Applied Machine Learning Project 3 MNIST+ Digit Images Classification

Kaggle Team: The Dark Knight Rises

Robin Yan
McGill University
huaqun.yan@mail.mcgill.ca

Haomin Zheng
McGill University
zheng.haomin@mail.mcgill.ca

Damien Goblot
McGill University
damien.goblot@mail.mcgill.ca

***Abstract*—The goal of this project was to implement different classification algorithms to recognize and classify handwritten digits (0-9). A training set consisting of 50,000 images of digits were given. This set was derived from the MNIST data set with several transformations applied to add extra challenges to the classification problem. We used logistic regression, neural network, support vector machine and convolutional neural network, and best result, 93.21% on Kaggle competition leaderboard, was given by convolutional neural network.**

I. Introduction

Image classification is the process of assigning images to predefined categories based on their contextual information. It is a form of pattern recognition from the field of computer vision, and also a very popular application of machine learning techniques. As with other classification problems, there are two types of image classification: supervised and unsupervised. Supervised classification requires users to specify classes *a priori* and select appropriate training examples to be placed into the classes. On the other hand, unsupervised training requires the user to simply specify the number of desired categories and the algorithms will automatically cluster test examples into the set of categories[12].

Under the umbrella of supervised image classification, handwritten digit recognition is an active topic in OCR (Optical Character Recognition) applications, and is the task we seek to tackle in this project. Examples of applications of handwritten digit recognition include postal mail sorting, bank check processing, form data entry, etc [8]. Of course, there are a number of challenges that exist. The digits are not always the same size, thickness, or possess the same orientation and position with respect to the margins. Furthermore, many of the digits look alike, such as '1' and '7', '8' and '9', and different individuals will have unique style of handwriting which can cause further confusion in the classification problem[10].

The approach to solve this problem is divided up into three tasks:

1) Pre-processing
2) Classificaion
3) Validation and optimization

Some pre-processing options include how the interpolation of pixel values is performed, adjustments to the size and aspect ratio of the normalized image, etc. In the case of feature extraction, a large number and variety of feature types and extraction techniques are available. PCA (principle component analysis), PCA with whitening, and ZCA (zero-phase component analysis) are considered in this project. Likewise, there are a large number of classifiers available for the classification process. The classifiers selected for this project are logistic regression, neural network, linear SVM, and convolutional neural network.

II. Related Works

There have been plenty of research performed on the classification of hand-written digits, especially on the MNIST dataset[14]. Yann Lecun *et. al* has compiled a list of the performance of many well-know machine learning algorithms on the MNIST database[5]. It is observed that neural network classifiers, more specifically convolution neural networks, tend to have the best performance. For instance, Ciresan *et. al* was able to achieve 0.27% error on a 10,000 test set using a committee of convolution neural nets with elastic distortion in augmenting the training set [13]. The use of elastic distortion to augment the trainig set proved very important to achieve low error rates. As Jarrett *et. al* discovered that the error rate of increased from 0.35% to 0.53% when distortion was not used [11]. Apart from neural net, k-nearest neighbour and virtual SVM also produced low error rates according to [4]. However, pre-processing methods is required for both cases. The dataset used in this report has be modified to add more challenge, currently, there is no publish results on this dataset yet.

III. Feature extraction

The MNIST database (Mixed National Institute of Standards and Technology database) is a large database of handwritten digits that is commonly used for training various image processing systems.[5] The database is also widely used for training and testing in the field of machine learning. It was created by "re-mixing" the samples from NIST's original dataset.[6]

MNIST+ is a more challenging dataset that modified the original MNIST dataset by rotating, projecting into 48x48 resolution, adding background pattern from Brodatz texture dataset, which is a widely used standard for signal process and computer visions.[2] Finally, perform embossing operation with a fix lighting angle and elevation.[16]

To make the basic representation of the data, we simply convert the image into grey-scale value of every pixel, which has total of 2304 features. This simple representation is good enough for some simple predictors such as logistic regression and linear SVM for the original MNIST set.[4] But it is proven that these linear methods have very poor results in MNIST+ dataset, as shown in our baseline algorithm analysis in Section VI. We need to find more advanced algorithms and more ways to reprocess the dataset.

IV. Feature pre-processing

To better represent the raw dataset, we used PCA, PCA with whitening and ZCA whitening methods to improve our results.

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The number of principal components is less than or equal to the number of original variables. In this case, we do not want to sacrifice the quality of the original image, thus we choose to use the same dimension as the raw data. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components. [7]

To compute PCA transformation, first we take the whole dataset ignoring the class labels, then compute its covariance matrix Σ.

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$$

We can then produce eigenvectors for each feature dimension and their corresponding eigenvalues. Sort the eigenvalues in decreasing order, then apply the transformation to the original dataset.[19] To select fewer dimensions from the dataset, simply ignore the least eigenvalues as needed, then perform the transform. To show that PCA indeed can conserve the majority of the information for performing feature reduction, we use basic logistic regression to predict on PCA dataset with different dimensions. For comparison, we also use the identical algorithm and parameters on the original dataset, using part of the whole features. As the result shown in Fig. 3, the PCA have much better result on every dimension settings. Moreover, reducing dimensions to 800 features does not have significant impact on the prediction results.

In addition to PCA, we also used PCA with whitening to further optimize our feature representation. Because our raw data is from images, the adjacent pixel values are highly correlated. To ensure the transform will produce uncorrelated features, we assume the covariance matrix will become identity matrix. This whitening process will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.[15]

If we just look at the whitening process, which is to find the uncorrelated outputs, PCA is not the unique solution. In Zero-phase Component Analysis (ZCA) whitening process, in stead of finding an orthogonal solution, we assume the transform matrix to be symmetric. This solution, will in fact preserve the spatial arrangement of the image and flatten its frequency spectrum.[3] Comparing to PCA which rotates the coordinates and scramble the image completely, ZCA whitened image will very much resemble the original image because the decorrelation filter was applied locally instead of globally, as we can see in Fig. 1.

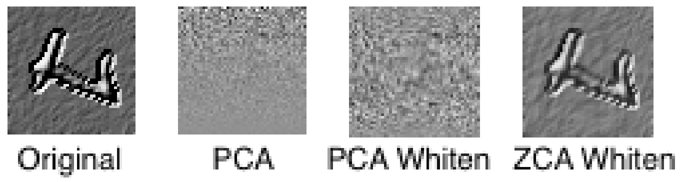

Fig. 1: The first image from the MNIST+ dataset being processed by PCA, PCA whiten and ZCA whiten

V. Algorithm Selection, Optimization, and Parameters

We selected the following algorithms for classification:

A. Baseline Algorithm

We used the logistic regression model as our baseline algorithm. Logistic regression is a powerful probabilistic classifier for binary classification, but it has to be expanded in order to cover the scope of this project. Thus, we chose to implement a one-versus all model where one logistic regression classifier is trained to distinguish examples of a certain class versus examples from all other classes. As a result, ten logistic regression classifiers were implemented to determine the probability of an example belonging in each of the ten possible classes (digits from 0-9). Examples are then assigned to the class with the logistic regression classifier that produces the highest probability.

1) Implementation: The implementation of the logistic regression classifier was based on Jason Rennie's derivation [9]. In this case, each example is labeled as +1 or -1 instead of the more popular notation y = {1,0}. An example is represented by: *x*, which is a set of *m* number of features $(x_1...x_m)$. An additional feature of $x_0 = 1$ was added to to feature set as a bias. The probability of an example belonging in the class is given by:

$$P(y=+1|x;\theta) = g(\sum_{k=0}^{m}\theta_k x_k)$$

where *g(z)* is the logistic (or sigmoid) function $\frac{1}{1+e^{-z}}$. θ_k represents the weight for the k^{th} feature.

The goal of training is to learn weights to maximize the log-likelihood of the data. Because y = {+1,-1} and the log property that 1-*g(z) = g(-z)*, the log-likelihood function can be written as:

$$L(\theta) = \sum_{i=1}^{n} log(g(y^i z^i))$$

where *n* represent the number of examples and $z^i = \sum_k \theta_k x_k^i$.

The weights are determined using gradient descent until convergence is reached. The gradient is calculated by taking the partial derivative with respect to θ. Given that $\frac{dg(z)}{dz}$ *=g(z)g(-z)dz* the gradient of the log-likelihood with respect the the k^{th} weight is:

$$\frac{dL}{d\theta_w} = \sum_{i=1}^{n} y^i x_k^i g(-y^i z^i)$$

Increasing the weight vector, θ, in the direction of the gradient would increase the log-likelihood. Thus, the update function for the weight set θ is represented by:

$$\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} y^i x_k^i g(-y^i z^i)$$

Where α is the learning rate.

2) Selection of learning rate and number of iterations: We experimented with many different learning rates e.g. 0.1, 0.01, 0.001, 0.0001, the results are similar because the fast converge property we see in PCA dataset. To show clearly the convergence process, we used very small learning rates with decaying. Ultimately, the learning rate was chosen to be $\frac{0.0005}{Number\ of\ iterations}$ to produce Fig. 3 with convergence being reached after 300 iterations.

B. Neural Network

The neural network allows to compute much more complex functions given the input and therefore process data to get better prediction results. We used a fully-connected neural network, which means every hidden unit has the same number of weights as the number of feature inputs. We implemented it as follows:

1) Initialize the weights vectors: We define the number of layers for the neural network (n) , and the number of hidden units per layer (h). To each layer is added a bias term so the weight vector will have a dimension increased by 1 for each input.Therefore each unit of the first layer would have a weight vector of dimensions $(2304 + 1) \times h$, each unit of the hidden layers would have a weight vector of dimensions $(h + 1) \times h$ and finally each unit of the output layer would have a weight vector of dimensions $(h + 1) \times 10$.

To initialize these weights, we pick random numbers that are for each layer in the interval $[-\epsilon, \epsilon]$ where

$$\epsilon = \sqrt{\frac{6}{L_{in} + L_{out}}}$$

with L_{in} the number of inputs for this layer and L_{out} the number of outputs for this layer. This allows us to speed up the learning process from the beginning (this idea comes from the machine learning course online of Stanford given by Andrew Ng)[18].

2) Train with training data: Given the input data ($x = 2304$ integers from 0 to 255 scales of grey) for each example, we feed it to the first hidden layer and get a vector of size equal to the number of hidden units in this layer j with values that are equal to

$$z_{i,j} = \sigma(w_{i,j} z_{i-1}) = \frac{1}{1 + e^{(-w_{i,j} z_{i-1})}}$$

where $w_{i,j}$ is the weight of the hidden unit i in the layer j and z_{i-1} the vector of inputs for this hidden unit. σ is the transfer function from the previous to the current layer. Recall that in our case the neural network is fully connected so every unit in the layer takes as input the outputs of every unit in the previous layer.

We go on using a feedforward propagation to get the prediction for the input given. The final layer is the output layer where the output vector is this time of size 10 to predict the given example. The output is the index of the maximum of the last output vector. We use the dataset by batches of size 100 and try to learn from each of these batches how to modify the weights in order to get a better prediction.

For the choice of the transfer function we began with the sigmoid function which was suggested in class, and then we moved to a mix of hyperbolic tangent $\tanh$ for the input and output layers and the softmax function for the output. We need such a function since in our case we are doing a multi-classification neural network, so the output activation function must satisfy the following conditions: [20]

$$0 < softmax_i < 1$$
$$\sum_i softmax_i = 1$$
$$softmax_i(x) = \frac{e^{x_i}}{\sum e^{x_i}}$$

The softmax function satisfies the following conditions and has allowed us to get better results. We experimented with different transfer functions to find the best combination, the result is shown in Fig. 5.

3) Minimizing the cost function: A first approach is to compute the derivative of the cost function for every weight of each unit in each layer. This is possible by performing a back propagation and computing the error that each weight is responsible for. It has been shown in class that the derivative can be easily computed depending on the error computed at the next layer and the inputs and outputs of the current layer. This back propagation gives us the $\delta_{i,j}$ for each weight of unit i in layer j that measures how far it was from the predicting the correct value.

Then we can update the weights by using a gradient descent and therefore do the following update :

$$w_{i,j} = w_{i,j} + \alpha\delta_{i,j}$$
$$\delta_{i,j} = w_{i-1,j}^{T}\delta_{i+1,j}.g'(z_i)$$

Here g is the transfer function and z_i is the input to layer i. Another approach which allows us to converge faster is to use a minimizing function such as $scipy.minimize$ using the conjugate gradient method, feed the computed gradients to the minimizing function to speed up the process and therefore get the best possible update of the weights. This is what we used to get better results.

4) Observations: Even though our implementation supports more than one hidden layer, we discovered that even with one layers it takes around 50 minutes to finish one iteration, using 500 per batch. We elected not to use multi hidden layers, but attempted to use different numbers of hidden units. The result is shown in Fig. 6.

C. Linear Support Vector Machine

We also used SVM as another baseline algorithm. We investigated the linear kernel and Gaussian kernel using scikit learn in python[15], the results are 31.78% and 38.73%, which are much worse than neural networks. We decided not to spend much time on this method, in stead focusing on convolutional neural network.

D. Convolutional Neural Network

Fully connected multilayer neural network is a very powerful learner that can be capable of learning very complex patterns in images. But in practice, this can be a computation challenge. A large number of parameters increase the capacity of the system and therefore requires a lager training set. In addition, the memory requirement to store so many weights may rule out certain hardware implementations. On the other hand, this architecture will entirely ignore the topology of the input. The input variables can be represented in any order without affecting the outcome of the training. On the contrary, images have a strong 2D local structure: pixels that are spatially nearby are highly correlated. Local correlations are the reasons for the well-known advantages of extracting and combining local features before recognizing spatial or temporal objects, because configurations of neighboring variables can be classified into a small number of categories. Convolutional networks force the extraction of local features by restricting the receptive fields of hidden units to be local.[4]

Convolutional Networks combine three architectural ideas to ensure some degree of shift , scale, and distortion invariance: local receptive fields, shared weights, and spatial sub-sampling. Each unit in the convolution layer receives inputs from a set of units located in a small neighborhood in the previous layer. This idea of connecting units to local receptive fields on the input is in line with Hubel and Wiesel's discovery of locally sensitive, orientation-selective neurons in the cat's visual system.[1] With local receptive fields, neurons can extract elementary visual features such as oriented edges, end-points, corners. These features are then combined by the subsequent layers in order to detect higher-order features.[4] This process can be illustrated in Fig. 2.[17]

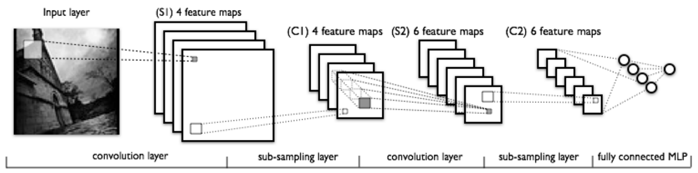

Fig. 2: The composition of a typical convolutional neural network

We used Theano to help implement this algorithm and experienced with many different configurations.[17] Because training this learner and finding convergence is very time comsuming, we did not try out every combinations to find the optimal configurations. The first configuration we did was a two convolution layers with a fully connected layer. Because the example was used on images with resolution of 28x28, our dataset is 48x48, we have to make some adjustments. In the end, we used (7,7) as the filter of the first convolution layer, and used max pool among (3x3) features, with 20 feature maps; the second layer also used (7,7) as the filter, but used max pool among (2x2) features, with 50 feature maps. The last hidden layer has 512 hidden units. This configuration gave us 87.32% accuracy on Kaggle leader-board.

We also used 3 convolution layers on the ZCA dataset. We used (2x2) max pool on all three layers, the first one we used (5x5) filter, and (3x3) for the later two. The kernel for each layer is (20, 40, 80). This took significantly more time to train, but it gave us 93.21% on the leader-board.

We also tried 4 convolution layers, more hidden units in the last layer, more fully connected layers , all had worse results. We argue that too many parameters may have caused overfitting. The comparison is illustrated in Fig. 7.

VI. Testing and Validation Results

All results are shown below, unless noted otherwise, used first 90% as training data and last 10% as validation data.

To compare PCA processed dataset with the original dataset, we train these two datasets with limited features. In PCA we discard some features at the end, in original, we discard the same number of features randomly, and use logistic regression with softmax as transfer function. The result is shown in fig 3. PCA has clear advantage in every number of features. We can also see that with discarding almost 3/4 of the features, PCA dataset can still make almost optimal prediction.

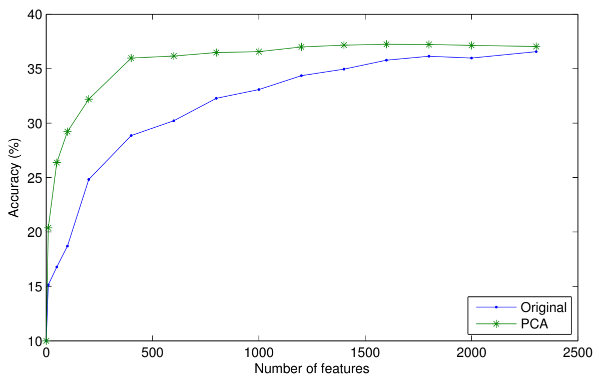

Fig. 3: Logistic regression on different numbers of features in original dataset and PCA dataset

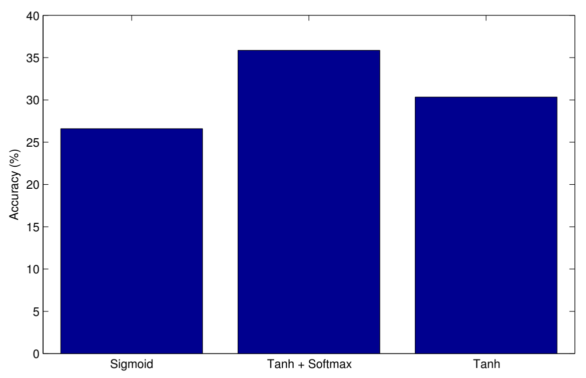

Fig. 5: Different transfer functions affect the result of a neural network

To understand how different data preprocessing methods affect convergence, we used 4 different dataset with sigmoid logistic regression, and plot their convergence pattern over the gradient descent process. A decaying learning rate of $\frac{0.0005}{Number\ of\ iterations}$ was used for gradient descent. We can see that ZCA and PCA not only gave us better final results, they also converges much faster, saving computation time. PCA with whiten does not produce better results in the end, this can be explained by arguing PCA whiten will actually remove some information as stated in Section IV.

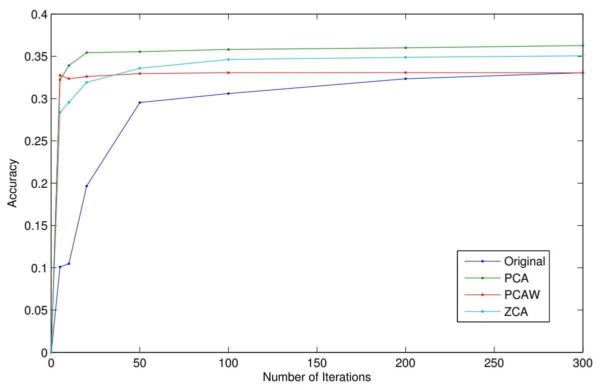

Fig. 4: Logistic regression with gradient descent on different preprocessed data

For neural network, we experimented on different transfer functions in the hidden units and the output units. Because the training time for our implemented version of neural network takes very long, we used 32 hidden units and doing only 20 iterations. The results are shown in the following graph. We can see that tanh for hidden units and softmax for output units is the best solution. From now on we will use this combination for the reminder of the report.

To investigate how the number of hidden units affects the results, we used Theano framework to implement neural network with different configurations, in favor of its efficiency. We can see from the following graph that more hidden units tend to have better predictions. From the experiments we also found out it took significantly more time to converge for more hidden units.

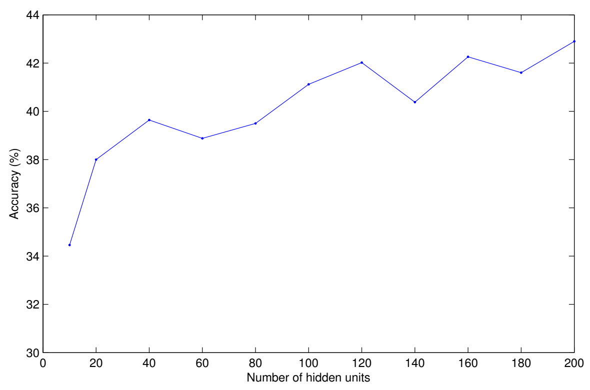

Fig. 6: Neural network using different hidden units

We used different configurations of convolutional neural networks, the best performance was given by a 3-convolution-layers configuration, which is more than 93% accuracy on Kaggle competition. Because these methods takes very long to train, we did not use a separate validation set, in stead, all data shown here is based on Kaggle result. In the case of PCA dataset, the result is much less than the rest, almost close to random guessing, this abnormality will be discussed in Section VII.

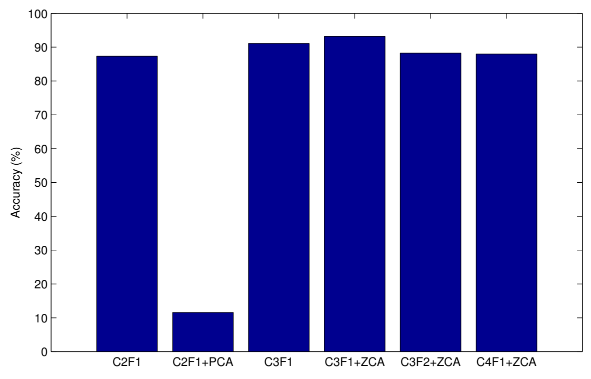

Fig. 7: Performance of different convolutional neural networks, based on Kaggle result. CxFy denotes x convolution layers, usually with max pool, and y fully connected layers.

Comparing all the best results from each algorithm, we can see that convolutional neural networks excels the rest, thanks to its architecture of combining adjacent pixels into patterns.

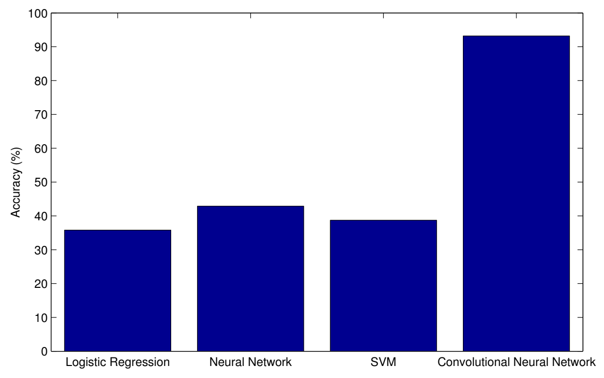

Fig. 8: Performance of each algorithms with best configuration, based on Kaggle result.

VII. Discussion

A. BFGS algorithm on gradient descent

Other than basic gradient descent, the BFGS (Broyden–Fletcher–Goldfarb–Shanno) algorithm was used to determine the weights of θ. The scipy.optimize.fin_bfgs package was used for this purpose. This package takes a function f to be minimized (negative log-likelihood), an initial guess of θ which was initialized to 0s), and the *fprime* function which is gradient of f (the negative gradient of log-likelihood). The parameters trained with BFGS produced an accuracy of 32.66% on the validation set and 34.58% on the public test set on Kaggle. Interestingly, these performances were slightly lower compared to the results of using parameters trained with basic gradient descent. After discussion, we conclude that this method may benefit from a more complex algorithm, for logistic regression, this introduced more hyper parameters and thus more uncertainty. This way, logistic regression may not be able to benefit from such gradient descent method.

B. Unstable results in variant number of hidden units

We expected to see constant increase of accuracy for increasing hidden units, as in increasing complexity of the model in other algorithms. But in stead the result increases with obvious oscillation as shown in Fig. 6. We argue that this may be caused by our use of mini-batch gradient descent method (we used 100 samples per batch and a maximum of 30 epochs), this method saves much time but introduced new variance into the model. Because we only made limited iterations through whole data, the results can be more unstable, causing oscillation in the graph.

C. PCA preprocessing with convolution neural network

When implementing convolution neural network on PCA dataset, we get nearly random results, as shown in Fig. 7. This is in direct contradiction to what we saw in logistic regression. To understand this phenomenon, we can refer to the image data after each transformation in Fig. 1. Even though PCA compressed the most useful data onto the first few rows, but it messed up the entire image space. In ZCA however the image stay roughly the same, just higher contrast. Because convolutional neural network assumes there is a strong correlation between adjacent pixels, PCA data no longer preserve this information. Thus PCA performs much worse than ZCA and original data on convolutional neural network.

D. Continue training complex models

More complex learning algorithm takes much longer to train, for example convolutional neural networks takes around half an hour to go through one iteration of the entire training samples. Sometimes when we need to tweak the learning rate, or use different proportion of training set, we need to rerun from the start. To combat this problem, we modified the program save the weights matrices on disk every time it finds a better solution, this way we can continue running the gradient descent even after restarting the program, this saved us a lot of time.

✓ **We hereby state that all the work presented in this report is that of the authors.**

Reference

[1] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex,” *Journal of Physiology (London)*, vol. 160, 1962.

[2] R. W. Picard, T. Kabir, and F. Liu, “Real-time recognition with the entire brodatz texture database,” in *IEEE Conf. on Comp. Vis. and Pattern Recognition*, 1993, pp. 638–639.

[3] A. Bell and T. J. Sejnowski, “Edges are the ‘independent components’ of natural scenes.,” in *In Advances in Neural Information Processing Systems*, MIT Press, 1996, pp. 831–837.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[5] C. C. Yann LeCun and C. J. Burges, *The mnist database of handwritten digits*, 1998.

[6] J. C. Platt, *Advances in neural information processing systems*. 1999, vol. 1.

[7] J. I.T., *Principal Component Analysis*, 2nd, ser. Springer Series in Statistics. Springer, NY, 2002.

[8] Y. Lee, Y. Lin, and G. Wahba, "Handwritten digit recognition: Benchmarking of state-of-the-art techniques," *Pattern Recognition*, vol. 36, pp. 2271–2285, 2003.

[9] J. Rennie, *Logistic regression*, Massachusetts Institute of Technology, 2003.

[10] G. Jain and J. Ko, "Handwritten digits recognition," in *ECE462 Project Report*, University of Toronto, 2008.

[11] M. K.Jarret K.Kavukcuoglu and Y.LeCun, "What is the best multi-stage architecture for object recognition," in *IEEE international conference for computer vision*, 2009.

[12] E.Goumhei, "Contextual image classification with support vector machine," in *Master's thesis*, University of Twente, 2010.

[13] J. D.Ciresan U.Meier, L.M.Gambardella, and J.Schmidhuber, "Flexible high performance convolution neural networks for image classification," in *IJCAI*, 2011.

[14] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE signal processing magazine*, pp. 141–142, 2011.

[15] 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.

[16] D. Krueger, *Make_mnistplus.py*, 2013.

[17] T. D. Team, *Convolutional neural networks (lenet)*, 2013.

[18] A. Ng, *Https://class.coursera.org/ml-007*, 2014.

[19] S. Raschka, *Implementing a principal component analysis (pca) in python step by step*, 2014.

[20] P. Giudici, *Applied Data Mining: Statistical Methods for Business and Industry*.