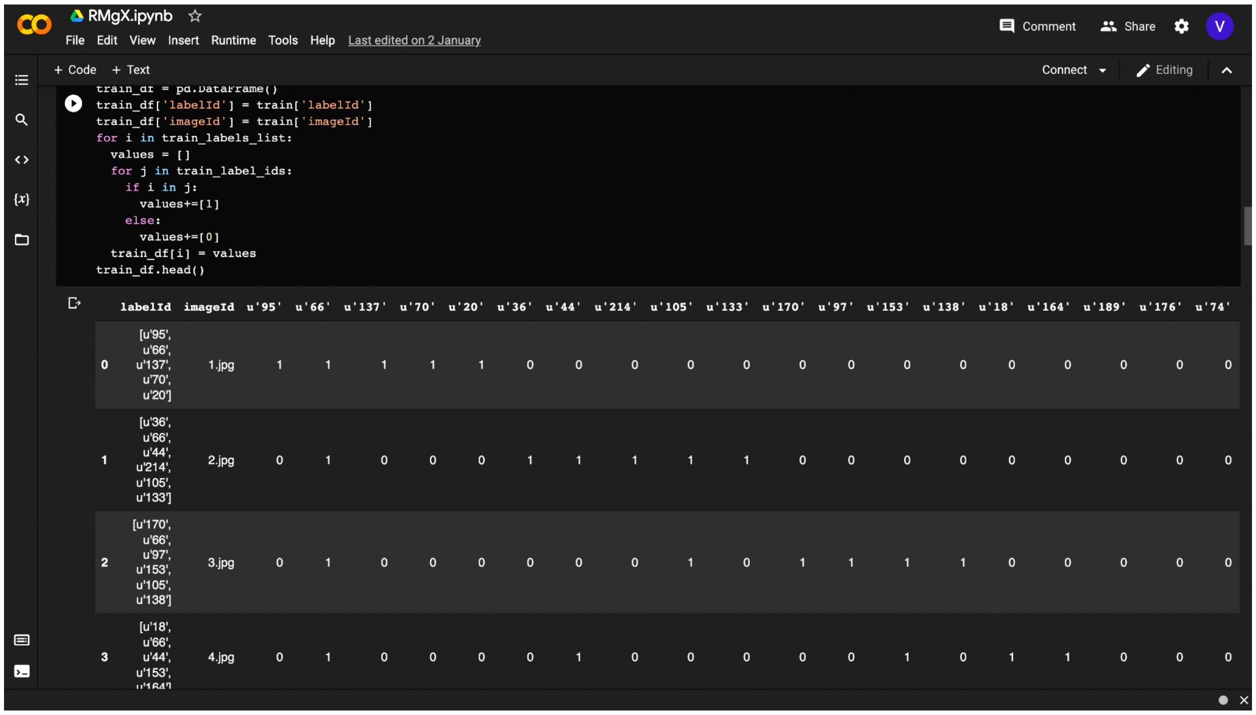
GitHub Repo link for code:-

https://github.com/varun202/Multi-Label-Classification

Data Preparation

- Notebook link:-
- To train the model we need to have a train and validation dataset along with their respective images and their labels.
- To load the images we can directly pass the path to the directory in which train and validation images.
- If we load the labels directly into the model there will be 15000 different labels for the 15000 images of the training dataset, and 3000 labels for the 3000 images in the validation dataset.
- **So we need to understand sending the raw labels could not be helpful to train the model**
- We process the data and if we look clearly at each image has a list of labels given to them which are basically classes that they belong to. Thus we need to create a data frame that can resemble all the labels in the dataset. It means classes from **[1 - 228]**.


RMgX.ipynb
File Edit View Insert Runtime Tools Help Last edited on 2 January
Comment Share V
+ Code + Text
Connect Editing

```
train_df = pd.DataFrame()
train_df['labelId'] = train['labelId']
train_df['imageId'] = train['imageId']
for i in train_labels_list:
  values = []
  for j in train_label_ids:
    if i in j:
      values+=[1]
    else:
      values+=[0]
  train_df[i] = values
train_df.head()
```

	labelId	imageId	u'95'	u'66'	u'137'	u'70'	u'20'	u'36'	u'44'	u'214'	u'105'	u'133'	u'170'	u'97'	u'153'	u'138'	u'18'	u'164'	u'189'	u'176'	u'74'
0	[u'95', u'66', u'137', u'70', u'20']	1.jpg	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	[u'36', u'66', u'44', u'214', u'105', u'133']	2.jpg	0	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
2	[u'170', u'66', u'97', u'153', u'105', u'138']	3.jpg	0	1	0	0	0	0	0	0	1	0	1	1	1	1	0	0	0	0	0
3	[u'18', u'66', u'44', u'153', u'164']	4.jpg	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	0	0

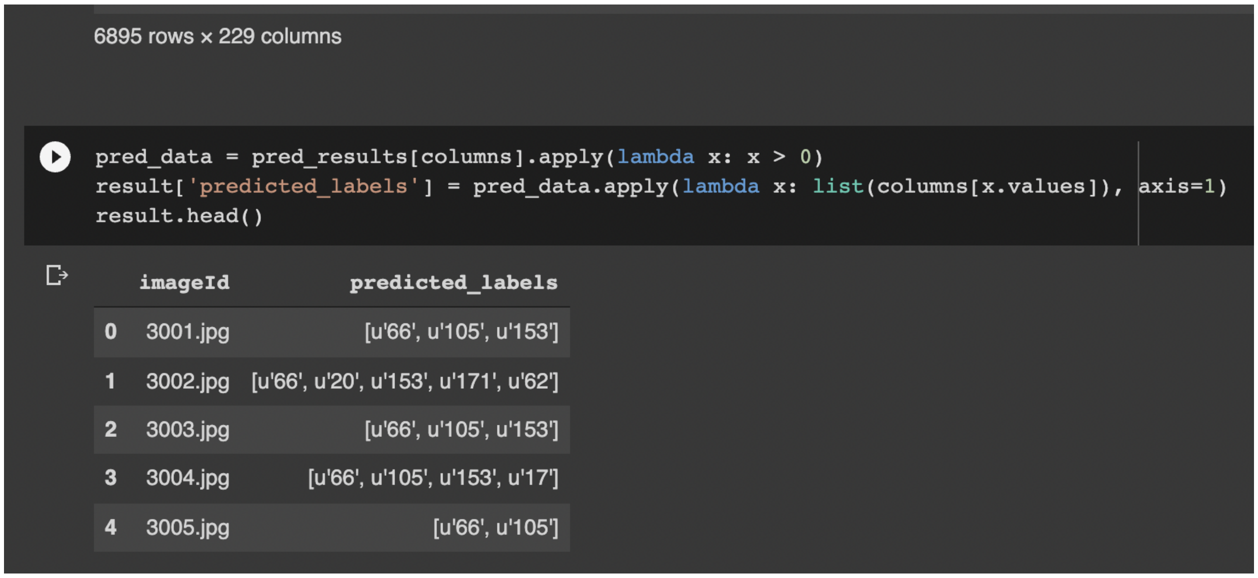
- So we implement some pre-processing of train and validation labels files and create respective data frames
- These dataframes can represent a cell value with **1** if the respective column value is present in that particular label of the image, else it is marked as **0**
- Next, we create dataset preparation using **imagedatagenerator()** and **flow_from_dataframe()** methods. We chose the flow_from_dataframe method because have the labels in dataframes, thus the model can know for a respective image it can look to the respective cell values that we have passed.
- We have set the target size to be (128, 128) to minimize the load on the model.
- Similarly, we create the test dataset

Build and Compile the Neural Network.

- We create the model network by using the Sequential layer as the base layer and adding a few convolutional layers on top of it.
- For every convolutional layer, we set the kernel size as (5, 5) with the activation function as 'relu'
- After a set of convolution layers, we need to convert the outputs returned by the last layer into a vector which is known as the flattening layer. This flattens layer determines the number of input nodes to the neural network.
- We need to have a certain number of nodes in the dense layer such that it could handle the data received from the previous layer. We calculate it by using the average of the number of nodes in flatten layer and the number of nodes in the output layer.
- Flatten has 1024 nodes and we need 228 nodes in the output layer because we have 228 classes. So [1024 + 228]/2 -> 626. Thus we add a certain number of nodes close to this average value.
- Then we train the model using the training dataset and validation dataset for 10 epochs.

Model Evaluation

- We can plot the accuracy and loss function for train and validation dataset by the model using matplot library
- To predict the results we pass the test dataset to **model.predict()**
- We normalize the predicted values to [0, 1] using lambda expressions and display the predictions with associated images name in a dataframe
- Finally, we process the dataframe to store the names of the images with a list of column names of cell values that are having value =1



6895 rows × 229 columns

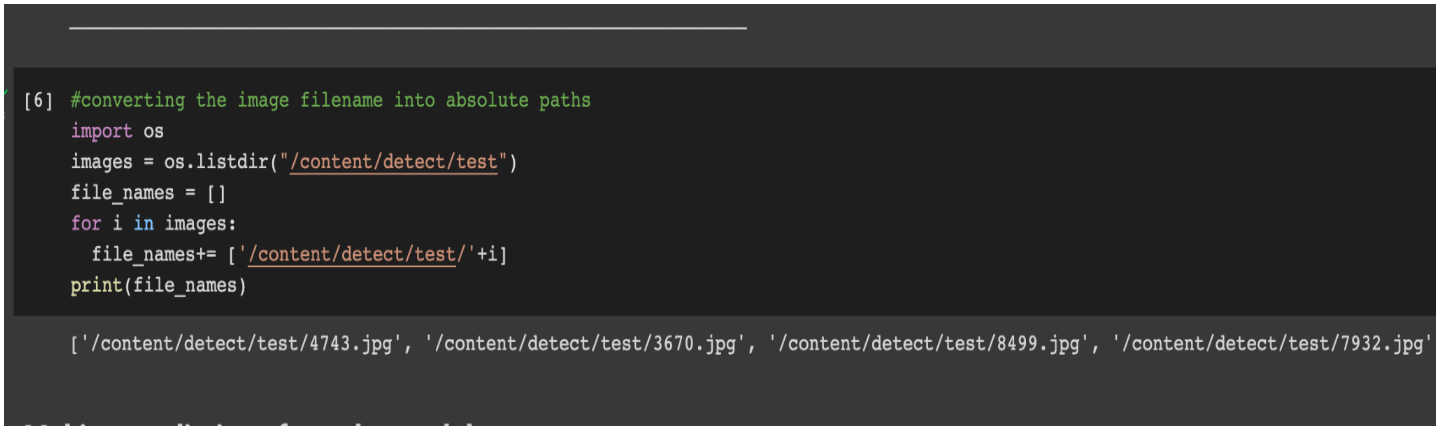
```
pred_data = pred_results[columns].apply(lambda x: x > 0)
result['predicted_labels'] = pred_data.apply(lambda x: list(columns[x.values]), axis=1)
result.head()
```

	imageId	predicted_labels
0	3001.jpg	[u'66', u'105', u'153']
1	3002.jpg	[u'66', u'20', u'153', u'171', u'62']
2	3003.jpg	[u'66', u'105', u'153']
3	3004.jpg	[u'66', u'105', u'153', u'17']
4	3005.jpg	[u'66', u'105']

- Export these predictions dataframe to the **result.csv file**

Object Detection

- For object detection, we use a VGG-19 pre-trained network that can classify up to 1000 objects.
- To perform detection on 6000 test images might take up quite a bit of time so in order to ease the process I have performed detections on randomly chosen 1000 images from the test dataset
- To load the image we use load_image from keras.preprocessing.image that accepts the path of the image file. So we create a list of 1000 image file names along with their path as file names.

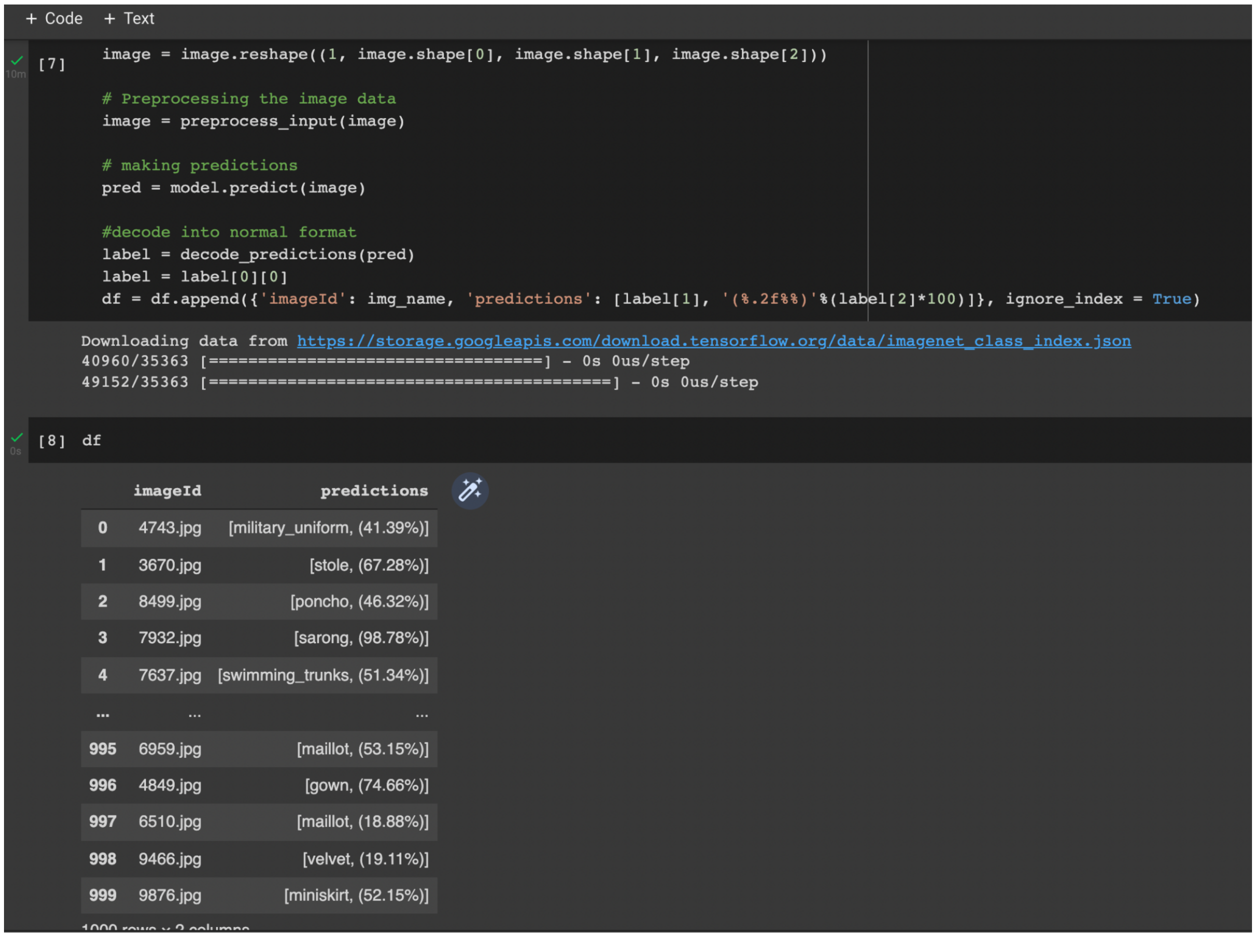


```
[6] #converting the image filename into absolute paths
    import os
    images = os.listdir("/content/detect/test")
    file_names = []
    for i in images:
      file_names+= ['/content/detect/test/'+i]
    print(file_names)

    ['/content/detect/test/4743.jpg', '/content/detect/test/3670.jpg', '/content/detect/test/8499.jpg', '/content/detect/test/7932.jpg'
```

- The VGG-19 model accepts images in the form of a NumPy array that has 4-dimensions. So we transform the pixels of the image into a NumPy array with each cell having a value ranging between **[0 to 256]** and add one more dimension to this array.

- We can predict the objects in the image now and we iterate through each file name present in the list that we have created previously.



```
[7]    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

       # Preprocessing the image data
       image = preprocess_input(image)

       # making predictions
       pred = model.predict(image)

       #decode into normal format
       label = decode_predictions(pred)
       label = label[0][0]
       df = df.append({'imageId': img_name, 'predictions': [label[1], '(%.2f%%)'%(label[2]*100)]}, ignore_index = True)

    Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
    40960/35363 [=================================] - 0s 0us/step
    49152/35363 [========================================] - 0s 0us/step
```

```
[8]  df
```

```
        imageId                   predictions
0      4743.jpg  [military_uniform, (41.39%)]
1      3670.jpg             [stole, (67.28%)]
2      8499.jpg           [poncho, (46.32%)]
3      7932.jpg           [sarong, (98.78%)]
4      7637.jpg  [swimming_trunks, (51.34%)]
...         ...                           ...
995    6959.jpg          [maillot, (53.15%)]
996    4849.jpg             [gown, (74.66%)]
997    6510.jpg          [maillot, (18.88%)]
998    9466.jpg           [velvet, (19.11%)]
999    9876.jpg        [miniskirt, (52.15%)]
```

- Finally, we can save the predictions to the csv file.

Hopefully, I have mentioned everything clearly about the approach for the Multi-label classification and object detection from the dataset. Please do revert back to me if there are any doubts/clarifications regarding my code.