CISC 867 Project2: Deep learning

Name: Hanan Fared Mohamed Omara.

ID: 20398559

DR/ Hazem Abbas

Eng/ Asif Mahfuz

Data Preparation 1

In this project, you will use the Fashion-MNIST dataset using a CNN neural network architecture.

1- First, download the data file and load it.

2- Describe the data.

[9]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Columns: 785 entries, label to pixel784
dtypes: int64(785)
memory usage: 359.3 MB
```

[10]:
```
train_df.describe()
```

[10]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...
count	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	60000.000000	...
mean	4.500000	0.000900	0.006150	0.035333	0.101933	0.247967	0.411467	0.805767	2.198283	5.682000	...
std	2.872305	0.094689	0.271011	1.222324	2.452871	4.306912	5.836188	8.215169	14.093378	23.819481	...
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	2.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	4.500000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	7.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
max	9.000000	16.000000	36.000000	226.000000	164.000000	227.000000	230.000000	224.000000	255.000000	254.000000	...

8 rows × 785 columns

3- Check the data for missing values or duplicates and carry out proper correction methods.

Check Missing Values

it seems that there is no missing values

[12]:
```
# data has no nulls
train_df.isnull().sum().sum()
```

[12]: 0

Check Duplicates

it seems that there is 43 duplicates

[13]:
```
# data has no duplicated rows
train_df.duplicated().sum()
```

[13]: 43

4- Clean the data.

Remove Duplicates

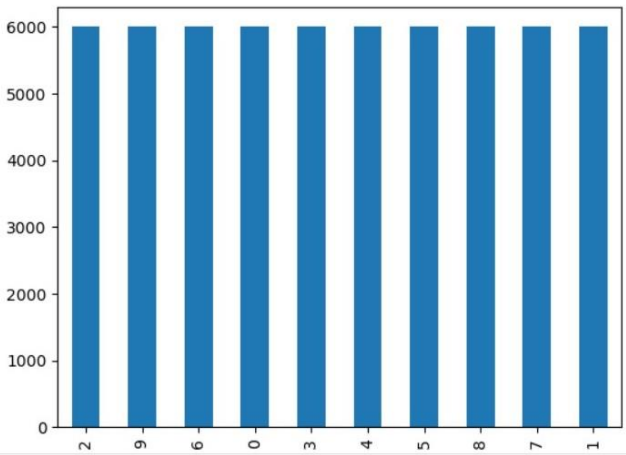
[14]:
```
train_df.drop_duplicates(inplace=True)
```

5- Visualize the data using proper visualization methods.

Data Visualization

```
[11]: train_df.label.value_counts().plot(kind='bar')
```

```
[11]: <AxesSubplot:>
```



6- Draw some of the image.

Display some Images

```
[20]: plt.figure(figsize=(10,10))

for index in range(49):
    plt.subplot(7, 7, index+1)
    plt.imshow(X_train[index], cmap='binary')
    plt.title(lebels[int(y_train[index])])
    plt.axis('off')
plt.subplots_adjust(hspace=0.7, wspace=0.8)
plt.show()
```

7- Carry out required correlation analysis.

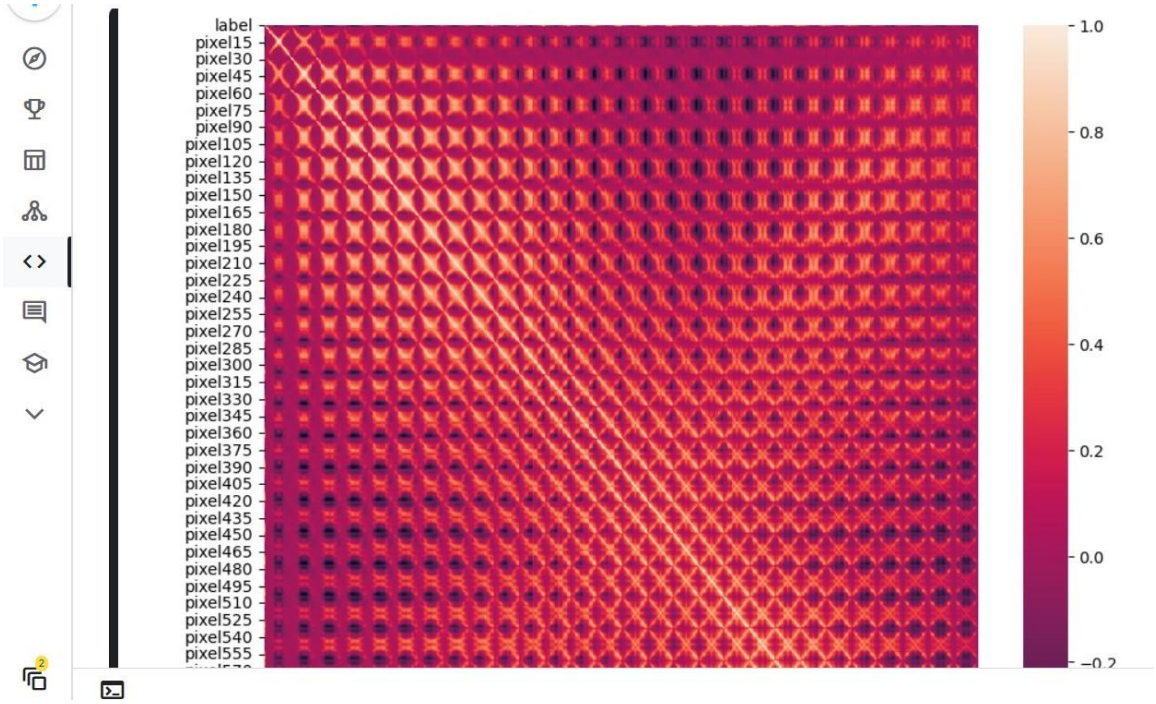
show correlation

[18]:
```
# check the correlation between all features to benefits of it if there is any high correlation between fea
corr = train.corr()
corr
```

[18]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777
label	1.000000	-0.000674	0.002944	-0.010439	-0.007111	-0.004632	-0.011045	-0.036765	-0.085211	-0.161552	...	-0.362614	-0.258380	-0.183131
pixel1	-0.000674	1.000000	0.297899	0.067550	0.046608	0.026630	0.026172	0.012095	0.012218	0.009637	...	-0.000642	0.004625	0.004605
pixel2	0.002944	0.297899	1.000000	0.575029	0.138710	0.054354	0.033185	0.022763	0.017127	0.016809	...	0.000492	0.004857	0.006811
pixel3	-0.010439	0.067550	0.575029	1.000000	0.387466	0.118135	0.087300	0.060927	0.035920	0.029712	...	0.010131	0.016743	0.018358
pixel4	-0.007111	0.046608	0.138710	0.387466	1.000000	0.573172	0.325684	0.242954	0.140955	0.085253	...	0.009684	0.018697	0.023373
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
pixel780	-0.066936	-0.002441	-0.002342	-0.001501	0.010095	0.021505	0.032305	0.027024	0.019414	0.024837	...	-0.074375	-0.009124	0.042812
pixel781	-0.018038	-0.000108	0.004273	0.006860	0.023942	0.025974	0.038563	0.030464	0.026227	0.034158	...	-0.042349	0.008555	0.045394
pixel782	0.045598	0.008765	0.014218	0.013152	0.012392	0.028863	0.044121	0.029457	0.022724	0.020953	...	-0.002807	0.034069	0.073942
pixel783	0.059960	0.026389	0.021297	0.009946	0.003075	0.022941	0.030806	0.016126	0.005938	0.000074	...	0.001898	0.026530	0.054193
pixel784	0.021772	0.041582	0.022162	0.015657	0.008423	0.007124	0.004950	0.001792	0.000591	0.000279	...	0.016603	0.028391	0.037301

785 rows × 785 columns



8- Carry out any required preprocessing operations on the data.

Split Data to train and test

```
# split to features and label
data = np.asarray(train_df, dtype=np.float32)
X = data[:, 1:]
y = data[:, 0]
```

+ Code + Markdown

Reshape data

Convert data from columns form to image matrix 28×28 form then add padding to be 32×32 because it is the minimum acceptable image size for pretrained models.

[16]:
```
# reshape to 28×28
X = X.reshape(-1, 28, 28)
# add padding
X = tf.pad(X, [[0, 0], [2, 2], [2, 2]])
# add the channels dimention so it will become (..., 32, 32, 1) instead of (..., 32, 32)
X = tf.expand_dims(X, axis=3, name=None).numpy()
X.shape
```

Convert data from columns form to image matrix 28×28 form then add padding to be 32×32 because it is the minimum acceptable image size for pretrained models.

[16]:
```
# reshape to 28×28
X = X.reshape(-1, 28, 28)
# add padding
X = tf.pad(X, [[0, 0], [2, 2], [2, 2]])
# add the channels dimention so it will become (..., 32, 32, 1) instead of (..., 32, 32)
X = tf.expand_dims(X, axis=3, name=None).numpy()
X.shape
```

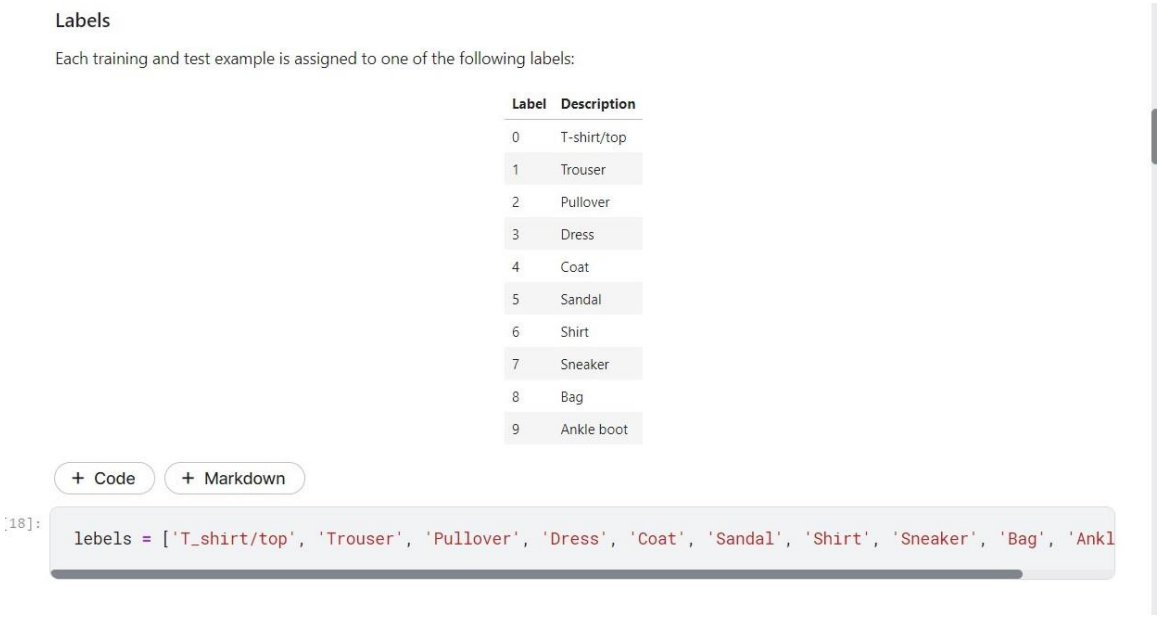
[16]:
```
(59957, 32, 32, 1)
```

Normalization

dividing by 255

[17]:
```
X = X / 255
```

9- Encode the labels.



Labels

Each training and test example is assigned to one of the following labels:

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

+ Code + Markdown

[18]:
```
lebels = ['T_shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankl
```

Training a CNN neural network

I will implement a LeNet-5 network to recognize the Fashion MNIST dataset.

1- Split data to train and validation
2- Build Model

Split data to train and validation

```
X_train_tune, X_val_tune, y_train_tune, y_val_tune = train_test_split(X_train, y_train, test_size=0.1, shuffle=True, stratify=y_train, random_state=seed)
```

+ Code + Markdown

Build Model

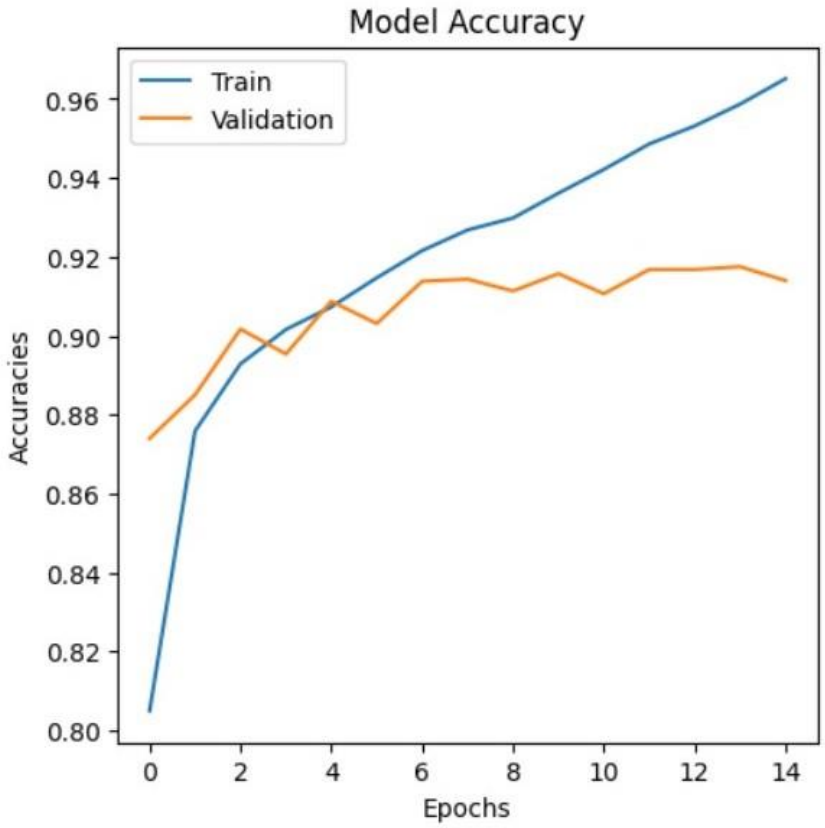
Add the ability to tune hyperparameters ex:

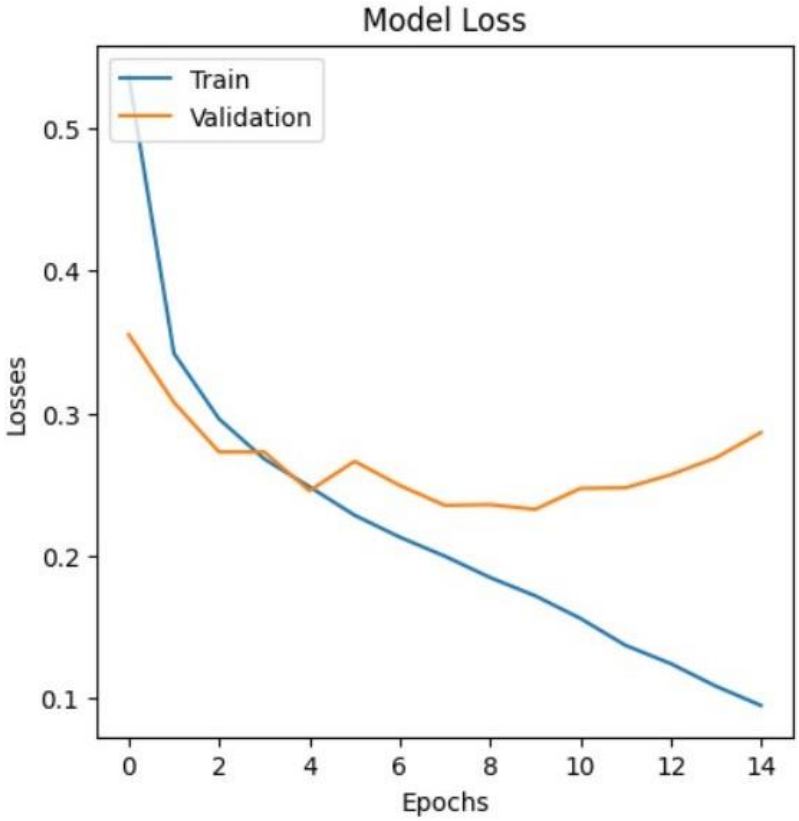
- Hidden Layer Size
- Learning Rate

3- Modify hyperparameters to get to the best performance and evaluate the model. (as shown in code)
    - As we build method to determine the best hyperparameter to train the model with them to get

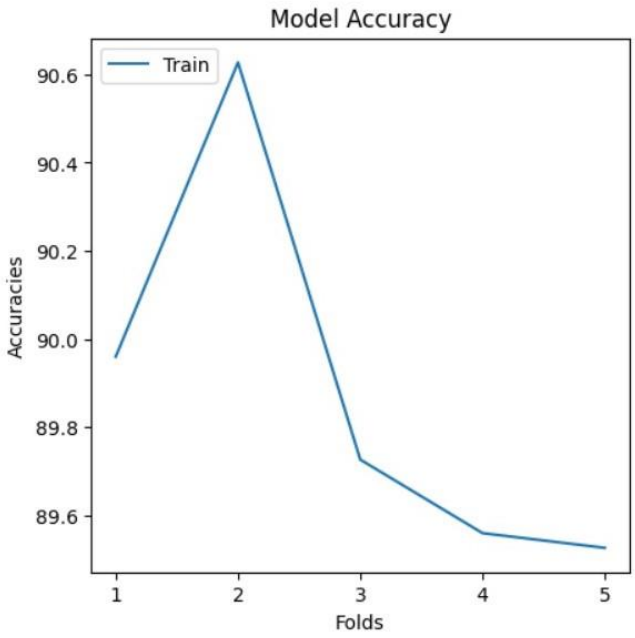
the highest accuracy and we found The optimal number of units in the first densely-connected layer is 240 and the optimal learning rate for the Adam optimizer is (0.001) for the Best val_accuracy So Far: (0.8807935118675232)

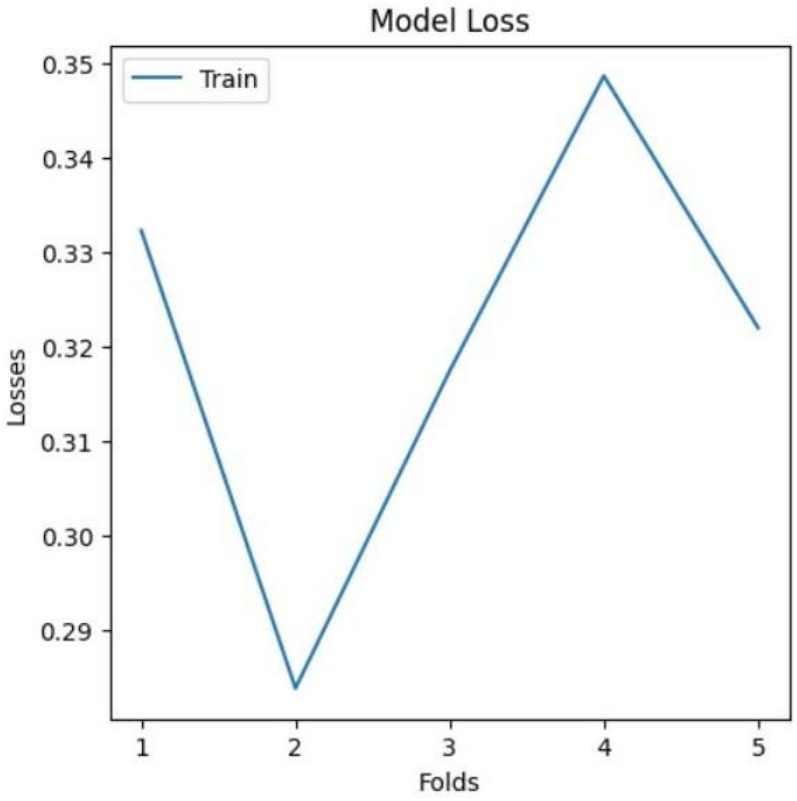
- When evaluate the model we found Score for: loss of 0.32196974754333496; accuracy of 89.52634930610657% but there is main disadvantage of LeNet-5 is **overfitting** in some cases and no built-in mechanism to avoid this .
- The plot of accuracy improvement using the previously mentioned techniques.





4- Evaluate the model using 5-fold cross-validation.
As shown in code and that is the plot of loss and accuracy.





- Graphs show the best evaluation accuracy and loss for each model during the cross validation process.

- We can notice from previous graphs of loss and accuracy that the LeNet-5 model is not good enough only so I will try to use transfer learning to improve the accuracy of model.

Transfer Learning

- replace the fully connected layer of a pre-trained model with a fully connected layer suitable for our problem.
- I will the following three pre-trained model Transfer Learning.

Model	Top-1 Accuracy	Top-5 Accuracy	Parameters
VGG19	71.3%	90.0%	143.7M
RESNet152V2	78.0%	94.2%	60.4M
DenseNet201	77.3%	93.6%	20.2M

1- VGG19

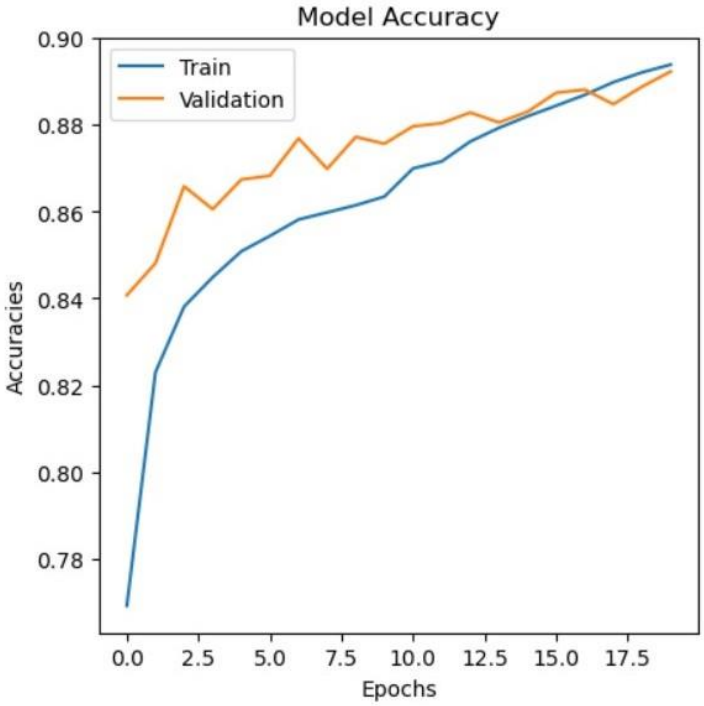
use a pretrained VGG19 and replace its fully connected network with a new one that I hope it will produce a good accuracy.

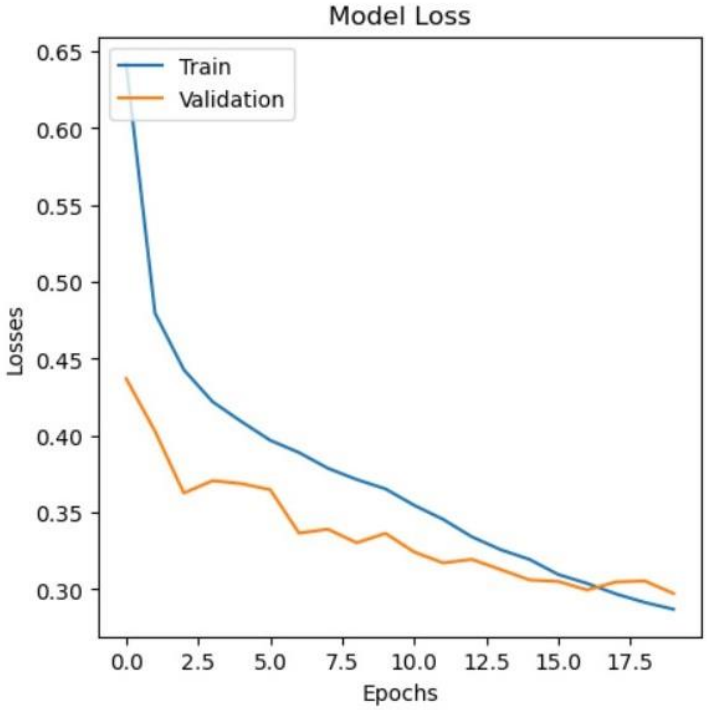
As shown in code we got loss: 0.2876 - accuracy: 0.8923 - val_loss: 0.2989 - val_accuracy: 0.8896 when

learning rate = 3.6788e-04, which is good .also on test Score for: loss of 0.355573892593383 8; accuracy of 87.92528510093689% using 20 epochs.

and I think it is enough good.

And its graphs are


Model Accuracy
Train
Validation
Accuracies
Epochs
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

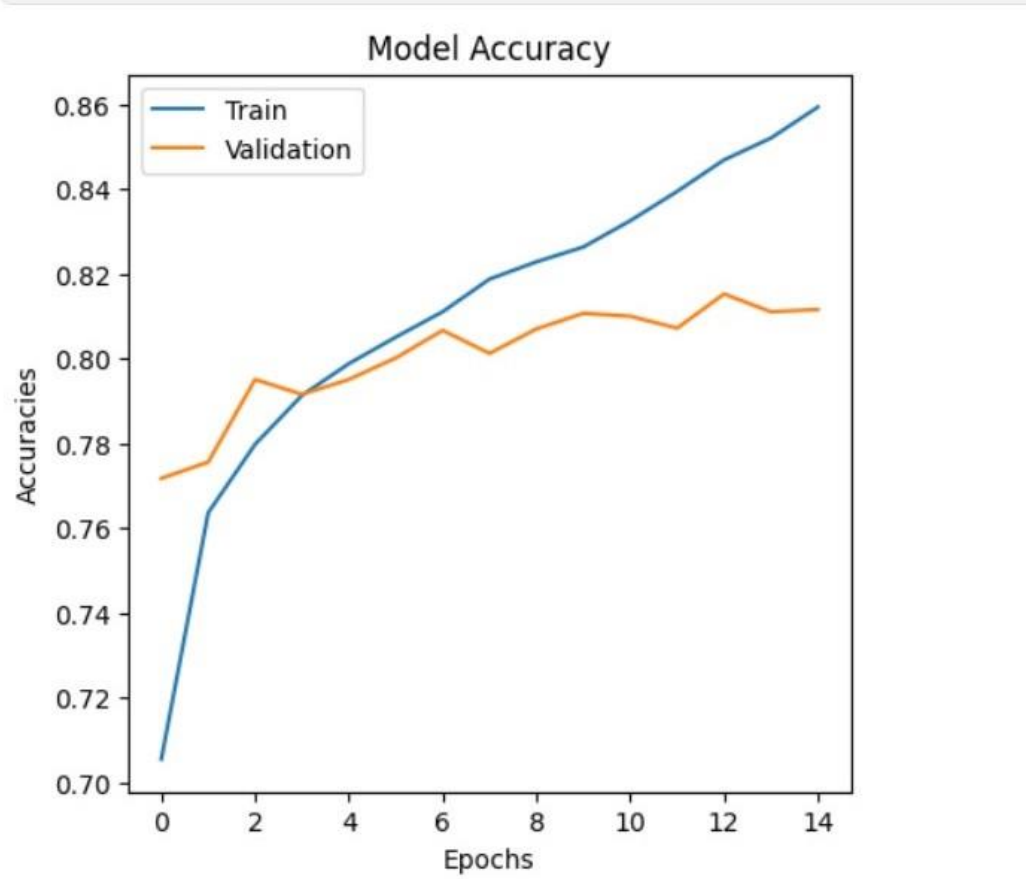

Model Loss
Train
Validation
Losses
Epochs
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

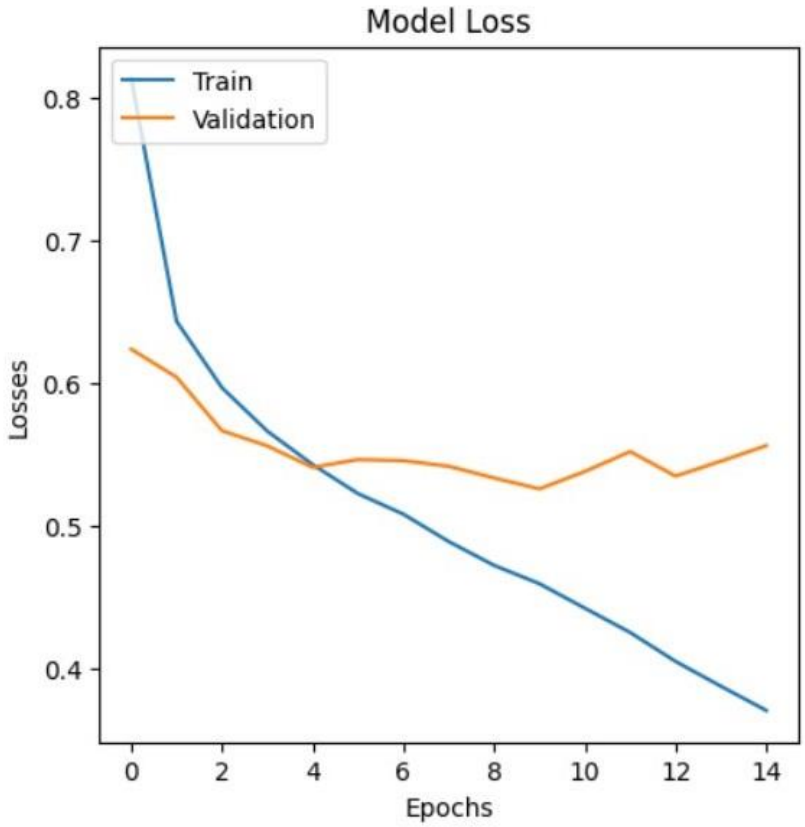
2- RESNet152V2

use a pretrained RESNet152V2 and replace its fully connected network with a new one that I hope it will produce a good accuracy. And as shown we got less accuracy than vgg19 but using only 10 epochs we got loss: 0.4463 - accuracy: 0.8320 - val_loss: 0.5357 - val_accuracy: 0.8127 - lr: 0.0010, and we got on test

Score for: loss of 0.5768388509750366; accuracy of 80.25349974632263%

and its graphs are





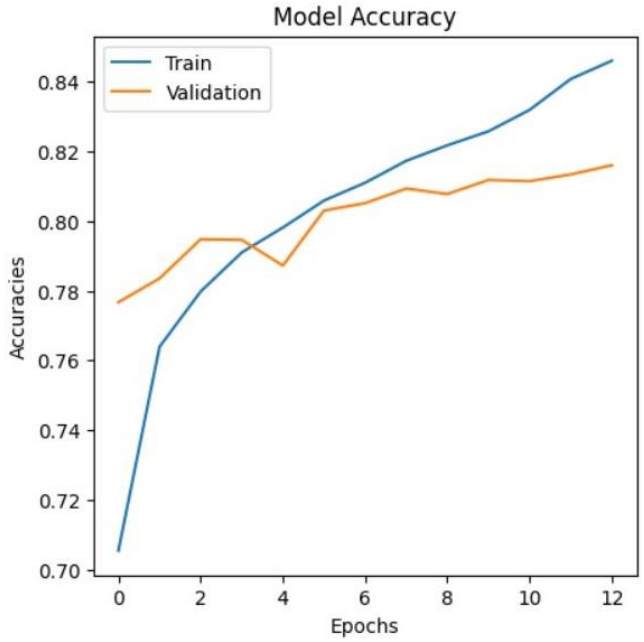
3- DenseNet201

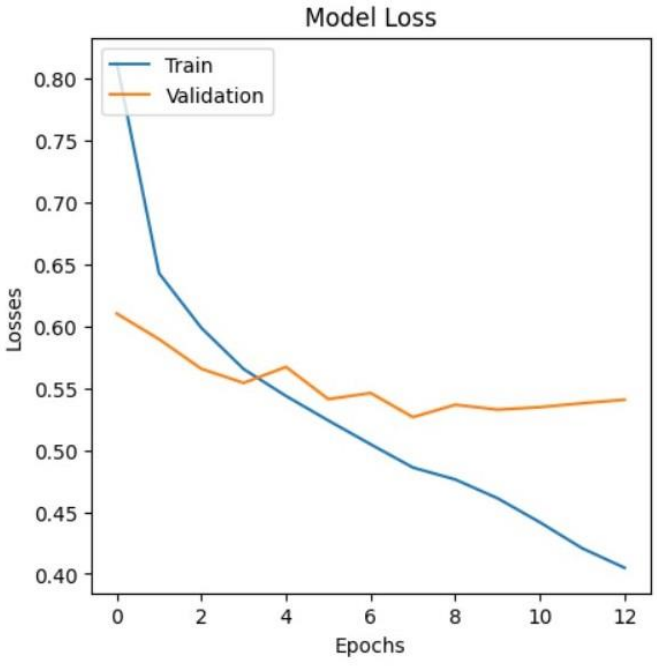
use a pretrained DenseNet201 and replace its fully connected network with a new one that I hope it will produce a good accuracy. And as shown we got less accuracy than other models

loss: 0.4614 - accuracy: 0.8264 - val_loss: 0.5260 - val_accuracy: 0.8146 - lr: 0.0010 when apply 10 epochs.

And on test we got Score for: loss of 0.5560103058815002; accuracy of 80.58705925941467%

and its graphs are





- **Observation**

  we notice that using transfer learning to tune and improve the model is a good choice as we got the best accuracy and the lowest loss when using VGG19 model.