NAME: OMAR SAMEH SAID

ID: 20399132

Project 2 Report

Part I: Data Preparation

• Describe the data

Fashion-MNIST is a dataset of Zalando's article images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. It is split in 10,000 as test and 50,000 as train datasets

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

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

```
In [4]: (x_train, Y_train), (x_test, y_test) = fashion_mnist.load_data()
```

```
In [5]: print("Fashion MNIST train shape:",x_train.shape," Label Shape Train : ",Y_train.shape)
        print("Fashion MNIST test shape:",x_test.shape," Label Shape Train : ",y_test.shape)

        Fashion MNIST train shape: (60000, 28, 28)  Label Shape Train :  (60000,)
        Fashion MNIST test shape: (10000, 28, 28)  Label Shape Train :  (10000,)
```

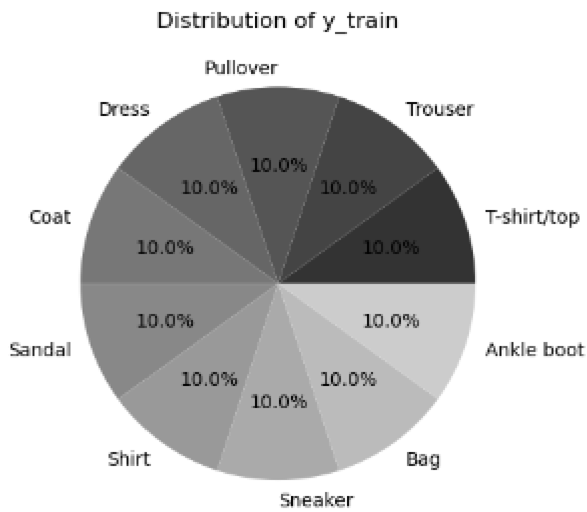
```
In [6]: np.unique(Y_train)

Out[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

• Check the data for missing values or duplicates

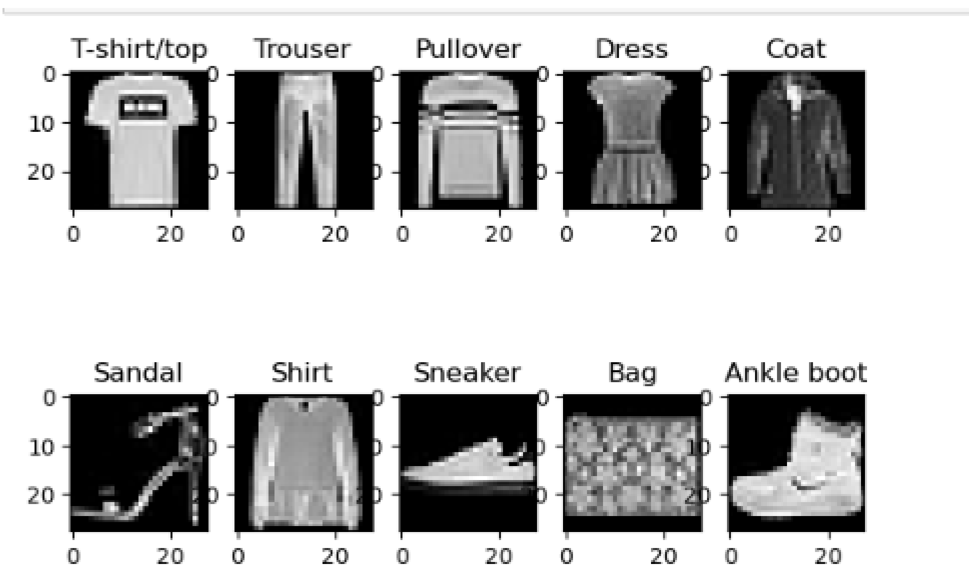
```
In [10]: x_train_df = pd.DataFrame(x_train.reshape(-1, 28*28))

         # Check for null values
         x_train_df.isnull().sum().sum()

Out[10]: 0
```

```
In [11]: x_train_df.duplicated().sum()

Out[11]: 0
```

- Visualize the data



- Draw some of the images



- Correlation (sample)

In [12]:
```
corr = x_train_df.corr()
corr.style.background_gradient(cmap="RdBu_r")
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	1.000000	0.323753	0.104725	0.045164	0.025112	0.013816	0.011129	0.012302	0.009424	0.000213	-0.001209	-0.001018	-0.002669	-0.003623	-0.00
1	0.323753	1.000000	0.562747	0.059439	0.070743	0.043161	0.027905	0.019830	0.019246	0.011690	0.001439	-0.005628	-0.009426	-0.010132	-0.01
2	0.104725	0.562747	1.000000	0.342158	0.152222	0.109535	0.074838	0.046004	0.040848	0.032191	0.021744	0.015129	0.002278	-0.003185	-0.00
3	0.045164	0.059439	0.342158	1.000000	0.615420	0.346866	0.249687	0.143525	0.087865	0.051165	0.024056	0.014962	0.007956	0.005426	0.00
4	0.025112	0.070743	0.152222	0.615420	1.000000	0.701805	0.417610	0.226949	0.133628	0.075431	0.033712	0.018272	0.013388	0.014077	0.01
5	0.013816	0.043161	0.109535	0.346866	0.701805	1.000000	0.656006	0.331723	0.187708	0.099852	0.047977	0.029858	0.021817	0.022242	0.02
6	0.011129	0.027905	0.074838	0.249687	0.417610	0.656006	1.000000	0.631489	0.321444	0.159015	0.058546	0.028897	0.018291	0.017348	0.01
7	0.012302	0.019830	0.046004	0.143525	0.226949	0.331723	0.631489	1.000000	0.661779	0.312727	0.105007	0.028479	0.004578	0.002513	0.00
8	0.009424	0.019246	0.040848	0.087865	0.133628	0.187708	0.321444	0.661779	1.000000	0.624614	0.221676	0.073262	0.012015	0.003846	0.00
9	0.000213	0.011690	0.032191	0.051165	0.075431	0.099852	0.159015	0.312727	0.624614	1.000000	0.584826	0.267237	0.129727	0.094782	0.08
10	-0.001209	0.001439	0.021744	0.024056	0.033712	0.047977	0.058546	0.105007	0.221676	0.584826	1.000000	0.703232	0.401258	0.331035	0.30
11	-0.001018	-0.005628	0.015129	0.014962	0.018272	0.029858	0.028897	0.028479	0.073262	0.267237	0.703232	1.000000	0.722325	0.569628	0.54
12	-0.002669	-0.009426	0.002278	0.007956	0.013388	0.021817	0.018291	0.004578	0.012015	0.129727	0.401258	0.722325	1.000000	0.878077	0.82
13	-0.003623	-0.010132	-0.003185	0.005426	0.014077	0.022242	0.017348	0.002513	0.003846	0.094782	0.331035	0.569628	0.878077	1.000000	0.94
14	-0.003647	-0.010131	-0.003130	0.005951	0.015063	0.022721	0.017968	0.002915	0.001666	0.089490	0.306943	0.541607	0.824611	0.948868	1.00
15	-0.003989	-0.010245	-0.001958	0.004915	0.008767	0.017167	0.013356	-0.001333	0.004603	0.111022	0.353151	0.607007	0.865679	0.903658	0.91
16	-0.002659	-0.006742	0.008050	0.009829	0.010823	0.022105	0.019709	0.013011	0.045832	0.213408	0.537737	0.758823	0.776620	0.706848	0.67

- The labels are already encoded

```
np.unique(Y_train)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8)
```

Training a CNN neural network

- implement a LeNet-5 and Modify hyperparameters

The famous LeNet-5 architecture had the following layers:

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

ref : hands on machine learning 3rd edition chapter 14 LeNet-5 architecture

I used Keras tuner to choose the best hyperparameters.

Tuned hyperparameters:

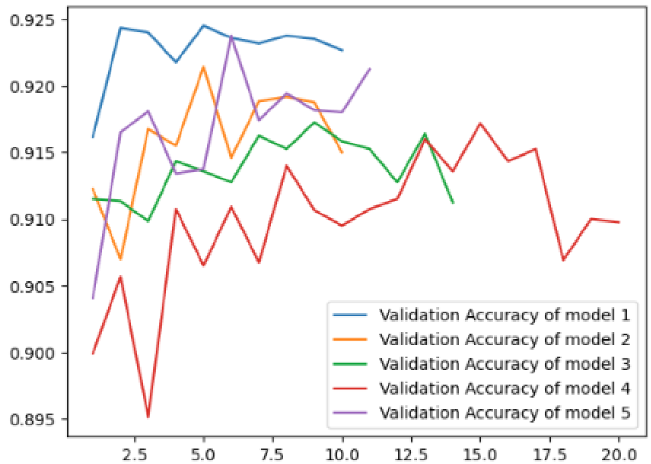
- Activation Function
- learning rate
- number of filters
- padding
- number of units in each hidden layer

Out[22]:

	Learning rate	activation function first convo	activation function second convo	best activation function first hidden	activation function second hidden	number of filters first convo	number of filters second convo	padding value first convo	padding value second convo	number of neurons first hidden	number of neurons second hidden
0	0.001	relu	relu	tanh	tanh	16	128	valid	same	448	192
1	0.001	tanh	relu	tanh	relu	8	64	same	same	128	320
2	0.001	tanh	relu	relu	sigmoid	128	16	valid	same	224	64
3	0.001	relu	tanh	sigmoid	tanh	32	64	valid	same	128	384

After getting the best hyperparameters Then train and evaluate their performance

In [25]:
```
# here we plot the first fifth models for compare Validation Accuracy
for i in range (5):
  plt.plot(np.arange(1,len(list_Val_accuracy[i])+1), list_Val_accuracy[i], label = "Validation Accuracy of model "+str(i+1))
plt.legend()
plt.show()
```



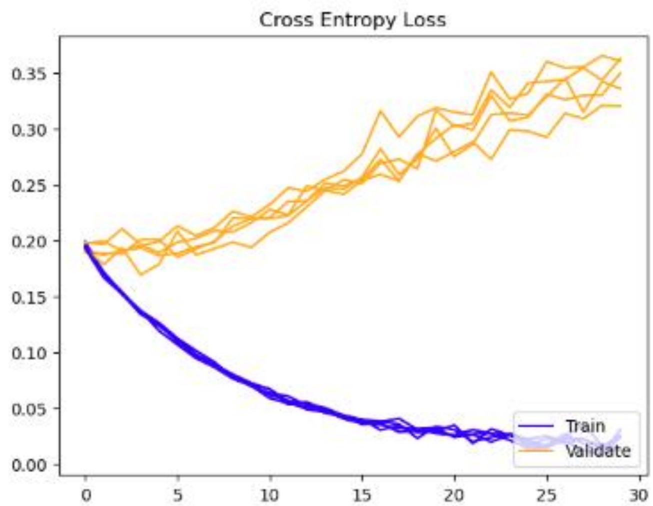
In [26]:
```
# Evaluate the best model.
loss, accuracy = best_model[0].evaluate(X_Test, y_test)
print("Accuracy of Testing : ",accuracy)
print("Loss Of Testing : ",loss)
```

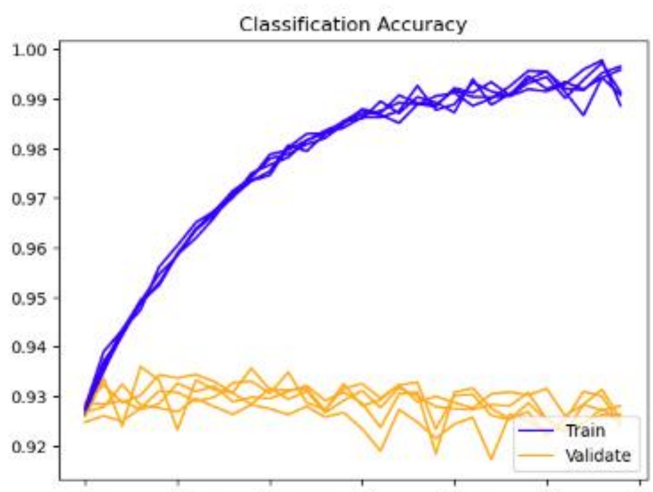
```
313/313 [==============================] - 1s 2ms/step - loss: 0.2586 - accuracy: 0.9214
Accuracy of Testing :  0.9214000105857849
Loss Of Testing :  0.2585798501968384
```

Evaluate the model using 5-fold cross-validation.

```
In [33]: scor

Out[33]: [0.9240000247955322,
          0.9279999732971191,
          0.9253333210945129,
          0.9262499809265137,
          0.9245833158493042]
```

```
In [34]: #  Learning curves
         for i in range(len(his)):
             # plot loss
             plt.title('Cross Entropy Loss')
             plt.plot(his[i].history['loss'], color='blue', label='train')
             plt.plot(his[i].history['val_loss'], color='orange', label='validate')
         plt.legend(["Train", "Validate"], loc ="lower right")
         plt.show()

         for i in range(len(his)):

             # plot accuracy
             plt.title('Classification Accuracy')
             plt.plot(his[i].history['accuracy'], color='blue', label='train')
             plt.plot(his[i].history['val_accuracy'], color='orange', label='validate')
         plt.legend(["Train", "Validate"], loc ="lower right")
         plt.show() # For Accuracy
```





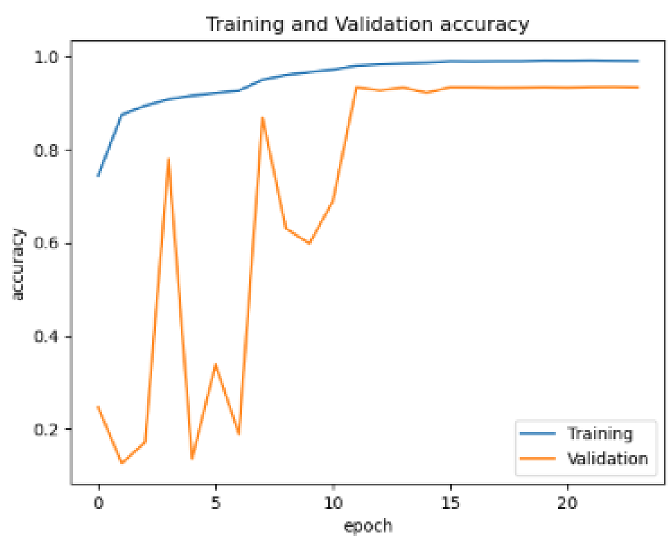
- It tends to overfit but it still has good accuracy on test set at the end LeNet-5 further improved the accuracy to **0.921** in small amount of time

using other two CNN models

EfficientNetV2B3 (slightly better than LeNet-5 acc: 0.925)


```
In [6]: history.history.keys()
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
```

```
Out[6]: Text(0, 0.5, 'accuracy')
```

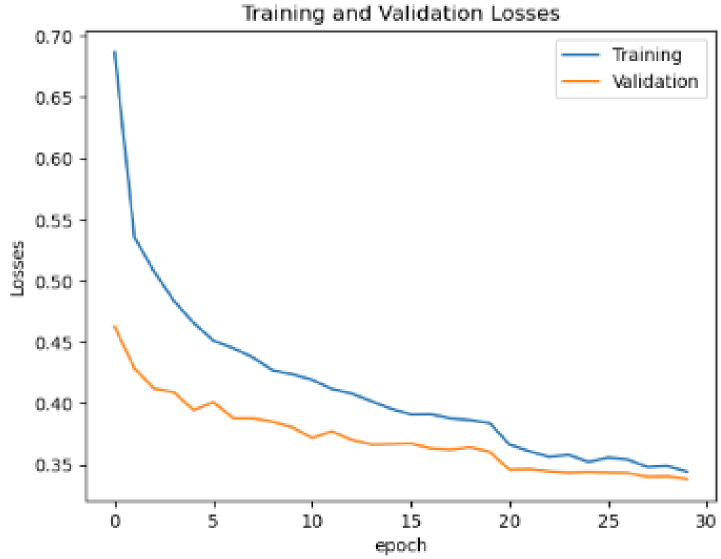



```
In [11]: # Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = model_final.evaluate(x_testE, y_test, batch_size=128)
print("test loss, test acc:", results)
```

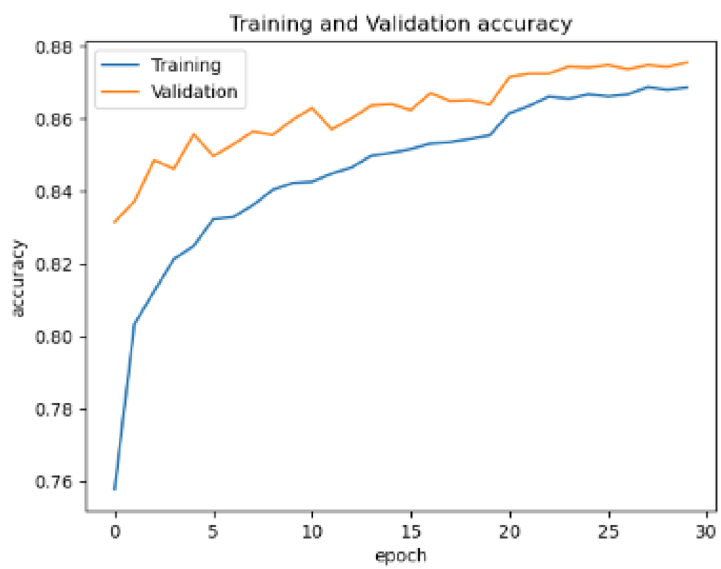
```
Evaluate on test data
79/79 [==============================] - 3s 26ms/step - loss: 0.3484 - accuracy: 0.9258
test loss, test acc: [0.3483932912349701, 0.9258000254631042]
```

DenseNet201 (much better than LeNet-5 acc: **0.931**)

First try with freezing (acc: 0.873)




```
In [41]: history.history.keys()
         import matplotlib.pyplot as plt
         %matplotlib inline
         plt.plot(history.history['accuracy'])
         plt.plot(history.history['val_accuracy'])
         plt.legend(['Training', 'Validation'])
         plt.title('Training and Validation accuracy')
         plt.xlabel('epoch')
         plt.ylabel('accuracy')

Out[41]: Text(0, 0.5, 'accuracy')
```

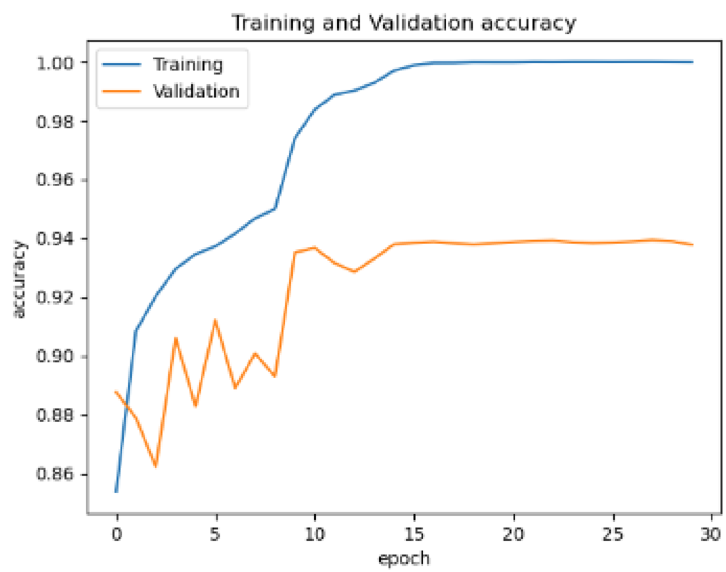



```
In [42]: # Evaluate the model on the test data using `evaluate`
         print("Evaluate on test data")
         results = DenseNet201_model.evaluate(x_testd, y_test, batch_size=128)
         print("test loss, test acc:", results)

         Evaluate on test data
         79/79 [==============================] - 2s 30ms/step - loss: 0.3501 - accuracy: 0.8730
         test loss, test acc: [0.3500940799713135, 0.8730000257492065]
```

Second try without freezing (acc: 0.931)

In [9]:
```
history.history.keys()
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training', 'Validation'])
plt.title('Training and Validation accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
```

Out[9]: Text(0, 0.5, 'accuracy')



In [10]:
```
# Evaluate the model on the test data using `evaluate`
print("Evaluate on test data")
results = DenseNet201_model.evaluate(x_testd, y_test, batch_size=128)
print("test loss, test acc:", results)
```

```
Evaluate on test data
79/79 [==============================] - 3s 39ms/step - loss: 0.4398 - accuracy: 0.9319
test loss, test acc: [0.43980199098587036, 0.9319000244140625]
```

Conclusion

Model	LeNet-5	EfficientNetV2B3	DenseNet201_model freezing	DenseNet201_model No freezing
Accuracy	0.921	0.925	0.873	0.931