ECE 471: Intro to Pattern Recognition
Final Project

Fashion-MNIST

Carl Edwards, Alec Yen
University of Tennessee, Knoxville
Electrical Engineering and Computer Science Department
cedwar45@utk.edu, ayen1@vols.utk.edu

Abstract

This report focuses on the performance of several classifiers on the Fashion-MNIST dataset. Fashion-MNIST is a more difficult version of the classic MNIST benchmark image dataset. This dataset consists of 10 classes of 28x28 grayscale images of fashion items. The data is normalized and principal component analysis and Fisher's Linear Discriminant are applied. MPP cases 1, 2, and 3, k-nearest neighbors, and decision trees are evaluated on the dataset. Additionally, 3-layer backpropagation neural networks and a convolutional neural network (CNN) are also tested. Performance for these classifiers is compared using the data's built-in train/test split and using 10-fold cross validation. Additionally, k-means and winner-takes-all clustering techniques are investigated for visualizing and reproducing the clusters in the data. The CNN classifier achieves the best result of 92.9%.

1 Introduction

Sight is the sense which humans rely on the most. It's used in all facets of life from driving to recognizing objects and faces. While this is an intuitive task to a human, image recognition is much more difficult for computers. This is due to the high dimensionality of images, the large number of classes of objects, and the potential variation within classes of things.

Computer vision has applications in many areas. It can be used in transportation, such as self-driving cars [1]. Another field which can see a big, life-changing impact is medical imaging [2]. In radiology, computer vision can potentially be used to detect cancers from tissue scans [3]. Since computer vision is very important application of pattern recognition, we decided to investigate it in this project using the Fashion-MNIST dataset.

1.1 Background

The original MNIST dataset from 1998 is a popular 10 category dataset consisting of 70,000 examples of handwritten digits. It was first introduced in [4] by LeCun et al. MNIST is a modified version of handwritten data obtained from NIST, the National Institute of Standards and Technology. The black and white images from NIST were normalized into a 20x20 pixel box, which preserved their aspect ratio. This resulted in grayscale images due to the anti-aliasing technique used in normalizing the images. Finally, the images were centered in a 28x28 image based on the mass of the pixels [5]. Originally created in 1998, MNIST has become an extremely popular benchmark dataset for its ease of use in prototyping and testing new classifiers. It is particularly popular in deep learning due to it small size compared to other datasets [6]. In 2012, a record high accuracy of 99.77% was achieved using deep neural networks [7].

Fashion-MNIST was introduced in 2017 by Xiao et al.[6] in order to provide a similarly sized alternative to MNIST which poses a more challenging classification challenge.

Fashion-MNIST retains the same data structure of grayscale images as MNIST. Hence, it is possible to use it as a prototyping dataset in the same scenarios as MNIST. Since classification accuracies have become so high for the MNIST dataset, Fashion-MNIST provides a more challenging alternative which can easily replace MNIST, allowing for better comparisons between modern deep learning models.

1.2 Challenges

The non-Gaussian nature of the dataset will pose a challenge to many of the classifiers we have learned in class. Classifiers based on parametric learning will also struggle because the images will likely not follow a given model.

Another challenge is the high dimensionality of the dataset. Since each image is 28x28 pixels, the final dimensionality of the dataset is 784. Such a high number of features causes the curse of dimensionality, which results in a massive feature space on the order of $255^{784} \approx 5 \times 10^{1886}$. This can cause difficulty in training classifiers because it can cause overfitting. Additionally, the data samples are farther spread apart in the space. With greater dimensionality, care needs to be taken to reduce the dimensions in a manner that minimizes the loss of information. Dimensionality reduction will need to be employed.

The samples in Fashion-MNIST are more difficult to classify than regular MNIST. This is because the 28x28 images were downsampled from color pictures of clothing taken from the Zalando shopping catalog [6]. In the original MNIST, the pictures were downsampled from black and white pictures of the same aspect ratio. In Fashion-MNIST, articles of clothing can vary more significantly inside of classes. For example, a T-shirt can have different pictures on it. This is also partially because these fashion products come from different gender groups: men, women, kids, and neutral [6]. The vizualization in Figure 2 shows how the classes in Fashion-MNIST are more difficult to separate.

With regard to high dimensionality, one study has shown that the least squares support vector machines (LS-SVM) decision function can be approximated by a normally distributed random variable if the dimension and size of a training set is very large. This found to be the case with both MNIST and Fasion-MNIST, datasets with a large number of training samples and dimensions, despite their non-Gaussian nature [8].

1.3 State of the Art

The non-Gaussanity of the Fasion-MNIST dataset has led researchers to favor more complex classifiers, such as CNNs. A study on the dataset using CNNs found that they were able to achieve an accuracy of 92.54% using a two layer CNN with batch normalization and skip connections [9]. According to the study, this strategy reduced training time while improving accuracy.

A study on the hierarchical convolutional neural networks (H-CNN) achieved an accuracy of 93.3% using the Fashion-MNIST dataset. This study modifed an existing base model VGG19 model that achieved an accuracy of 92.9% and improved its accuracy to 93.3% using a modification to the model as H-CNN VGG19 [10].

On the Fashion-MNIST's GitHub, there is a leaderboard of some of the best classification accuracies to date. The current leader, Andrew Brock, has achieved a remarkable accuracy of 96.7% wide residual networks (WRN) in PyTorch and 8,900,000 parameters; preprocessing was done using standard normalization and augmentation [11]. He uses a novel technique of training networks in which he progressively freezes layers, thereby accelerating training [12].

Two-layer convolutional neural networks had become very successful with the original MNIST library, achieving accuracies higher than 99%. By comparison, on Fashion-MNIST, these classifiers were around 93% accurate [11].

Fashion-MNIST has the possibility to provide a more challenging dataset to machine learning researchers. By addressing some of the drawbacks of the original MNIST, Fashion-MNIST has the possibility to become the introductory dataset that people turn to. The high dimensionality and complexity could pose a significant challenge to many of the classifiers we have learned, but we hope to investigate and see which classifiers do perform well.

1.4 Project Objective

The objective of this project was to integrate and evaluate various techniques from class on the dataset in order to achieve the best performance. MPP case 1, 2, and 3, k-nearest neighbor (kNN), back-propagation neural networks (BPNN), and decision trees are tested. Additionally, a convolutional neural network (CNN) is also used to reach performance results accesible by modern deep learning. We also employ two dimensionality reduction techniques: principal component analysis (PCA) and Fisher's Linear Discriminant (FLD). This helps improve some of the classifiers and alleviates the problem of the high-dimensionality of the dataset. To evaluate the classifiers, we used a built-in training/test split of the data and also 10-fold cross-validation. Finally, we used the k-means and winner-takes-all (WTA) clustering algorithms to test if we could find the original clusters in the dataset.

2 Methodology

A comprehensive description of all algorithms used in this project follows. Some of the descriptions and formulas are referenced from past projects [13], [14], [15].

2.1 Preprocessing

Before the data can be classified it is often desirable to undergo two steps of preprocessing: normalization and dimensionality reduction. Normalization often converts the data to a scale between -1 and 1. In the case of pixel values between 0 and 255, a simple division by 255 is enough to normalize the data.

Second, the data can undergo dimensionality reduction. In many instances of pattern recognition, one may be provided many features (dimensions) for each data point. This is often referred to as the curse of dimensionality. With the addition of each dimension, we

need exponentially more training data to obtain a truer understanding of the data. Also, with more dimensions, we risk overfitting and using features that in actuality are not as important. In this paper, we investigate two methods of reduction, including Fisher's Linear Discriminant and principal component analysis.

Two methods of dimensionality reduction were implemented in this project.

- Fisher's Linear Discriminant: FLD is a supervised approach which uses the training data set to establish a projection matrix W that will best discriminate the testing data. Mathematically, given c classes with d dimensions, we reduce the number of dimensions to $c-1$ dimensions. The equation in (1) was used to calculate W.

$$\begin{aligned} S_W &= \sum_{k=1}^{K} (x - m_k)(x - m_k)^T \\ S_B &= \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T \\ W &= \max_D(eig(S_W^{-1} S_B)) \end{aligned} \tag{1}$$

- Principal Component Analysis: PCA is an unsupervised approach that aims to minimize information loss. The projection matrix P can be calculated as seen in (2). The number of dimensions m is chosen so that it does not exceed a given maximum error.

$$\begin{aligned} \Sigma_{x,dxd} &= \text{cov}(X) \\ E_{dxd} &= \text{eig}(\Sigma_{x,dxd}) \\ P_{dxm} &= \begin{bmatrix} e_1 & e_2 & e_3 & \dots & e_m \end{bmatrix} \end{aligned} \tag{2}$$

In both FLD and PCA, the projection vector W and basis vectors P are found using the training set and are then applied to the testing set.

2.2 Bayesian Discriminant Functions

After the data is preprocessed, classification can be performed. The first classification method explored in this paper are the Baysian discriminant functions $g_i(\mathbf{x})$, in which a given test sample's feature vector $\mathbf{x}$ are passed through functions for each class and the outputs are compared. Namely, the feature vector $\mathbf{x}$ is assigned to class i over class j if $g_i(\mathbf{x}) > g_j(\mathbf{x})$. In this project, we use the discriminant function seen in (3) [16].

$$g_i(\mathbf{x}) = p(\mathbf{x}|\omega_i) + \ln P(\omega_i) \tag{3}$$

Three discriminant functions were developed, each with different assumptions. This report will refer to them as Case I, Case II, and Case III. Case I represents the most simplified case, in which all of the following three assumptions were made.

- Assumption 1: The distribution is Gaussian and follows the distribution given in (4), where μ_i is the mean (average) feature column vector and Σ_i is the covariance

square matrix. Note, the discriminant functions found this way are categorized as parametric learning, because a parametric model is used.

$$p(\mathbf{x}|\omega_i) = \frac{1}{(2\pi)^2|\Sigma_i|^2} \exp -\frac{1}{2}(\mathbf{x} - \mu_\mathbf{i})^T \Sigma_i^{-1}(\mathbf{x} - \mu_\mathbf{i}) \tag{4}$$

- Assumption 2: The classes share the same distribution, as seen in (5).

$$\Sigma_i = \Sigma \tag{5}$$

- Assumption 3: The two features, x and y, were assumed to be completely independent of each other, as seen in (6), where σ^2 is the variance and I is the identity matrix.

$$\Sigma = \sigma^2 I \tag{6}$$

Using (3), we can derive three different discriminant functions.

- Case I: Using all of the previous assumptions, (7) can be derived. This is the equivalent to the minimum Euclidean distance classifier.

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2}(\mathbf{x} - \mu_\mathbf{i})^T(\mathbf{x} - \mu_\mathbf{i}) + \ln P(\omega_i) \tag{7}$$

- Case II: Using only Assumptions 1 and 2, (8) can be derived. This is the equivalent to the minimum Mahalanobis distance classifier.

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_\mathbf{i})^T\Sigma^{-1}(\mathbf{x} - \mu_\mathbf{i}) + \ln P(\omega_i) \tag{8}$$

- Case III: Using only Assumption 1, (9) can be derived. Unlike the previous cases, the decision boundary with Case III is nonlinear.

$$g_i(\mathbf{x}) = -\frac{1}{2}\ln|\Sigma_i| - \frac{1}{2}(\mathbf{x} - \mu_\mathbf{i})^T\Sigma_i^{-1}(\mathbf{x} - \mu_\mathbf{i}) + \ln P(\omega_i) \tag{9}$$

2.3 Non-Parametric Learning

K-nearest neighbors (kNN) performs classification without assuming a model. With kNN, a test sample's Euclidean distance from each training sample is measured. The k nearest training samples' class labels are found and the majority is assigned to the test sample. The posterior probability of a tests sample is justified as in (10), where k_i are the k closest training samples of class i. n_i is the total number of training samples of class i and n is the total number of training samples, and thus $\frac{n_i}{n}$ represents the prior probability. Thus, kNN assumes a prior probability based on the distribution of the training data.

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} = \frac{\frac{k_i/n_i}{V}\frac{n_i}{n}}{\frac{k/n}{V}} = \frac{k_i}{k} \tag{10}$$

2.4 Clustering

2.4.1 K-Means Clustering

K-means clustering is one of the more popular unsupervised clustering techniques. In this iterative technique, a certain number of classes (or clusters) k is assumed in a dataset. k cluster means (centroids) are arbitrarily chosen to begin with. For each sample, the nearest cluster mean is found. For each cluster mean, a new cluster mean is recalculated using the nearest data samples [16]. The algorithm is outlined in Algorithm 1.

Algorithm 1 k-Means Clustering

```
x : {x_1, x_2, ..., x_n}                          ▷ data x has n elements
μ : {μ_1, μ_2, ..., μ_k}                          ▷ cluster centers μ has k elements
while μ_i changes do                               ▷ Stop when μ_i does not change
    for all n in x do
        find nearest μ_i
    recalculate all μ_i                            ▷ Based on which x are nearest to it
return μ = {μ_1, μ_2, ..., μ_k}
```

2.4.2 Winner-Take-All

Winner-takes-all is a similar technique to the k-means algorithm with a small modification. In this process, when determining the closest cluster μ_i to a data sample x, the cluster is also pulled towards x using the equation in (11). ϵ is the "learning parameter" and is typically a small value, on the order of 0.01.

$$\mu_i^{new} = \mu_i^{old} + \epsilon(x - \mu_i^{old}) \tag{11}$$

The new algorithm is shown in Algorithm 2. This is a form of online learning, since the cluster centers μ_i are changing as one loops through each of the data samples.

Algorithm 2 Winner-Takes-All Clustering

```
x : {x_1, x_2, ..., x_n}                          ▷ data x has n elements
μ : {μ_1, μ_2, ..., μ_k}                          ▷ cluster centers μ has k elements
while μ_i changes do                               ▷ Stop when μ_i does not change
    for all n in x do
        find nearest μ_i
        μ_i^new = μ_i^old + ε(x − μ_i^old)
    recalculate all μ_i                            ▷ Based on which x are nearest to it
return μ = {μ_1, μ_2, ..., μ_k}
```

2.5 Decision Trees

Decision trees are a non-statistical approach to pattern classification. All samples start at the root node and are divided up and classified as one progresses through the tree. At each node N, a property query is made to distinguish each sample into two groups. The objective of decision trees is to maximize the change in impurity from each node to the next layer. This change in impurity is defined as in (12). Decision trees in the project were implemented using `scikit-learn`'s `DecisionTreeClassifier` class [17].

$$\Delta i(N) = i(N) - P_L i(N_L) - (1 - P_L) i(N_R) \tag{12}$$

2.6 Backpropagation Neural Networks (BPNN)

The fundamental building block of neural networks is the perceptron (a single layer network), which are inspired by the neurons of the human brain. They are composed of inputs $\vec{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_d & 1 \end{bmatrix}$ and weights $\vec{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_d & -w_0 \end{bmatrix}$, where w_0 is the bias. The output $z = \vec{w}^T \vec{x}$. If $z > 0$, the perceptron outputs 1. Otherwise, it outputs 0. Assuming the ground truth is $\vec{T}$ and the network's output is $\vec{z}$, gradient descent can be used as in (13) to converge to a solution [18].

$$\vec{w}^{k+1} = \vec{w}^k + \sum_{i=1}^{n} (T_i - z_i) x_i \tag{13}$$

A BPNN is simply the combination of these perceptron units with the addition of backpropagation. In this project, Keras was used to implement the BPNN [18].

2.7 Convolutional Neural Networks (CNN)

Convolutional neural networks are formed mainly from three types of layers: convolutional, max pooling, and fully-connected layers. The first stages of convolutional nets consist of max pooling and convolutional layers [19]. First, convolutional layers attempt to extract features by sliding a filter over the previous layer. Next, max pooling is used to merge semantically related features together [20]. These layers are often used together in order to extract more and more complex features from the original image. After convolutional and max pooling layers have been applied, fully-connected layers are used to predict the class of the data sample. In this project, Keras is used to implement the CNN [18].

2.8 Classifier Fusion

Classifier fusion is a classifier in it of itself, as it uses the classification results of other classifiers to classify test samples. In this project, we use Naive Bayes combination. It takes the confusion matrices of the classifiers to be fused and creates a lookup table. The lookup table is created by performing pair-wise multiplication of the confusion matrices' columns.

An example lookup table is shown below, where a column with header "12" means that Classifier 1 predicted Label 1 whereas Classifier 2 predicted Label 2. The values in that column indicate the posterior probabilities of each corresponding label. The MPP is chosen as the label by the classifier fusion.

	11	12	21	22
Label 1	0.7	0.3	0.1	0.1
Label 2	0.1	0.2	0.2	0.8

Table 1: Example of classifier fusion lookup table using Naive Bayesian

For example, if given a test sample that classifier 1 believed to belong to label 1 but that classifier 2 believed to belong to label 2, according to the lookup table, the fusion classifier will assign it to Label 1, since $0.3 > 0.2$.

2.9 M-Fold Cross Validation

M-fold cross validation is a very useful form of performance evaluation. It serves not to help build a classifier, but rather to evaluate a classifier's performance and maximize the data available. In this process, the dataset is partitioned into m sets. $m-1$ sets are used for training the classifier while the remaining 1 set is used for evaluating the classifier. The process is then repeated m times so that each set is used for testing at least once. The overall accuracy of the classifier is the average of the accuracies found on each of the m test sets. For our experiments, a value of $m = 10$ was often used.

3 Experiments and Results

3.1 Dataset

The Fashion-MNIST dataset consists of 28x28 images, like the original MNIST dataset. Mimicking MNIST's large magnitude of data samples, the Fashion-MNIST dataset consists of 60,000 testing samples and 10,000 training samples. The 28x28 images translate to 784 features. Each pixel, valued between 0 and 255 grayscale, is considered a feature. There are 10 classes total: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot. The data samples are uniformly split between the ten classes for both training and test sets.

The 28x28 images were obtained through the following process. Original color images of the different items were converted to PNGs and trimmed at the corners. The longest edge was then resized to 28 pixels and the pixels were sharped. The shorter edge was then extended (adding whitespace) to 28 pixels. Finally, the image was negated and converted to grayscale [6].

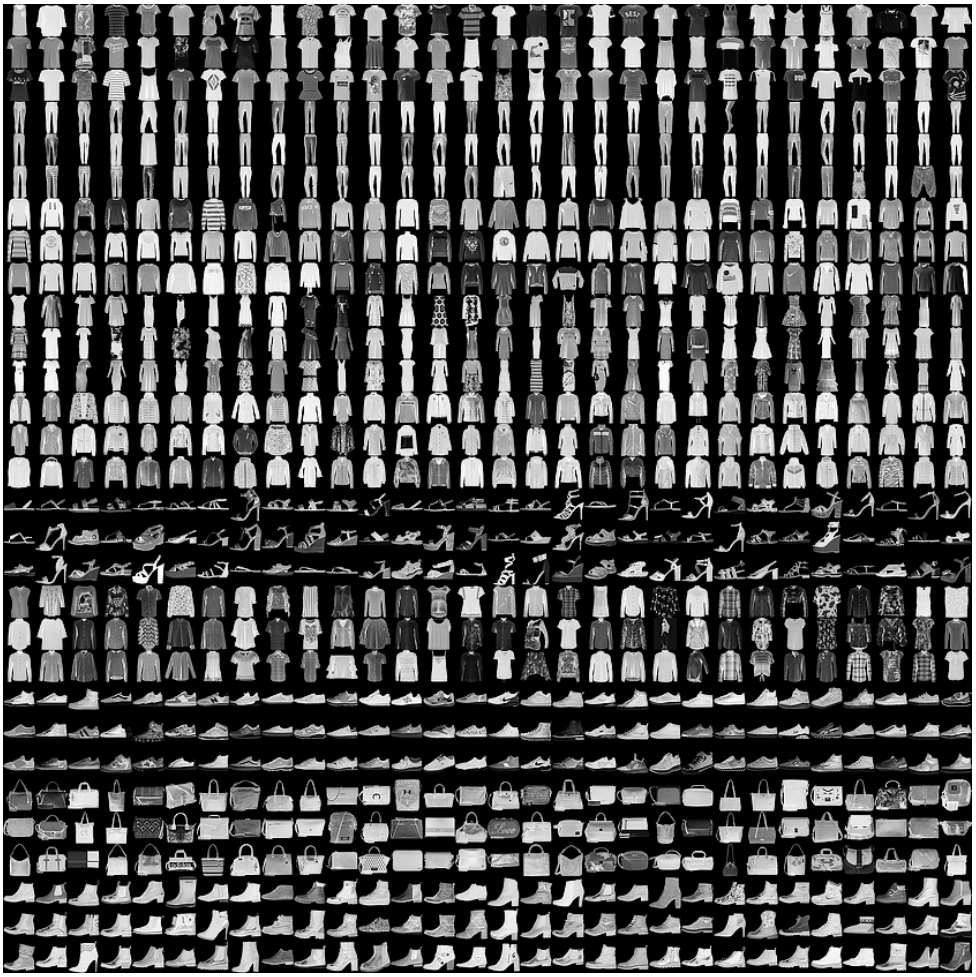
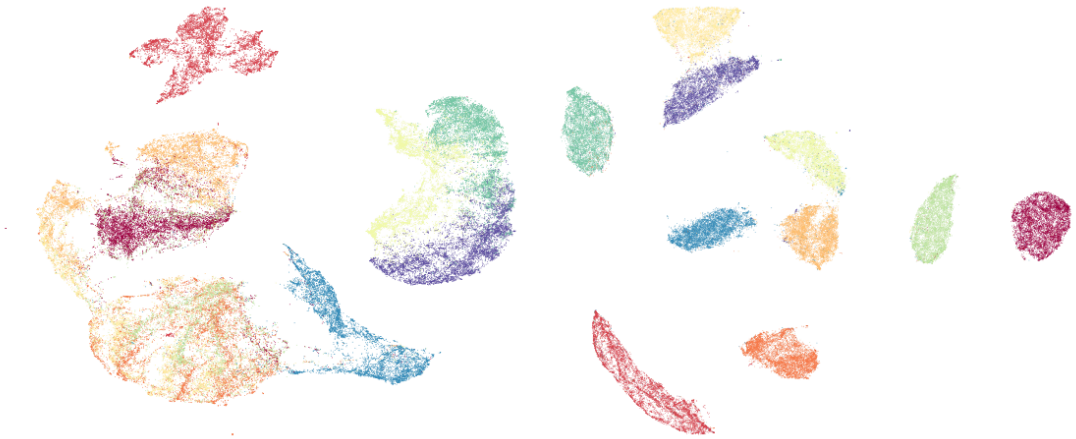


Figure 1: These example pictures [21] show various examples of each category from the dataset.



(a) Fashion-MNIST (b) MNIST

Figure 2: Vizualization of MNIST and Fashion-MNIST using Uniform Manifold Approximation and Projection (UMAP) from [21].

3.2 Summary of Results

Several different classifiers are evaluated on the dataset. These results are presented in Table 2. Among these classifiers, CNN achieves the highest accuracy with a value of 92.9%. Additionally, these values are compared in Figure 3. It is important to note that the results from k-means and WTA are not directly comparable to the other classifiers. This is further discussed in Section 3.6.

Accuracy							
				10-Fold			
Classifier	Norm	PCA	FLD	Norm	PCA	FLD	Notes
Case 1	0.6768	0.6759	0.7906	0.6857	0.6851	0.804	
Case 2	0.815	0.7951	0.8151	0.8242	0.8034	0.8324	
Case 3	0.7242	0.8072	0.8064	0.7279	0.8084	0.8173	
kNN 5, 1	0.8623	0.8636	0.8203	0.8632	0.867	0.8335	k=5, p=1 (Manhattan)
kNN 5, 2	0.8554	0.8603	0.8203	0.8569	0.8633	0.8334	k=5, p=2 (Euclidean)
kNN 5, 3	0.8402	0.8544	0.8182	0.8436	0.859	0.832	k=5, p=3
kNN 5, ∞	0.6366	0.8365	0.8144	0.6406	0.8434	0.8271	k=5, p=∞
kNN 10, 2	0.8515	0.8619	0.8264	0.8554	0.8643	0.8404	k=10, p=2
kNN 20, 2	0.8415	0.8541	0.829	0.8459	0.8579	0.8425	k=20, p=2
kNN 50, 2	0.8262	0.843	0.8288	0.8315	0.8465	0.8427	k=50, p=2
kNN 100, 2	0.8164	0.8314	0.8263	0.8184	0.8346	0.8401	k=100, p=2
kNN 250, 2	0.7949	0.8121	0.8221	0.7986	0.8157	0.8377	k=250, p=2
k-means	0.6115	0.5943	0.8996				10 clusters
WTA	0.6086	0.5937	0.8997				10 clusters
BPNN 5	0.848	0.8155	0.8065	0.8492			5 hidden nodes
BPNN 8	0.8593	0.8498	0.8092	0.852			8 hidden nodes
BPNN 10	0.8573	0.8602	0.8079	0.8502			10 hidden nodes
BPNN 15	0.8435	0.8474	0.8129	0.8543			15 hidden nodes
Decision Tree	0.7887	0.7694	0.7663	0.7952			from sklearn
CNN	0.9287						based on Keras

Table 2: Accuracy results from given train/test splits and 10-fold cross validation. 10% of the training set is used as a validation set for BPNN and CNN. BPNN is a 3-layer neural network implemented in Keras. The CNN is based on code from Keras.

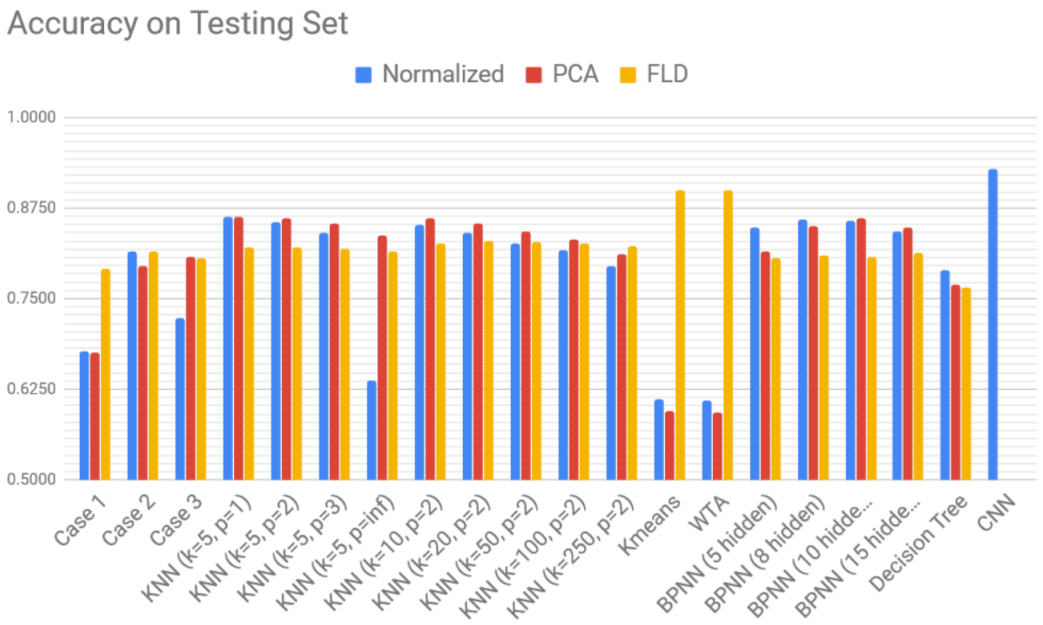


Figure 3: Comparison of classifiers accuracies on the testing set. Also includes kmeans and WTA algorithms accuracies for finding the original clusters.

3.3 Dimensionality Reduction

Each of the classifiers was evaluated on three differently preprocessed datasets (normalized, normalized and PCA reduced, and normalized and FLD reduced). We set a maximum allowed error 0.1 for PCA, and the result reduced the original 784 dimensions to 85 dimensions. FLD reduced the number of dimensions from 784 to 9.

From Table 2, different classifiers favored different datasets. The discriminant functions and clustering did the best with FLD, whereas kNN performed with highest accuracy using PCA. The neural networks had varying results with good performance with only normalized and with PCA, but overall did not favor FLD.

The manner in which the dimensions were reduced had a strong influence on the accuracy. Since FLD aims to best discriminate the data, the linear classifiers and clustering algorithms, both of which rely heavily on the position of the data in the hyperspace, had a stronger performance.

3.4 Discriminant Functions

The discriminant functions were implemented as discussed previously and the label with the highest posterior probability, per MPP, was chosen.

The discriminant functions attained a surprisingly high accuracy, given the challenging problem of image classification, averaging around 70-80% accuracy. This high accuracy validates the findings of the [8], who predicted the normal distribution of the data, given the large number of training and testing samples. This demonstrates that classifiers perform well under the assumption of a Gaussian distribution. It is also interesting to note that for the linear classifiers, cases 1 and 2, FLD performs best. This is likely because the goal of FLD is the best at discriminating the data; it aims to best separate the classes.

3.5 k-Nearest Neighbors

The kNN classifier was tested with different parameter sweeps. First, different values of k were used: $k = 5, 10, 20, 50, 100, 250$. Different orders p of the Minkowski distance metric were also used: $p = 1, 2, 3, \infty$. $p = 1$ corresponds with Manhattan distance, and $p = 2$ corresponds with Euclidean distance.

Despite this strong performance of parametric learning previously, non-parametric learning using kNN still outperformed all of the discriminant functions, averaging accuracy of around 85%. The accuracy of kNN was highest using $k = 5$ and using Manhattan distance (p=1). As the order of the Minkowski distance increased, the accuracy decreased; this was true, regardless of whether it was PCA or FLD reduced. Larger values of k also did worse - in fact, the highest accuracy was achieved using the smallest value of k tested, $k = 5$.

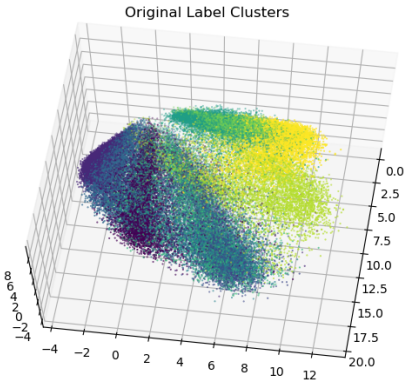
3.6 Clustering

K-means and winner-takes-all (WTA) were tested on the 10,000 testing samples and tested to form 10 clusters from the data. Once these clusters were formed, the most common testing label in each was assigned to the entire cluster. The accuracy of this approach is what was reported in Table 2.

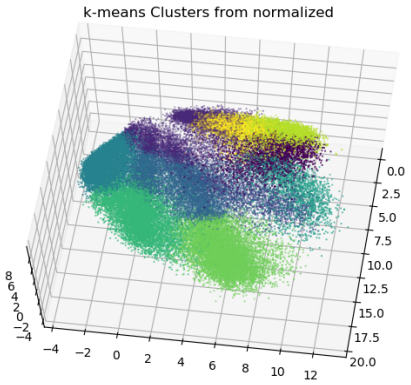
This approach of clustering using FLD was incredibly high, achieving an accuracy of almost 90%. The accuracies using the normalized and PCA reduced datasets was significantly worse, averaging at around 60% accuracy. This suggests that the supervised dimensionality reduction approach of FLD is much better suited for clustering, since its goal is to best discriminate the data. The high performance using FLD is an expected result and is validated by our results.

Although our clustering algorithms achieve excellent results, they are not directly comparable to the other classifiers. Instead, the clustering algorithms show that high percentages of original clusters can be found using the structure of the data.

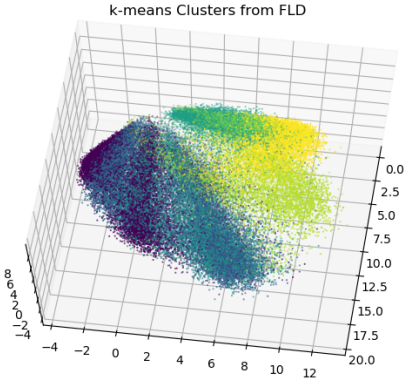
The clustering algorithms were also used visualized by using PCA (Figure 4). Note that the clustering algorithms do not have labels, so the colors are different. Additionally, the clusters created from the 2-dimensional PCA representation do not capture the classes well as well as the clusters created from the 784-dimensional normalized data.



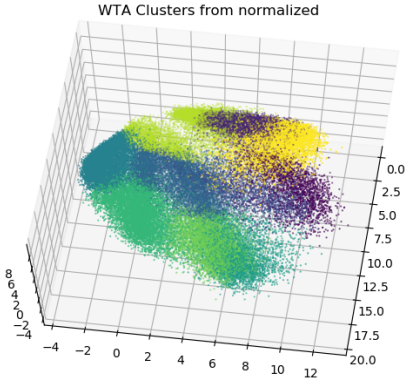
(a) Original Clusters.



(b) Clusters obtained from using k-means on normalized dataset.



(c) Clusters obtained from using most-common-label assignment method on k-means from FLD. For example, note the left cluster is incorrectly classififed.



(d) Clusters obtained from using WTA on normalized dataset.

Figure 4: Clusters from original labels, k-means, and WTA vizualized visualized on PCA.

3.7 Decision Trees

Decision trees were implemented using `scikit-learn`'s `DecisionTreeClassifier` class [17].

The decision tree did slightly worse than the discriminant classifiers, achieving a max accuracy of 79% using the normalized dataset. The decision tree algorithm did not seem to benefit from dimensionality reduction and seemed to struggle overall. These challenges suggest that decision trees are not a strong classifier for purely numerical data, and perhaps are better suited for categorical or qualitatively-defined data.

3.8 BPNN and CNN

The BPNN and CNN worked very well. They are implemented in Keras [18] using a Tensorflow 1.13 backend running on a NVIDIA GeForce 1070 GPU with 6.1 compute power. Both use a validation split of 10% with early stopping using validation loss. Additionally, batch sizes of 128 are used. Early stopping is given a patience value of 4. A convergence curve for CNN can be seen in Figure 5.

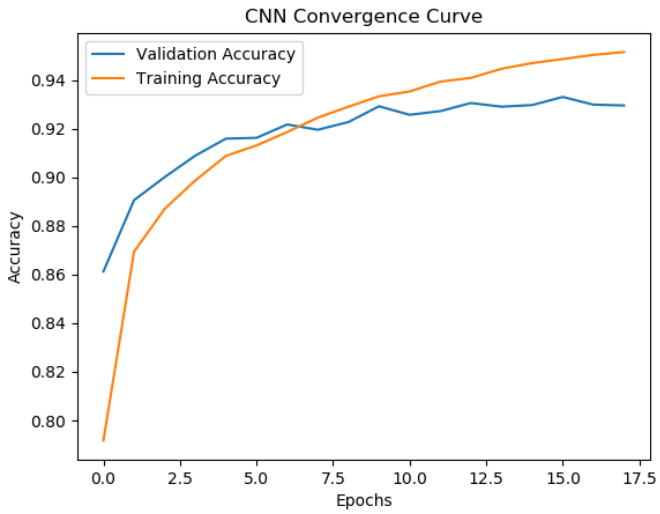


Figure 5: A convergence Curve for CNN.

In this project, BPNN was implemented as a 3-layer neural network with variable hidden nodes. Standard gradient descent was used as an optimizer, and the input and hidden layers used ReLU for an activation function. Values of 5, 8, 10, and 15 were tested. As can be seen in Table 2, BPNN with 10 hidden layers achieved the highest accuracy of 86%. Performance between the normalized data and PCA seemed to be similar for all the BPNN's. Sometimes the BPNN PCA performed better and sometimes the normalized dataset did. However, FLD generally performed worse. This is perhaps because the neural network model has the capability to handle the additional information present in the higher-dimensional original normalized dataset and PCA dataset. This hypothesis is supported by the fact that the simplest classifiers, case 1 and 2, perform better using FLD without the excess information in the data.

The architecture of the CNN (see Figure 6) is based on the Keras MNIST example [18]. First, two 2D convolutional layers, with 32 and 64 filters respectively, are applied. Both layers use 3x3 kernels. Next, a 2x2 max pooling layer is applied. A dropout of 25% is applied and the output is flattened. This is fed into a 128 node dense layer. 50% dropout is applied and a final dense layer is used to produce class predictions using softmax. All other layers use ReLU as an activation function. The network is optimized with Adadelta using categorical crossentropy.

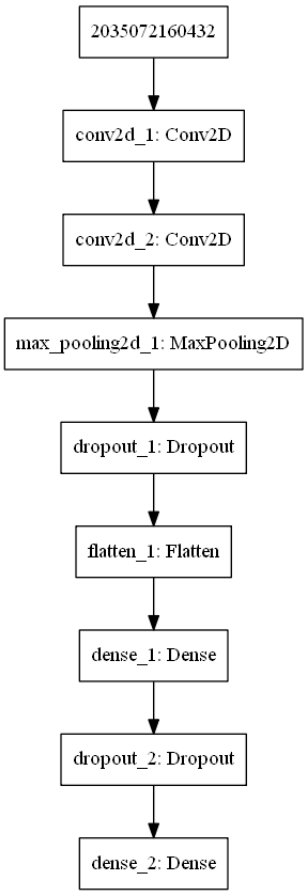


Figure 6: Keras generated visualization of the CNN. 203507... is the unnamed input.

The CNN achieved an accuracy of 92.9%, which is several percent better than other classifiers! This was expected, since convolutional neural networks are a modern deep learning approach which have led to drastic improvements in computer vision. However, our own experiments confirm that even with a small convolution net, performance is significantly better than traditional pattern classification techniques, including backpropagation neural networks.

3.9 Classifier Fusion

Classifier fusion was implemented by utilizing both the confusion matrices and the predicted labels from the classifiers we wanted to fuse. The confusion matrices were combined to form the lookup table using the Naive Bayes method previously discussed. The predicted labels determined which column of the lookup table to use and gave us an accuracy for our classifier fusion algorithm.

We found that, in general, the fusion algorithm did not improve the accuracy of the strongest classifier (see Table 3). For example, using the CNN and two other strong

networks, our fused accuracy was only 90.1%, slightly worse than the CNN. This could indicate that the misclassified test samples were often shared between the classifiers and thus classifier fusion did not benefit the best classifier's accuracy.

However, using just BPNN and kNN, we did achieve a higher accuracy than with either of the individual classifiers alone (about 86% each). We were inspired to try this combination by looking at Figure 7. We noticed that the two algorithms were better at classifying different categories, so fusing them provides a better result of 87.1%.

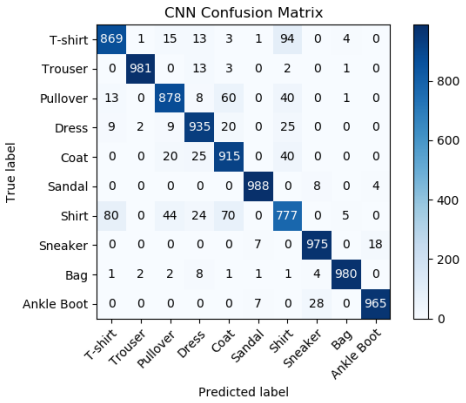
Classifier Fusion			
Classifiers			
1	2	3	Acc
CNN	BPNN Norm (8 hidden)	kNN PCA (k=5, p=2)	0.9014
Case 2 PCA	kNN PCA (k=5, p=2)	Decision Tree Norm	0.8561
CNN	Case 3 FLD	BPNN PCA (10 hidden)	0.8996
BPNN PCA (10 hidden)	kNN PCA (k=5, p=1)		0.8713

Table 3: Accuracy results from classifier fusion.

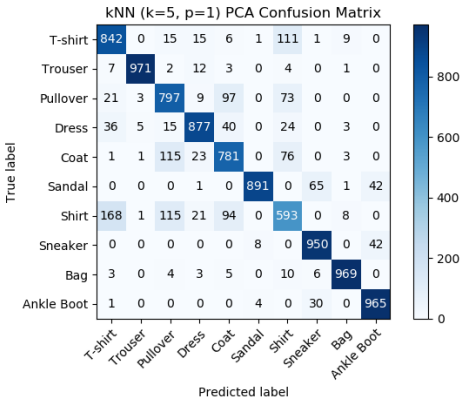
3.10 M-Fold Cross Validation

For all of our m-fold cross validation, we set $m = 10$. Our cross validated accuracies matched the trend of the testing set accuracy fairly well (see Table 2), indicating that the testing and training set are relatively unbiased.

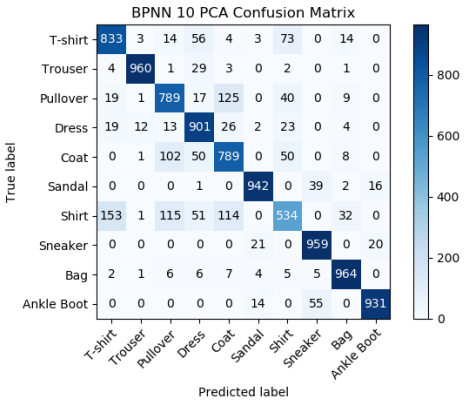
3.11 Common Misclassifications



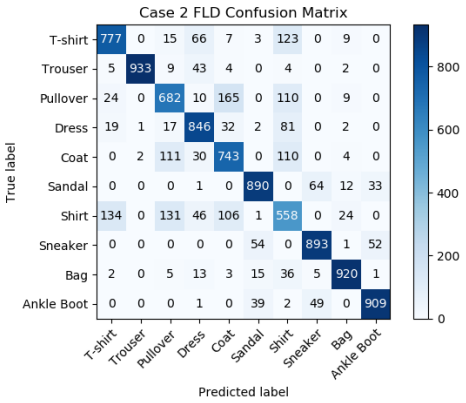
(a) Confusion Matrix for CNN.



(b) Confusion Matrix for kNN (k=5, Manhattan).



(c) Confusion Matrix for BPNN (hidden = 10).



(d) Confusion Matrix for Case 2 FLD.

Figure 7: Confusion matrices for various classifiers.

From observing the confusion matrix for CNN in Figure 7, it is clear that the most commonly misclassified label is the shirt class. It is commonly confused with the T-shirt, pullover, and coat classes. These are difficult classifications to make, even for humans. Some common misclassifications are shown in Figure 8. For example, the boot in the top right is classified as a sandal. This is perhaps due to the apparent black hole in the middle. Additionally, coats, pullovers, and shirts are often confused with each other. Even by examining the shapes of these clothes items with the human eye, the distinction is hard to find. This shows why Fashion-MNIST is more difficult than MNIST. In order to distinguish some of these classes, specific features must be extracted that differentiate - for example, T-shirts from shirts.

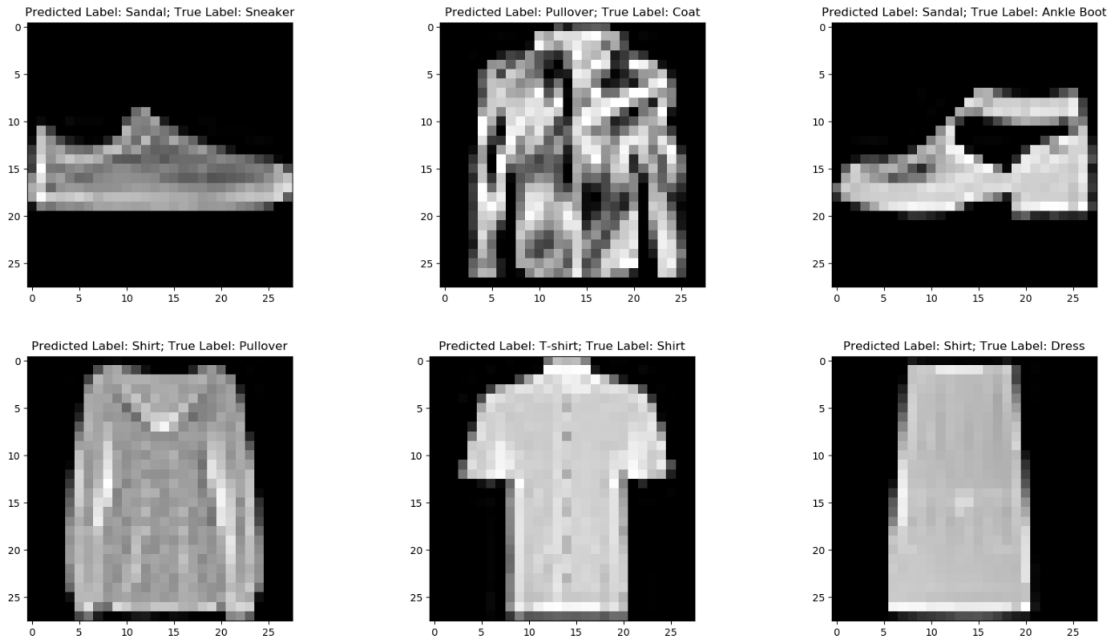


Figure 8: Examples of misclassified pictures.

3.12 Comparison To State of the Art

While the models from class perform better than expected, they still fall short of modern deep learning approaches. The CNN we implemented comes the closest to the state of the art approach: it achieves nearly 93% accuracy. This is comparable to other simple convolutional neural networks from the literature. From the algorithms implemented in class, kNN and BPNN both achieve 86% accuracy in certain cases.

Classifier Fusion	
Classifier	Acc
State of the Art using Wide Residual Networks [12]	96.7 %
CNN with Normalized Data	92.9 %
kNN with PCA using k=5 and Manhattan distance	86.4 %
BPNN with PCA using 10 hidden nodes	86.0 %
Case II with FLD	81.5 %

Table 4: Accuracy results from classifier fusion.

4 Summary

In this project, we evaluated the accuracy of the classifiers we've learned in class to the Fashion-MNIST dataset, a dataset created with the intention of providing a more

challenging classification problem than the original handwritten digits MNIST.

We implemented normalization, principal component analysis (PCA), and Fisher's linear discriminant (FLD) to create three preprocessed datasets, each with 10,000 testing samples: normalized, normalized with PCA, and normalized with FLD. These three test sets were classified using the Case I, II, and III discriminant functions, k-nearest neighbors (kNN), k-means, winner-takes-all (WTA), backpropagation neural network (BPNN), decision trees, and convolutional neural networks (CNN). 10-fold cross validation was also performed to evaluate each classifier's performance.

The best classifier we obtained was the CNN, achieving an accuracy of nearly 93%. Using techniques learned in class such as BPNN and kNN, we achieved over 86% accuracy. Using FLD + k-means and WTA, we also managed to group 90% of the samples into clusters which resembled the original dataset.

Since many of these algorithms were implemented in past projects, implementing them was not of particular difficulty. The most challenging implementations were classifier fusion. Additionally, we used the GPU version of Tensorflow, which provided significant speedups but was difficult to set-up and configure. Although there were not too many challenges in the implementations of the classifiers, this project's challenge was applying them to hundreds of features. We saw some of the simpler algorithms such as case 1 and 2 have difficulty with larger number of features. They saw performance improvements using FLD, which reduced the number of features to 9. Additionally, this dataset was challenging due to its content. The classes, such as shirt, T-shirt, and coat, were difficult to distinguish. Even within a classes, there was potentially substantial variation between samples. For example, one t-shirt might look very different from another, especially after being converted to grayscale.

This project has allowed us to better understand the struggles in computer vision and the modern machine learning techniques for approaching these problems. In future work on this dataset, we would further investigate the usage of CNNs and other deep learning technique. Although methods from class performed well, they still perform significantly worse than the CNN and deep learning techniques in the literature. Although we did not achieve state of the art results, we achieved a high accuracy using a deep learning technique (convolutional neural networks). This came much closer to the state of the art than other classifiers. This will undoubtedly be an invaluable experience, especially as the importance and prominence of computer vision continues to increase.

Contributions

Alec Yen: MPP Cases, kNN, creating clusters with kmeans and WTA, m-fold, PCA and FLD, NB classifier fusion
Carl Edwards: BPNN, CNN, Decision Tree, creating visualizations with kmeans and WTA, PCA and FLD

References

[1] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[2] M. N. Wernick, Y. Yang, J. G. Brankov, G. Yourganov, and S. C. Strother, "Machine learning in medical imaging," *IEEE signal processing magazine*, vol. 27, no. 4, pp. 25–38, 2010.

[3] S. Wang and R. M. Summers, "Machine learning and radiology," *Medical image analysis*, vol. 16, no. 5, pp. 933–951, 2012.

[4] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[5] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[6] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 08 2017.

[7] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.

[8] Z. Liao and R. Couillet, "A large dimensional analysis of least squares support vector machines," *IEEE Transactions on Signal Processing*, vol. 67, no. 4, pp. 1065–1074, Feb 2019.

[9] S. Bhatnagar, D. Ghosal, and M. H. Kolekar, "Classification of fashion article images using convolutional neural networks," in *2017 Fourth International Conference on Image Information Processing (ICIIP)*, Dec 2017, pp. 1–6.

[10] Y. Seo and K.-s. Shin, "Hierarchical convolutional neural networks for fashion image classification," *Expert Systems with Applications*, vol. 116, pp. 328–339, 2019.

[11] Zalandoresearch, "zalandoresearch/fashion-mnist," Oct 2018. [Online]. Available: https://github.com/zalandoresearch/fashion-mnist

[12] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Freezeout: Accelerate training by progressively freezing layers," *arXiv preprint arXiv:1706.04983*, 2017.

[13] A. Yen, "Classication with dimensionality reduction and performance evaluation, project report, ece471," Feb 2019.

[14] ——, "Color image compression using unsupervised learning (clustering), project report, ece471," March 2019.

[15] ——, "Neural network, decision tree, and performance evaluation, project report, ece471," April 2019.

[16] R. O. Duda, P. E. Hart, and D. G. Stork, "Pattern Classification 2nd Edition," 2001.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[18] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[19] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, Aug 2018. [Online]. Available: https://doi.org/10.1007/s13244-018-0639-9

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

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