CS273A Final Project
University of California, Irvine

FASHION-MNIST MACHINE LEARNING

March 21, 2022

Ashley Schwartz, avschwa1, 47243547
Anand Srinivasan, anands3, 57442628
Erik Urzua, urzuae, 26566236

Github Repository:
https://github.com/ashleyvsch/cs273A_project

1 Introduction

Machine learning models have become increasingly useful in a variety of applications within both academia and industry from biology to defense, and from natural language processing to modeling the stock market and many other areas. Image classification problems are interesting for their applications in object detection and image recognition for both still images and video capture. Throughout this work we will be implementing supervised machine learning techniques for which a learner must be able to map an input image of a fashion item to an output label of item type based on many different examples of input-output pairs from the Fashion-MNIST data set.

1.1 Data Exploration

The Fashion-MNIST data set contains images of 10 different fashion items. An example image of each fashion item contained in the data set is shown in Figure 1.

Figure 1: **Fashion Items.** Fashion items included within the Fashion-MNIST dataset.

Each image in the data set is comprised of 784 pixels, or a matrix of size 28x28; therefore, our data set is comprised of 784 features. This is a high-dimensional problem which is not easy to tackle for any machine learning model. Within this project, we explore a few different models but largely focus on forward feed fully connected neural networks. High dimensional problems, and large data sets, make it increasingly difficult for a model to learn efficiently. Neural networks are the most common choice to handle high-dimensional problems and their are many packages and tools available to help with computational efficiency.

2 Methods

2.1 Data preprocessing and base model design

Learning a model is inherently difficult. We can improve how well a model learns by carefully assessing the data and making choices that are best suited to the problem at hand. In preprocessing and base model design, we must take into account elements such as data set splitting, normalization, loss assessment, and optimization.

Any model can be trained and do well, but the real question is how well does it perform on a testing data set. In other words how well it does at extrapolating. For our model building, we have decided to use a training set, a validation set, and a testing set for final testing. The training set contains 60,000 images which were split by an 80/20 ratio giving 48,000 images for training, and 12,000 images for validation, and the testing set contains 10,000 images. All model explorations were done on the testing and validation so that the model never touches the testing data until final model performance testing.

Learning a model can be difficult if the data is sparse. Gray-scale images are represented by pixels that take on integer values between 0 and 255, where 0 is black and 255 is pure white. The spread of these values can introduce complications into the machine learning model, specifically in optimization. To prevent this we have normalized our data to reduce the spread. We have modified the pixels in each image to take values between 0 and 1, where 0 is black and 1 is pure white. In addition to this, we have "flattend" the 28x28 grid, or array, of normalized pixel values

into a one dimensional grid (i.e a sequential list of integer values) and used this grid as the input to each of our models. We note here that in doing so, the models are unable to detect certain features of the data image such as edges, contours and curves that could be used to differentiate the one fashion item from another such as a coat from a pullover jacket.

The data tells us we are tackling a classification problem. A common choice for assessing loss in classification is cross entropy loss due to it being convex. Cross entropy loss assesses the softmax function and makes prediction decisions based on the probability of each class while also combining the negative log likelihood loss function.

Finally, the optimizer will be the driving force in optimizing the model parameters. The optimizer must minimize the loss function. We have our choice of typical gradient descent optimizers, and adaptive optimizers. The learning rate of the optimizer is the largest difference between these choices as adaptive optimizers alter the learning rate during optimization. We have chosen to utilize the adaptive optimizer Adam so that we can investigate the role batches play in model performance and not have to focus too much on the learning rate.

2.2 Model exploration

Throughout this section, we explore model building using different machine learning libraries. We aim to analyze the differences in performance in the varying libraries, along with ease of use, and computational efficiency. We will be building a forward-feed neural network in each library for comparison. Model decisions such as optimizer and loss functions have been chosen using the structure defined in the previous section while other model parameters are chosen through exploration and discovery.

2.2.1 Model Exploration with ML Tools

The python codes from MLtools provided as part of the lecture notes were utilized in order to train and test the Fashion MNIST dataset. In particular, the kNN-classifier, decision trees and neural network architectures were implemented with the Fashion MNIST data. In each case, the training was performed with a data subset of 10,000 rows, and the validation was performed with 1,000 samples. The training and validation samples were taken as smaller batches in order to minimize on computation time.

The kNN-classifier was implemented by parametrically varying the k-value and the regularization value α. Table 1 below shows the validation accuracy as a percentage.

k-value	$\alpha = 0$	$\alpha = 1$	$\alpha = 2$	$\alpha = 4$
5	84.4	82.1	81.7	81.4
10	85.0	82.1	81.9	81.4
20	83.0	82.1	81.9	81.4
50	81.1	82.0	81.9	81.4
100	78.9	82.0	81.9	81.4

Table 1: **K-value and Regularization.** Validation accuracy with kNN classifier.

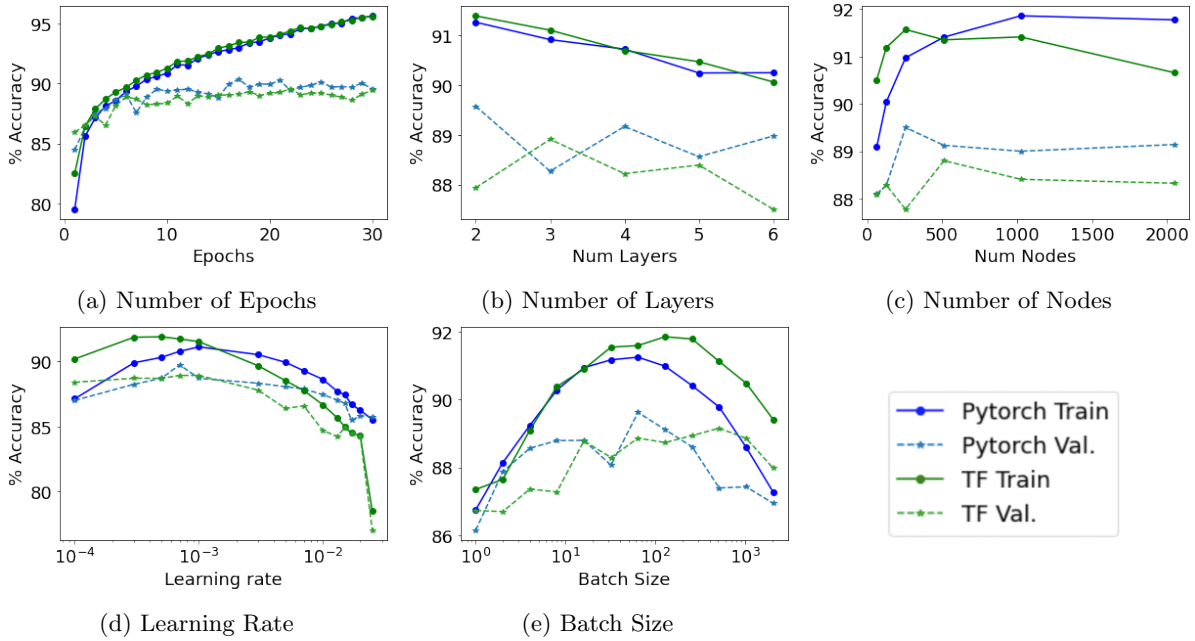
The decision tree implementation for the dataset was performed by varying the depth and minParent variables, and the results are shown in Table 2.

The neural network implementation was performed using 20,000 training samples and tested on a 1,500 sample dataset. The hyperbolic tangent activation function was used, with 500 nodes in each layer and 10 hidden layers, along with one input and output layer. The calculated validation accuracy was obtained to be 78.9 %.

From this exploration, we found that the kNN-classifier provided the best accuracy rate, with a k-value of 10, and regularization value α of zero.

depth	minParent = 2	minParent = 5	minParent = 10	minParent = 20
5	73.4	73.3	73.3	73.2
10	78.7	79.3	79.4	79.0
20	78.8	78.5	78.4	78.6
50	78.5	78.6	77.0	79.2

Table 2: **MinParent.** Validation accuracy with decision tree classifier.



(a) Number of Epochs (b) Number of Layers (c) Number of Nodes

(d) Learning Rate (e) Batch Size

Figure 2: **Parameter Exploration**. Forward feed fully connected neural network hyperparameter exploration with Keras and Pytorch.

2.2.2 Forward Feed Neural Network with Keras and Pytorch

Throughout this model exploration with Pytorch and Keras[1], we have analyzed the number of epochs needed for optimal training, number of nodes and layers within the network, the learning rate, and the batch size for optimizer parameter updates.. The different combinations of these hyperparameters can get increasingly large. We opted to look at each parameter individually and determine our best model from this analysis. The goal is to optimally choose a parameter value that performs well for both the training and validation set without over fitting the model. To prevent over fitting, we ultimately choose the parameter that maximizes the validation accuracy, even if the training accuracy is not at its maximum.

Library	Epochs	Number of Layers	Number of Nodes	Learning Rate	Batch Size
Keras	22	3	512	0.0007	512
Pytroch	17	2	256	0.0007	64

Table 3: **Final Hyperparameters**. Chosen hyperparameters for the Keras and pytorch models that prevent overfitting and maximize validation accuracy

[1]We note that Keras is a deep learning API that runs on top of the TensorFlow library. Specifically it is a the high-level API of Tensorflow and made to be more approachable by providing the essential abstractions and building blocks for developing machine learning models to run on varying platforms.

Library	Train Accuracy	Validation Accuracy	Testing Accuracy
ML KNN	85.22	85.00	NA
Keras	95.50	89.92	89.31
Pytorch	93.26	90.16	89.07

Table 4: **Final Model Results.** Comparison Of accuracy rates between libraries.

3 Results

Choosing the best machine learning model is, quite frankly, an impossible task to do. As a reminder in machine learning, no model is perfect, but some models are useful. A useful model is one that performs well on both the training and validation data sets and the model is not overfitting the training data. The final chosen models have been tested on the testing data set of 10,000 images that were not used in determining model accuracy during hyperparameter exploration. The results from the best chosen model in each library is shown in Table 4. We note here that even the best validation accuracy rate that was obtained was from the kNN-classifier was below 85% accuracy while the neural networks in pytorch and Keras are near 90%.

Throughout this project we aimed to analyze the similarities and differences between difference machine learning libraries. Throughout this process we have found that computational efficiency greatly differs between the three libraries investigated. Keras performed the fasted, utilizing the full capability of a GPU. Pytorch has a more complex architecture and a more hands-on approach to building a network model. For example, the training loop in Pytorch must be written, while Keras has these architectures pre-built. We feel that the complexity of Pytorch is at times more work than Keras as Keras has many features and functions that we had to custom build for the Pytorch predefined and written into various functions, classes, and class methods. However, one is more likely to understand each part of the Pytorch model than the parts of the Keras model by having to write and customize each of the features that were already predefined in Keras API.

Regarding our best model's classification and misclassification results, we have created a bar chart for each fashion item showing the item the model wrongly classified instead of the correct item as well as the model's accuracy on that fashion item alone. The charts can bee seen in figure 3 below. Overall, the best model has the high accuracy classifying trousers and sandals and had the lowest accuracy classifying shirts and pullover. Most often, the best model would misclassify the shirts as a T-shirt/top (over 100 times) or a pullover (80-90 times) and misclassify the a pullover as a coat (over 100 times) or a shirt (about 60-70 times). Throughout our model and data exploration we saw that the created models consistently had the lowest accuracy classifying either shirts, pullover, T-shirt/tops and coats with the majority of the misclassification for each class made up of the other three classes. This consistent trend amongst all the models is most likely down to how similar each of the four fashion items look to each other in a low-resolution grey-scale image and indicates that models created throughout are unable to pick up on the little details that differentiate the item unlike a convolutional neural network may be able to.

4 Appendix

The models developed in this project were built using the documentation of Keras and Pytorch as references. Each person was responsible for the exploration of one library: Anand – ML Tools, Erik – Keras (and Tensorflow), Ashley – Pytorch. We all worked together to decide which hyperparameters to investigate, make base model decisions, and generate joint plots.

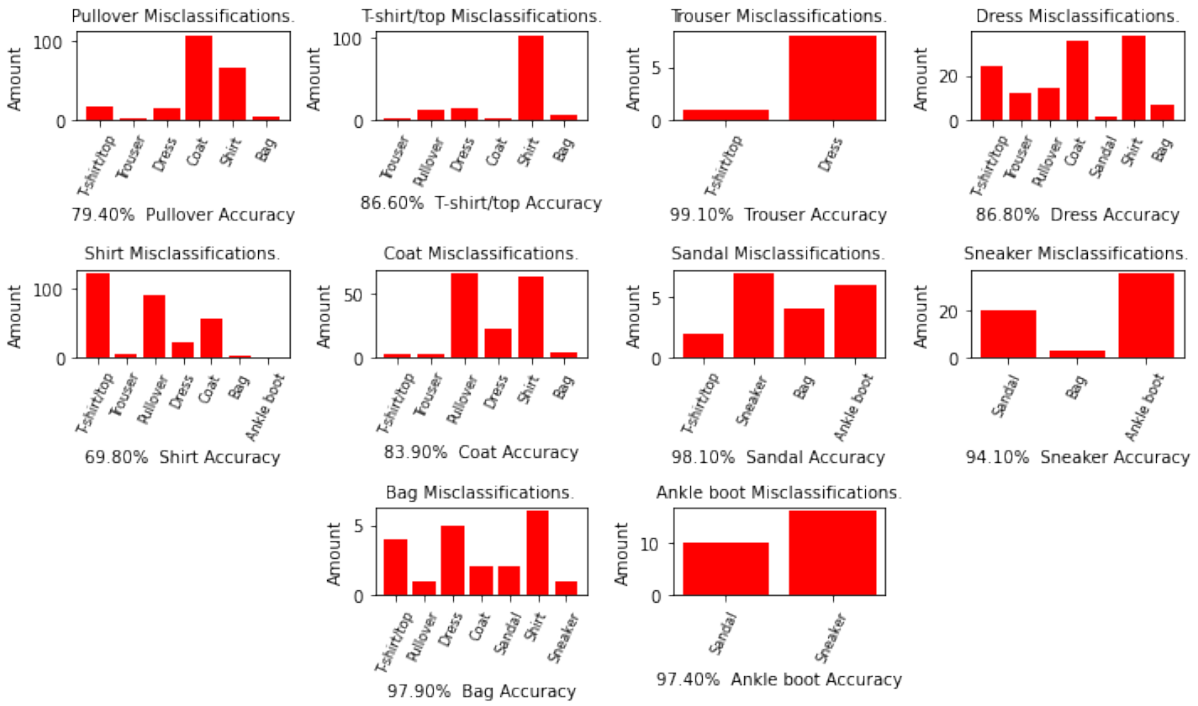


Figure 3: Classification statistics for the best Keras model. Each bar chart for each fashion item displays the items that where wrongly classified instead and the number of times so. We also give the prediction accuracy for each fashion item alone.