Fashion-MNIST Project Report

Project Overview

The Fashion-MNIST clothing classification problem is a new standard dataset used in computer vision and deep learning.

Problem statement

The objective is to identify (predict) different fashion products from the given images using 3 different transfered learning models (LeNet 5, VGG19, AlexNet) with some preprocessing for each model of them.

The target dataset has 10 class labels, as we can see from above (0 – T-shirt/top, 1 – Trouser,,….9 – Ankle Boot).

Given the images of the articles, we need to classify them into one of these classes, hence, it is essentially a **'Multi-class Classification'** problem.

We will be using CNN to come up with a model for this problem and will use "**Accuracy**" as the performance measure.

Data description

The ***Fashion-MNIST dataset*** consists of images of that originate from Zalando's image directory. Zalando is a European e-commerce company founded in 2008.
The researchers in Zalando have created the Fashion-MNIST dataset that contains 70,000 images of clothing. More specifically, it contains 60,000 training examples and 10,000 testing examples, that are all grayscale images with the dimension 28 x 28 categorized into 10 classes.

The dataset is proposed as a more challenging replacement dataset for the MNIST dataset.

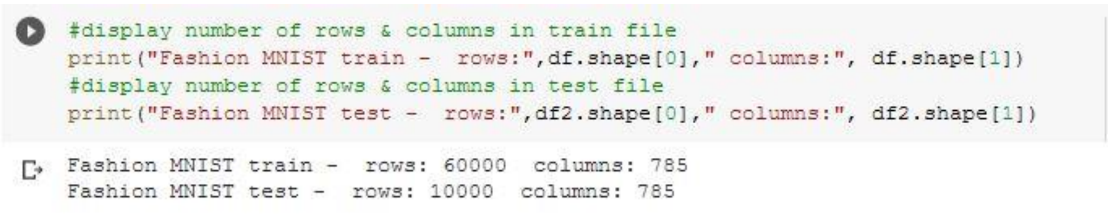
It is a dataset comprised of 60,000 small square 28×28 pixel grayscale images of items of 10 types of clothing, such as shoes, t-shirts, dresses, and more. The mapping of all 0-9 integers to class labels is listed below.

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

It is a more challenging classification problem than MNIST and top results are achieved by deep learning convolutional neural networks with a classification accuracy of about 90% to 95% on the hold out test dataset.

Data Exploration

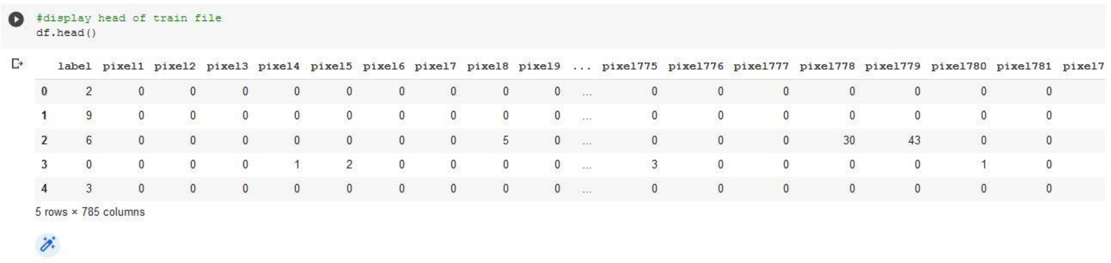
Shape of dataset:



```
#display number of rows & columns in train file
print("Fashion MNIST train -  rows:",df.shape[0]," columns:", df.shape[1])
#display number of rows & columns in test file
print("Fashion MNIST test -  rows:",df2.shape[0]," columns:", df2.shape[1])
```

```
Fashion MNIST train -  rows: 60000  columns: 785
Fashion MNIST test -  rows: 10000  columns: 785
```

Look at sample of our dataset:

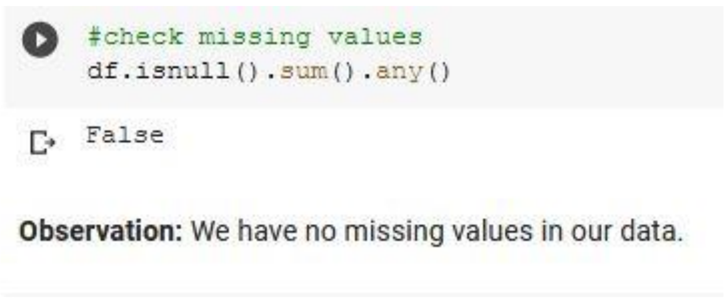


```
#display head of train file
df.head()
```

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixel7
0	2	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
1	9	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
2	6	0	0	0	0	0	0	0	5	0	...	0	0	0	30	43	0	0	
3	0	0	0	0	1	2	0	0	0	0	...	3	0	0	0	0	1	0	
4	3	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	

5 rows × 785 columns

Exploring missing values:

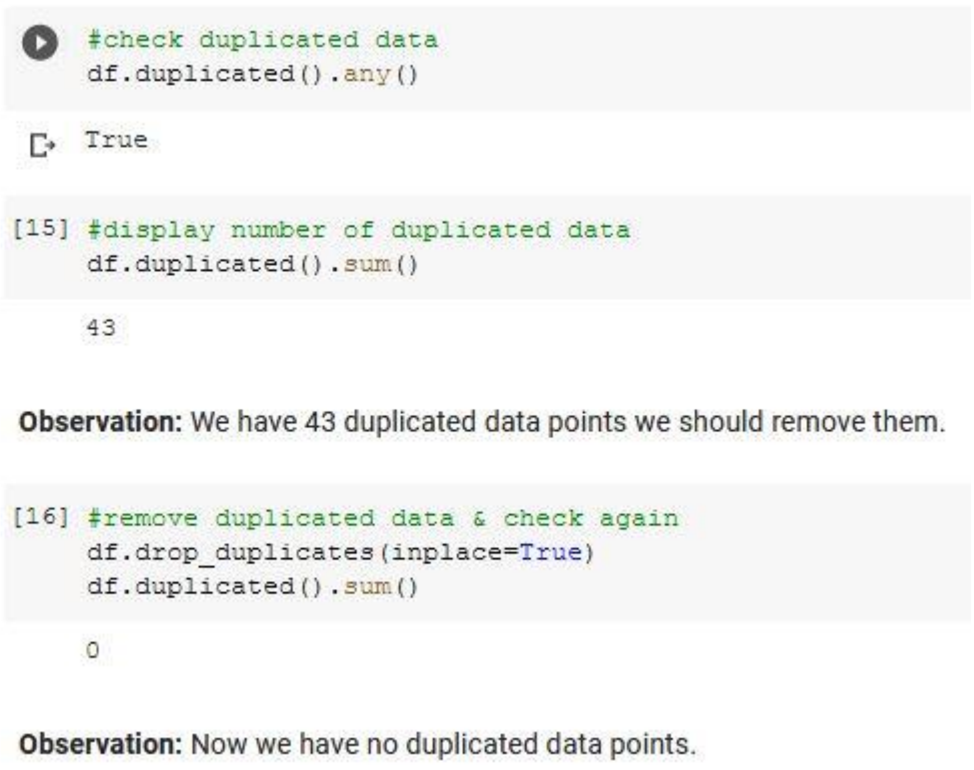


```
#check missing values
df.isnull().sum().any()
```

```
False
```

Observation: We have no missing values in our data.

Exploring duplicated data:



```
#check duplicated data
df.duplicated().any()
```

```
True
```

```
[15] #display number of duplicated data
     df.duplicated().sum()
```

```
43
```

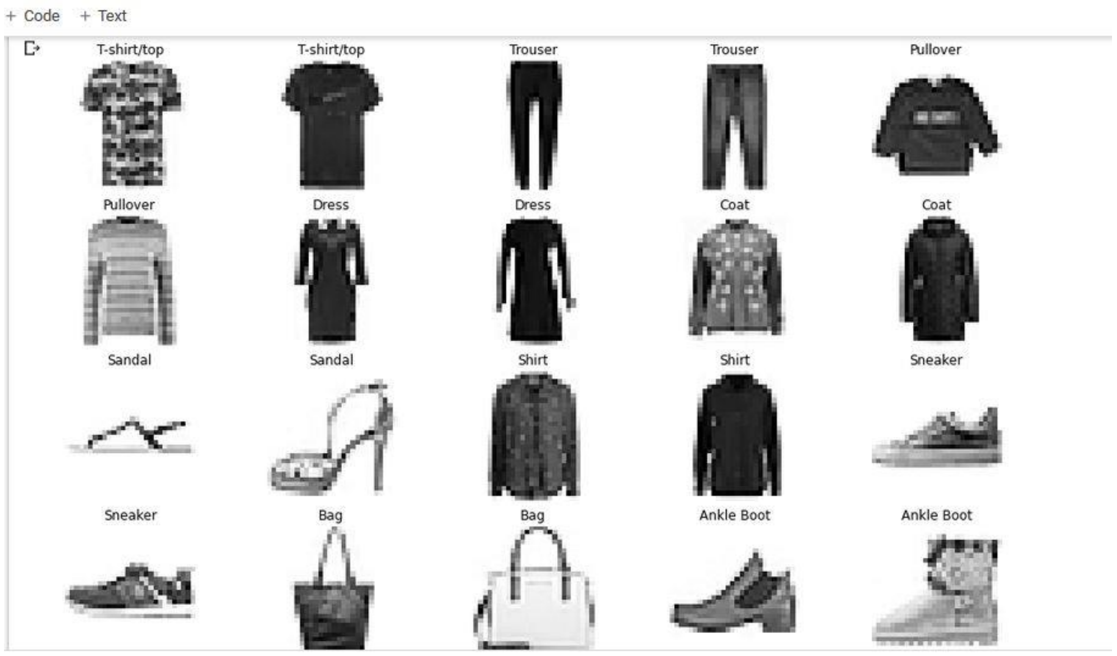
Observation: We have 43 duplicated data points we should remove them.

```
[16] #remove duplicated data & check again
     df.drop_duplicates(inplace=True)
     df.duplicated().sum()
```

```
0
```

Observation: Now we have no duplicated data points.

Some train Examples:



Preprocessing

For the 3 models we will:

- Split the data to features (image pixels) and labels (type of clothing) with range from 0:9.

- Change features data type from int to float.

- Normalize the features by dividing each image by 255.

    Each image (instance) in the dataset has 784 pixels (features) and value of each feature(pixel) ranges from 0 to 255, this range is too wide, hence we have performed Normalization on the training and test dataset, by dividing the pixels by 255, so that values of all features (pixels) are in a small range (0 to 1).

- Change labels data type from int to categorical.

For LeNet 5:

- We will convert the image shape to (n, 28, 28, 1).

For VGG19:

- We will convert the image shape to (n, 48, 48, 3).

- Apply the preprocess_input function is **to adequate the image to the format the model requires**.

For AlexNet:

- We will convert the image shape to (n, 28, 28, 1).

Then we will compiles our models with different optimizers then fit them to our training data.

Model Evaluation

- We used K-Fold Cross-validation which is **a resampling procedure used to evaluate machine learning models on a limited data sample**. With a number of groups that a given data sample is to be split into is **5 K-folds**.

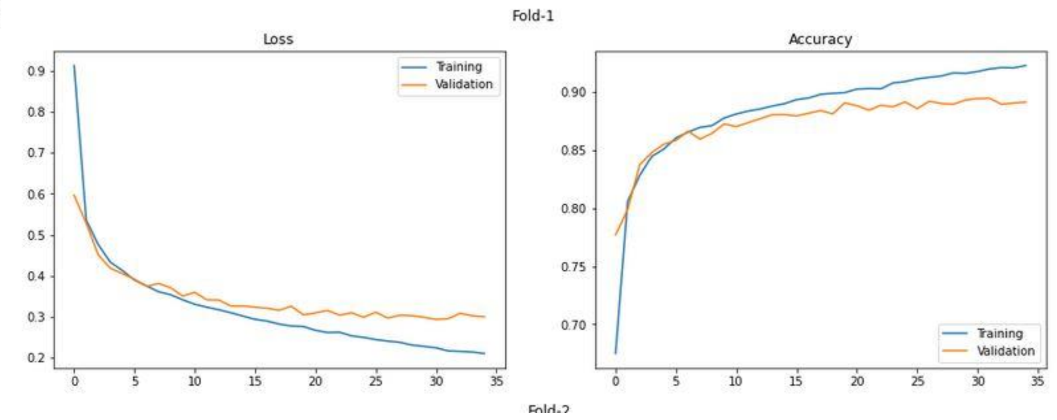
- Also we provided plot of accuracy improvement using the previously mentioned techniques for each model.

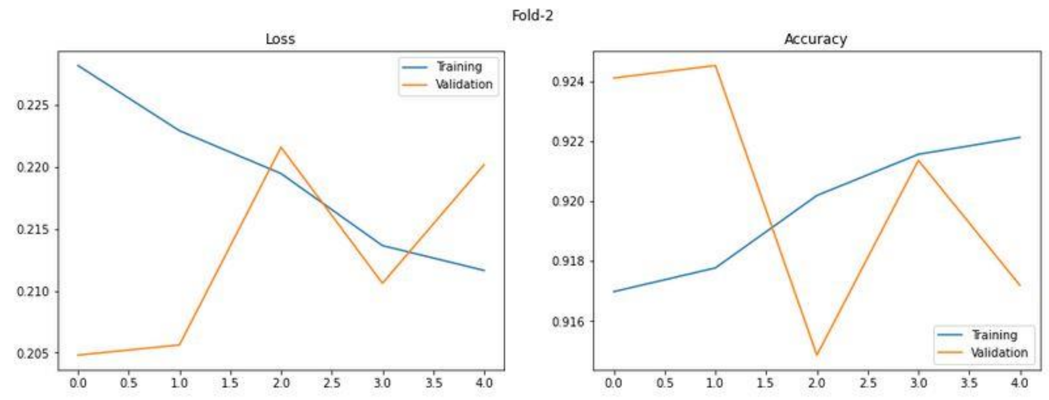
- Also plot for the convergence curve for each model.

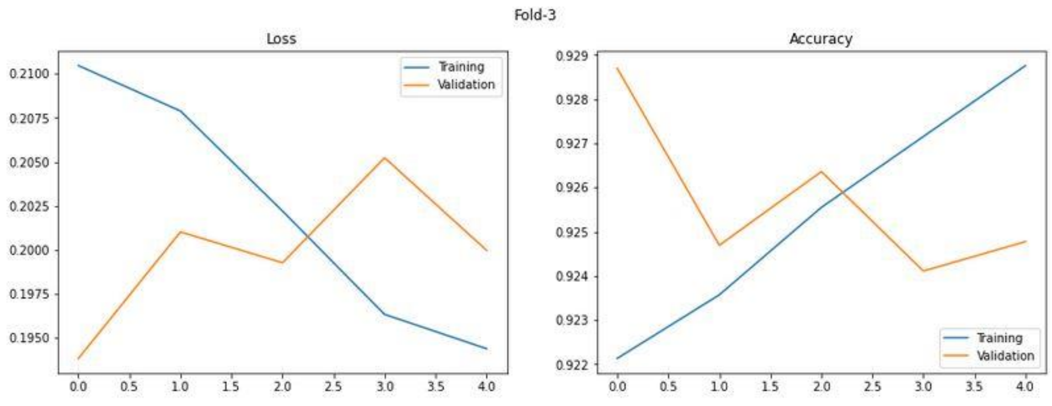
LeNet Model

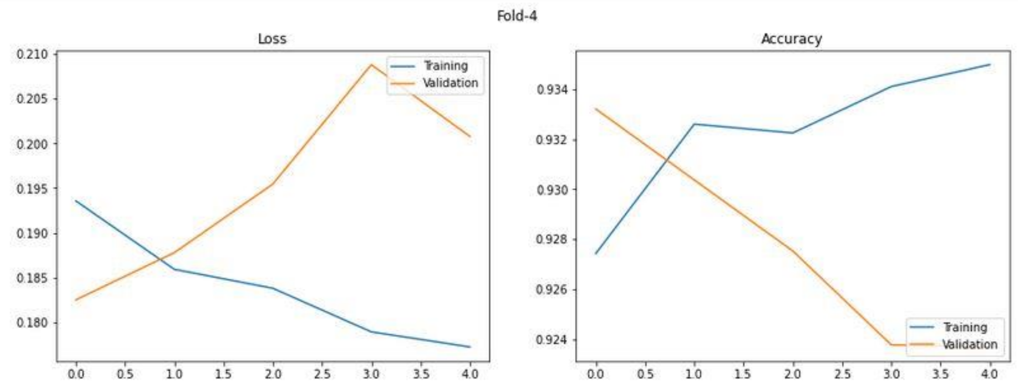
Visualizing loss and accuracy graphs from every fold.

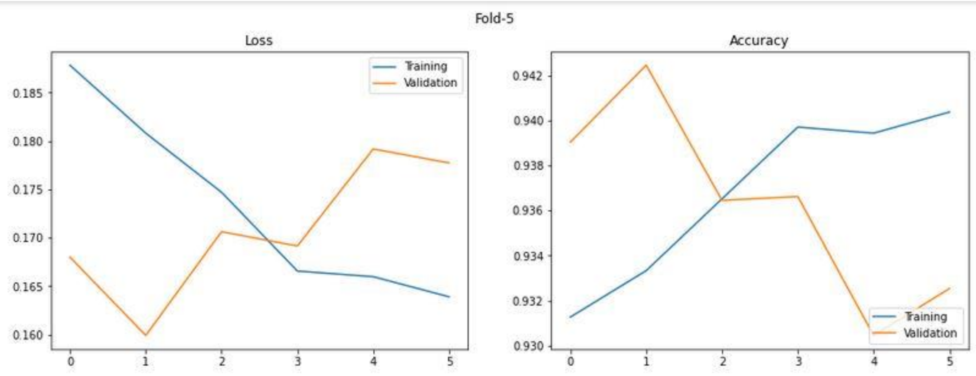
```python
# Displaying the graph results
for history in histories:
    display_kfold_result(history, (histories.index(history)+1))
```



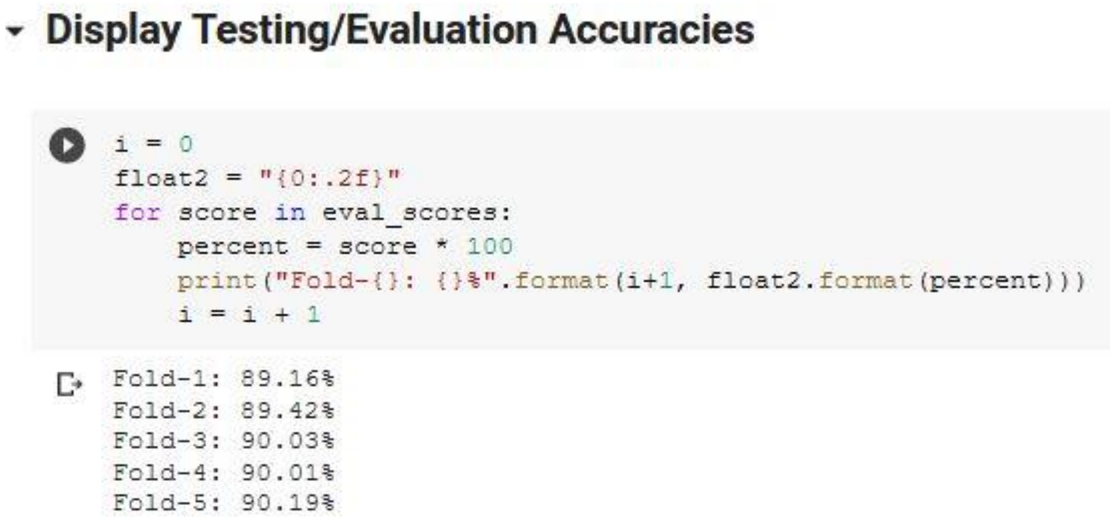








The train & validation accuracy range from 90:95% which is acceptable accuracy regardless the benchmark for this dataset.



Display Testing/Evaluation Accuracies

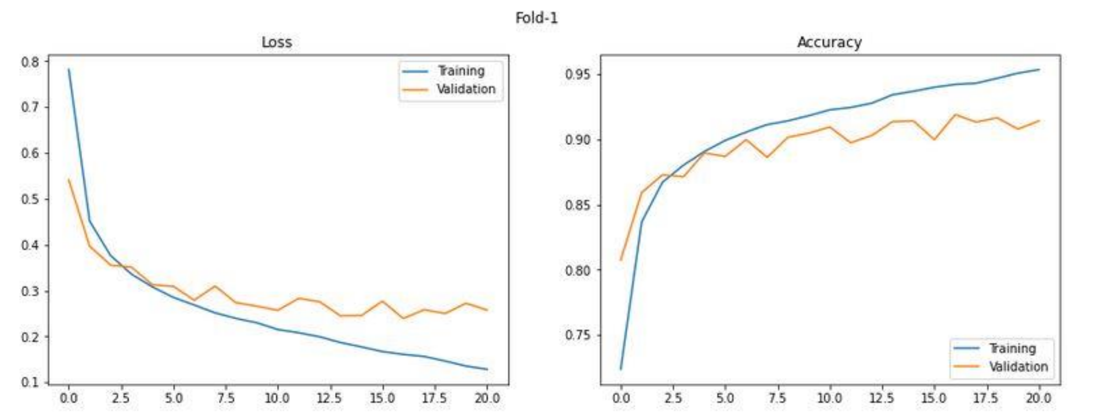
```
i = 0
float2 = "{0:.2f}"
for score in eval_scores:
    percent = score * 100
    print("Fold-{}: {}%".format(i+1, float2.format(percent)))
    i = i + 1
```

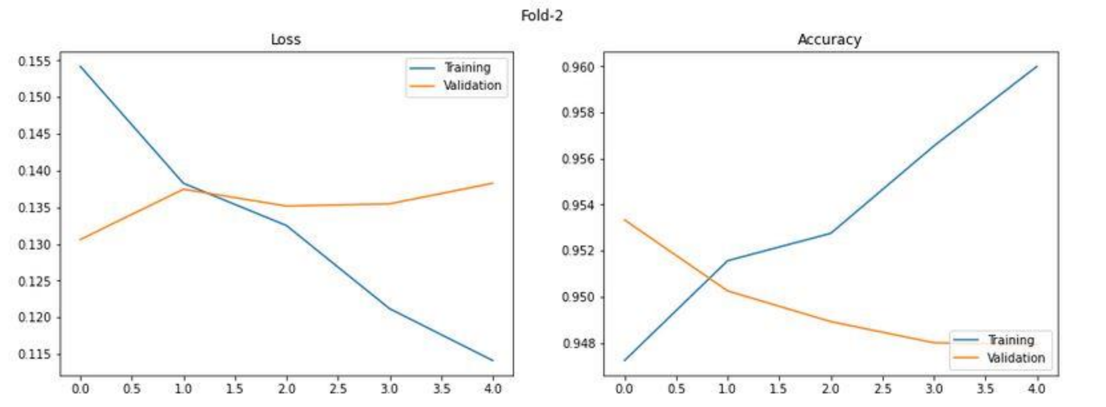
```
Fold-1: 89.16%
Fold-2: 89.42%
Fold-3: 90.03%
Fold-4: 90.01%
Fold-5: 90.19%
```

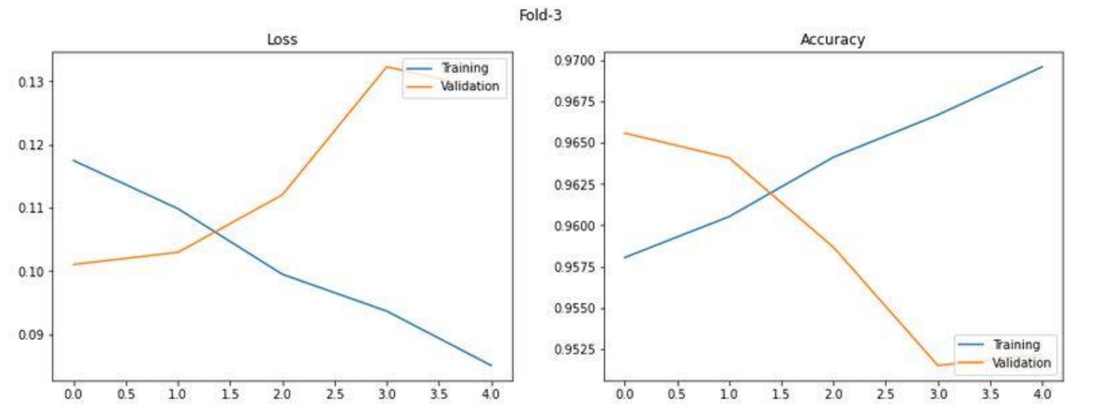
Observation: The best model instance was from fold-5.

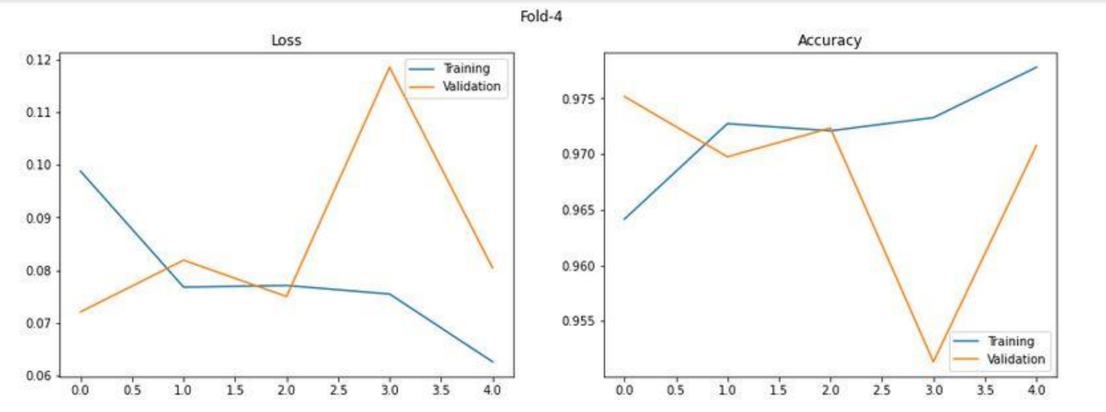
LeNet Model after adjustments

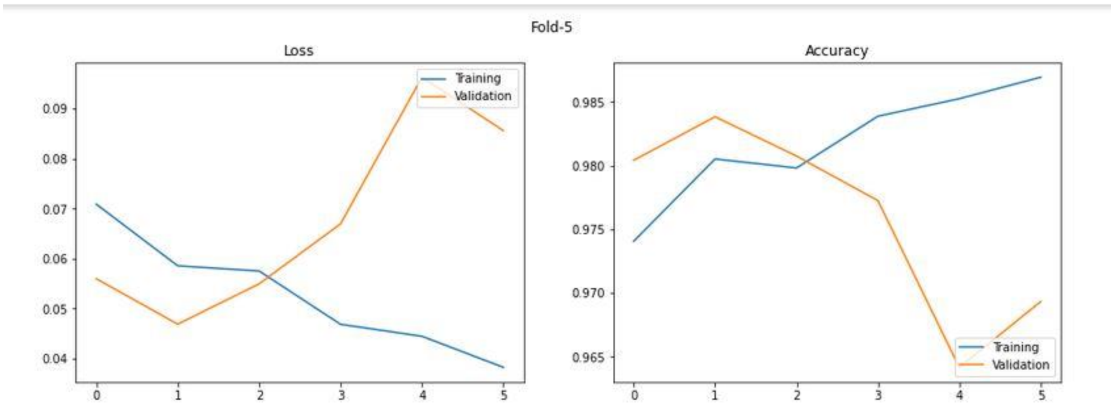
Visualizing loss and accuracy graphs from every fold.


Fold-1
Loss
Training
Validation
Accuracy
Training
Validation

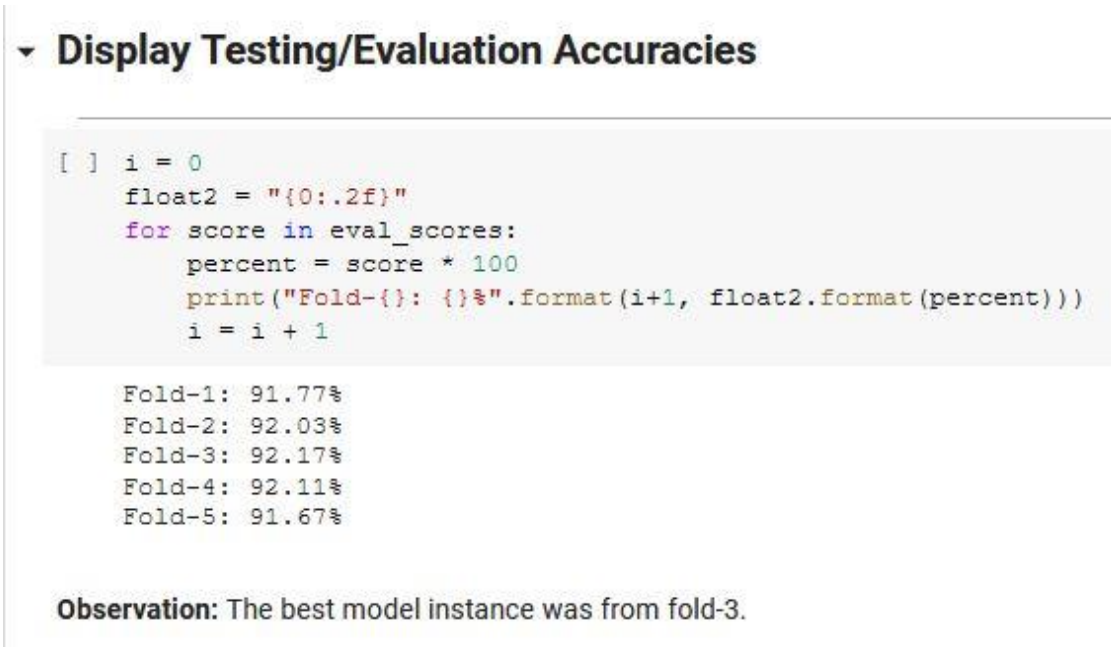

Fold-2
Loss
Training
Validation
Accuracy
Training
Validation


Fold-3
Loss
Training
Validation
Accuracy
Training
Validation


Fold-4
Loss
Training
Validation
Accuracy
Training
Validation



The train & validation accuracy range from 90:98% which is much better than the pure LeNet-5 structure.



Display Testing/Evaluation Accuracies

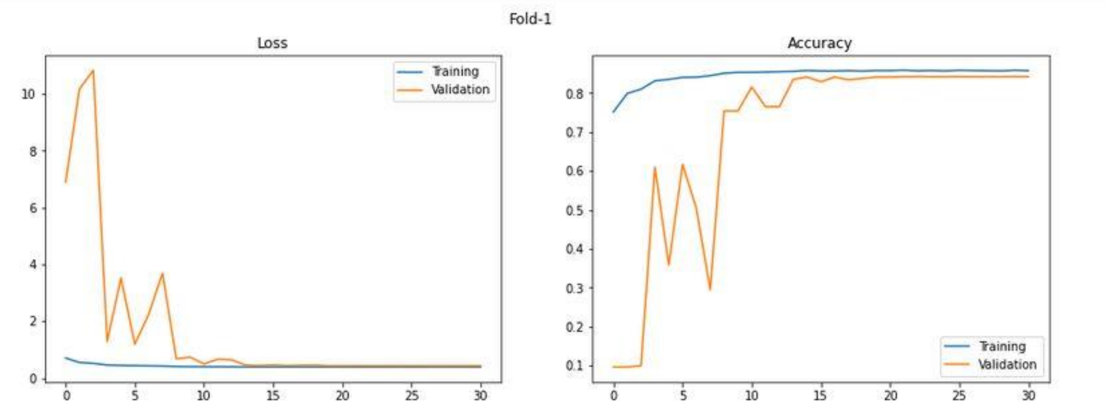
```
i = 0
float2 = "{0:.2f}"
for score in eval_scores:
    percent = score * 100
    print("Fold-{}: {}%".format(i+1, float2.format(percent)))
    i = i + 1
```

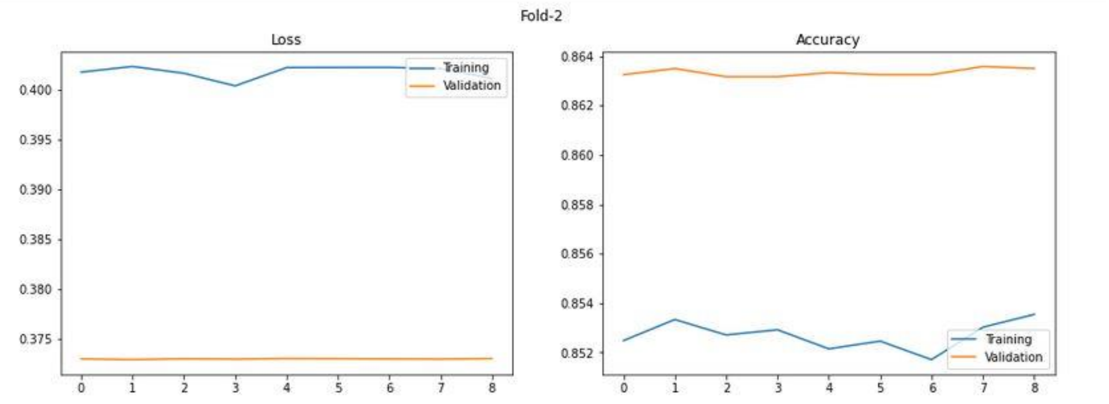
```
Fold-1: 91.77%
Fold-2: 92.03%
Fold-3: 92.17%
Fold-4: 92.11%
Fold-5: 91.67%
```

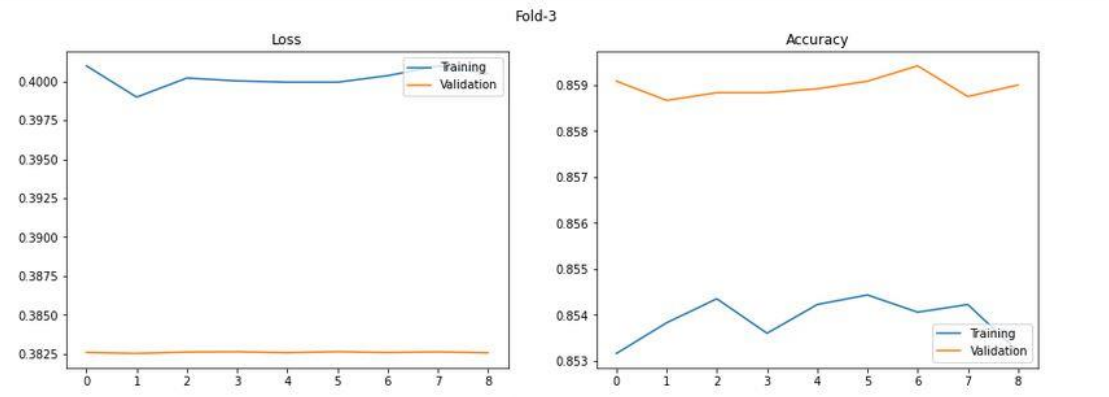
Observation: The best model instance was from fold-3.

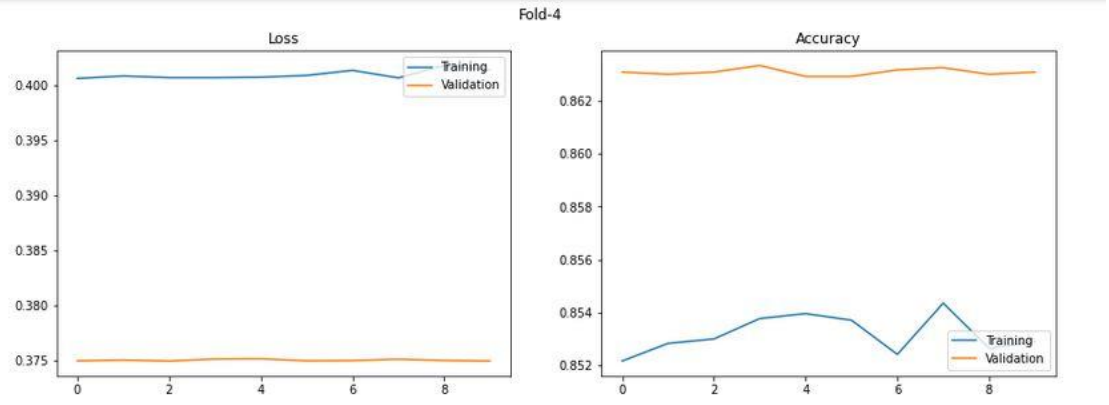
VGG19 Model

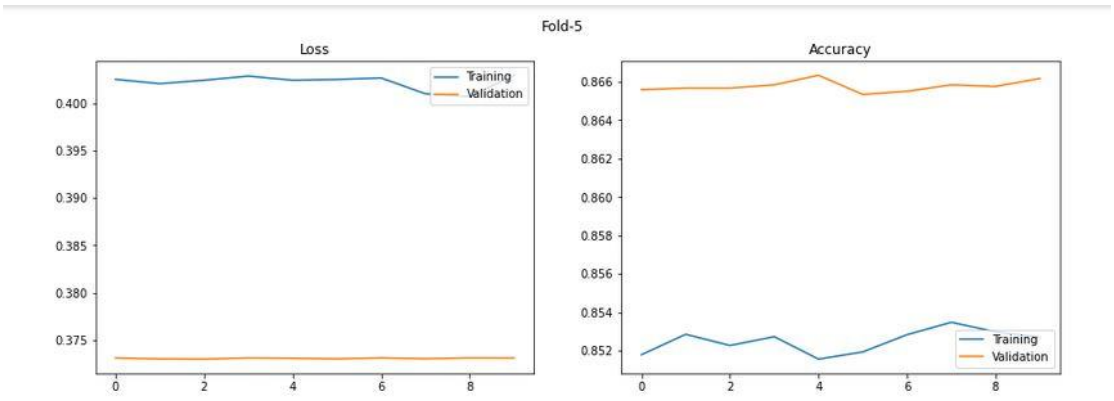
Visualizing loss and accuracy graphs from every fold.


Fold-1
Loss
Training
Validation
Accuracy
Training
Validation

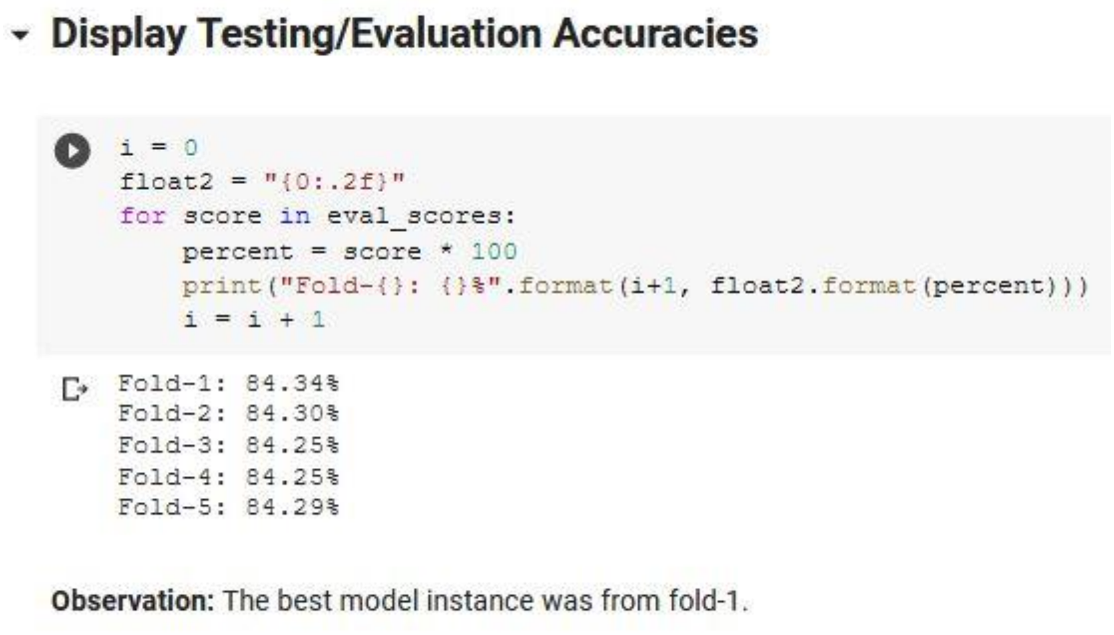

Fold-2
Loss
Training
Validation
Accuracy
Training
Validation


Fold-3
Loss
Training
Validation
Accuracy
Training
Validation


Fold-4
Loss
Training
Validation
Accuracy
Training
Validation



The train & validation accuracy range from 84:85% which indicated that LeNet is better than VGG19 for this dataset.



Display Testing/Evaluation Accuracies

```
i = 0
float2 = "{0:.2f}"
for score in eval_scores:
    percent = score * 100
    print("Fold-{}: {}%".format(i+1, float2.format(percent)))
    i = i + 1
```

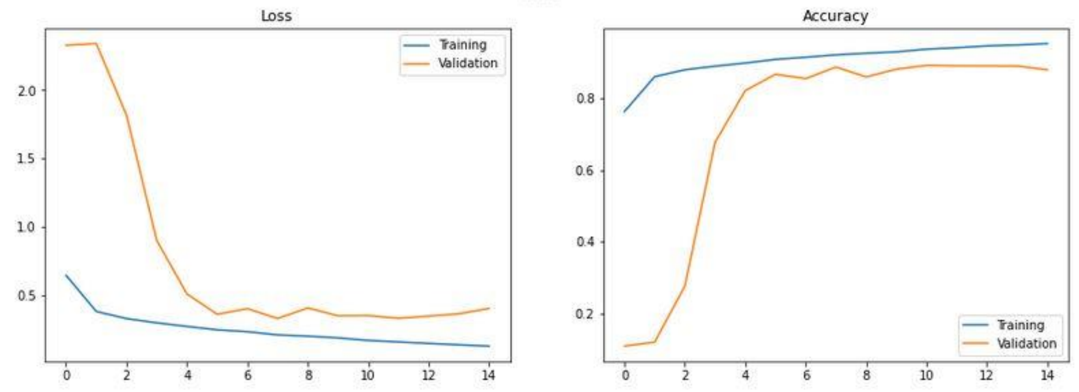
```
Fold-1: 84.34%
Fold-2: 84.30%
Fold-3: 84.25%
Fold-4: 84.25%
Fold-5: 84.29%
```

Observation: The best model instance was from fold-1.

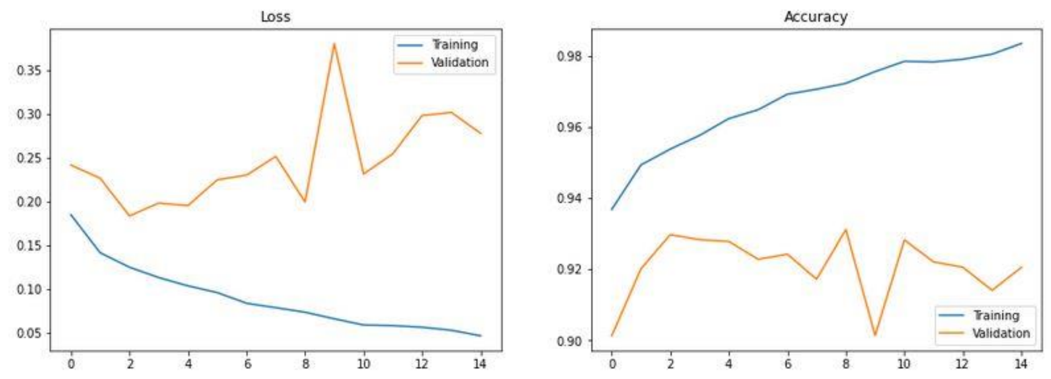
AlexNet Model

Visualizing loss and accuracy graphs from every fold.

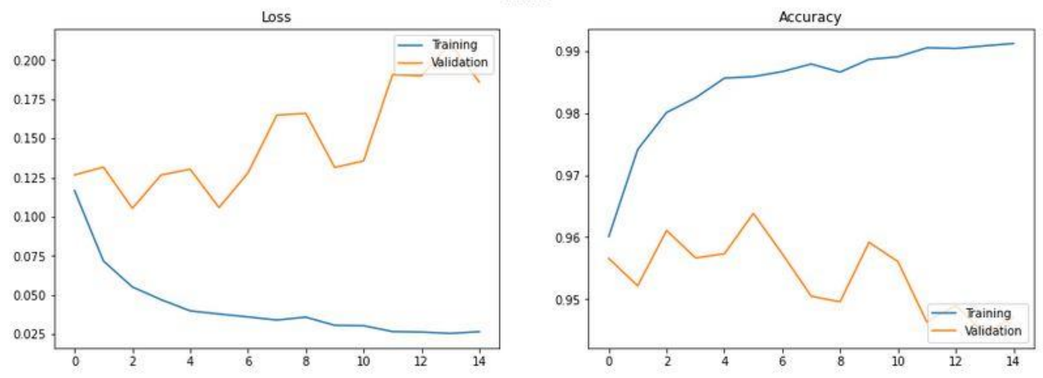
Fold-1



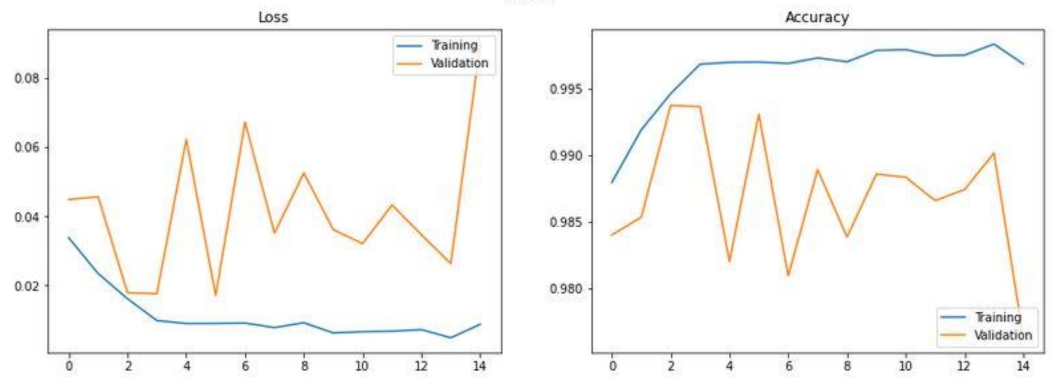
Fold-2



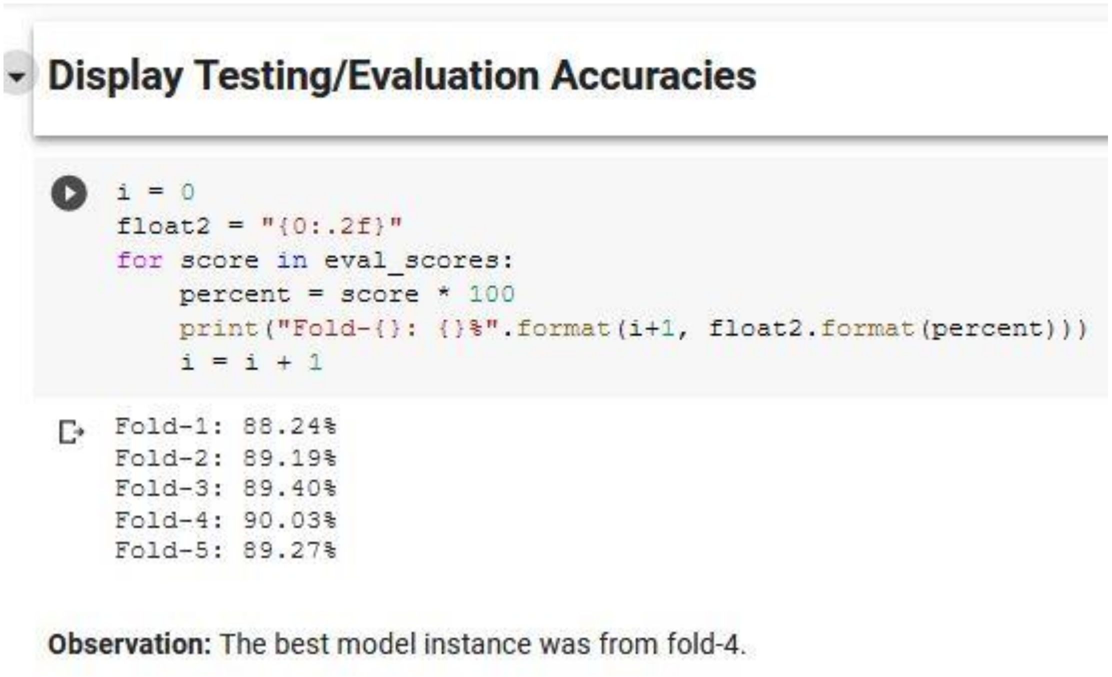
Fold-3



Fold-5



The train & validation accuracy range from 89:98% which is good as LeNet-5 modified structure but not good at test data as it.



Display Testing/Evaluation Accuracies

```
i = 0
float2 = "{0:.2f}"
for score in eval_scores:
    percent = score * 100
    print("Fold-{}: {}%".format(i+1, float2.format(percent)))
    i = i + 1
```

```
Fold-1: 88.24%
Fold-2: 89.19%
Fold-3: 89.40%
Fold-4: 90.03%
Fold-5: 89.27%
```

Observation: The best model instance was from fold-4.

Conclusion

- We compared different optimizers used in training neural networks and gained intuition for how they work. We found that **SGD with Nesterov Momentum and Adam** produce the best results when training different transfer learning models on Fashion MNIST data in TensorFlow.

- K-Fold Cross Optimization is a good way for evaluating the model in our case and we used 5 groups to split the data into.

- LeNet modified model was the best model among the four models we tested with the best accuracy 92.17% on test data confirming that model is fine **with no overfitting**. And training & validation accuracies 96% & 95% respectively. That returned to that it was the best suitable architecture used to classify MNIST dataset.