Name: Anukriti
E-mail id: anu1999kriti@gmail.com

PROJECT REPORT

Problem Statement: To develop a CNN-based classification architecture for classifying a given chart image to one of five chart classes, namely “Line”,“Dot Line”,“Horizontal Bar”, and “Vertical Bar”, and “Pie” chart.

[Completed as a sub-step for Task 2, Task 3 and Task 4]
Task 1: [Pre-processing steps]
Downloaded the dataset from the drive link mentioned in the mail as charts.zip and created a csv file ‘train.csv’ to map image file name with the corresponding chart type. Also, created a categorical variable named ‘code’ for chart type. { “vbar_categorical” : 4, “pie”:3, “line”: 2, “hbar_categorical”: 1, “dot_line”: 0}
Image size: 128 X 128
Used the train and Val images for training and validation in an appropriate ratio (80% for training and 20 % for validating) while training.

Task 2 (explained later)

Task 3: Fine-tuned a pre-trained network for this task and calculated accuracy, loss and then, plotted the obtained loss.

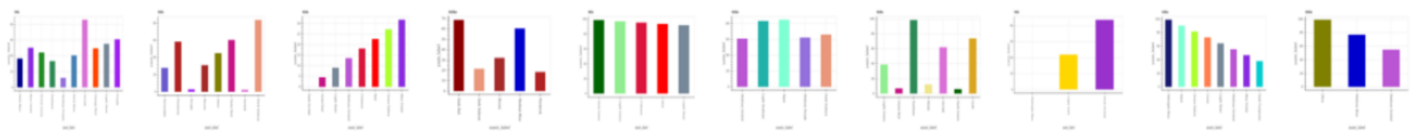
Code Explanation [Sub-section Task 3 of ChartClassification.ipynb]:
- Used a pre-trained VGG model with imagenet weights and fine-tuned it.
- Rest steps are all same as Task 2

#Training
Trained the model using train_val dataset on data generated batch-by-batch by a Python generator and used Stratified K-Fold cross validation along with shuffling. Also, recorded mean accuracy value, loss and also, plotted the accuracy and loss values.
Trained for only 30 epochs per iteration (5).

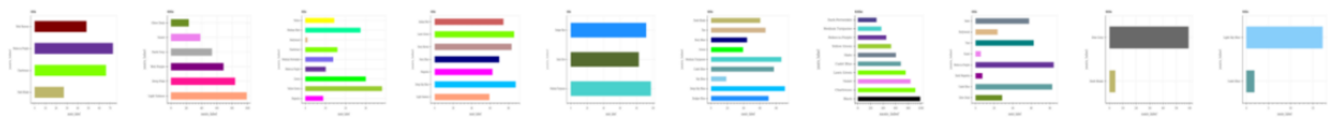
#Testing
Loaded the trained model for testing and saved all the predictions in the results folder.

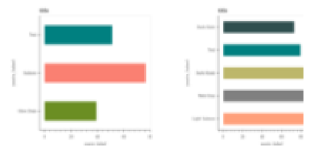
#Results

1. Vbar_categorical graphs predicted by the model:



2. Hbar_categorical graphs predicted by the model:

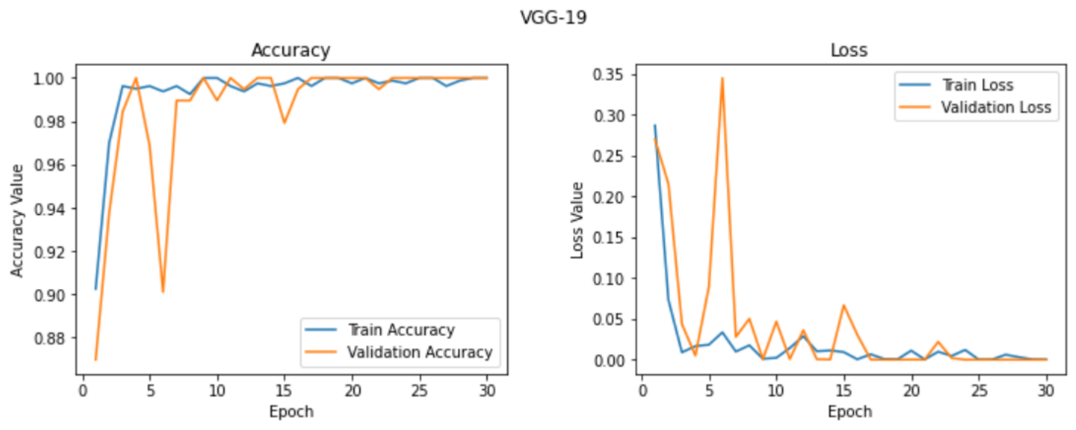




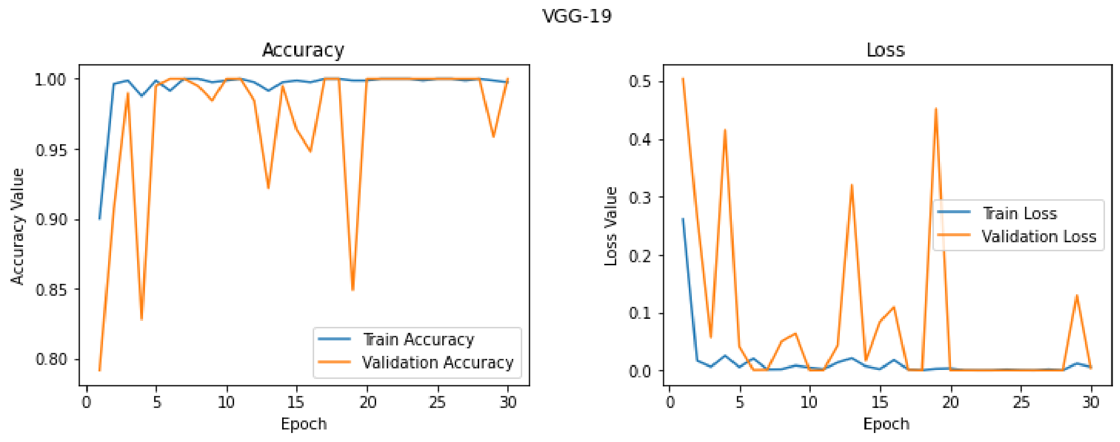
Similarly, others can be plotted as well.

Results from all iterations:
Accuracy obtained after fine-tuning: 1.0 (100%) → Also, checked for over-fitting but not that case, predictions also 100% accurate as can be seen from above plots.

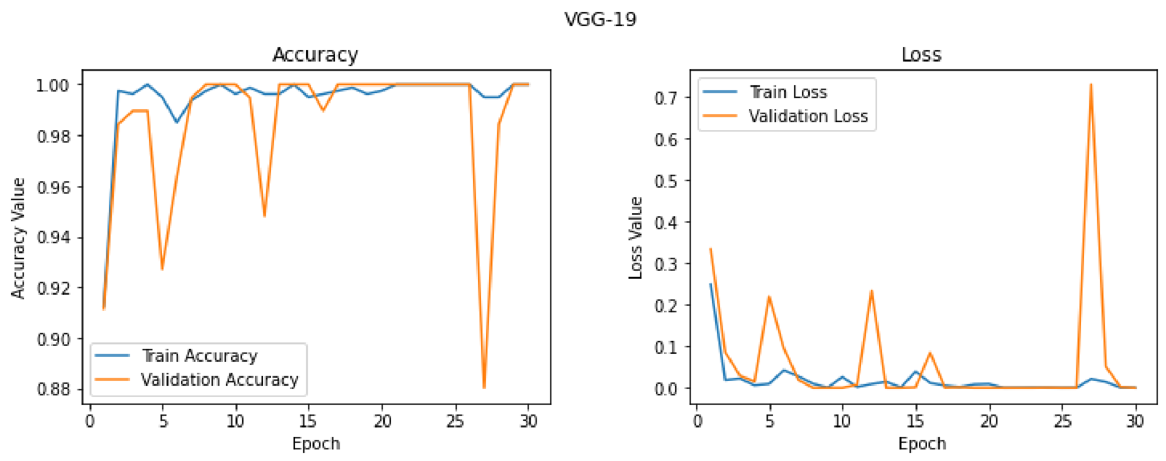
Iteration 1:



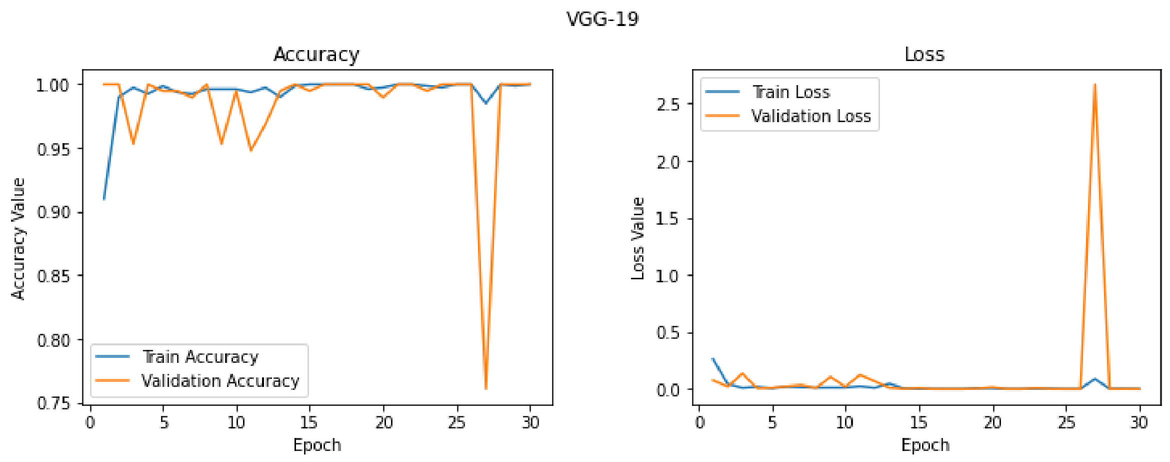
Iteration 2:



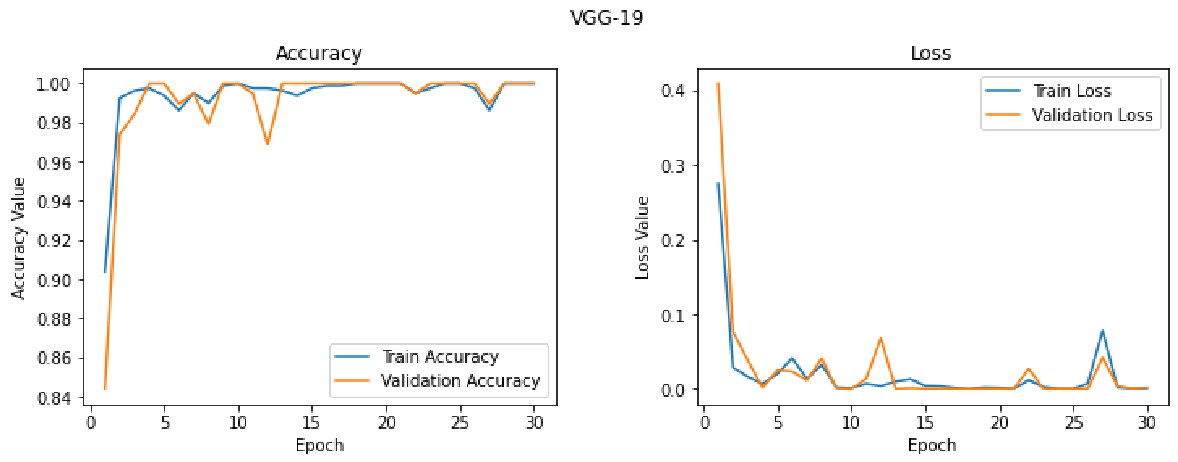
Iteration 3:



Iteration 4:



Iteration 5:



Task 2: Implemented a two-layer Convolutional Neural Network, and calculated accuracy, loss and then, plotted the obtained loss.

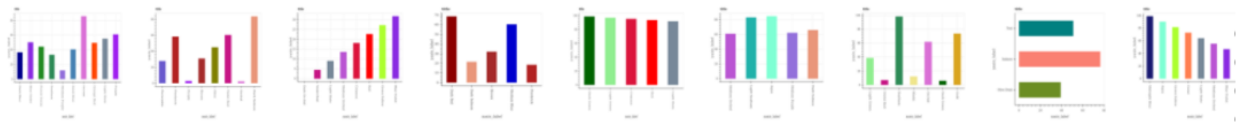
Code Explanation [Sub-section Task 2 of ChartClassification.ipynb]:
# Model Creation

- Initialized the CNN, added first layer with 32 feature detectors (with 3*3 dimensions so that convolution layer compose of 32 feature maps) [Convolution layer]
- Used tensorflow backend and reduced the size of feature maps i.e. reduced the number of nodes in the future fully connected layer to reduce time complexity, less compute intense without losing the performance. [MaxPooling layer]
- Flattened all the feature maps in the pooling layer into single vector [Flattening]
- Added all the layers in a classic ANN which composed of fully connected layers, chose the number of nodes in hidden layers in the power of 2 (common practice)
- Finally, added the output layer with number of nodes = 5 (no. of chart types)
- Note: Activation function for hidden layers: relu, output layer: softmax
- Compiled the CNN model : Loss function - Categorical cross entropy, Optimizer - AdaDelta with learning rate = 1 and chose accuracy as the metrics.

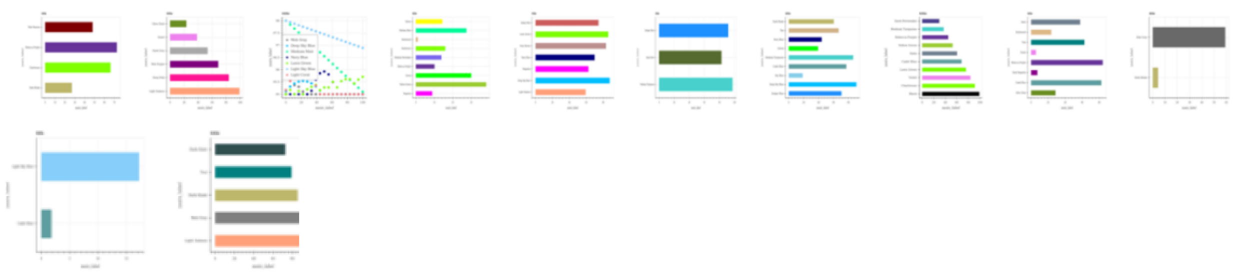
#Training
Trained the model using train_val dataset on data generated batch-by-batch by a Python generator and used Stratified K-Fold cross validation along with shuffling. Also, recorded mean accuracy value, loss and also, plotted the accuracy and loss values.
Trained for only 30 epochs per iteration (5).

#Testing
Loaded the trained model for testing and saved all the predictions in the results folder.

#Results
1. Vbar_categorical graphs predicted by the model:



2. Hbar_categorical graphs predicted by the model:

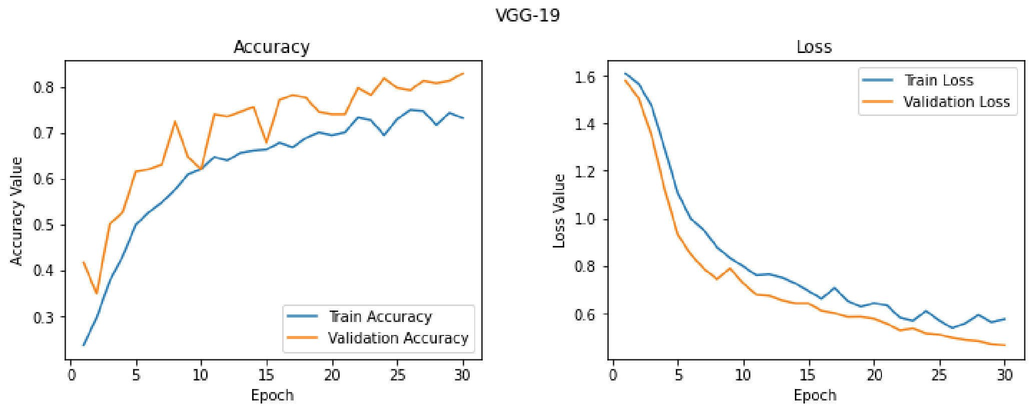


Similarly, other chart types can be plotted as well.

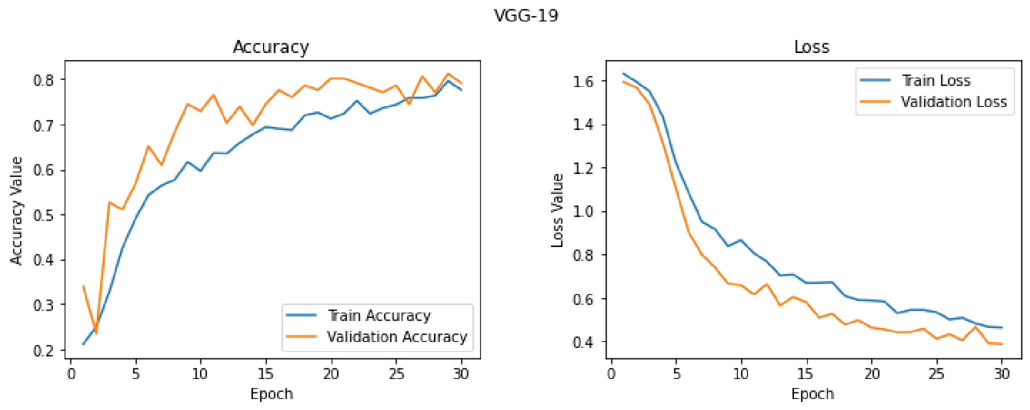
Results from all iterations:

Accuracy obtained from a simple CNN model: 81.67% (+/- 3.04%) → Also, checked for test predictions and validated similar results from the same.

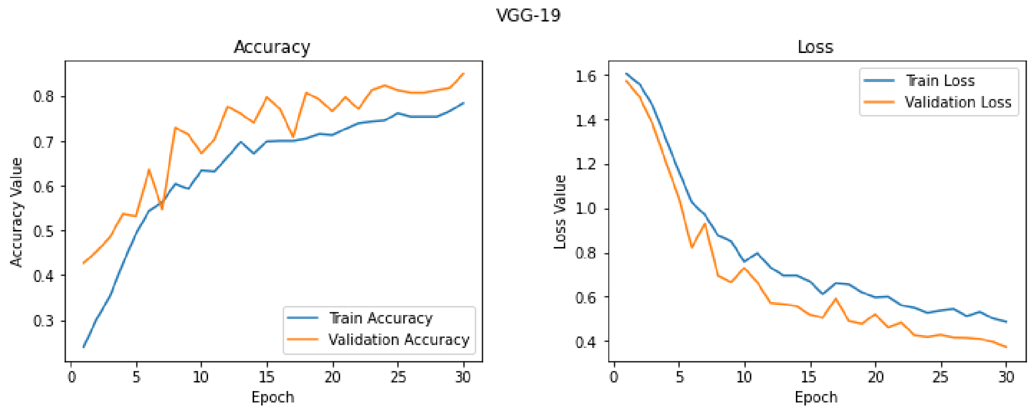
Iteration 1:



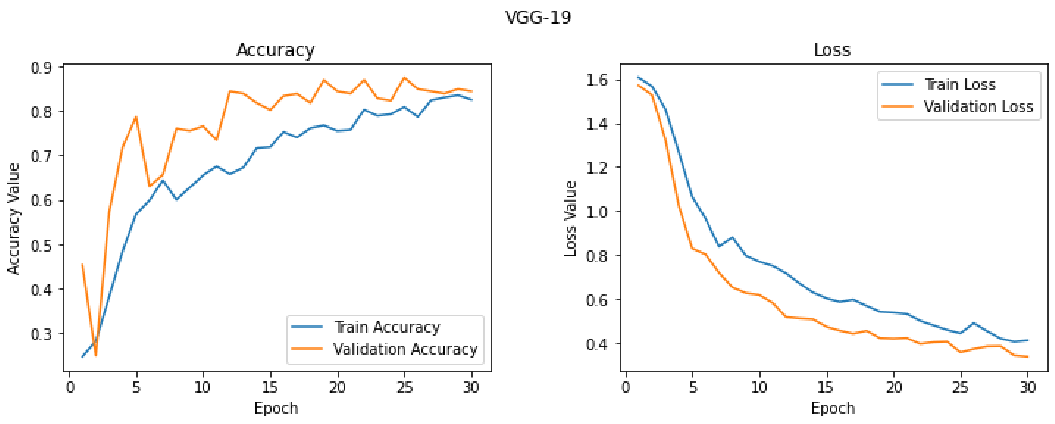
Iteration 2:



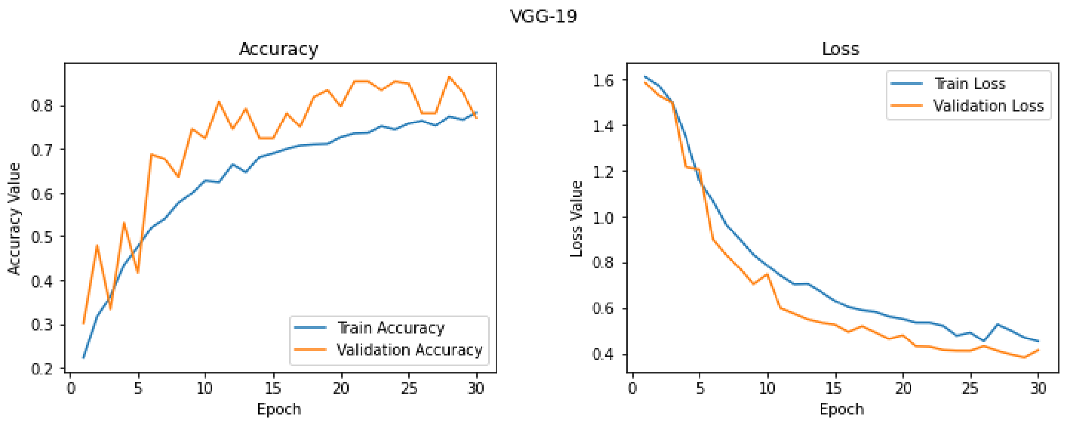
Iteration 3:



Iteration 4:



Iteration 5:



Some take-aways:

- Initially, while training the CNN model, we saw there was over-fitting due to a significant difference between the training accuracy and validation accuracy. However, after applying a dropout with 0.5 probability, we saw the validation accuracy started zigzagging but at the same time, it followed a similar accuracy with training. So the difference between training and validation accuracy was gone. Hence, it doesn't look like over-fitting now.
- I should have kept the learning rate lower to solve the vanishing gradient descent problem as observed earlier. Hence, also started batch by batch data processing instead using a python generator.
- I decided to not change the depth of the model and continued with this one since we didn't have an under-fitting problem. The primary objective here was just to find a good size neural network with an optimal learning rate to handle underfitting.
- Also, since the accuracy was low - Task 3 showed a significant improvement as expected because fine-tuning works well even with a very small dataset (as provided).

Task 4: Tried data augmentation

Overfitting occurs when the model “memorizes” the data instead of “generalizing” and we can understand it by examining train and validation accuracy where the training accuracy is much better than validation accuracy. Data Augmentation, Regularization, and decreasing the depth of the model are some of the possible solutions. Hence, **augmented the data** with few possible combinations including rotation, width shifting, height shifting, shearing, zooming and horizontal flipping.

Due to time-constraint, I couldn't work much on this task. However, I really wanted to try out some state-of-the-art pre-trained models for image classification including ResNet50, Inceptionv3 and EfficientNet.