DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Machine Learning and Data Science

ENCS5341

# Project – Fashion-MNIST

Team members:

- Mahmoud Qaisi - 1190831
- Osama Rihami - 1190560

Instructor: Dr. Yazan Abu Farha.

Section: 2

Date: 4/2/2023

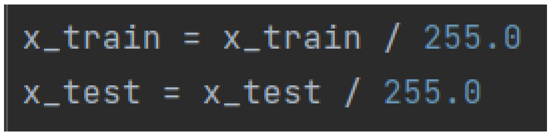
## Fashion-MNIST dataset:

The data set used in this project is called fashion-MINST. It contains a training set that is made up of 60000 28x28 greyscale images. There is also a testing set that contains 10000 images with the same features. The images in the dataset are classified using 10 labels that are referred to as numbers from 0 to 9.

The main task here is to train different learning model using this data set. It is required to observe how these different models predict the testing set and discuss the reasons behind its ability or inability to perform well.

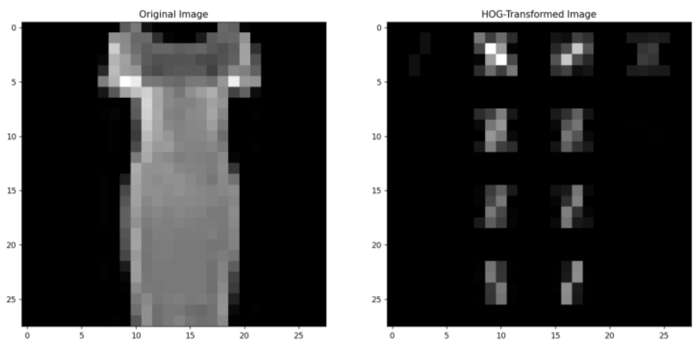
## Feature Extraction:

When importing the data set, a list containing each photo in the form a list of 28 lists that each contain 28 pixels of a row int the image. The first feature used will be the raw pixels of the image. The data was normalized using min-max transformation. The max value is 255 which makes the normalization process simply as follows:



```
x_train = x_train / 255.0
x_test = x_test / 255.0
```

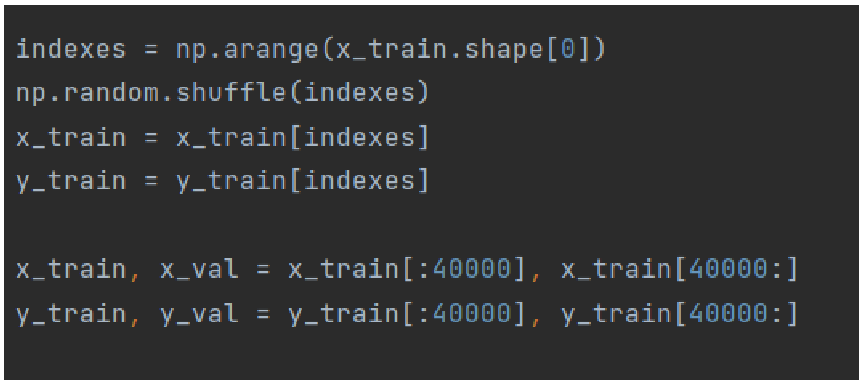
A few alternate features were considered to use in training the models. The chosen feature is the Histogram of Oriented Gradients. Histogram of Oriented Gradients (HOG) is a feature descriptor used in computer vision and image processing to represent the shape of an object. It is particularly useful for object detection and recognition tasks, such as pedestrian detection in images and videos.



The HOG feature descriptor works by dividing an image into small cells and computing the gradient orientation histograms for each cell. The gradient orientation histograms represent the distribution of edge directions in the cell and capture the object's shape information. The histograms for all the cells in the image are concatenated to form a feature vector that represents the entire object.

- ## Validation set:

In order to test out the most suitable hyper-parameters used in our models a validation set was created by splitting the training data into a 2/3 training set and 1/3 validation set to check the accuracy on different hyper parameters used. The training set was shuffled before splitting to make sure the data isn't being over-fitted.



```
indexes = np.arange(x_train.shape[0])
np.random.shuffle(indexes)
x_train = x_train[indexes]
y_train = y_train[indexes]

x_train, x_val = x_train[:40000], x_train[40000:]
y_train, y_val = y_train[:40000], y_train[40000:]
```

- ## Classification Models:

- ### The baseline model:

The baseline model used in this project is K-nearest neighbor. It classifies an element based on a majority vote of the closest k neighbors to it. The are two hyper parameters to be considered for this model which are the k (the number of neighbors being considered) and the distance measure being used.

- ### Random forest:

The random forest classification model is an ensemble method that uses multiple decision-trees generated by randomizing the data set and random node optimization. This model is built on the idea of ensemble learning, which is a method of combining multiple classifiers to solve difficult problems and enhance model performance. It's a popular machine learning algorithm, and it usually give good performance and its suitable for big datasets. The hyper parameter taken into consideration here is the number of estimators which is number of trees to be generated.

- ### Support machine vector:

Support machine vector is a linear classifier that tries to find the boundary with the largest margin between the classes, which means that the distance between the closest data points of each class and the boundary is maximized. These closest data points are known as support vectors, and they play a key role in determining the decision boundary. The hyper parameters taken into consideration are the kernel and the regularization parameter (C).

## • Technical Details:

In each model there exist a number of hyper parameters that need to be provided for the model to insure the best possible performance.

### • K- nearest neighbor:

Two hyper parameters to were considered for this model. K which is the number of neighbors taken into consideration for the majority vote. This parameter plays a huge role in overfitting or under fitting the training data. The second parameter is the distance measure used. The choice of distance measure depends on the characteristics of the data, such as the number of dimensions, the presence of outliers.

### • Random forest:

One hyper parameter was considered here which is the number of estimators. It represents the number trees to be generated it. Is chosen based on the complexity of the problem and the size of the training dataset. The more you increase it the higher the accuracy will be but the at the expense of higher cost in time, space and other resources.

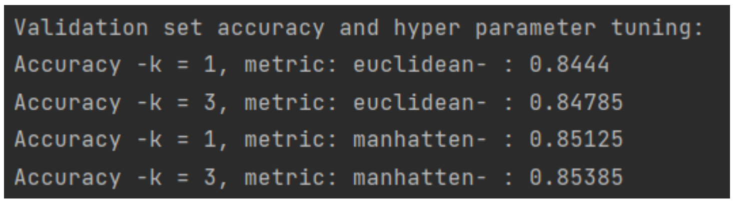
### • Support machine vector:

Two hyper-parameters were considered. The kernel is a function that maps the input data into a higher-dimensional space, where it becomes easier to find the best decision boundary. The regularization parameter (C) controls the trade-off between the model's ability to fit the training data well and its generalization ability to new data. A smaller value of C results in a larger margin, but may lead to more misclassifications on the training data. On the other hand, a larger value of C results in a smaller margin and fewer misclassifications on the training data, but may lead to overfitting.

## • Experiments and Results:

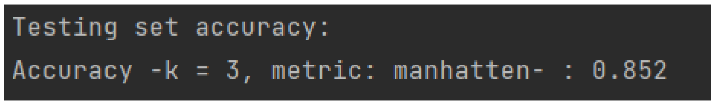
### • K-nearest neighbor:

The validation set was used to tune our hyper parameters. Four combinations of k = {1,3}and the distance metric = {Euclidean, Manhattan} were tested.



```
Validation set accuracy and hyper parameter tuning:
Accuracy -k = 1, metric: euclidean- : 0.8444
Accuracy -k = 3, metric: euclidean- : 0.84785
Accuracy -k = 1, metric: manhatten- : 0.85125
Accuracy -k = 3, metric: manhatten- : 0.85385
```

The accuracy was fairly similar in all cases. The Manhattan metric with 3 nearest neighbors showed a slightly better performance than the other s so it was used for the final model.



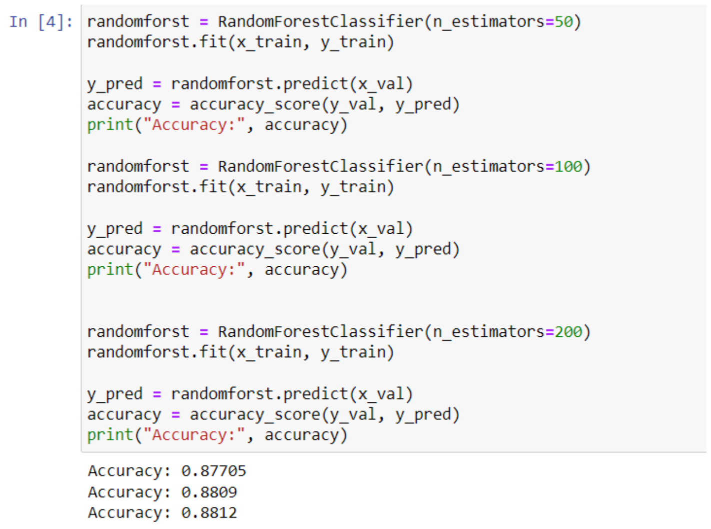
```
Testing set accuracy:
Accuracy -k = 3, metric: manhatten- : 0.852
```

The testing set accuracy was very similar to the training accuracy.

### • Random forest:

The hyper-parameter was tuned using three different values (50,100,200) and by observing the output accuracy on the validation set.

It can be noticeable that the best value for the parameter is 200 with validation accuracy of 88.12%



```
In [4]: randomforst = RandomForestClassifier(n_estimators=50)
        randomforst.fit(x_train, y_train)

        y_pred = randomforst.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)

        randomforst = RandomForestClassifier(n_estimators=100)
        randomforst.fit(x_train, y_train)

        y_pred = randomforst.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)


        randomforst = RandomForestClassifier(n_estimators=200)
        randomforst.fit(x_train, y_train)

        y_pred = randomforst.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)

        Accuracy: 0.87705
        Accuracy: 0.8809
        Accuracy: 0.8812
```

As it can be seen above:

Random Forest training accuracy is: 95.985%

Random Forest validation accuracy is: 88.12%.

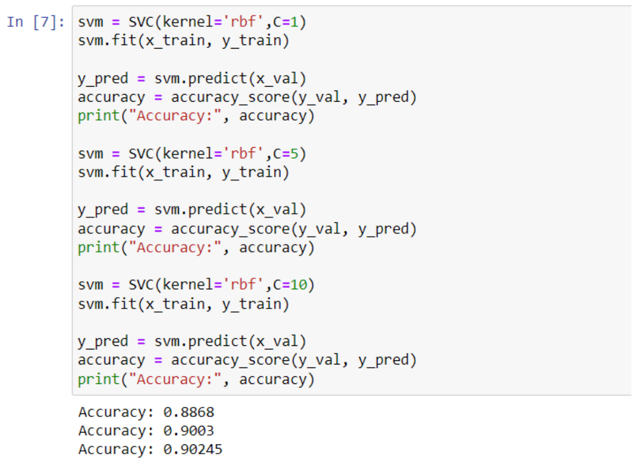
Random Forest testing accuracy is: 87.22%.

Using this model gave us high performance results which are better than the KNN model because it’s more complicated and stronger than KNN algorithm. Also, it has short training time (about 4-5 minutes) for a large dataset like the one used.

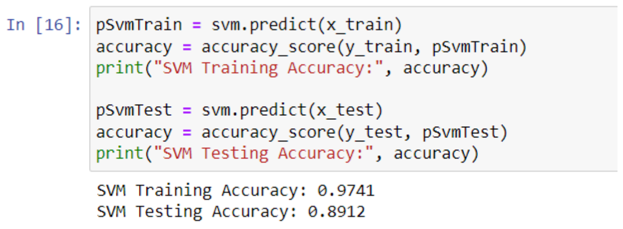
- ## Support machine vector:

SVM classifier with kernel type of radial basis function (RBF) aka Gaussian was trained on the dataset. The hyper-parameter (C) was tuned using three different values (1,5,10) and by observing the output accuracy on the validation set.

It can be noticeable that the best value for the parameter is 10 with validation accuracy of 90.0245%



```
In [7]: svm = SVC(kernel='rbf',C=1)
        svm.fit(x_train, y_train)

        y_pred = svm.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)

        svm = SVC(kernel='rbf',C=5)
        svm.fit(x_train, y_train)

        y_pred = svm.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)

        svm = SVC(kernel='rbf',C=10)
        svm.fit(x_train, y_train)

        y_pred = svm.predict(x_val)
        accuracy = accuracy_score(y_val, y_pred)
        print("Accuracy:", accuracy)

        Accuracy: 0.8868
        Accuracy: 0.9003
        Accuracy: 0.90245
```



```
In [16]: pSvmTrain = svm.predict(x_train)
         accuracy = accuracy_score(y_train, pSvmTrain)
         print("SVM Training Accuracy:", accuracy)

         pSvmTest = svm.predict(x_test)
         accuracy = accuracy_score(y_test, pSvmTest)
         print("SVM Testing Accuracy:", accuracy)

         SVM Training Accuracy: 0.9741
         SVM Testing Accuracy: 0.8912
```

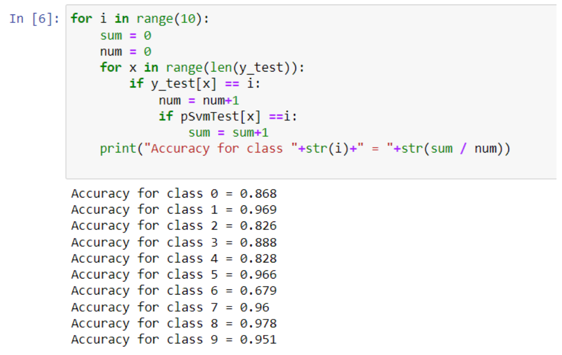
As it can be seen above:

- SVM training accuracy is: 97.41%
- SVM validation accuracy is: 90.0245%.
- SVM testing accuracy is: 89.12%.

Using this model gave us the best performance results which are better than Random Forest and KNN models, because SVM can handles outliers better than the others. However, it takes longer training time than other models (about 10 minutes).

- ## Errors and evaluation metrics:

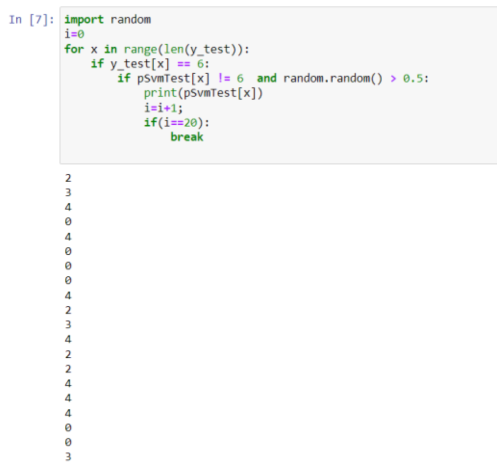
The accuracy of the best model (SVM) on each class separately from test set:



```
In [6]: for i in range(10):
            sum = 0
            num = 0
            for x in range(len(y_test)):
                if y_test[x] == i:
                    num = num+1
                    if pSvmTest[x] ==i:
                        sum = sum+1
            print("Accuracy for class "+str(i)+" = "+str(sum / num))

        Accuracy for class 0 = 0.868
        Accuracy for class 1 = 0.969
        Accuracy for class 2 = 0.826
        Accuracy for class 3 = 0.888
        Accuracy for class 4 = 0.828
        Accuracy for class 5 = 0.966
        Accuracy for class 6 = 0.679
        Accuracy for class 7 = 0.96
        Accuracy for class 8 = 0.978
        Accuracy for class 9 = 0.951
```

Class 6 has the lowest accuracy with 67.9%.

Below are 20 random images where the prediction is wrong for class 6:



```
In [7]: import random
        i=0
        for x in range(len(y_test)):
            if y_test[x] == 6:
                if pSvmTest[x] != 6  and random.random() > 0.5:
                    print(pSvmTest[x])
                    i=i+1;
                    if(i==20):
                        break

        2
        3
        4
        0
        4
        0
        0
        0
        4
        2
        3
        4
        2
        2
        4
        4
        4
        0
        0
        3
```

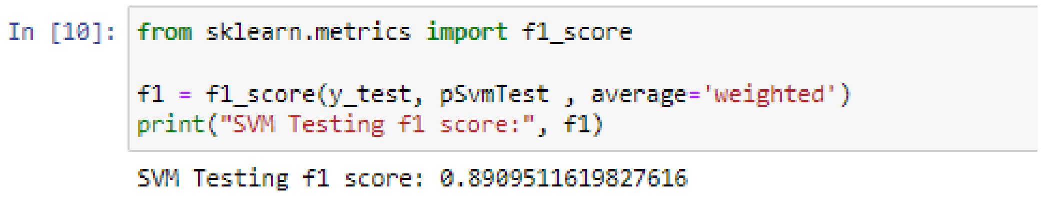
It can be seen that most of the wrong the wrong predictions are classified as 2, 0, 3 or 4 class type.

Possible reason behind that is that class 6 refers to Shirt type of clothes, and this type can sometimes be hard not to mistake it with another similar types of clothes like pullovers, T-shirts/tops, Dresses and Coats like in our case even for human beings.

One of the possible improvements to solve this problem is to give more weight to the middle area of the image, because shirts mostly recognized with buttons and by doing that the model can give more attention to the middle area where the buttons are.

The metric used here is the accuracy score. And it's actually a good metric for this dataset, because the dataset is balanced between different classes, and all the classes have the same weight, also the output is error tolerant and the errors are not fatal.

Another metrics can be used also like F1 score:



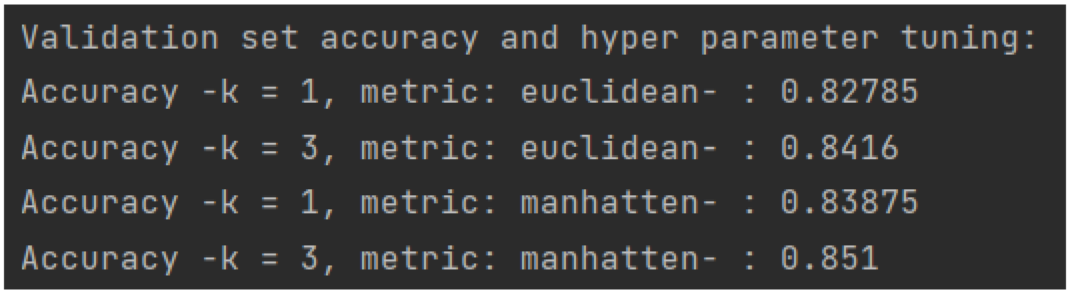
```
In [10]: from sklearn.metrics import f1_score

         f1 = f1_score(y_test, pSvmTest , average='weighted')
         print("SVM Testing f1 score:", f1)

         SVM Testing f1 score: 0.8909511619827616
```

F1 score for SVM on test set is: 89.095%

It can be noticed that f1 score has similar value to the accuracy score.

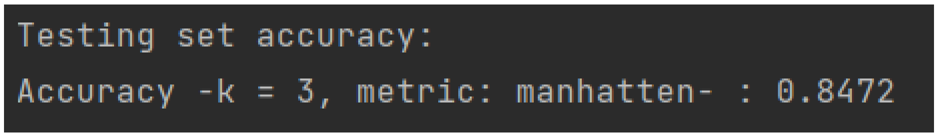
- ## New Feature Extraction:

After transforming the data into Histograms of Oriented Gradients, the new data was fed to the baseline model the KNN and displayed very similar result although slightly worse.  And the testing data accuracy displayed similar behavior.



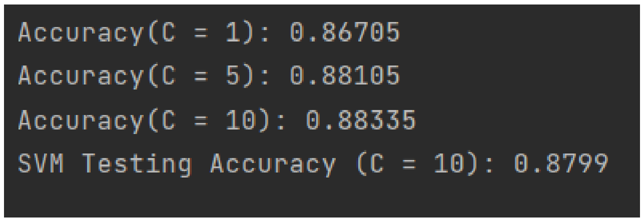
```
Validation set accuracy and hyper parameter tuning:
Accuracy -k = 1, metric: euclidean- : 0.82785
Accuracy -k = 3, metric: euclidean- : 0.8416
Accuracy -k = 1, metric: manhatten- : 0.83875
Accuracy -k = 3, metric: manhatten- : 0.851
```

The accuracy of the model on the testing set also slightly decreased by bit. Which makes the effort of extracting the hog feature from the data not worth it with unnoticeable results in the KNN model.



```
Testing set accuracy:
Accuracy -k = 3, metric: manhatten- : 0.8472
```

Also when using the hog feature with the svm model the resulted training accuracy was fairly similar to the raw pixels feature and also slightly less.



```
Accuracy(C = 1): 0.86705
Accuracy(C = 5): 0.88105
Accuracy(C = 10): 0.88335
SVM Testing Accuracy (C = 10): 0.8799
```

But the difference displayed between the validation accuracy and testing accuracy could mean that the hog feature is more reliable than the raw pixels in producing a more genialized and less biased training model.

Also both models were significantly faster due to the hog feature using it's histogram which cares more about the shape than the details of the picture which increases the calculation process.

## • Conclusions and Discussions:

After Observing different models' performance on the Fashion-MNIST dataset, and how these different models predict the testing set. We saw that between KNN, SVM and Random Forests, SVM model had the best prediction accuracy with 89.12%, then comes the Random Forests model with slight difference and accuracy of 87.22%, and the last one is KNN model with accuracy of 85.2%.

However, these models were too slow and ineffective for real-time predictions, and the stronger the model was the more time it takes for it to learn the dataset, so weaker models can be used in order to trade accuracy with time

Also, each model has its limitations like: KNN which its accuracy depends on the quality of the data. And for Random Forests a large computational power as well as resources are required as it builds numerous trees to combine their outputs. And SVM algorithm is not very much suitable for large data sets. Also, if the dataset was imbalanced, accuracy is not a good as a metric for model performance.