DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Machine Learning and Data Science

ENCS5341

Project

Prepared By:

Student Name: Ibrahim Nobani

Student ID: 1190278

Student Name: Mahmoud Nobani

Student ID: 1180729

Instructor: Dr. Yazan Abu Farha

Section: 1

Date: 10/2/2023

Contents

Introduction:

In this project we are going to test several machine learning models for a popular Fashion-MNIST dataset, that contains over 60000 images of clothes represented in 28x28 gray scale, each one of those images is associated with a label from 10 classes.

Many machine learning models were tackled in this project, each one to find its own pattern and makes decisions from the unseen dataset, to evaluate the results generated from the used models we decided to use accuracy as our main metric, this decision comes as accuracy is the most popular and used widely by people who worked on this dataset, these are the three chosen models:

1) **K-Nearest Neighbor KNN:**

   KNN is a lazy learner, non-parametric Supervised Learning technique, the stores all training data and classifies the new data based on the similarity/distance to the nearest k-neighbors [1]. This model was chosen for the following reasons:

   - Great baseline for any model.

   - No training time needed.

   - Simple and easy to implement.

   - Known for its good performance and should not be underestimated.

2) **Random Forest:**

   Random forest is a Supervised Ensemble Tree-based Machine Learning Algorithm, which means it combines multiple models to make its prediction. Ensemble has two methods, Boosting (which will be discussed in the 3rd algorithm) and Bagging which the random forest follows. Bagging methods take multiple models and train them on randomly selected subset of the training data, using their combined prediction to give a final prediction.

   This model was chosen for the following reasons:

   - Relatively fast (Compared to other algorithms it has less training time).

   - Handles Large datasets with high accuracy.

   - Reduces overfit.

3) XGBoost:

XGBoost is an implementation of Gradient Boosted decision trees, In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model [2]. This algorithm uses Boosting ensemble method. Boosting is a method of combining weak learners in a sequential way while iteratively improving the output, this reduces the bias in the data. This model was chosen for the following reasons:

-To show the difference between Bagging and Boosting ensemble methods.

-It's an optimized gradient boosting library.

-Efficient Flexible and portable.

-It won many big Machine Learning competitions (Because it's fast?).

Technical Details:

-Dataset:

The first thing we have done is normalizing the dataset by dividing its values by 255 (max value for gray scale), this was done to help speed up the process of training and testing.

Secondly, we shuffled our training set, this was done to ensure we get a sample from every class in the testing data before splitting, speaking of splitting, the dataset was then split into a training set of 40000 image and a validation set of 20000 images, we used the validation set to test our different combination of hyper parameters before applying the final test.

-KNN:

Hyperparameters:

1- K: A parameter that refers to the number of nearest neighbors to include in the majority of the voting process.
2- Distance: The distance between two data points which helps finding the nearest neighbors, and the ones we decided to test are: Euclidean and Manhattan because they the most famous.

We finally to choose the following combinations to test the hyperparameters:

Test	K	Distance
1	1	Euclidean
2	1	Manhattan
3	3	Euclidean
4	3	Manhattan
5	5	Manhattan

-Random Forrest:

Hyperparameters:

1) N-Estimators: The number of trees in the forest.
2) criterion: defines the function which determine the quality of the split for the trees, we will the two most common criteria: gini and entropy.

This is the combination of hyperparameters we want to test:

Test	criterion	N-Estimators
1	gini	100 (default)
2	entropy	100 (default)
3	gini	250
4	entropy	250
5	gini	500
6	entropy	500

-XGBoost:

Hyperparameters:

1) Learning rate: it is a regularization parameter that shrinks feature weights in each boosting step.

2) N-Estimators: number of trees to be generated and boosted.

And the combination of the hyperparameters we want to test are:

Test	N-Estimators	Learning Rate
1	50	0.1
2	50	0.01
3	100	0.1
4	100	0.01
5	100	1

Experiments and results:

-KNN:

Test Number	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
1	0	0:50	100%	0:26	84:75%
2	0	6:13	100.00%	2:59	84.82%
3	0	0:50	91:50%	0:26	85.17%
4	0	5:59	91.92%	2:57	85.64%
5	0	6:04	89.98%	2:59	85.86%

From the results we obtained above, we can notice that the distance metric 'Manhattan' gives better results in general compared to 'Euclidian' distance (although Euclidian was considerably faster), if we look at the K pairing, we can notice that the higher k is, the better accuracy we had, and less over-fitting effect we got in our model, this can be highlighted if we compare test 2 (k=1, Manhattan) where we our model overfit (training accuracy=100%, validation accuracy = 84.75%), and test 5 which had our best performing model (k=5, Manhattan).

Thus, we decided to choose KNN with k = 5, and Manhattan distance as our baseline model

and now we test the baseline model on the test images:

time taken to predict: 1:37, accuracy = 85.65%

from now on we will aim to beat that score with the other models.

-Random Forrest:

Test Number	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
1	1:15	0:03	100%	0:01	87.90%
2	1:45	0:02	100%	0:01	88.09%
3	3:24	0:04	100%	0:02	88.17%
4	4:10	0:05	100%	0:03	88.14%
5	5:48	0:09	100%	0:05	88.28%
6	8:53	0:09	100%	0:05	88.21%

First look reaction: **overfitting,** all the different hyper parameters for the random forest model produced overfitting results, which is not desirable.

Now looking at the hyper parameters, we can see that when we increase N-Estimators, the accuracy increases, and although at the start '**entropy'** was better than '**gini'** (for N-Estimators = 100), the other test proved that **gini** is a better criterion for evaluation than **entropy.**

Thus, we can see that our best model is 5, with hyperparameters of: (criterion = gini, N-Estimators = 100), with an accuracy for the validation set of 88.28% which surpassed our baseline (85.86%).

Now let's test our model on the testing data:

time taken to predict: 0:02, accuracy = 87.29%

and we were able to surpass our baseline.

-XGBoost:

Test Number	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
1	3:50	0:01	93:44%	0:00	87.61%
2	4:28	0:01	87:39%	0:01	84.30%
3	9:03	0:00	97:41%	0:00	88.81%
4	9:16	0:01	88:37%	0:01	84.75%
5	10:02	0:02	100%	0:01	89.34%

Looking at XGBoost result, and we can only describe it by saying, its all over the place, its lacks the consistency the random forest provided, but it reach a ceiling that random forest didn't reach, which is an accuracy score for the validation set of 89.34%, and that thanks to test 5 (LR:1, N-Estimators = 100), we can see that a bigger LR produce a better results, whereas bigger N-estimators produce both a better results but the model overfits, which is the bad news here.

But the good news is that we beat both the baseline model, and our best performing model, now let's put our best-performing model on the test vs the testing data:

time taken to predict: 0:01, accuracy = 88.62%

and we were able to both surpass our baseline and former best-performing model.

-Studying the XGBoost (the best performing model):

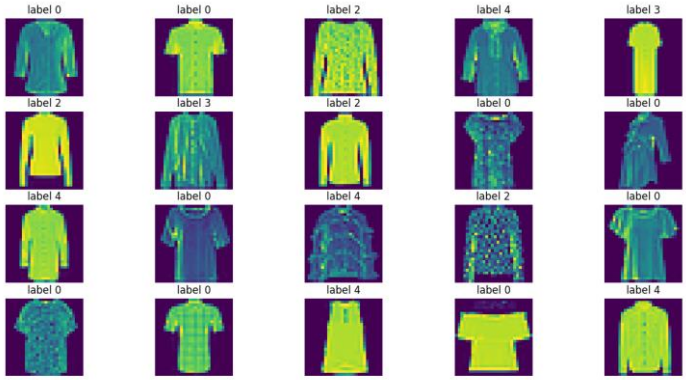
Now we want to study the best performing model even further, we want to understand how the accuracy score was distributed on each separate class, so we first divided the testing data to 10 separate ones (representing each class/label) and we got the following results:

Label	Accuracy score
(T-shirt/top) class 0	83.90%
(Trouser) class 1	97.00%
(Pullover) class 2	82.10%
(Dress) class 3	91.10%
(Coat) class 4	81.10%
(Sandal) class 5	96.20%
(Shirt) class 6	65.50%
(Sneaker) class 7	95.40%
(Bag) class 8	96.50%
(Ankle boot) class 9	95.10%

As we can see, the only bad classification we got was with class 6 or shirts, other classes accuracy scores were great, all of them were over 80%, with some reaching the 95%, but why did class 6 give such a bad score?

The first idea we have is the fact that it's hard to classify a shirt when u also have pullovers (class 2), t-shirts (class 0) and coats (class 4) which all of these 3 classes had 80% score unlike the other ones who reached the 90%.

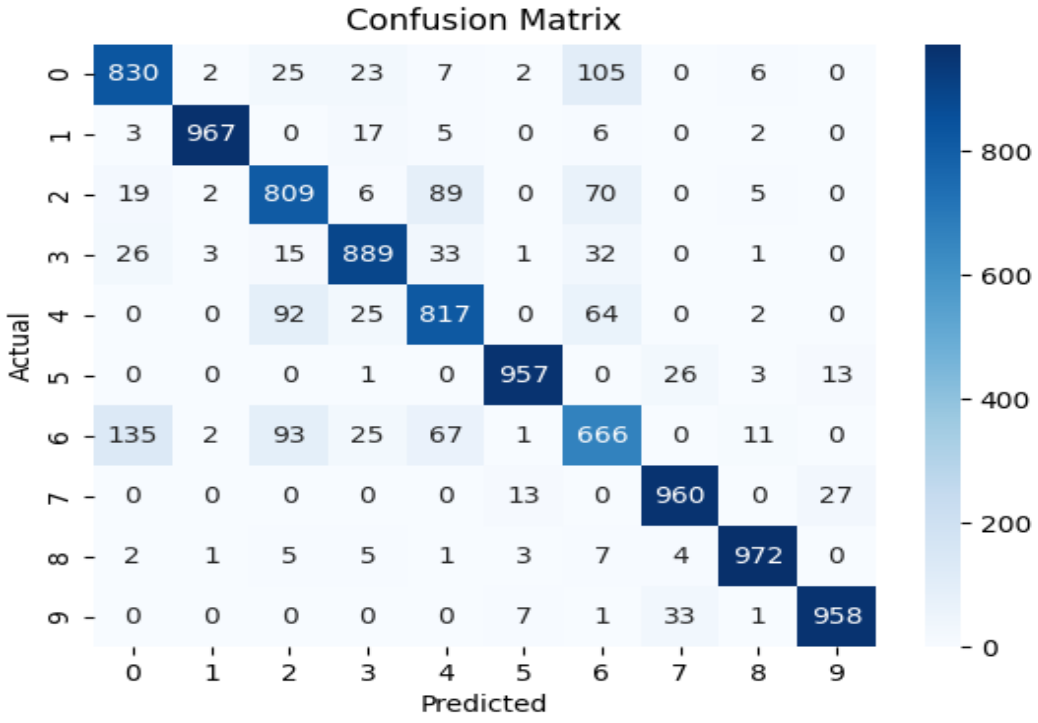
So, to understand what happened, we printed 20 images from class 6 as shown below:



Looking at the output, our initial assumption that class 0, 2, 4 affected class 6 is true with class 0 being the main one.

This little experiment shows us the problem with accuracy as a classifier, it does give a good indication (it gave over 88% score for all test cases), but it doesn't tell u the whole story, as it can't predict if all class work effectively (class 6 (shirt) had 66% score).

I think a better way to analyze and understand the data is by using another evaluation metric, the confusion matrix, so we displayed it as shown below:



From the confusion matrix, we can see Cleary each class accuracy score, and where the misclassification happened, for class 6, the confusion matrix reaffirms out initial assumption that class 0, 2, and 4 is the reason class 6 had such a bad accuracy score with class 0 being the main culprit, it seems our model isn't strong enough to differentiate between these classes.

How we can face this problem?

Increase the weights of the weak class, we have done that and the results we got is as shown below:

Label	Accuracy score W for all = 1	Accuracy score (W for class 6 = 2, other 1)	Accuracy score (W for class 6 = 5, other 1)	Accuracy score (W for class 6 = 10, other 1)
(T-shirt/top) class 0	83.90%	83.70%	82.80%	78.20%
(Trouser) class 1	97.00%	96.90%	96.80%	97.10%
(Pullover) class 2	82.10%	81.90%	80.30%	79.50%
(Dress) class 3	91.10%	90.00%	89.60%	87.70%
(Coat) class 4	81.10%	81.60%	89.60%	79.70%
(Sandal) class 5	96.20%	95.20%	96.00%	95.60%
(Shirt) class 6	65.50%	68.20%	71.40%	72.80%
(Sneaker) class 7	95.40%	96.20%	95.90%	95.90%
(Bag) class 8	96.50%	96.50%	96.30%	95.90%
(Ankle boot) class 9	95.10%	95.00%	95.30%	95.30%

Well, it kind of worked, the accuracy for class 6 did increase each time it weight increase, and we managed to reach an accuracy score over 70% for it, which is a huge improvement, but sadly it came at the expense of the other class, especially class 0, 2 and 4, and the reason is that the overall accuracy for the model didn't change, the model we have here still can differentiate well between these 4 classes, what better solutions we can have? New features, maybe.

-New features:

So far, we used 784 feature to represent the data, which is too high and that lead to high time of testing, so we want to find new features to represent the data:

1) Avg:

When u study how the data is distributed, we can Cleary see its a 28*28 array, how about we replace the values of each row with the avg of it, and the results we got?

Test	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
Xgboost (LR =1, n-estimators =100)	13:11	0:01	100%	0:00	77.15%
KNN (k=5, Manhattan)	0:00	10:32	81.59%	3:25	75.25%
RF (n-estimators = 500, gini)	15:03	0:09	100%	0:03	77.14%

And as we can, the accuracy we got her, isn't promising, so we will try a new, different approach to calculate the features.

2) PCA:

PCA or Principal Component Analysis is a dimensionality reduction technique used to reduce the features while retaining as much information as possible, the reason we choose this algorithm is because of a paper we found on the internet comparing multiple algorithms against each other on fashion MNIST dataset and we saw how good PCA performed **[3].**

U might be thinking, isn't the reduction of the number of features a bad thing? Why would you do it? The answer is no, but a little, as the reduction of features comes at the expense of accuracy of course, but the trick is that we trade the small percentage of accuracy for simplicity, and simple is always better than complex, because simple and smaller dataset is easy to use, faster, and a good way to avoid overfitting, and PCA is the perfect way insure the best results.

Now before we perform PCA, we need to standardize our data (normalize our feature by making the mean = 0, and the variance = 1) and the reason we did that is because PCA is quite sensitive to variance **[4]**.

Now how does PCA work?

A. Covariance Matrix:

PCA aims to keep as much information as possible, this is done first by Covariance matrix which give the relation between each feature, and how strongly they are connected**[4]**..

The covariance matrix is p x p symmetric matrix where p is the dimension of the data.

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Each variable in the matrix has a sign, where + is correlated and – is inversely correlated.

B. Eigenvalues and Eigenvectors:

After that the eigenvalues (indicate the magnitude of the variance in each eigenvector direction) and eigenvectors (directions in the data that correspond to the maximum variance) are calculated (with the help of the covariance Matrix) **[4]**..

The reason we calculate the eigenvalues and eigenvectors is to find the principle components, which are a new features that represent the maximum amount of variance and hold the information of the old features, (principle components are basically eigenvectors with the largest eigenvalues), after calculating all the principle components, we choose the ones we need (the number of the new features) with highest values eigenvalues, these will be stored in a matrix called feature matrix where the number of column equal to the number of eigenvalues we decided to keep**[4]**..

C. Transform the data:

Lastly, we will change the input data using the feature matrix we constructed, this is done by finding the dot product between the transpose of the data and the transpose of the feature matrix.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

D. Testing using PCA:

- **KNN with PCA:**

And after this long procedure, we managed to shorten the number of features we have to a desired one, first we will apply it on KNN our baseline with the hyperparameters **(k=5, Manhattan)**

Test	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
New features = 50	0:05	0:48	89:81%	0:29	85.50%
New features = 85 (recommend by **[3]**)	0:05	1:30	90:19%	0:45	86.65%
New features = 100	0:05	1:41	90.31%	0:48	86:06%
Original	0	6:04	89.98%	2:59	85.86%

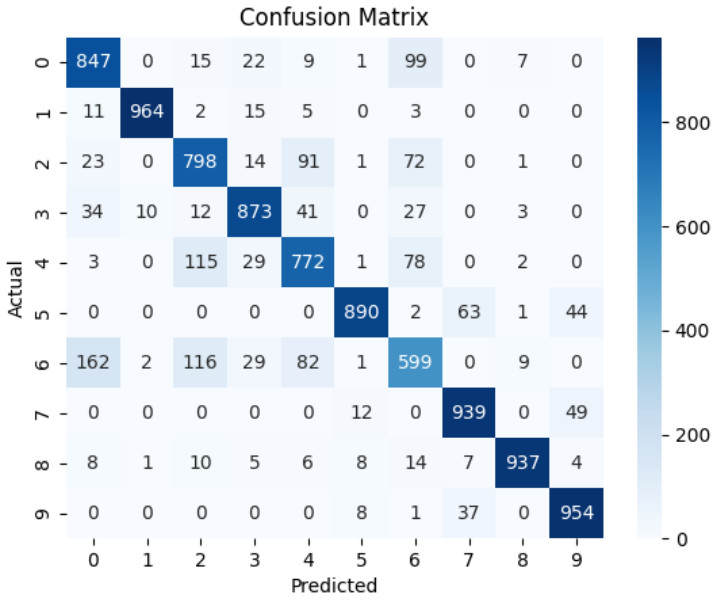
Well, the results we got with PCA are great, not only faster but also has very high accuracy score and as we can see, having 85 features as recommended gave a high validation set accuracy, even higher than the original, now let's find the test accuracy:

For original KNN: test set prediction time: 1:59, test set accuracy: 85:73%

For KNN with PCA: test set prediction time: 0:22, test set accuracy: 85:60%

even when using the test data, KNN with PCA performed better in terms of speed and accuracy, the reason KNN worked well with PCA is probably because PCA helped reducing the noise in the feature of the data (which KNN is sensitive to), thus makes it easier for KNN to the most accurate prediction.

Now let's calculate the confusion matrix for our new best KNN model:



sadly, our new model, still can't classify class 6 well.

- **Xgboost with PCA**

Now let's try PCA with our best performing model Xgboost with hyperparameters of **LR = 1, N-estimators= 100.**

Test	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
New features = 50	5:08	0:01	100%	0:01	86.38%
New features = 85	6:58	0:01	99.98%	0:01	86.84%
New features = 100	8:35	0:02	99.98%	0:01	87.22%
original	10:02	0:02	100%	0:01	89.34%

Although the speed for Xgboost did improve, the accuracy didn't unlike KNN, and the reason for that might be that Xgboost, is more advance model, less sensitive to noise, and can derive more

complex relations between the feature, so by reducing them, we reducing the range it can work with.

We also can see that we still have overfitting problem.

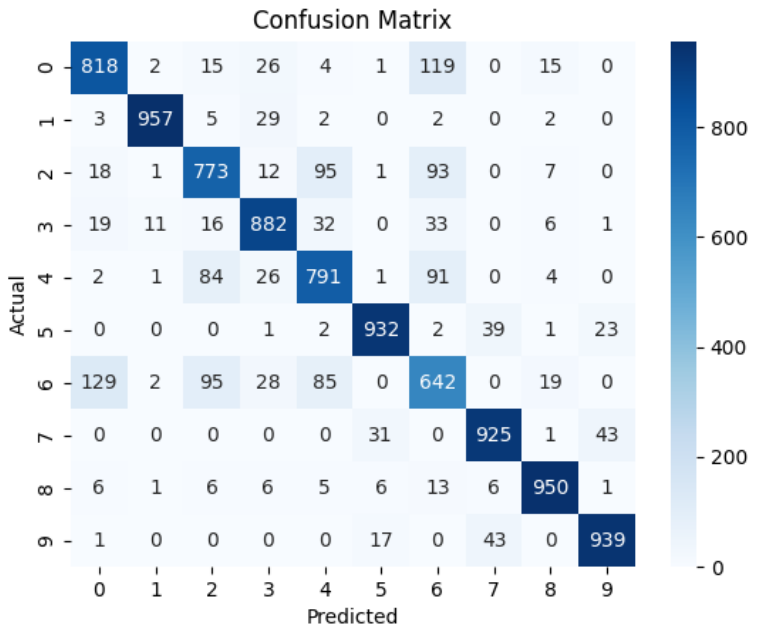
now let's find the test accuracy on the test set:

For original Xgboost: test set prediction time: 0:01, test set accuracy: 88.62%

For Xgboost with PCA=100: test set prediction time: 0:01, test set accuracy: 86:02%

As expected, it didn't perform better than the original model.

And finally let's find the confusion matrix for the test set:



And the problem we faced with class 6 is still present.

Maybe we can solve if we had a better model, a model that can possibly reach the 90% test accuracy, let's try CNN.

3) New Approach CNN:

So far, all the approaches we used are traditional and shallow model for machine learning, but now we will try a new approach a deep learning approach, which we hope it will help us reach the 90% accuracy mark, and thus increase the accuracy of the week class.

One of the advantages of deep learning approach is their ability to find pattern and extract features which will benefits us a lot.

Deep learning approaches has a lot of techniques, for the sake of this project we will use, CNN (convolutional neural network), and the reason for using it are:

1) Deep learning approach
2) Simple and not as complex as other approaches
3) Highly recommended for computer vision and image recognitions

how does CNN work? CNN is a combination of layers, each layer performs a certain task, in general it has 3 layers **[5]**:

1) convolutional layer: the majority of the computation happens here, using kernels or filters and across multiple iterations, a convoluted feature is produced, which allows CNN to extract pattern from the data and thus the features, there can be multiple layers of it in one model.
2) pooling layer: the aim of this layer is to reduce complexity and improve efficiency using kernels or filters.
3) Fully Connected Layer: the final layer, where everything comes together thus producing the classification

Other types of layers can also be used like dense, dropout and flatten.

These layers come together to form a CNN model, where each layer detects a different feature of the input data using filters/kernels scattered through the model, after each layer, the complexity increases, and the identification capability increases, this process is repeated multiple time until our model is capable of classifying the data we have.

For our variation of the CNN model, we decided to use these layers **[6]:**

Number	Layer	Description
1	Conv2D(filters=64, kernel_size=2, padding='same', activation='relu', input_shape=(28, 28, 1))	A 2D convoluted layer with an 64 filter of size 2*2 used to extract the features, the padding is applied here to avoid information lose, the input shape was specified to grayscale, with activation function “relu”
2	MaxPooling2D(pool_size=2)	Next we perfom pooling operation to reduce the spasital size of the data, thus reducing paramters while keeping the most important data and reducing overfitting, which choose a size of 2 (2*2) that fits the previous layer
3	Dropout(0.3)	To regularize our model, where a percantge of neurons are droped out (0.3), this helps reduce overfitting
4	Conv2D(filters=32, kernel_size=2, padding='same', activation='relu')	These next three layers are similar to the previous ones, the repetition allows for a better extraction of information and its recommended to use with image based data, this will help avoid overfitting and act as a regulator
5	MaxPooling2D(pool_size=2)	
6	Dropout(0.3)	
7	Flatten()	Convert the input to a one dimensional one
8	Dense(256, activation='relu')	Convert the input to a readable one using activation relu
9	Dropout(0.5)	Another dropout layer to make sure everything is good and to reduce overfitting as much as possible
10	Dense(10, activation='softmax')	The classification happens her, with 10 neurons (represent the number of classes) and softmax which is a great function for multiclass classification as it will find the probability of the best classification

Now we will try running This model with different epochs values or iterations where it will with each time try to reduce the loss function, and we will also try different batch sizes (samples).

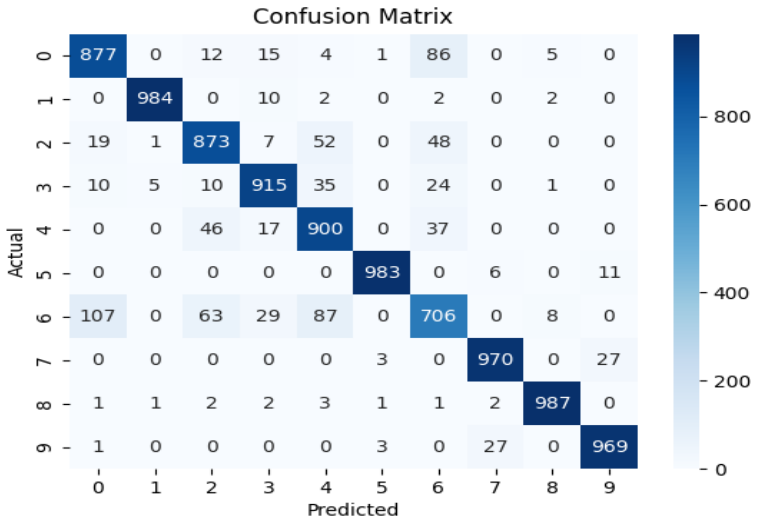
After running the model, we got the following results:

Test	Training time	Training set prediction time	Training set Accuracy	Validation set prediction time	Validation set Accuracy
Epoch = 10, batch size = 32	6:31	0:12	92.83%	0:06	91.29%
Epoch = 10, batch size = 64	7:44	0:11	92.45%	0:05	90.97%
Epoch = 18, batch size = 32	14:37	0:19	95.09%	0:07	91.97%
Epoch = 18, batch size = 64	17:08	0:15	94.01%	0:08	91.75%

The results we got are great, all of our tests managed to reach a validation set accuracy of over 90%, which just shows how strong CNN is,

The best performing test was, the third one (epoch = 18, batch size = 32), it did take some time, which is to be expected, but it performed the best and now we want to put it to the test on the testing set: **test set prediction time: 0:03, test set accuracy: 91.64%**

finally, we managed to reach 90% on our testing set, and we now can hope for a better result for class 6, to find that out, lets print the confusion matrix:



Well, the results here is better than we used to get, the class 6 accuracy is higher and its over 70%, and with an overall accuracy of 91.64% I can say this was a success.

Conclusion:

In this project we tested several machine learning models for a popular Fashion-MNIST dataset, the models tackled in this project were KNN, Random Forrest and Xgboost.

First we started with KNN, 5 tests were conducted, after analyzing the results we choose KNN (k=5, Manhattan distance) as our baseline model, then testing on the test images we generated the following (time taken to predict: 1:37, accuracy = 85.65%), followed by that we ran 6 tests on the Random Forrest model, the best model for this had the following parameters :(criterion = 'gini', N-Estimators = 100),), with an accuracy for the validation set of 88.28% which surpassed our baseline (85.86%), test images were tested for this and the following results were obtained (time taken to predict: 0:02, accuracy = 87.29%), XGboost was the final model, 5 tests were conducted with the 5^{th} test being the best with hyperparameters of (LR:1, N-Estimators = 100), so this was tested on the test images and produced the following results (time taken to predict: 0:01, accuracy = 88.62%), thus we were able to both surpass our baseline and former best-performing model.

There were many limitations concerning the models we used, for KNN, we noticed that it didn't produce great accuracy scores, and it did also require a considerable amount of time to perform the needed computation, and the huge number of features our dataset had didn't help (Curse of dimensionality), as for random forest, although it had better accuracy scores, the main problem we faced is overfitting, all the random forest parameters we tried made the model overfit, lastly for boost our (former) best performing model, it did have the best accuracy score out of the previous model, badly sadly it didn't reach over 90 and it did overfit, another limitation it had was the long training time, which made it harder to tune our model

Since XGboost was our best preforming model, we decided to dive deep into its computations and calculated the accuracy for each of the 10 classes, we noticed that although our model had a relatively high accuracy it failed to efficiently predict class 6 (shirt) which had an accuracy of 65.5% which speaks of the limitations our traditional model and metric had, we concluded that accuracy metric although it gives good indication it doesn't tell everything about the model, therefore we used the confusion matrix to further our knowledge about the model performance, other metrics we can use are like (AUC, ROC), in order to improve the performance with class 6 we tried to increase its weight, this has indeed increased the class accuracy but other classes were affected and their accuracy went down, so instead we tried other features to analyze our model,

the first feature we tried is by averaging each row in the 2D representation of the image, this attempt produced even worst results than the ones that led us to use the average metric. So we tried using a more popular way to find new features (Dimensionality reduction called PCA), we noticed after using the improvement we got on our baseline model (KNN), while the accuracy of the XGboost got slightly worse, but the speed of each of these two models got higher with PCA. So we tried one last approach, a deep learning approach (CNN), which was our best preforming model so far and reached an accuracy of 91.64%, while the accuracy of class 6 improved to 70.6%, an improvement over the last few attempts.

Overall, we can call this little experiment a success, but we can a little further, in the future we can try other machine learning model like svm (which a lot of people recommend for image classification), and we can also try other CNN variations as we have seen success with the one we used, other techniques we can implements are data augmentation (which can help increase the accuracy of the weak class), and maybe fld (another dimensionality reduction technique)

References:

[1]: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

[2]: https://www.geeksforgeeks.org/xgboost/

[3] https://cnedwards.com/files/Edwards_Yen_Final_Project.pdf

[4] https://builtin.com/data-science/step-step-explanation-principal-component-analysis

[5] https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

[6] https://keras.io/