 [==============================] - 3s 467ms/step - loss: 0.3523 - accuracy: 0.9268 - val_loss: 
0.8165 - val_accuracy: 0.8684
Epoch 7/10
6/6 [==============================] - 3s 475ms/step - loss: 0.6169 - accuracy: 0.9146 - val_loss: 
1.4527 - val_accuracy: 0.7895
Epoch 8/10
6/6 [==============================] - 3s 460ms/step - loss: 0.5719 - accuracy: 0.9024 - val_loss: 
0.8640 - val_accuracy: 0.8684
Epoch 9/10
6/6 [==============================] - 3s 487ms/step - loss: 0.5212 - accuracy: 0.9146 - val_loss: 
0.3705 - val_accuracy: 0.9211
Epoch 10/10
6/6 [==============================] - 3s 490ms/step - loss: 0.2382 - accuracy: 0.9451 - val_loss: 
1.4001 - val_accuracy: 0.8684
```

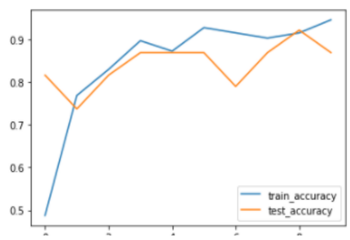
Train and Test Loss and Accuracy

```python
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='train_loss')
plt.plot(r.history['val_loss'],label='val_loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

#plot the accuracy

plt.plot(r.history['accuracy'],label='train_accuracy')
plt.plot(r.history['val_accuracy'],label='test_accuracy')
plt.legend()
plt.show()
plt.savefig('Accuracy over Epochs')
```





Predicting the test dataset:

```
In [22]: import numpy as np
         y_pred = np.argmax(y_pred, axis=1)

In [23]: y_pred

Out[23]: array([2, 0, 1, 2, 2, 1, 2, 0, 1, 2, 2, 1, 1, 2, 0, 0, 1, 2, 1, 2, 2, 1,
                1, 2, 1, 0, 1, 2, 0, 1, 2, 2, 2, 1, 2, 1, 0, 2])

In [24]: from tensorflow.keras.models import load_model
         from tensorflow.keras.preprocessing import image

In [67]: test_img , label =test_set.next()

In [72]: def plotImages(y_pred, label):
           """
           input:image array
           output:plot images
           """
           for idx ,img in enumerate(y_pred):
             if idx <= 10:
               plt.figure(figsize=(5,5))
               plt.imshow(img)
               plt.title(img.shape)
               plt.axis=False
               plt.show()

In [74]: plotImages(test_img, label)
```

- As we can see that in the prediction , we got 2 which means its belong to the trouser as we can see in the image that it predicted right.

Saving the model

```
In [19]: # Save the Model

         model.save('model_vgg16_ImageClassificationcomplete')
         model.save('model_vgg16_tf',save_format='tf')
         model.save('model_vgg16.h5')

INFO:tensorflow:Assets written to: model_vgg16_ImageClassificationcomplete/assets
INFO:tensorflow:Assets written to: model_vgg16_tf/assets
```