Fashion-MNIST: Clothing Items Classification

Lilly Kumari
University of Washington
lkumari@uw.edu

Narendra Shivaraman
University of Washington
narensh@uw.edu

Abstract

Image classification is a supervised learning problem where we need a set of target classes, and train a model to recognize them using labeled example images. In this project, we have trained different machine learning models for multi-class classification on Fashion-MNIST dataset and compared their performance with respect to different evaluation metrics. The dataset is very similar to the MNIST handwritten digits dataset, and embodies fashion related clothing images belonging to overall 10 different classes. We have experimented with SVM (Support Vector Machines), XGboost (Extreme Gradient Boosted Decision Trees), MLP (Multi-layer Perceptron) and CNN (Convolutional Neural Network). On comparison of prediction results, we found out that the CNN model is performing the best since unlike other methods, it takes care of the spatial pixel information of input images data.

1. Introduction

Image classification is the process of assigning categories or classes of interest to the pixels in an image. For successfully doing that, the classifier needs to learn the relationship between the data features (or attributes) and the information classes. When domain knowledge is available, the classification falls into supervised learning category. The prior knowledge helps in mapping of pixel attributes to classes.

The complete image classification pipeline can be formalized as follows:

- **Input:** The input which is called a training set consists of a set of N images which are each labeled with one of the K classes.
- **Learning:** The agenda is to learn a model (using the training set) which recognizes what each class looks like.
- **Evaluation:** Here we assess the quality of the learnt model by comparing its predictions for a completely new set of images (test set) against the ground truth.

2. Dataset

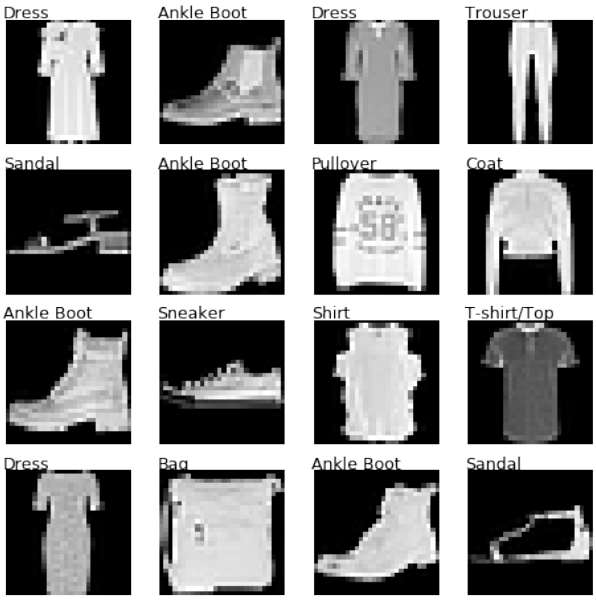

Figure 1. Sample Images from Fashion-MNIST

Fashion-MNIST is a dataset provided by Zalando Research [4]. This dataset is very similar to the MNIST (handwritten digits) dataset which is arguably the most used dataset in the image classification domain. Fashion-MNIST dataset comprises of 70,000 images which are represented as 28 x 28 grayscale images. Some sample images are shown in Figure 1. These images correspond to some fashion products from the following 10 categories:

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

Originally the dataset has been split into training set (size: 60000 images) test set (size: 10000 images). For hyper-parameter tuning, we have split further the training

set into our training set and validation set which contain 50,000 and 10,000 images respectively. The splitting was done after random shuffling so that the validation set mimics the original label/category distribution. Apart from that, we have not changed the test dataset in any manner.

2.1. Exploratory Data Analysis & Visualizations

In order to get an idea of the data distribution, we did some exploratory data analysis. The label distribution within the data as well as within the splitting is even, for example, the training data set contains 6,000 images of each of the 10 categories as shown in Figure 2. Since the dataset is balanced, we don't need to worry about different class weighting techniques to deal with dataset label skewness.

We wanted to visualize the clustering of our data in lower dimensions. Since the pixel values are arranged in 28 x 28 array fashion, the input data has a dimension of 784. We use Principal Component Analysis (PCA) [3] which is a dimensionality reduction technique to transform our dataset into 2 dimensions for 2D visualization, where those 2 dimension represent the maximum variance within the dataset. Prior to applying PCA using python's scikit-learn package [1], we normalize the input dataset since PCA is affected by the scale of different features. Also we randomly choose a subset of 1,000 images from the normalized training dataset such that they contain images from all given categories and then apply PCA on top of that. The PCA visualization on a random subset is shown in Figure 3

As it can be seen in Figure 3, there are certain classes which are more interweaved into one another, for example {t-shirt, shirt and pullover} can hardly be clustered in the linear subspace. Same goes for {sneaker and sandal} and {pullover and coat}. The remaining classes are mostly clustered, hence this suggests that we should also evaluate the micro-accuracy (classification accuracy corresponding to each class) while assessing our models.

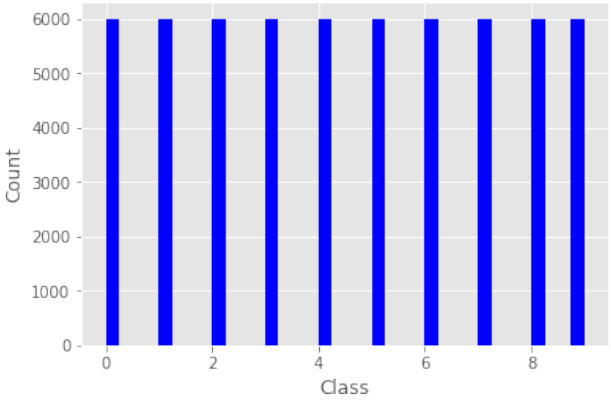

Figure 2. Label distribution in training data

3. Modelling for Classification Task

3.1. SVM

Support Vector Machines are supervised machine learning algorithms which are quite well known for both classification and regression tasks. They are based on the idea of constructing a set of hyper-planes which best separate out the data points. The points which are closest to such hyper-planes are called support vectors. SVM finds its optimal decision boundary by maximizing the distance (called margin) between the hyper-planes and their support vectors and ensuring that different class data points lie on the opposite side of the hyper-plane.

A hyper-plane in an n-D feature space can be represented by the following equation:

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w} + b = \sum_{i=1}^{n} x_i w_i + b = 0$$

Dividing by $||\mathbf{w}||$, we get

$$\frac{\mathbf{x}^T\mathbf{w}}{||\mathbf{w}||} = P_{\mathbf{w}}(\mathbf{x}) = -\frac{b}{||\mathbf{w}||}$$

indicating that the projection of any point $\mathbf{x}$ on the hyper-plane onto the vector $\mathbf{w}$ is always $-b/||\mathbf{w}||$, i.e., $\mathbf{w}$ is the normal direction of the plane, and $|b|/||\mathbf{w}||$ is the distance from the origin to the plane.

The hyper-plane partitions the n-D space into two regions. Specifically, we define a mapping function $y =$

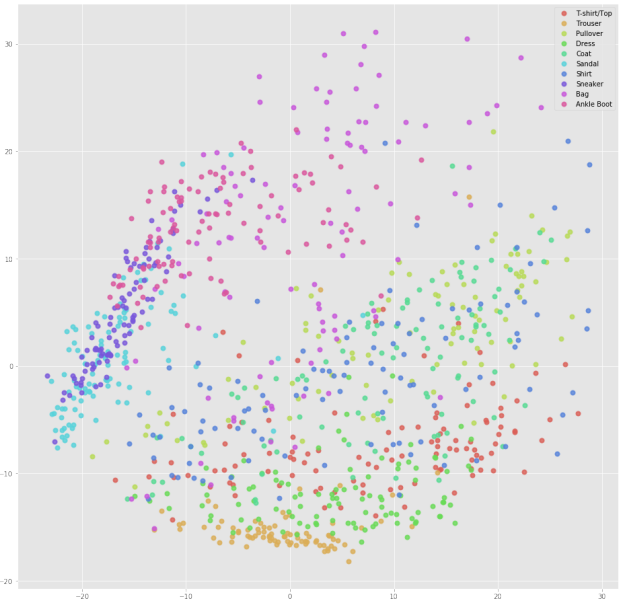

Figure 3. PCA Visualization into 2 components on Fashion-MNIST dataset

$sign(f(\mathbf{x})) \in \{1, -1\}$,

$$f(\mathbf{x}) = \mathbf{x}^T\mathbf{w}+b = \begin{cases} > 0, & y = sign(f(\mathbf{x})) = 1,\ \mathbf{x} \in P \\ < 0, & y = sign(f(\mathbf{x})) = -1,\ \mathbf{x} \in N \end{cases}$$

Any point $\mathbf{x} \in P$ on the positive side of the plane is mapped to 1 (positive class), while any point $\mathbf{x} \in N$ on the negative side is mapped to -1 (negative class). A point $\mathbf{x}$ of unknown class will be classified to P if $f(\mathbf{x}) > 0$, or N if $f(\mathbf{x}) < 0$.

We can write the SVM classifier with *K* as the kernel function as follows

$$g(x) = \text{sign}\left(\sum_i y_i w_i K(x_i, x) + b\right) \tag{1}$$

$$= \text{sign}\left(\sum_{i:y_i=1} w_i K(x_i, x) - \sum_{j:y_j=-1} w_j K(x_j, x) + b\right) \tag{2}$$

$$= \text{sign}\Big(h_+(x) - h_-(x) + b\Big). \tag{3}$$

3.1.1 Implementation

For learning a SVM on the Fashion-MNIST dataset, we used a python library called *svm* provided by *Python Scikit-Learn* package [1]. The *SVC* class for fitting the data and ground truth labels takes the following arguments:

- **C:** This is the penalty parameter of the SVM's error term. When C is small, SVM chooses a large margin hyper-plane at the expense of misclassifying greater number of data points. When C is large, it chooses a small margin hyper-plane which classifies the data points more correctly. So, in a way, it controls the trade-off between finding a smooth decision boundary and classifying data points correctly.

- **Kernel type:** A linear kernel (function) is used when the data points are linearly separable. But for the problem cases where the data points are not linearly separable, SVM converts them into a separable points by using kernels which transform the low dimensional input space into a higher dimensional space. The non-linear kernel types are *poly* and *rbf*.

- **Gamma (γ):** It is the kernel coefficient for rbf and poly type kernels. Higher values of gamma tend to exactly fit the training dataset and hence lead to possible over-fitting.

- **Degree:** It is the degree of the polynomial kernel function which is used to transform the data to higher dimensional space so that it becomes separable.

3.1.2 Hyperparameter Tuning

Since machine learning models are very prone to over-fitting and under-fitting problems, we need to carefully choose a combination of hyper-parameters giving best cross-validation score which we could pass onto to the final algorithm. This grid search for hyper-parameters was implemented using *GridSearchCV* method provided by python *Scikit-Learn model selection* library. Instead of fitting the entire training data, we used a subset of data (called validation dataset) for the best hyper-parameter search. Table 1 shows the test accuracy against different sets of *C*, γ and *Degree*. Based on that, a poly kernel with degree 3, error penalty term (C) of 3 and kernel coefficient (γ) of 0.1 are chosen for the final learning on training data points.

Table 1. SVM - Grid Search

C	Degree	γ	Test Acuracy
1	3	0.1	0.859200
1	3	0.01	0.859200
1	4	0.1	0.845800
1	4	0.01	0.845800
1	5	0.1	0.838200
1	5	0.01	0.838200
10	3	0.1	0.859200
10	3	0.01	0.859200
10	4	0.1	0.845800
10	4	0.01	0.845800
10	5	0.1	0.838200
10	5	0.01	0.838200

3.1.3 SVM: Results

Since this is a Multi-Class classification problem, we look at both macro accuracy and micro accuracies, although the classes are evenly distributed throughout the dataset. The overall accuracy of the learned SVM classifier on the provided test dataset is 0.8894. Figure 4 shows the confusion matrix corresponding to the test dataset. Also, Table 2 provides details about the per-class (micro) Precision, Recall and F1 score of the learned SVM on the test dataset.

3.2. Extreme Gradient Boosting Decision Trees - XGBoost

XGBoost which stands for Extreme Gradient Boosting is an advanced gradient boosted decision trees algorithm which is optimized so as to build the decision trees in a parallel fashion. It is a supervised ensemble learning method which is used for both classification and regression tasks. In contrast to algorithms like Random Forests which use bagging technique for aggregating the outputs from multiple learners, XGBoost uses boosting technique in which trees

are built sequentially to reduce the residual errors coming from the previous trees.

$$F_1(x) = F_0(x) + h_1(x)$$

where F_0 is the initial model which is is used to predict target variable y. It gives a residual error of $(y - F_0(x))$. h_1 is the new model which is fitted to the residuals of previous step and finally, F_0 and h_1 are combined to give a boosted up model F_1. This boosting step is carried on for m iterations until the residuals have been minimized as much as possible:

$$F_m(x) = F_{m-1}(x) + h_m(x)$$

So, in this way, each consequent learner learns something from its predecessors and the model keeps on getting updated via gradient descent on the residual errors. The final learner thus performs best with respect to keeping down both bias and variance in the predictions.

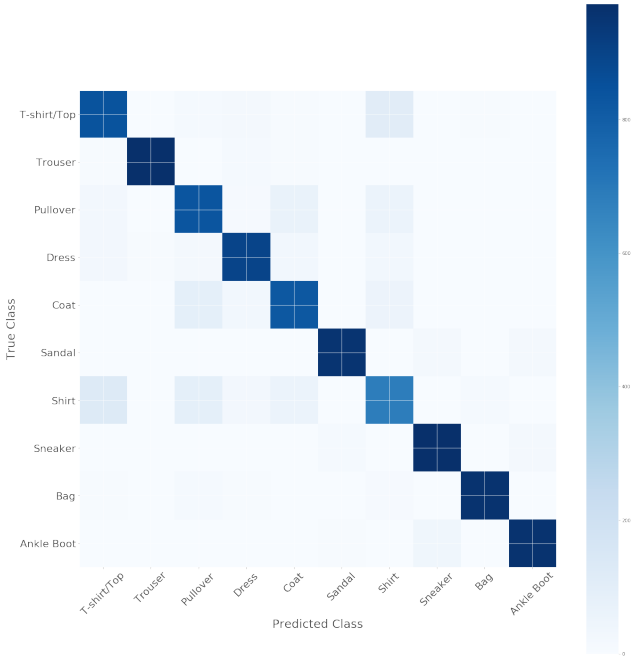

Figure 4. Confusion Matrix - SVM

Table 2. Evaluation Metrics - SVM

Class Name	Precision	Recall	F1 Score
t-shirt	0.82	0.84	0.83
trouser	0.99	0.97	0.98
pullover	0.78	0.83	0.81
dress	0.90	0.90	0.90
coat	0.83	0.82	0.83
sandal	0.98	0.96	0.97
shirt	0.72	0.68	0.70
sneaker	0.94	0.97	0.96
bag	0.97	0.96	0.97
ankle boots	0.97	0.96	0.96
overall	0.89	0.89	0.89

3.2.1 Implementation

For learning an ensemble of gradient boosted decision trees, we used the *xgboost* python package [2]. This implementation has an in-built regularization method to prevent overfitting via L1 or L2 regularization. It provides custom optimization objectives and evaluation criteria along with built-in cross-validation at each iteration. The *train* method takes the following arguments:

- **eta:** This is the boosting learning rate by which the model weights are adjusted at each step.
- **Minimum Child Weight:** This defines the minimum sum of weights of all the observations for a child node. Higher values can prevent the model from learning specific relations existing between certain data samples, hence can lead to under-fitting if not tuned properly using cross-validation.
- **Maximum Depth:** This defines how deeply a tree can be grown during the boosting process. Higher values can make the model learn relations specific only to certain data samples, thus can lead to over-fitting if not tuned properly using cross-validation.
- **Subsample:** This denotes the fraction of data points to be randomly sampled for each tree. Lower values typically lead to under-fitting.
- **Column Sample by Tree:** This determines the fraction of columns (features or attributes) to be randomly sampled for building a tree. High values of this can lead to over-fitting.
- **Alpha (α):** This denotes the L1 regularization coefficient on child weights.
- **Gamma (γ):** This denotes the L2 regularization coefficient on child weights.

3.2.2 Hyperparameter Tuning

Since the choice of different parameters can lead to possible under-fitting and over-fitting problems in our problem space, we need to diligently choose them. Hence, we ran a grid search on the most vulnerable two parameters (*Maximum Depth* and *Minimum Child Weight*) which can make the model prone to the two problems mentioned above. This was done using *GridSearchCV* method provided by python *Scikit-Learn model selection* library. The search results are shown in Table 3. So, at last, we learned a XGBooost model

with learning rate as 0.1, subsampling fraction as 0.8, column sampling by tree fraction as 0.8, L1 regularization coefficient as 8, L2 regularization coefficient as 2, maximum tree depth as 4 and minimum child weight as 3.

Table 3. XGBoost - Grid Search

Maximum Tree Depth	Minimum Child Weight	Test Acuracy
4	1	0.860100
4	3	0.860200
6	1	0.859000
6	3	0.859300

3.2.3 XGBoost: Results

For evaluating our model, we looked at both the overall accuracy as well as the micro (per-class) precision and recall values. The overall accuracy of the XGBoost classifier on the provided test dataset is 0.894. Figure 5 shows the confusion matrix corresponding to those predictions. Also, Table 4 provides details about the micro precision, recall and F1 scores of the classifier on the test dataset.

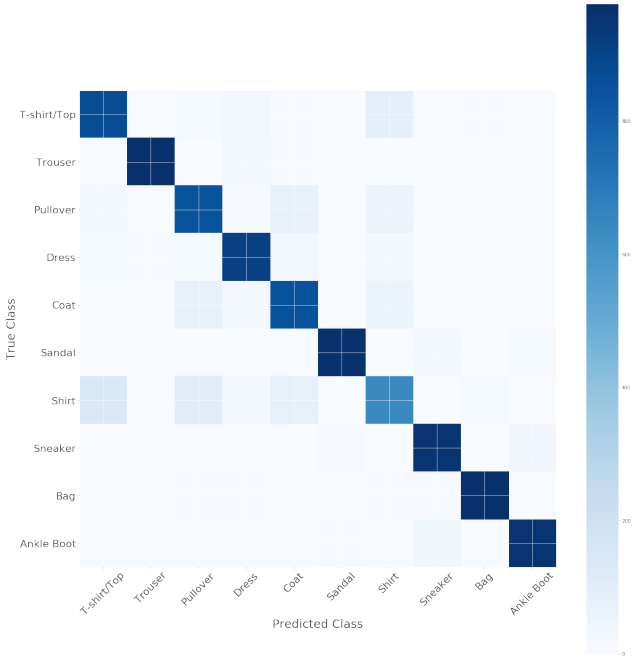

Figure 5. Confusion Matrix - XGBoost

3.3. Multi Layer Perceptron (MLP)

The multi layer perceptron model, also known as a vanilla neural network is the first model that is chosen in the class of neural networks for supervised learning and classification tasks. The multi layer perceptron model consists of input, output, and hidden layers that are connected sequentially. A loss function is evaluated and minimized in order to ensure that the predictions are as accurate as possible. The loss is propagated backwards in order to update the weights of the neural network. The loss function used is the categorical cross entropy loss function. In this case, M takes the value 10.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

In order to perform the classification task, a MLP model is designed with one input, hidden, and output layer, and trained for 20 epochs. The test accuracy obtained with this model is 0.8682. The performance of this model can be observed in the loss plot (Figure: 6), accuracy plot (Figure: 7), and confusion matrix (Figure: 8) respectively.

This model doesn't capture the spatial relations present in the neighboring pixels of an image, due to the fully connected nature of the multilayer perceptron model.

Table 4. Evaluation Metrics - XGBoost

Class Name	Precision	Recall	F1 Score
t-shirt	0.83	0.87	0.85
trouser	0.99	0.97	0.98
pullover	0.80	0.84	0.82
dress	0.89	0.91	0.90
coat	0.82	0.85	0.83
sandal	0.98	0.96	0.97
shirt	0.74	0.64	0.69
sneaker	0.94	0.96	0.95
bag	0.97	0.98	0.97
ankle boots	0.96	0.96	0.96
overall	0.89	0.89	0.89

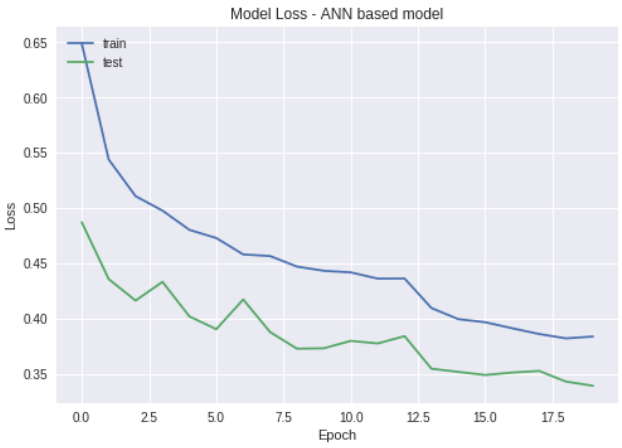

Figure 6. Plot: Loss vs Epochs - MLP

3.4. Convolutional Neural Network (CNN)

Convolutional Neural Networks are a class of feed-forward neural networks that are commonly used in image

related tasks due to their weight-sharing and spatial connectivity architecture. CNNs are often used in image classification tasks. In this project, different CNN models are evaluated in order to gain the best classification accuracy.

3.4.1 Design

To facilitate the learning of different features by the layers, each CNN layer has many filters which are spatially related. These sequential layers subsequently learn different features from the input images in order to classify those images into the known classes. Convolutional layers apply the convolution operation on the input and the output from this operation is forwarded to the next layer. Pooling layers are included to cluster the neuron outputs and reduce the dimensions of the outputs from the neurons. Fully connected layers are used to convert the 3D outputs from the hidden layers to a 1D vector in order to predict the class for each input.

Categorical cross entropy is used as the loss function in order to train the model. Rectified Linear Units (ReLUs) are used as the activation function for the neurons in the model. The output layer neurons uses softmax activation in order to predict a probability distribution of the classes for each input. The training is done on Google Colab using Keras with Tensorflow as the backend. The matplotlib and pylab packages are used to visualise the results.

Figure 7. Plot: Accuracy vs Epochs - MLP

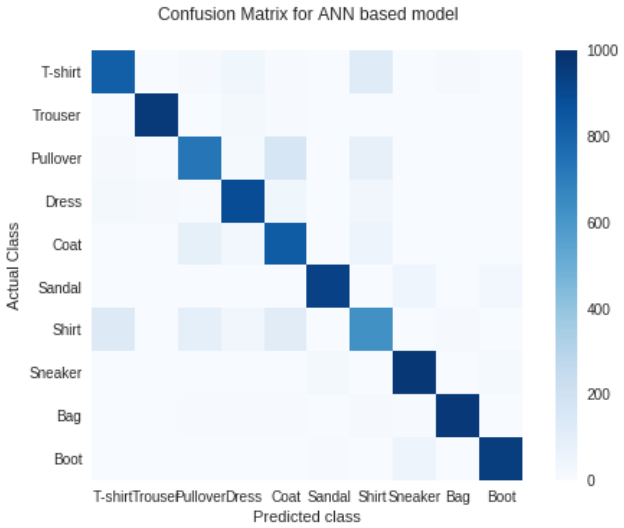

Figure 8. Confusion Matrix - MLP

3.4.2 Hyperparameter Tuning

In order to determine the various hyperparameters required to train the model, a grid search is performed using different choices of the hyperparameters. The best possible hyperparameter is chosen based on the accuracy obtained by training the model on different values of the hyperparameters. For all the hyperparameter searches, a fixed 2 layer CNN model is chosen, and the training is done over 5 epochs. The sklearn package is used to implement grid search.

3.4.2.1 Batch Size Instead of training the model with all the inputs, a subset of the inputs are chosen in order to expose the model to different variations in subsequent training cycles, and enable the model to generalize better. This is done by choosing a batch size for the model. Table 5 shows the test accuracy with different batch size values. Based on this, a batch size of 32 is chosen.

Table 5. Batch Size - Grid Search

Batch Size	Test Acuracy
32	0.895855
64	0.889982
128	0.885709
256	0.880018
512	0.872182

3.4.2.2 Optimizer Different optimization algorithms can be chosen in order to update the weights of the Convolutional neural networks using the back-propagation algorithm. Table 6 shows the test accuracy with different optimizers. Based on this, the Nadam optimizer is chosen.

3.4.2.3 Learning Rate Different learning rates can be chosen initially in order to update the weights of the model during training. Table 7 shows the test accuracy with different initial learning rates. Based on this, 0.1 is chosen as the initial learning rate. However, based on the validation accuracy performance, the learning rate is reduced to one-tenth of its value if the validation accuracy doesn't increase over three epochs of training.

3.4.2.4 Momentum Different momentum values can be chosen initially in order to assist the updation of the weights of the model during training. Table 8 shows the test accuracy with different momentum values. In the equations below, v represents the exponentially weighted past gradients. β is the momentum parameter. α is the learning rate parameter. W is the weight tensor and C is the cost. However, Keras documentation recommends that Nadam is utilized with the default values.

$$v_{dW} = \beta v_{dW} + (1 - \beta)\frac{\partial \mathcal{J}}{\partial W}$$
$$W = W - \alpha v_{dW}$$

3.4.2.5 Initializer Different momentum values can be chosen initially in order to assist the updation of the weights of the model during training. Table 9 shows the test accuracy with different initialization distributions present in Keras. Based on this, the initialization distribution in Keras known as *lecun_uniform* is chosen.

3.4.2.6 Batch Normalization Batch normalization is a technique used to enhance the performance of a model. This is done by scaling and normalizing the inputs to a layer of the neural network. It is observed that batch normalization does improve the accuracy of the model, as seen in Table 10

Table 6. Optimizer - Grid Search

Optimizer	Test Acuracy
Nadam	0.9128
SGD	0.8324
RMSProp	0.8861
Adagrad	0.9063
Adam	0.912
Adadelta	0.9102

Table 7. Learning Rate - Grid Search

Initial Learn Rate	Test Accuracy
0.1	0.84420
0.01	0.84400
0.001	0.84400

Table 8. Momentum - Grid Search

Momentum	Test Acuracy
0.0	0.8392
0.2	0.8440
0.4	0.8379
0.6	0.8442
0.9	0.8440

3.4.3 Model Architecures

Different CNN models are designed by adding or removing layers from the two-layer CNN model that was utilized in the hyperparameter evaluation. All the hyperparameters used are fixed. Each model is trained for 20 epochs with the same training and validation datasets. Each model is tested with the same testing dataset. The shape of the filter kernels in each layer is the same. The activation function used is the Rectified Linear Units activation function.

$$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$$

3.4.3.1 One CNN layer The test accuracy obtained with this model is 0.9054. The performance of this model can be observed in the loss plot (Figure: 9), accuracy plot (Figure: 10), and confusion matrix (Figure: 11) respectively.

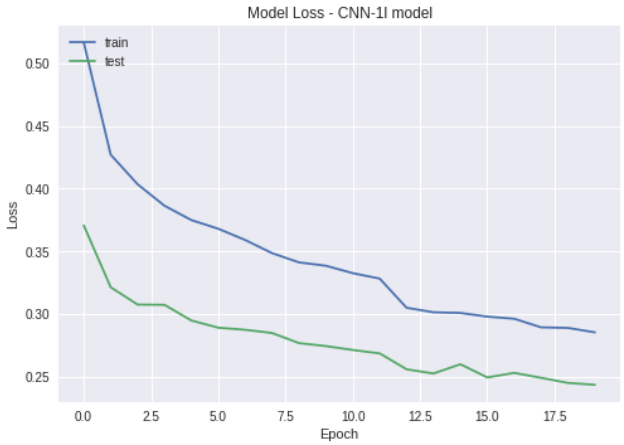

Figure 9. Plot: Loss vs Epochs - CNN 1 layer

Table 9. Initializer - Grid Search

Initializer	Test Acuracy
uniform	0.807700
lecun_uniform	0.844900
normal	0.824700
zero	0.105500
glorot_normal	0.839800
glorot_uniform	0.841400
he_normal	0.832500
he_uniform	0.828000

Table 10. Batch Normalization - Grid Search

Batch Normalization	Test Acuracy
Yes	0.8958
No	0.9208

3.4.3.2 Two CNN layers The test accuracy obtained with this model is 0.9208. The performance of this model can be observed in the loss plot (Figure: 12), accuracy plot (Figure: 13), and confusion matrix (Figure: 14) respectively.

3.4.3.3 Three CNN layers The test accuracy obtained with this model is 0.838. The performance of this model can be observed in the loss plot (Figure: 15), accuracy plot (Figure: 16), and confusion matrix (Figure: 17) respectively.

3.4.3.4 Four CNN layers The test accuracy obtained with this model is 0.8768. The performance of this model can be observed in the loss plot (Figure: 18), accuracy plot (Figure: 19), and confusion matrix (Figure: 20) respectively.

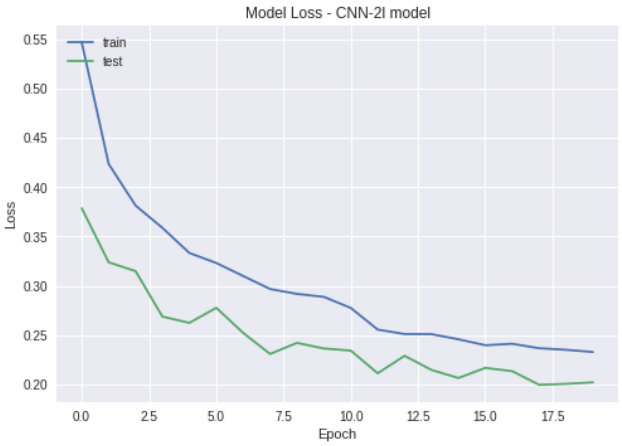

Figure 12. Plot: Loss vs Epochs - CNN 2 layer

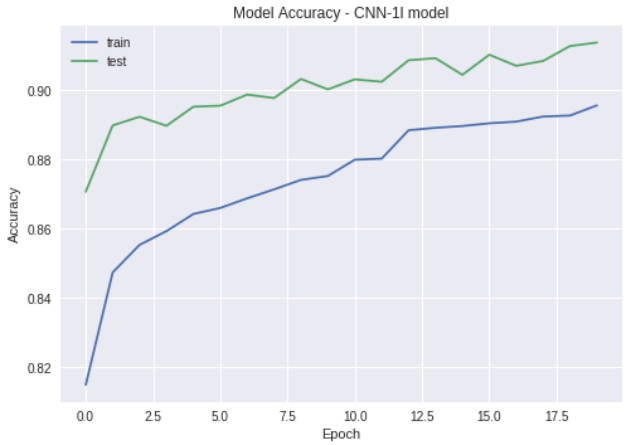

Figure 10. Plot: Accuracy vs Epochs - CNN 1 layer

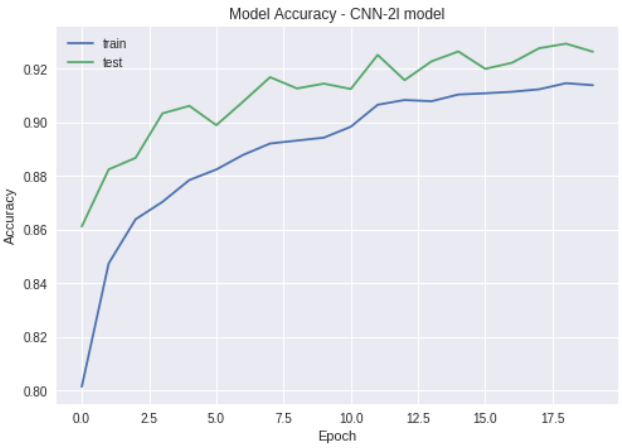

Figure 13. Plot: Accuracy vs Epochs - CNN 2 layer

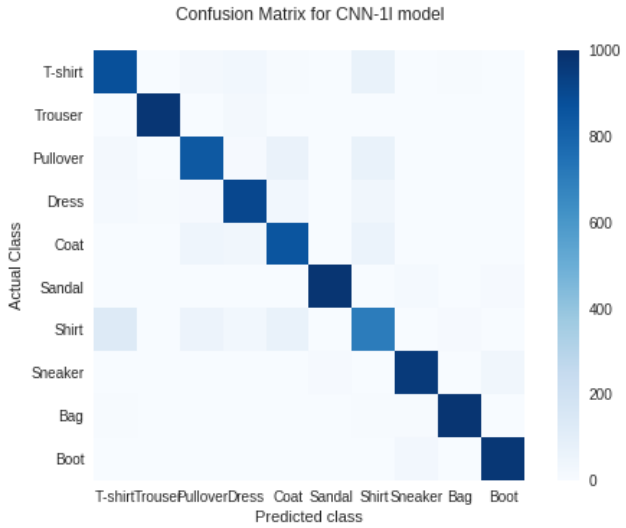

Figure 11. Confusion Matrix - CNN 1 layer

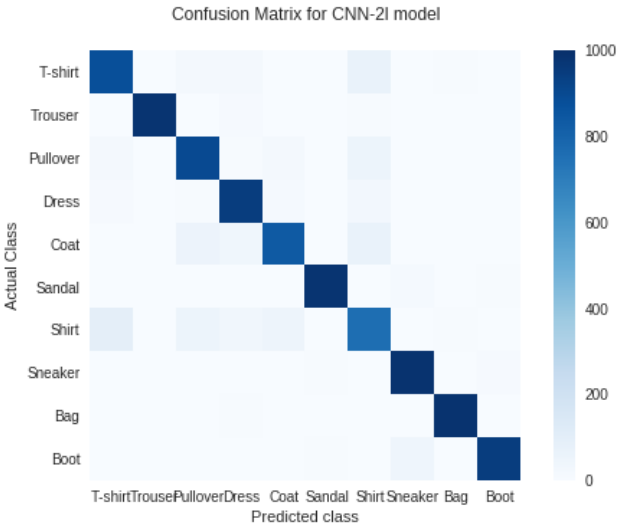

Figure 14. Confusion Matrix - CNN 2 layer

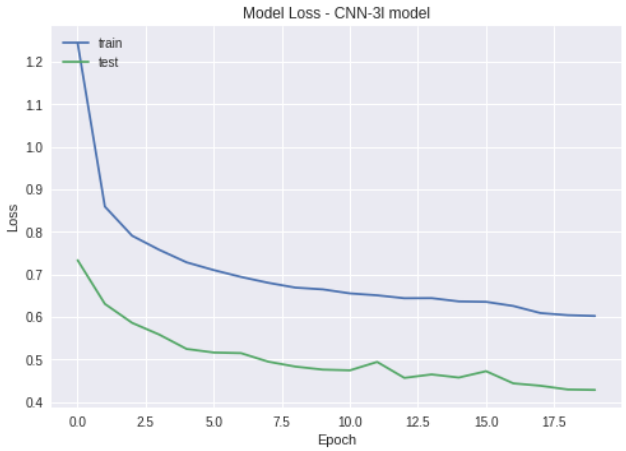

Figure 15. Plot: Loss vs Epochs - CNN 3 layer

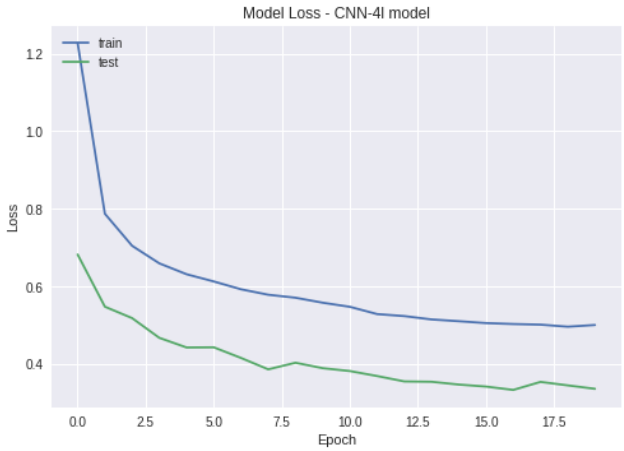

Figure 18. Plot: Loss vs Epochs - CNN 4 layer

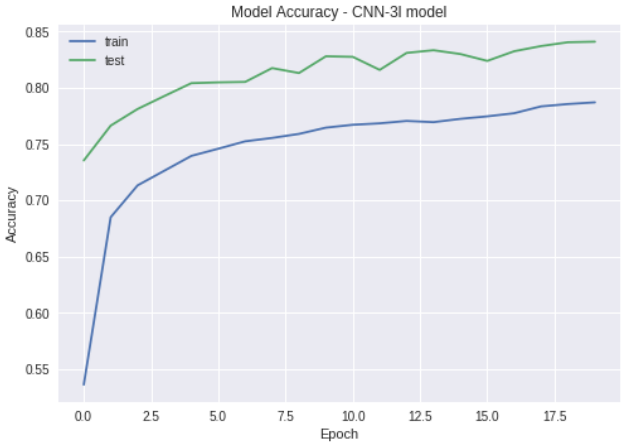

Figure 16. Plot: Accuracy vs Epochs - CNN 3 layer

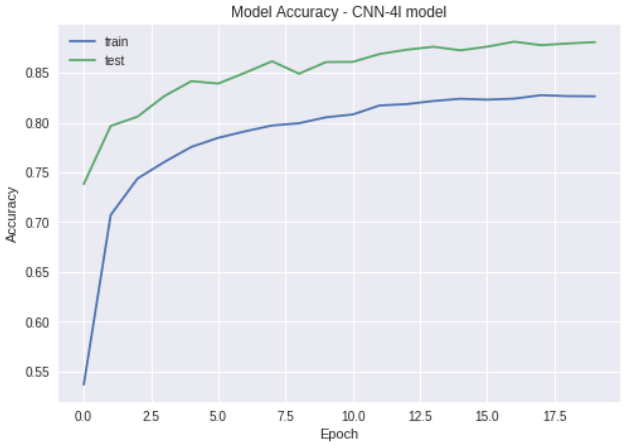

Figure 19. Plot: Accuracy vs Epochs - CNN 4 layer

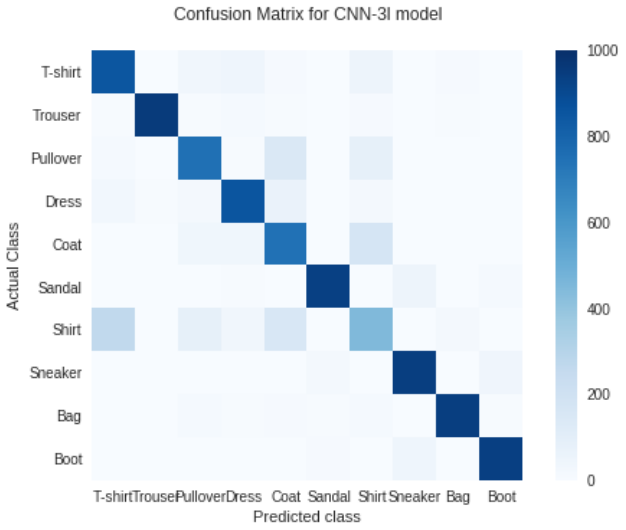

Figure 17. Confusion Matrix - CNN 3 layer

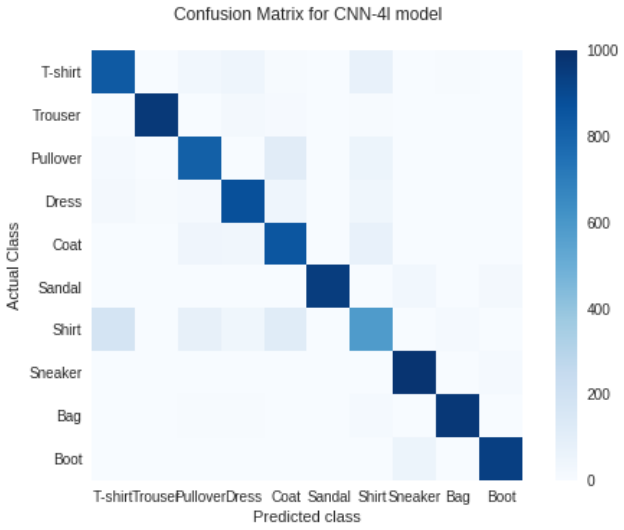

Figure 20. Confusion Matrix - CNN 4 layer

3.4.3.5 Comparison of CNN based models Based on the training of the different CNN models with hyperparameter tuning, it can be seen from Table 11 that the two layer CNN model performs the best, with a test accuracy of 0.9208 and validation accuracy of 0.9264. The larger models may perform better if trained for more number of epochs. However, this can also lead to overfitting.

Table 11. Comparison of CNN Models

Conv Layers	Validation Accuracy	Test Acuracy
1	0.9137	0.9054
2	0.9264	0.9208
3	0.8411	0.838
4	0.8809	0.8768

4. Comparison of Results

Table 12 shows the comparison of the different methods used in this project to perform classification of the Fashion-MNIST dataset. From this, it can be observed that the CNN based model performs the best.

Additionally, apart from the methods adopted, data augmentation can be utilized in order to enhance performance. Also, pretrained models can be used as implementations of transfer learning for this classification task.

Table 12. Comparison of Models

Method/Model	Test Accuracy
SVM	0.8894
Extreme Gradient Boosting	0.894
Multi Layer Perceptron	0.8682
Convolutional Neural Network	0.9208

4.1. Supplementary Material

All our code is in the form of IPython notebooks (Python 3) which can be executed at once using *Run all cells* option. Overall, we have used following python libraries and packages - keras, numpy, pickle, matplotlib, seaborn, pickle, pylab, h5py, xgboost and sklearn. The scripts are organized based on the task at hand and classification method as below:

- The *data_visualization.ipynb* is the script for exploratory data analysis and running Principal Component Analysis into 2 components.
- The *svm_classifier.ipynb* is the script for learning a Support Vector Classifier using polynomial kernel on the training data, getting the predictions on the test data and accordingly evaluating metrics and plots for visualizing the results quantitatively.
- The *xgboost_classifier.ipynb* is the script for learning a Gradient Boosted Decision Trees Classifier on the training data after doing hyper-parameter tuning on validation data, getting the predictions on the test data and accordingly evaluating metrics and plots for visualizing the results quantitatively.
- The *nn_based_classifiers.ipynb* is the script for training all the neural network based models used in this project. It has the following segments: data formatting, preliminary experiments, grid search experiments, convolutional neural network models, and the multilayer perceptron based model. It also contains the scripts and outputs to generate the loss plots, accuracy plots, and confusion matrices for different models.

References

[1] https://scikit-learn.org/stable/.

[2] https://xgboost.readthedocs.io/en/latest/python/python_api.html.

[3] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis, 1987.

[4] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.