Feature Extraction using PCA for Image Classification

Mary Aiyetigbo
School of Computing
Clemson University
maiyeti@clemson.edu

Venkatesh Velidimalla
School of Computing
Clemson University
vvelidi@g.clemson.edu

Abstract

Image datasets often have high dimensionality, and the application of machine learning algorithms to classify this high-dimensional data might require efficient memory storage and high computational cost. In this project, we propose to use Principal Component Analysis (PCA) for feature extraction to reduce the dimension of the data before passing it to a machine learning algorithm for classification. We will evaluate the model trained with reconstructed images against the model trained on the original data to compare the performance of our trained model.

1 Introduction

Principal Component Analysis, also known as PCA, is a linear technique used to reduce the dimensionality of data. It uses orthonormal transformation for reducing largely correlated variables with a small set of linearly independent features [1]. PCA has three important properties. First, it is an optimal linear strategy (related to mean squared error MSE) for compressing a set of high-dimension data to a low-dimension vector and then for reconstruction. Also, the model parameters are directly computed from the data itself, using the covariance, for example. Thirdly, PCA requires only matrix multiplication; hence, compression and decompression operations are easy to perform, given the model's parameters [6]. PCA creates uncorrelated feature space that can be used for further analysis. It increases interpretability and also minimizes information loss. In addition, it helps us find the most significant features in a dataset and makes the data easy to plot in low dimensional space (2D and 3D). As a feature reduction method, PCA can deal with over-fitting problems that may occur in datasets with many features [2].

Multilayer perception (MLP), also known as Deep Neural Network (DNN), is one of the deep learning algorithms composed of multiple layers of perceptions. MLPs are helpful in research for their ability to solve complex problems [3]. Neural Networks have been used to solve computer vision tasks such as image segmentation, object detection, image classification, and Natural language processing. In MLP, each neuron in the network layer is mapped to all the neurons in the next layer [4]. This deep connectivity makes the network complex, and the computational resources required to approximate the complex solution might be costly, especially when the input data has a high dimension. The approximation of too many complex solutions can also be prone to overfitting. Hence, extracting important features from input data before feeding them into the model for the classification task is important. Also, MLP has been known to require a large number of parameters for training. Since each pixel in the input image represents a feature, the neurons of the input layer can be very large, which will, in turn, requires a lot of weight parameters to map the input features to the hidden neuron. As the size of the input image increases, the total number of parameters also increases. Hence, MLP for image classification will require lots of computation resources for linear operation and might not be suitable for image classification, especially with large input images. PCA can mitigate this challenge by reducing the dimension of the input image. Another neural network architecture called convolutional neural network is also designed to work better with images by extracting important image features before feeding them to the MLP.

Convolutional Neural Network (CNN) has also been used extensively in the literature to extract important features from input images. CNN takes an image as input and processes it using a linear combination of filters (kernel) which are convolved with the input image. CNN usually contains multiple layers of convolutions and max-pooling to extract meaningful features and reduce the dimension of the input image [7]. CNN has been reported to achieve very high accuracies in computer vision tasks. However, it may also require high computational costs to process and might also take a lot of execution time, depending on the size of the input data.

In this project, we will use the PCA technique to reduce the data's complexity and extract important features while retaining most of the variation before feeding them as input to the MLP. We hope that this will help minimize overfitting and reduce computational costs. We will also perform image classification using CNN as a feature extractor to analyze if we will get a comparable result when PCA is used as the feature extractor. We carry out three experiments (listed below) in this project and evaluate the result of each experiment

1. Train the model using only MLP for image classification
2. Train a model using PCA for feature extraction and MLP for classification
3. Train a model using CNN and MLP for image classification

2 Methodology

Principal Component Analysis (PCA). The first approach is to conduct feature extraction to reduce the dimension of correlated features in the dataset by obtaining principle components from a raw dataset. Since PCA is an unsupervised learning approach, the labels of the samples will not be used for this approach. The main idea was to compute principal components from the input data, which we used to reconstruct the new image. Firstly, the dataset $\mathbf{X} \in \mathbb{R}^{m \times 784}$ was centered by computing the mean of the dataset ($\bar{\mathbf{X}}$) and subtracting the mean from each data sample to obtain a new data matrix $\mathbf{X}$. Next, we computed the covariance matrix $\mathbf{X}^T\mathbf{X}$ of the centered dataset. Singular value decomposition of the covariance matrix was used to obtain the principal components, also known as the Eigenvector U, as shown in equation (1) below where $\mathbf{S}$ is the singular value

$$[\mathbf{U}, \mathbf{S}, \mathbf{V}] = svd(\mathbf{X}^T\mathbf{X}) \tag{1}$$

The top k eigenvector sorted based on the largest singular value will be chosen as the principal component ϕ_k to form a matrix of $m \times k$ dimensional below.

$$\phi_k = U[:, 1:k] \tag{2}$$

The data sample is transformed onto the new subspace using the $m \times k$ eigenvector matrix. The newly reconstructed data $\mathbf{X}_r$ is computed using the formulation below

$$\mathbf{X}_r = \mathbf{X}\phi_k \tag{3}$$

This method aims to find the best summary of our data by using the least amount of principal components; by choosing our principal components, we minimize the distance between the original data and its projected values on the principal components.

Multilayer Perceptron (MLP) will be used to classify our dataset. MLP is a linear combination of the input with some learned weights to generate the output. The new subspace will be used as the image of the input layer. In this project, we built a very simple neural network model with two hidden layers; the first hidden layer has 256 neurons, while the second hidden layer has 64 neurons. The output layer has ten (10) neurons, and each neuron will be used to predict the probability of a sample input data belonging to a class (0 to 9). A simple equation of the model structure is shown below

$$y_i = f(W_3^T(f(W_2^T(f(W_1^T x_i) \tag{4}$$

Where y_i is the predicted output, x_i is the input sample data, W_1 is the weight of the first layer with shape $784 * 256$, W_2 is the weight of the second layer with shape $256 * 64$ and W_1 is the weight of the first layer with shape $64 * 10$

The ReLU function in equation 5 below will be used as the activation function in the hidden layers.

$$\textbf{Relu}(x) = max(0, x) \tag{5}$$

Since the dataset has multi-classes, the softmax function shown in equation 7 will be used as the output layer's activation function to output the probabilities of the output belonging to any of the ten classes.

$$\textbf{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{6}$$

The model was evaluated using accuracy metric. This is defined as the number of predictions the model can get correctly that the sum of the true positives (TP) and true negatives (TN) [9].

$$\textbf{Accuracy} = \frac{NumberofCorrectPredictions}{Totalnumberofpredictions} \tag{7}$$

3 Experiments

Python programming language was used to implement this project, and PyTorch framework libraries [8] were used to build and train our neural network.

3.1 Dataset

The dataset proposed for this project is the Modified National Institute of Standards and Technology (MNIST) public dataset. MNIST is an extensive collection of handwritten digits which contains 60,000 training samples and 10,000 validation samples. It is a subset of a more extensive NIST Special Database [5]. The dataset contains different human handwriting from digits zero (0) to nine (9). This is a multi-class classification task with ten (10) classes. Each data sample has an image dimension of 28 x 28 image matrix and, when flattened, has a dimension of 784 column vector. Fig 1 below is a dataset sample with the associated label. The dataset was loaded using the datasets function in the torchvision library, and the data was normalized (standardized) using a 0.5 mean value.

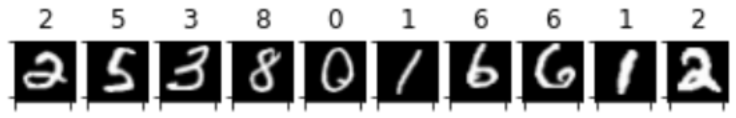


Figure 1: Sample MNIST Dataset

3.2 Implementation Details

3.2.1 Experiment 1: MLP Only

We created a Multilayer Perceptron (MLP) classifier model to classify handwritten digits. We used two hidden layers with 256 and 64 neurons, respectively, for the network architecture. In this experiment phase, we used the original 28 * 28 MNIST image as the input to the model. Because the input layer of an MLP takes 1D tensors, we reshaped (flattened) the two-dimensional input data into 1-dimensional data with 784 pixels, each pixel in the image representing a feature. Therefore, we need 784 neurons in the input layer of our MLP model. Despite using a small neural network of two hidden layers (256 and 64), the total number of parameters (weights and bias) used to train the network was **218,058**. The major reason for having such a large number of parameters for the model is the large size of the input layer (784 neurons).

Since we have multiple classes, cross-entropy loss L in equation 8 was used as the loss function to train our model. Where P^* and P are the probability of the predicted and true class distribution.

$$L = \sum P^*(i) log P(i) \tag{8}$$

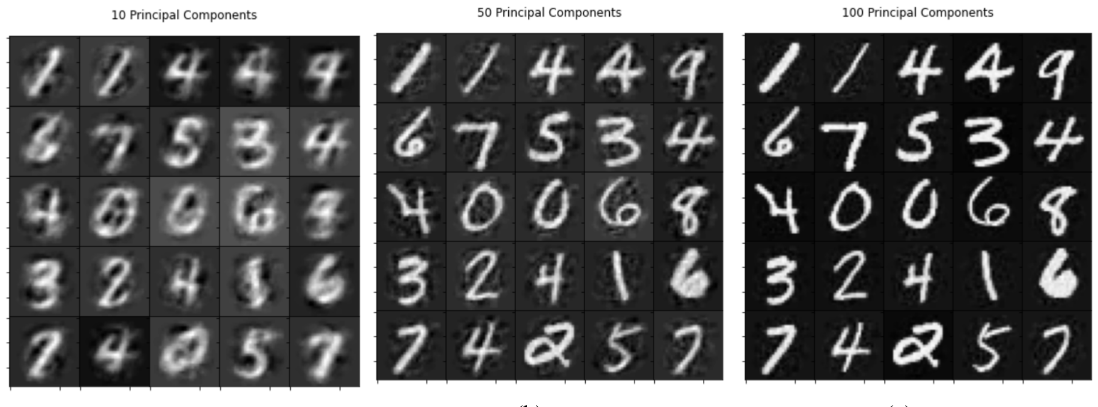
In this experiment, we trained the model for twenty (20) epochs. The model parameters were updated using a gradient descent algorithm in equation 9, where δ is the learning rate hyperparameter. A learning rate of 0.001 was used as the step size for updating the weight and bias parameters.

$$w = w - \delta \Delta w \tag{9}$$

The input dataset was trained in batches with a batch size of 128 sample data for each iteration, and the model was trained for a total of 9360 iterations. The model achieved a training accuracy of about 99% after 20 epochs.

3.2.2 Experiment 2: PCA + MLP

We implemented PCA on our data for feature extraction and image compression tasks. To implement PCA on our data, firstly, the data were normalized, and the eigenvectors and singular values of the covariance matrix (or correlation) of the data were computed. Eigenvectors (principal components) determine the direction of the new space, and eigenvalues determine its magnitude. The new reconstructed image was calculated with the eigenvector. Fig 2 shows the reconstructed images using the first 20, 50, and 100 principal components, respectively. From Fig 2c below, it was shown the first 100 principal components could retain good information about the image, and the sharpness of the reconstructed image is very close to the original, which has about 784 features.



(a) (b) (c)

Figure 2: PCA for Image Reconstruction

Fig 3 below is the plot of the cumulative variance retained across the number of principal components (784). It was also observed that the first 100 principal components could capture about 92.5% of the variability in the original image data, which is enough to retain the quality of the original images.

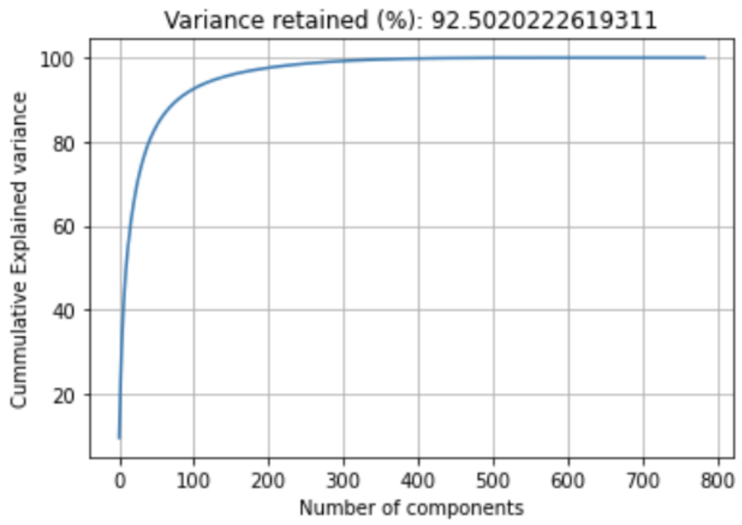
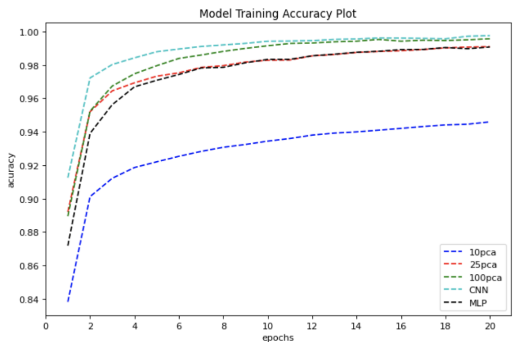


Figure 3: Explained variations of principal components

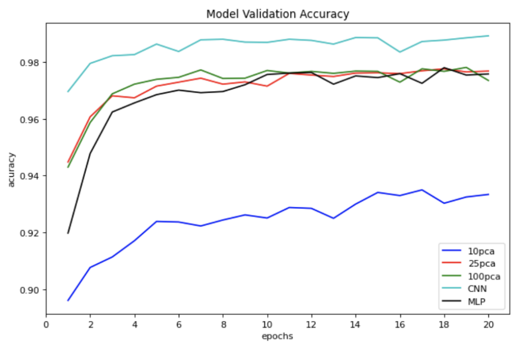
We carried out this experiment using different numbers of principal components 10, 25, and 100, respectively. We used the same network architecture defined in experiment 1 above except for the input layer. In this experiment, the number of input neurons varied based on the number of principal components (PCs). For 10 PCs, our model had only ten neurons in the input layer, and the total number of parameters used for the model was training this model was **19,914**. Compared with the experiment using just MLP, this is a significant drop in the number of parameters, and the model achieved a training accuracy of about 95%.

For experiments with 25 PCs, our model had only 25 features in the input layer with **23,754** number of parameters. The model trained with 25 components achieved a training accuracy of 99%, which is the same as the training accuracy obtained with MLP that used 784 input features.

Finally, we used 100 principal components of the input images to train our model since 100 PCs retained a good percentage of the variance of the input image, as shown in Fig 3 above. This means that the input features of this model were 100, and the model had a total of **42,954** parameters. Compared with the MLP model in experiment 1, this model had a higher training accuracy of 99.6% with fewer model parameters. This means using PCA to extract features is able to reduce the computation resources due to fewer parameters, and it can also achieve high performance during training.



(a)



(b)

Figure 4: Models Accuracies

3.2.3 Experiment 3: CNN + MLP

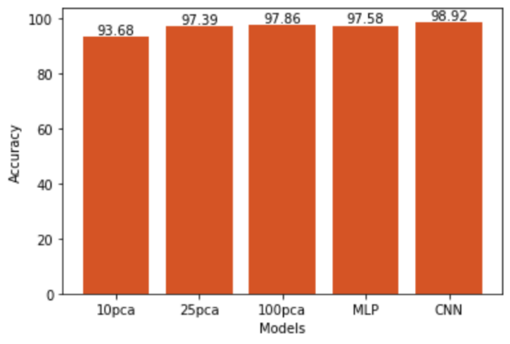
CNN is known to achieve very high accuracy for image classification. In this third experiment, we compare the performance of our PCA model with a model trained using CNN as the feature extraction. The same fully connected structure used in experiment 1 above was used for the classification task. However, two additional layers of convolutions were included in the model architecture. The first convolution layer has eight (8) kernels of size (3 x3) with stride and padding of 1 and a MaxPool layer with stride and kernel of 2. The second convolution layer used 16 (3x3) kernels with a stride and padding of 1. The CNN model has a total of **219,306** model parameters which is the highest parameters from all the experiments. However, from Fig 4a above, the CNN model achieved the highest training accuracy of 99.8%.

3.3 Evaluation

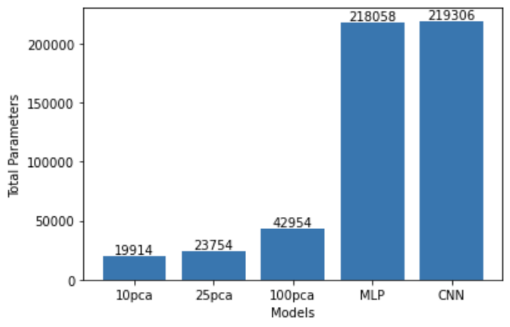
Fig 4b above shows the validation accuracy of all the trained models. Each model was evaluated using 10,000 test samples. For the plot, we can see that the model trained with 10 principal components of the input images had the least performance with an accuracy of 93.68% after 20 epochs. This is because using only 10 PCs is not a good representation of the original image. Fig 3 shows that 10 PCs could only retain about 60% variance in the original image; hence, the low performance compared to other models.

The MLP model, and model trained with 25PCs and 100PCs, respectively, all have similar accuracies on the test data after training for 20 epochs. However, it can be shown that the MLP model started with a lower accuracy at the start of the training, while both models trained with principal components of 25 and 100 could fit the test data earlier in training iterations. Also, these two models; 25pca and 100pca, were also trained with significantly few parameters of 23,754 and 42,954, respectively, and they still achieved a comparable test accuracy with the MLP model, which had 218,058 parameters.

The CNN model was shown to have the highest test accuracy, achieving an accuracy of 98.92% on the test data, and was able to fit the test data appropriately even after only a few iterations. Despite the high performance of the CNN model, the model will require large computation resources to train as the total number of parameters used to train the model was 219,306. Fig 5 below shows the total number of parameters for each trained model with their accuracies on the test data.



(a)



(b)

Figure 5: Plot of Accuracy and Model Parameters

In conclusion, the fig 5 above shows that using PCA to extract important features from image data greatly reduces the total number of parameters required for training while still achieving high accuracy on test data comparable to MLP and CNN models.

References

[1] Migenda, N., Möller, R., & Schenck, W. (2021). Adaptive dimensionality reduction for neural network-based online principal component analysis. *PloS one*, 16(3), e0248896. https://doi.org/10.1371/journal.pone.0248896

[2] Iftach A. What Is PCA and How Can I Use It? Retrieved from https://www.bigabid.com/what-is-pca-and-how-can-i-use-it/

[3] Wikipedia (2022). Multilayer perceptron. retrieved from https://en.wikipedia.org/wiki/Multilayer_perceptron

[4] Neural network and its uses retrieved from: https://www.altexsoft.com/blog/image-recognition-neural-networks-use-cases/

[5] Papers With Code (2018). MNIST. Retrieved from https://paperswithcode.com/dataset/mnist

[6] Roweis, S. (1997). EM algorithms for PCA and SPCA. Advances in neural information processing systems, 10.

[7] Agarwal1. A and Gupta N. (2020) PCA vs. CNN: An Approach of Image Fusion Process. http://proceeding.conferenceworld.in/Government_Eng_College_Bharatpur/9Bc6hwlvheb7B606.pdf

[8] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32 (pp. 8024–8035). Curran Associates, Inc.

[9] GoogleDevelopers (2022). Classification: Accuracy. Retrieved from https://developers.google.com/machine-learning/crash-course/classification/accuracy