PCA and Autoencoders

Tyler Manning-Dahan
INSE 6220 - Fall 2017
Concordia University

***Abstract*—In this paper, I compare two dimensionality reduction techniques for processing images before learning a multinomial logistic regression model. Principal component analysis is used as a linear mapping and autoencoders, a neural network technique, is used as a non-linear mapping. The reconstruction differences and classification errors are both compared using the fashion MNIST dataset.**

***Keywords*—Principal Components Analysis (PCA), Dimensionality Reduction, Autoencoders, Classification, Fashion MNIST**

I. Introduction

Many techniques exist for reducing the dimensionality of a dataset. Formalized in 1933 [1], Principal Component Analysis (PCA) is a multivariate statistical technique for reducing the number of dimensions in a dataset into a compressed representation called its principal components. One characteristic of PCA is that it creates a linear map and, thus, is limited to learning linear relationships between variables. For this reason, a neural network technique known as autoencoders can be used for encoding and decoding large representations of data with the flexibility of learning non-linear as well as linear mappings. This project presents an exploration of PCA and autoencoders for clustering image data and investigates the predictive power of each method when combined with a multinomial logistic regression.

A. Related Work

The connection between PCA and neural network representations is well known. Erkki Oja demonstrated in 1982 that a neural network with a linear activation function essentially learns the principal component representation of the input data [2]. Oja et al. further developed this work by examining one-unit learning rules for independent component analysis and the learning of minor components using neural networks as well [3] [4].

Furthermore, a German research team recently investigated initializing deep autoencoders with PCA, specifically for document image analysis [5]. They noted that when comparing an image classification task initialized with autoencoders to one using PCA, less samples or training data were necessary for computing the principal components and achieving good results.

B. Fashion-MNIST Dataset

In order to compare these two dimension reduction techniques, the Fashion-MNIST dataset released by Zolando research in 2017 will be used [6]. This dataset consists of 28 x 28 gray-scale images of $70,000$ fashion products. There are 10 categories of these fashion products, with $7,000$ images per category. The 10 categories of clothing are *T-Shirt/Top*, *Trouser*, *Pullover*, *Dress*, *Coat*, *Sandal*, *Shirt*, *Sneaker*, *Bag* and *Ankle Boot*. When fitting classification models or learning compressed representations of the data, we will use the standard training set of $60,000$ images and labels, and the test set of $10,000$ images and labels, both of which have been pre-split for bench-marking purposes.

To represent image data in a tabular format or matrix, we will create an n x m matrix where each row is an image, resulting in n rows, and each column is a pixel value of the image, resulting in m pixels. Given our images are 28 x 28, each image has 784 pixels where each pixel takes on a value between 1 and 255, where 1 is a pure white pixel and 255 is a pure black pixel. The labels for each image are integers in the range $[0, 9]$, where each number corresponds to a type of clothing and is summarized in table 1.

TABLE I
Image Classes and Labels

Type of Clothing	Class
t-shirt/top	0
trouser	1
pullover	2
dress	3
coat	4
sandal	5
shirt	6
sneaker	7
bag	8
ankle boot	9

II. Modelling Techniques

A. Principal Component Analysis

Using the training dataset without the labels, we effectively have a $60,000$ x 784 data matrix, denoted as X. The matrix representation of the data makes the variables not only difficult to visualize and interpret but also computationally taxing to deal with when applying predictive models. Therefore, we can use PCA to learn a transformation matrix, Z, that maps all the original data into a new coordinate system that projects the data onto a basis that maximizes explained variance of the data.

The first step of PCA is to standardize the data matrix, X. This is important because features or columns in the matrix that have larger scales relative to other columns will end up dominating the final principal component matrix, i.e.

attributing most of the explained variance to these features. This step is typically done by subtracting each column mean from each column and dividing by their respective standard deviation. In the case of our gray-scale image data, this is less crucial given the pixels are all on the same scale. However, standardization does have other benefits as well such as faster learning for neural networks which will become important later when implementing autoencoders, therefore it will be part of the procedure in this paper. The standardized matrix will be denoted as Y.

The second step is to calculate the covariance matrix of the standardized matrix. The covariance between the variables can be found by performing matrix multiplication between Y^T and Y and normalizing by $\frac{1}{n-1}$, where n is the number of features. It should be noted that this results in an 784 x 784 symmetric matrix, where each diagonal element is the variance of each feature and every non-diagonal element is a particular covariance between two features. Therefore, the covaraince matrix, S, will have the following structure:

$$S = \begin{bmatrix} var(x_1) & cov(x_1, x_2) & \dots & cov(x_1, x_{784}) \\ cov(x_1, x_2) & var(x_2) & \dots & cov(x_2, x_{784}) \\ \vdots & \vdots & \ddots & \vdots \\ cov(x_1, x_{784}) & cov(x_2, x_{784}) & \dots & var(x_{784}) \end{bmatrix}$$

The third step is to find the eigenvectors of the covariance matrix using eigenvalue decomposition such that we obtain a matrix, Z, whose columns are the eigenvectors of S. Here, Λ is a diagonal matrix whose elements are the eigenvalues of S in decreasing order. The eigenvector matrix Z has columns, $z_1, z_2, ..., z_{784}$, representing the first principal component, the second principal components, and so on and so forth until the last principal component. Each component is orthogonal to each other and are ordered in decreasing order of explained variance.

The last step is to use the transformation matrix, Z, to compute the principal component scores of the scaled data matrix, Y, by multiplying the two together.

Therefore, PCA can be summarized as follows:

1) Standardize data matrix: $Y = HX$
2) Calculate covariance matrix: $S = \frac{1}{n-1} Y^T Y$
3) Eigenvalue Decomposition: $S = Z \Lambda Z^{-1}$
4) Find PC Scores of original data: $T = YZ$

When reducing the dimensionality of our dataset, we denote the transformation matrix as Z_L, where L is the number of principal components such that $L < 784$. Therefore, we can compress the data using $T_L = YZ_L$, where T_L still has the same number of rows but only L columns, thereby resulting in a reduced dataset.

The transformed matrix, T, can be used to cluster the data in an unsupervised manner, but in our case, since we have labels, we will train a logistic regression model and then use this model to predict the class of an image and assess the accuracy of the model.

B. Multinomial Logistic Regression

Following dimensionality reduction, logistic regression will be used on the resulting transformed data, T_L, to fit a logistic regression model. Once the model is fitted, the model will be used to make predictions with the test data. A multinomial logistic model will be used because there are ten classes in the dataset. For a multinomial model, the response variables has 10 levels in the space $G = 0, 1, 2, ..., 9$, representing the set of possible classes.

Suppose the response variables has K levels in the space $G = \{0, 1, 2, ..., K\}$, representing the set of possible classes. The probability of determining a particular class is defined as [7],

$$Pr(G = k|X = x) = \frac{e^{\beta_{0k} + \beta_k^T x}}{\sum_{l=1}^{K} e^{\beta_{0k} + \beta_k^T x}}$$

Because multinomial logistic regression outputs a vector of probabilities given an input, we need to decide which class the the output probability vector belongs to. We will simply use the highest probability to assign the output to a predicted class.

C. Autoencoders

Autoencoders are a type of neural network architecture that take in an input matrix, compress (encode) the input to a reduced set of dimensions and then reconstruct (decode) the compressed data back to its original form. Therefore, a lossy transformation is applied to the data that may be used in applications like image compression.

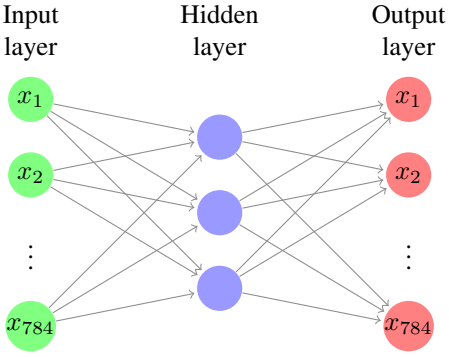

Fig. 1. Basic Autoencoder with 1 hidden layer that is trained to learn an approximation of the input

As seen in Figure 1, an autoencoder takes in a data matrix, X, and is passed through a hidden layer that is a lower dimension than the input layer. The goal is for the network to reproduce the original data as close as possible to the original.

The input layer will have 784 neurons, corresponding to the number of features the dataset has. Recall that each neuron represents a weight and an activation function. For our purposes, the rectified-linear unit (ReLU) activation function will be used for the hidden layer and a sigmoid activation function will be used in the final output layer. The network

will be trained on the entire training dataset, i.e. $60,000$ data points.

The goal of the autoencoder is to learn a function $f(x) \approx x$, where $f(\cdot)$ is made up of the weights and activation functions from the network layers. Therefore, a non-linear identity approximation will be output of this process.

I will not explicitly detail the mathematics of backpropogation for training an autoencoder, but it may be summarized in algorithm 1 [8].

for *each input x* **do**
  Feed-forward pass to compute activations of all hidden layers and store these in a cache-style memory. At the same time, compute the final output x′ in the last layer.
  Measure deviation of x′ from input x using loss function
  Backpropogate the error through the network and update the weights
  Repeat until resulting loss is acceptable or other factor is satisfied
end

Algorithm 1: Autoencoder training algorithm

Furthermore, as detailed by Andrew Ng in his notes [9], the idea with autoencoders is that there exists some correlations between pixels, indicating patterns in the images. If the input was simply random noise then learning a meaningful representation of the input would be very difficult. Since the data is essentially images of clothing, there are features that exist among the different examples, revealing a structure that can be learned. discover some of those correlations.

Like the PCA model, the autoencoder will be paired with a classifier in a supervised context to be able to make accurate predictions. This is done by combining the learned compression layer from the autoencoder and adding a softmax layer that will take in the compressed image representation and produce a relative probability of each class. The architecture of this can be seen Figure 2.

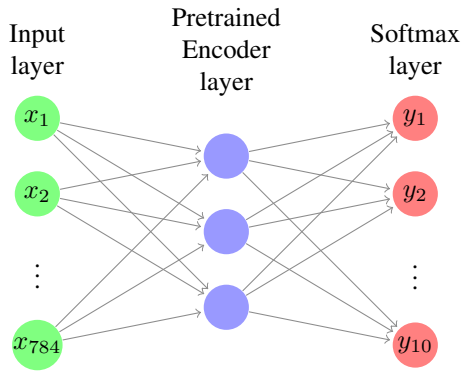

Fig. 2. Basic Autoencoder with 1 hidden layer that is trained to learn an approximation of the input

A softmax layer is exactly the same as a multinomial logistic regression, and is interchangeable in terms of notation. I will not demonstrate they are equivalent but it can be seen the references [10]. Therefore, I will use softmax and logistic regression interchangeably in this paper since there is no mathematical difference between the two in this context.

D. Relationship between PCA and Autoencoders

It can be shown that when PCA maximizes the variance to find the principal components, it is actually minimizing the following cost function [11]:

$$J = \sum_{n=1}^{N} |x(n) - \hat{x}(n)|^2$$

Where $\hat{x}(n)$ is the reconstructed result using the inverse of the transformation matrix, i.e. $\hat{x}(n) = Z^{-1}Zx(n)$ [12].

The output, z, of a layer in a neural network is the multiplication of the weight matrix, W, by the input, x, with the activation function, $f(\cdot)$ being applied following this multiplication, i.e. $z = f(Wx)$. In a single hidden layer autoencoder, after z is outputted by the encoder, it must be decoded by another layer to reconstruct the original data, which gives $y = g(Vz)$, where y is the final reconstructed output, $g(\cdot)$, is another activation function and V is the matrix of the weights of the decoding layer. Therefore, we can put both these equations together to show an autoencoder is applying the following to an input:

$$y = g(Vf(Wx))$$

If the activation functions are chosen to be the identity functions then the above reduces to $y = VWx$ and the cost function of the autoencoder is then,

$$J = \sum_{n=1}^{N} |x(n) - VWx(n)|^2$$

This is the same result as the PCA cost function, meaning PCA is simply a special case of an autoencoder where the activation functions are chosen linearly.

E. Model Evaluation

The main evaluation metrics that will be used to evaluate our classifiers is the F_1 score, which provides an encompassing measure that takes into consideration a model's precision and recall. It is defined as,

$$F_1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{1}$$

This definition of F_1 score is the harmonic mean of precision and recall. Precision is defined as the ratio of true positives to the number of true positives and false positives and recall is defined as the ratio the number of true positives to the number of true positives plus the number of false negatives. F_1 score is best when $F_1 = 1$ (perfect precision and recall) and reaches its worst value at $F_1 = 0$ [13].

Calculating the F_1 score in the multi-class, yields a score for every class. Therefore, to compare different classifiers we

will average the ten F_1 scores to yield an overall performance metric. In some cases, a comparison between F_1 scores at the class level will be made to determine where the classifier is under-performing.

As a benchmark comparison, a logistic regression will be trained on the dataset without any dimensionality reduction. The average F_1 score and the F_1 scores for each class will be recorded for comparing later models.

III. PCA Implementation

A. Scree plot

Plotting a modified Scree plot in Figure 3 of the first ten principal components in order of highest explained variance, it is clear the first three principal components account for roughly 40% of the total explained variance.

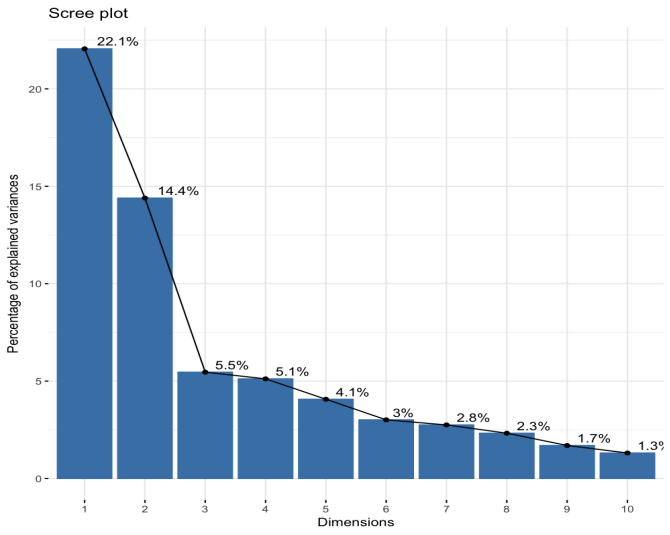

Fig. 3. Scree plot with bars displaying the explained variance as a percentage of total variance by the first ten PCs

for reducing the dimensionality of a dataset using the principal components that explain the most variance, one rule of thumb is to select principal components up to the "elbow" point in a Scree plot, i.e. where the gain in explained variance is first greatly reduced. Using the Scree plot in Figure 3, we can deduce this is the first three principal components, i.e. z_1, z_2, and z_3.

Therefore, using the reduced PC matrix, Z_3, we can compute the PC scores of matrix X and again fit a multinomial logistic regression and test the results. This yields an average F_1 score of 0.32, which is significantly worse than using all the principal components. In fact, the model was so primitive it had and $F_1 = 0$ for *sandal*, *shirt*, and *sneaker* classes.

B. Explained Variance

The previous heuristic for choosing a lower dimensional representation proved to be an oversimplification of the problem and loses too much information by dropping 781 principal components. Another technique to find the minimum number of principal components without a signification loss in signal is to use a grid search method for trying out several possibilities over some range of values.

To examine this deeper, the percentage of explained variance will be examined. Recall, we can calculate the percentage of explained variance by the ith principal component by using the eigenvalue λ_i as a ratio to the total sum of the 784 eigenvalues. Therefore, it is:

$$\text{Percentage of Variance} = \frac{\lambda_i}{\sum_{i=1}^{784} \lambda_i} * 100\%$$

To determine an appropriate range of values, all 784 principal components were plotted with their percentage of explained variance. This visualization makes it clear that a grid search between 130 and 145 principal components is ideal given the diminishing returns of variance at this point. Therefore, keeping the original logistic model fixed and trying the different numbers of principal components to yield the most accurate classifier yields 141 principal components. This can be seen in Figure 4.

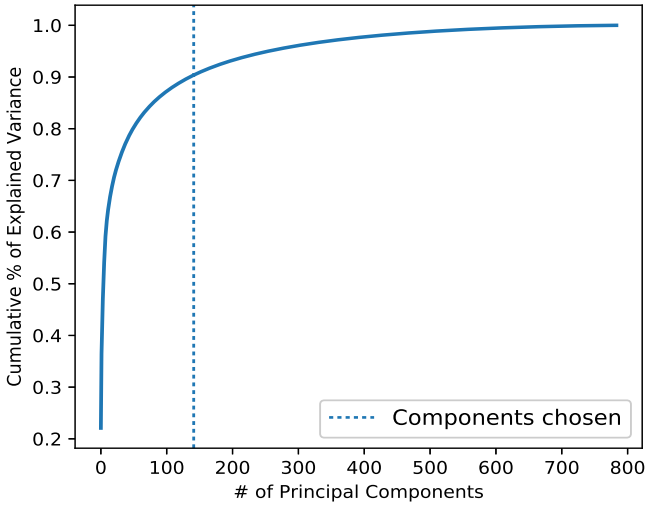

Fig. 4. Cumulative percentage of explained across all principal components, where the dotted line indicates the cut-off of 141 principal components, that was found to be most efficient

Using the first 141 principal components, $z_1, z_2, ..., z_{141}$, to again transform the original data and train it on a multinomial logistic classifier, yields an average F_1 score of 0.73. Therefore, with the number of dimensions reduced by a factor of 5 using PCA, we can still classify the images almost as accurately as the benchmark logistic regression that had an average F_1 score of 0.75. The F_1 score for each individual class is summarized in table 2 in the last section of this paper when compared to the autoencoder implementation. The 141 principal components correspond to a cumulative percentage of variance of approximately 90%. This means the remaining 643 principal components only add about 10% more information.

Lastly, to evaluate this classification model we can setup a confusion matrix for all the classes. A confusion matrix

displays the predicted label on one axis and the true label on the opposite axis. The diagonal elements of the matrix are the number of test examples for which the predicted label is the same as the true label, while the off-diagonal elements are test examples that are mislabelled by the classifier [14]. The better the classifier, the higher the values along the diagonal elements, indicating correct predictions of a classifier.

Another benefit of the confusion matrix is you can quickly see what labels are being incorrectly classified as rather than just know they are incorrect.

The values in the confusion matrix in Figure 5 have been normalized by the total of each class, therefore rather than see the number of counts in each element of the matrix, we see the values are the percentage of correct or incorrect predictions.

Overall the results in Figure 5 are promising given that *trouser*, *coat*, *bag*, and *ankle boot* that have an accuracy of 95%, 97%, 96%, and 98% respectively. However, 45% *pullover* predictions were incorrectly classified as a *coat*. A similar issue occurred with the *shirt* class who only had 23% correct predictions with many being classified as a *coat* too. The classifier seems to have trouble distinguishing between *coat*, *shirt*, and *pullover*, which is not surprising given they all three pieces of clothing look quite similar.

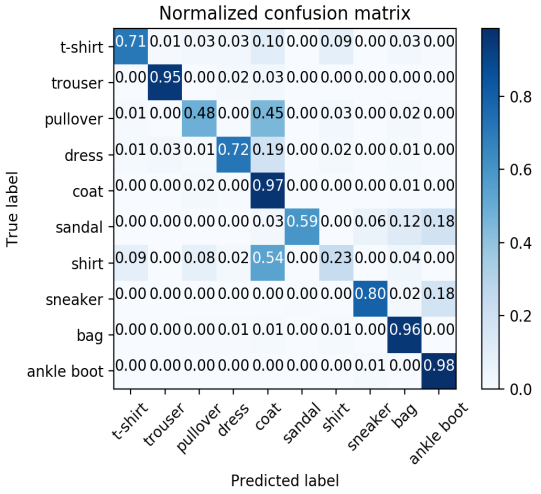

Fig. 5. Confusion matrix of the ratio of the true labels to the predicted labels for each class. These results are based on the 141-component PCA model with logistic regression.

IV. Autoencoder Implementation

A. Image Reconstruction

Using a simple autoencoder architecture described in Figure 1, the training set, X, can be encoded (compressed) by training X to learn itself. The encoding layer is chosen to have 32 neurons all with ReLU activation functions. The size of this layer was chosen arbitrarily but mainly based on neural networks trained with the classic MNIST dataset in the Keras documentation for autoencoders [15]. Note this results in a dimension reduction of 24 times less than the fully featured dataset.

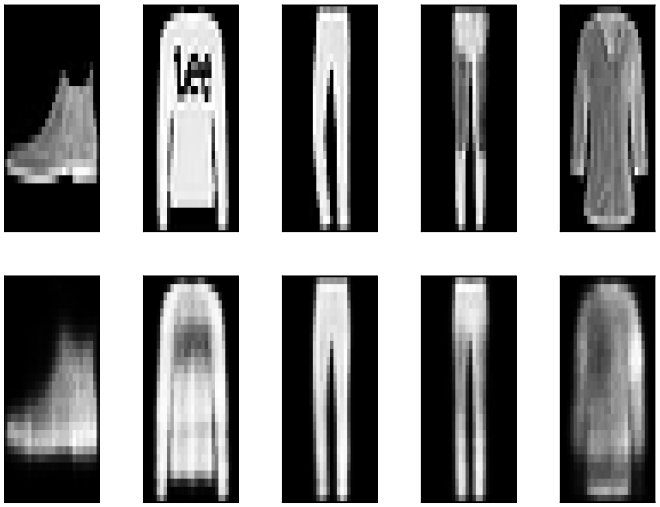

Fig. 6. Reconstruction of test images using the trained autoencoder. Images are (from left to right) ankle boot, pullover, trouser, trouser, and shirt.

Following the training of the autoencoder, we can visualize how it is reconstructing the image data by encoding and decoding images from the test set. A sample of these results can be seen in Figure 6. This confirms the autoencoder is in fact reproducing a compressed representation of the information given to it. The basic shapes of the data are clearly learned but details like the words on the clothing are not clear.

The decoding layer is then removed from the network and the softmax layer is added for training with the training set labels. Once completed, the model performance is evaluated using the test set.

B. Autoencoder Accuracy

Calculating the average F_1 score using the neural network yields 0.86, which is a 17% improvement from the optimal PCA model that used 141 principal components.

If we look at each the individual F_1 score of each class, we see there is an improvement in every single one. The biggest gains are in the *pullover* category with a 32% increase, *coat* with a 33% increase and *shirt* with a 103% increase in F_1 score. All these results are summarized in table 2.

Comparing the confusion matrix in Figure 7 with the confusion matrix in Figure 5, the neural network's classifier improvement in particular classes is noticeable for the *pullover* class and the *shirt* class. Both of these categories have much less false positives found in other classes, giving a higher value along the diagonal of the confusion matrix. Furthermore, while the PCA model struggled with the *sandal* class with an accuracy of 59%, the auotencoder has a score of 94% in the same category.

Lastly, we can break down the F_1 scores for every class and compare them for the benchmark logistic regression model that did not involve any data compression, the 141 PCA and

TABLE II
RESULTING F1 SCORES BY CATEGORY AND MODEL

Type of Clothing	141 PCA	Autoencoder
t-shirt/top	0.78	0.82
trouser	0.95	0.97
pullover	0.59	0.78
dress	0.8	0.85
coat	0.58	0.77
sandal	0.74	0.96
shirt	0.33	0.67
sneaker	0.85	0.95
bag	0.87	0.95
ankle boot	0.83	0.96

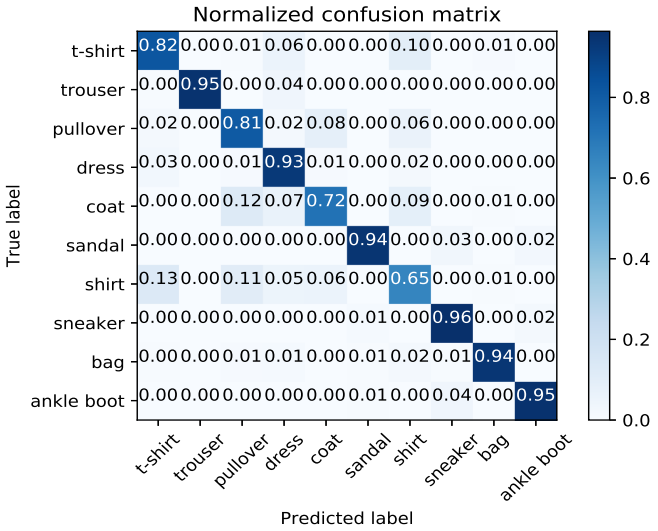

Fig. 7. Confusion matrix of the ratio of the true labels to the predicted labels for each class. These results are based on the autoencoder and softmax model.

logistic regression model, and the autoencoder with a softmax layer model. These are summarized in Figure 8. It is clear that the neural network model has the best F_1 score in every image category. While the PCA model was able to do nearly as well as the benchmark in every category, it was not a significant improvement. In fact it only eclipsed the benchmark in the *t-shirt/top* and *trouser* categories. Therefore, if accuracy was a concern then utilizing the entire dataset with a multinomial logistic regression would be better. However, if representing the data in a compressed, minimal format at the expense of some accuracy, then reducing the dataset using 141 principal components and a logistic regression would be the better choice.

The appeal of neural networks is clear from Figure 8. The autoencoder with softmax layer model is able to reduce the dataset to 32 key features and still outperform logistic regression and PCA models. The autoencoder significantly improved the F_1 score in the *shirt* and *coat* categories, both of which had very low scores for the other models.

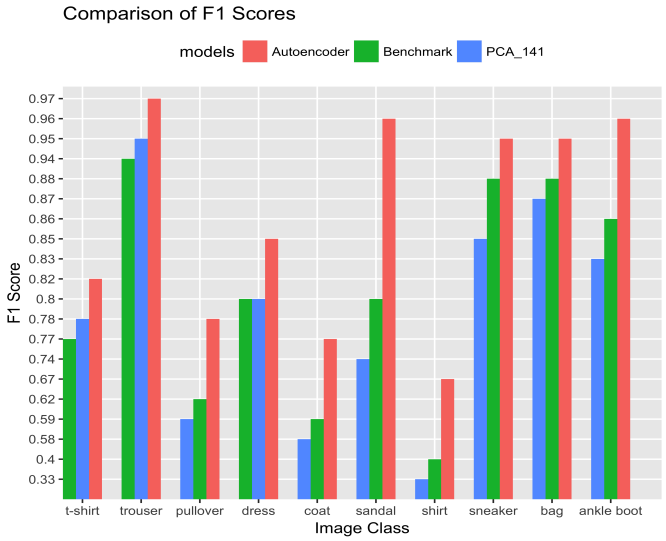

Fig. 8. Comparison of F_1 Scores across each image class and the various classifiers used. The Autoencoder classifier has the highest score in every category.

V. DISCUSSION

There are several more ways PCA and autoencoders could be compared for a deeper understanding of their differences. Looking at the details of the loss function of each classifier and how they evolve over time may provide additional information into the neural network superiority that generalizes better with so little training.

Given the simplicity of the autoencoder model used here, it may be worth exploring stacked autoencoders for a better performance. A stacked autoencoder would be trained one layer at a time learning to represent each compression sequentially be training on itself. Once each layer is trained, they are stacked with a classification layer in a supervised manner for training with the labels, similar to the use of a softmax layer in this paper. This will probably allow the model to learn more complex relationships in the images, leading to better classification results.

As for pure performance, it may be interesting to train the Fashion-MNIST dataset on more complex neural network architecture like convolutional neural networks that are typically very good at image recognition tasks.

Finally, while a neural network-style encoder and classifier was better at classifying images in this context, it may not always be the case that an autoencoder can outperform a PCA representation of the data. If a linear map can sufficiently explain the variance in the data without significant loss of detail, then a neural network may be overkill when a simpler model like PCA can be used effectively.

REFERENCES

[1] H. Hotelling, "Analysis of a complex of statistical variables into principal components." *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[2] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, no. 3, pp. 267–273, 1982.

[3] ——, "Principal components, minor components, and linear neural networks," *Neural networks*, vol. 5, no. 6, pp. 927–935, 1992.
[4] A. Hyvärinen and E. Oja, "One-unit learning rules for independent component analysis," in *Advances in neural information processing systems*, 1997, pp. 480–486.
[5] M. Seuret, M. Alberti, R. Ingold, and M. Liwicki, "Pca-initialized deep neural networks applied to document image analysis," *arXiv preprint arXiv:1702.00177*, 2017.
[6] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.
[7] "https://web.stanford.edu/˜hastie/glmnet/glmnet_alpha.html#log," https://web.stanford.edu/~hastie/glmnet/glmnet_alpha.html#log, (Accessed on 12/17/2017).
[8] "Autoencoder - wikipedia," https://en.wikipedia.org/wiki/Autoencoder, (Accessed on 12/15/2017).
[9] "sparseautoencoder_2011new.pdf," https://web.stanford.edu/class/cs294a/sparseAutoencoder_2011new.pdf, (Accessed on 12/13/2017).
[10] "Unsupervised feature learning and deep learning tutorial," http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/, (Accessed on 12/17/2017).
[11] "A tutorial on autoencoders for deep learning - lazy programmer," https://lazyprogrammer.me/a-tutorial-on-autoencoders/, (Accessed on 12/17/2017).
[12] "14_pca.pdf," https://www.cs.toronto.edu/~urtasun/courses/CSC411/14_pca.pdf, (Accessed on 12/17/2017).
[13] "F1 score - wikipedia," https://en.wikipedia.org/wiki/F1_score, (Accessed on 12/02/2017).
[14] 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.
[15] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.
[16] J. Shlens, "A tutorial on principal component analysis derivation," *Discussion and Singular Value Decomposition*, vol. 25, 2003.