Comp 551 Project 3: Modified digits

Karine Mellata
karine.mellata@mail.mcgill.ca
260741673

Isaac Chan
isaac.chan@mail.mcgill.ca
260624096

Xingwei Cao
xingwei.cao@mail.mcgill.ca
260572128

I. Introduction

Team Name: **Soldiers of Trottier**

The problem this report addresses is that of a supervised image classification task. In Machine Learning literature, the task of classification consists of predicting the class of an input. The image classification problem is one of the most often investigated research areas at the moment as it has a wide range of industry applications.

In this project, we were assigned to train three models for classifying data set containing modified handwritten digits. This data set is called modified MNIST. Following the original MNIST data [1], the modified MNIST data set consists of 50000 training images and 10000 test images. Each image consists of two digits and one operator. The operator is represented as either the letter "A" for addition or the letter "M" for multiplication, in either uppercase or lowercase. The input is a 64×64 pixel image overlaying a random texture and containing 2 digits and one letter, as opposed to the standard MNIST dataset containing one centered digit.

The first approach uses a **Logistic Regression** classifier as our baseline learner, implemented with the help of the scikit-learn library [2]. The second approach uses a manually-implemented **Fully Connected Feed-Forward Neural Network**. The third approach uses a **Convolutional Neural Network** (CNN) implemented with TensorFlow [3].

II. Related Work

As mentionned above, image classification, a specific branch of pattern recognition in computer vision, is the task of assigning a class label to an input image from a fixed set of categories. According to Machine Learning literature, image classification has been one of the most investigated fields over the past several decades. One of the most popular models used is the multiplayer perceptron model (MLP).

In particular, convolutional neural networks (CNN) achieve the highest performance among various machine learning applications [5], [6]. Particularly, Litjens et al. investigates the application of image classification models to medical image analysis. [5] In their recent work, they present that out of various statistical models such as Restricted Boltzman Machine (RBM) [7], Auto-encoders [8] and Recurrent Neural Networks, CNN remained the most frequently used for medical image analysis.

In fact, CNNs have been used for decades in image classification [9]. Sahiner et al. investigated the application of CNN for classifying various types of breast tissue. Although the classification is binary and recent algorithmic advances would allow this to be considered a relatively achievable task, the result presented shows that the accuracy 80% is a good example of how powerful CNNs are for image classification.

The algorithmic advances in image classification tasks have attracted a lot of attention lately. One of the most remarkable models is the Deep Belief Network (DBN). Hinton et al. successfully achieved high accuracy in image classification with a deep structure.[10] After such work, the trend of the deep model for image classification has certainly thrived until the time of this article. The structural advance of deep models has been one of the most investigated areas of image classification research.

One of the largest data sets available for image classification is ImageNet by Deng et al. [12]. As stated above, deep learning models have been highly successful in the field of image classification. Deep Residual Learning Networks (Resnet) [11] have been one of the most successful models for image classification. By introducing a residual structure into deep convolutional models, He et al. shows that deep residual model achieves 21% error rate for the ImageNet dataset, and as low as 3.57% error rate using an ensemble of these residual nets [11].

III. Problem Representation

As per the problem statement, the training set provided contains 50,000 grey-scale images of size 64×64 pixels. The output is a digit representing the result of the operation, and it can be one of 40 unique classes. It is possible to see in Figure 1 that the majority of the outputs lay between 0 and 24 within the given data set.

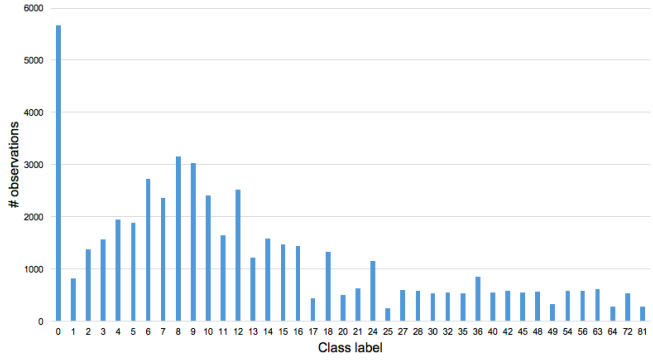

Figure 1: Distribution of classes over training set

(a) First five samples from training data.

(b) First five samples after pre-processing.

Figure 2: Visualization of first five samples before and after our pre-processing.

A. Pre-processing

Our image pre-processing strategy consists of two parts. The first part is to filter out the pixel values by a certain threshold. The second part is to normalize the image. In particular, we first set any pixel value that is less than 230 to be 0. We also set any pixel value that is larger than the threshold to be 255, maximum value. The reason of such operation is to filter out the random noise introduced to the image data set. After the operation described above, we normalized the input to the range $[0, 1]$. This pre-processing aims to let model learn faster. The images before and after pre-processing is shown in Figure 2.

B. Feature selection

With a data set this large (50,000 examples of 4096 features each), it is necessary to use a technique for dimensionality reduction. The Principal Component Analysis (PCA) method's goal is to reduce the dimensionality of a data set containing co-dependent data, while retaining as much variance as possible [4]. This method transforms the current features to a new set of variables, which are called the principal components and are independent. The principal components are ordered such that a low number of components retain most of the variation present in the data set. Although greatly shortening the computation time, this method can discard useful information since it is strongly focused on variance and there is not always a direct relationship between variance and predictive power. For this reason, this method was only used for feature extraction of our baseline learner (logistic regression).

IV. Algorithm Implementation

A. Logistic Regression

1) Selection: The first and simplest algorithm implemented was a logistic regression classifier, used as a baseline. This classifier measures the relation between the binary output and the features using a sigmoid curve, with $\sigma(w^T x_i)$ being the probability y_i = 1 given x_i :

$$\sigma(w^T x) = \frac{1}{1 + e^{-w^T x}} \tag{1}$$

The logistic regression classifier was implemented using the scikit built-in SAGA solver, which helped achieve the fastest convergence, with a L1-norm regularization. Contrary to what the name suggests, logistic regression is a binary classifier, which is why a method for reducing the multi-class problem to a binary problem was used. The One-vs-Rest (OVR) solver was used, which involved training the logistic regression binary classifier for each class with samples of that class as positive examples and all other as negative examples. In addition, using the built-in Standard Scaler function in sci-kit, the input data was normalized by removing the mean and scaling to unit variance.

2) Paramater selection: As mentioned above when discussing feature selection, PCA was used for dimensionality reduction. Through cross-validation, we tested with different principal components values. The data set was divided with 80% for training and 20% for testing. The optimal number of principal components chosen for logistic regression was 128. As displayed in Figure 3, this number was chosen over n=24, as we realized n=24 may be discarding too much valuable information as it was simply outputting zeros (0 being the most prominent class as shown in Figure 1). This meant that this would lead to an over-fitted model that generalizes poorly.

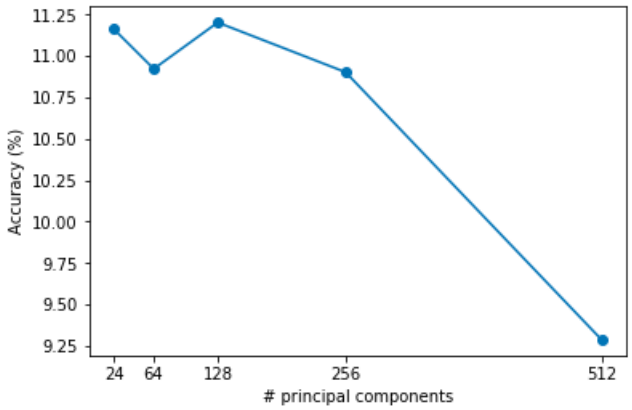

Figure 3: Accuracy through cross-validation achieved with different numbers of principal components.

The regularization hyper-parameter (inverse of the L1 penalty) has also been chosen through cross-validation, with a 5-fold cross-validation scheme. The lowest value tested, which was $C = 100/testing\_samples$ with 10000 testing samples ($C = 0.01$) gave us the best results. This was to be expected, because a higher regularization hyper-parameter allowed us to avoid over-fitting the data.

B. Feed-forward Neural Network

The feed-forward neural network relies on the usage of a set of fully connected neurons. We apply a logistic activation function as our nonlinear activation function using a single hidden layer consisting of 40 neurons. The output utilizes a softmax function to generate a probability distribution of the outputs, and applies backpropagation to compute the gradients of the cross-entropy loss function defined by:

$$E = -\sum_{i=1}^{nout}(t_i log(y_i) + (1-t_i) log(1-y_i)) \quad (2)$$

where the output is defined by a one-hot encoding of the classes. The pre-activation of each of those neurons is defined by

$$s_i = \sum_{j=1} h_j w_{ji} \quad (3)$$

where w_{ji} is the weight connecting layer j to layer i and h_j is the output of layer j. Our output layer utilized the softmax activation function to derive the final probability distribution

$$y_i = \frac{e^{-s_i}}{\sum_c^{n_{class}} e^{s_c}} \quad (4)$$

And the derivative of the output with respect to the weight that goes from the previous hidden layer to the output layer, with the addition of the L2 Regularization term can be defined by

$$\frac{\delta E}{\delta w_{ji}} = (y_i - t_i)h_j + 2\lambda W^2 \quad (5)$$

The derivation of which is available in the Appendix. The derivative of the weights, with addition of the L2 Regularization term, going into the hidden layer can be computed using the chain rule.

$$\frac{\delta E}{\delta w_{kj}} = \frac{\delta E}{\delta s_j}\frac{\delta s_j}{\delta w_{kj}} = \sum_{i=1}^{nout}(y - t_i)(w_{ji})(h_j(1-h_j))(x_k) + 2\lambda W^2 \quad (6)$$

Where the derivation for this is available in the Appendix.

C. Convolutional Neural Network

Similar to a feed-forward neural network, a convolutional neural networks consist of neurons that have learnable weights and biases. The difference between the feed-forward neural network and the CNN is the assumption of input data. A CNN will assume that the input data is aligned with a certain axis of the data. For example, a colored image is a third order tensor that is separated by the channels of basic colors. By making such assumptions, CNNs are able to learn kernel matrices that takes a certain number of input channel and outputs into another number of channel.

V. Testing and Validation

Model	Training Acc(%)	Validation Acc(%)	Test Acc(%)
CNN	**99.8**	N/A	**65.2**
MLP	11.8	**11.5**	12.1
Logistic Regression	11.2	11.3	8.8

Table I: Summarizing models we adopt for this project with training, validation and test accuracy.[1]

[1]Note that the test accuracy is partial as we are able to test 30% of all test data on Kaggle submissions.

A. Logistic Regression

The classifier was trained with 40000 and tested on 10000 samples. Using 5-fold cross-validation with each time 20% of the data randomly selected for testing, the model achieved an accuracy of approximately 11.3%.

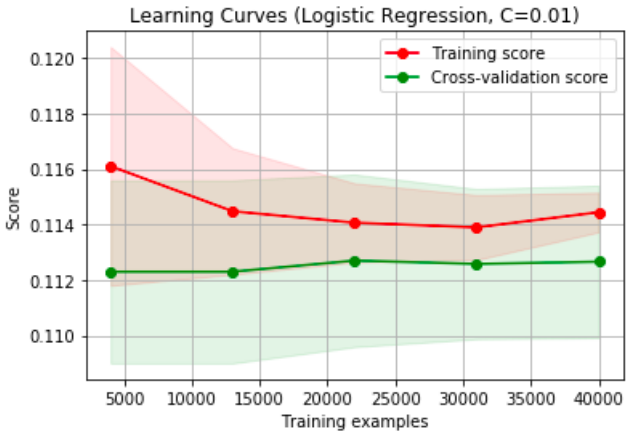

Figure 4: Learning curve: Accuracy versus training examples

As expected, the training score starts higher and slowly decreases (for the exception of a small increase at the last iteration), and the validation score slightly increases. Such low score is due to the fact that the decision boundary for a logistic classifier is linear. Therefore, the classifier needs the inputs to be linearly separable, when they are not within our data set.

B. Feed-forward Neural Network

The feed-forward neural is constructed with a single hidden layer. The weights going into the output layer are drawn from a uniform distribution ranging from -0.05 to 0.05, and the weights going into the hidden layer are drawn from a uniform distribution ranging from -0.01 to 0.01. The process of cross validation for all the hyperparameters (L2 Regularization (lambda) parameter, learning rate(alpha), nodes in hidden layer) were all validated using a 60/10/10/10/10 split of the data. The last 10% of the data were used as a means of estimating the model's accuracy on a test set.

Hyperparamater	Value
Learning Rate	0.0003
L2-regularization coefficient	0.0001
Hidden layer size	40

Table II: Hyperparameters found through cross validation

The methodology that was exercised in the cross validation is that an estimate would first be made on the appropriate magnitude (in orders of 10) for each of the hyperparameters, then the adjustment would be made on a more minute level until we found the hyperparameter values that gave the highest validation set accuracy.

C. Convolutional Neural Network

We constructed a deep convolutional model with 5 convolutional layers and 2 fully connected layers. Each weight is

initialized with normal distribution with $\mu = 0.0$ and $\sigma = 0.1$. We inserted dropout after the first convolutional layer with a rate of 0.25 [13]. The second dropout was inserted between the first fully connected layer and the second fully connected layer with a dropout rate of 0.5. We used the ADAM optimizer [14] with a learning rate of 0.001 and trained the model for 750k iterations with a batch size of 100. The first layer maps 1 channel to 16 channels. At the third layer, the output channel is set to be 32 and at the two last convolution layers, the output channel is set to be 64 and 128 respectively. At the first, third and last layer, we inserted a pooling layer by factor 2 to make the training of the CNN more efficient.

After the convolution layers, we flattened the output into a vector of size 8192. The vector is then fed to a fully connected layer that maps to the space $\mathbb{R}^{1024}$. After a dropout layer, the output vector from the first densely connected layer is fed to another fully connected layer that maps to a space $\mathbb{R}^{40}$. Following the tradition, we calculate the soft-max operation of the output layer then adopt cross entropy for the loss function. The graph of cross entropy loss to the training steps is shown in Figure 5. It is observable that at training step $40k$ that the training loss is nearly converged to 0.

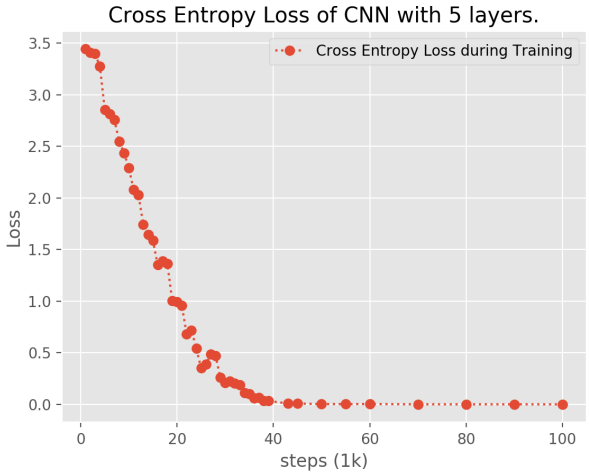

Figure 5: Figure of cross entropy loss of CNN with five layers.

VI. Discussion

In this section, we discuss our methodology of pre-processing, model selection and experimenting results followed by a conclusion.

For data pre-processing, as stated in Section 3, we filtered the input images by a certain threshold and normalized them. The primal reason that we adopted this method over others is because of the performance of our models. As it was shown in the report, CNN performs the best among various machine learning models, hence why we tested different pre-processing methods.

Instead of normalizing the input, we fed the input into the CNN without normalization, meaning that the value of pixel is in a space $[0, 255]$, not $[0, 1]$. We observed during the experiment that such data would make the model take a longer time to converge to global minimum, and even often diverged or failed to train.

During the experiment, we tested various other threshold values for pixel filtering such as 120, 200 and 220. Among all the thresholds, we observed that the training is most stable when we adopt a threshold of pixel value of 230 or 235. We consider this being due to the magnitude of noise added to the input. Since the noise is not consistent throughout the data set, it is natural that a higher threshold such as 230 or 235 will filter most of the noise out while keeping the information about the digits and characters valid and readable.

For this project, we were allowed to select and build our own model to tackle the problem. As described in Section 2, CNN has been shown by some researchers [5], [6], [9] that it is one of the best performing methodologies for image classification. We followed the result from the survey [5] and adopted CNN as our last optional model.

Given the results, we considered our logistic regression model and feed-forward neural network model to under-fit to the training data respectively. The test error of logistic regression and MLP is quite high compared to that of CNN. The simple explanation to the symptom of under-fitting is because the model complexity of both of logistic regression and MLP are not enough to capture the complexity of data. In other words, the number of model parameters were not enough to fully train data set with. In addition, for a logistic regression algorithm to be successful, an extremely large number of training data is necessary, especially for a $64 \times 64 pixel simage$.

As shown by He et al. [11], state-of-the-art deep models require approximately 1.7 millions number of model parameters to successfully learn data set such as CIFAR-10 [16]. We consider the complexity of the given data set to be approximately similar to that of CIFAR-10. Both of our model have far less number of model parameters than 1.7 millions. If our hypothesis is correct, it is most likely impossible to successfully train logistic regression and feed-forward network.

The test error by our CNN model shows that it does not generalize well to the test data set. Although we consider that we do not need very large model such as [11], the symptom of over-fitting is shown in Figure 5. We know that the typical indication of over-fitting is having high training accuracy while having low test-accuracy. Although we cannot fully reason the gap between our training and test error, we consider it is solvable by lowering learning rate or adding L2 regularization terms.

In conclusion, we were assigned to train three machine learning models to learn modified handwritten digit data set. We developed logistic regression model, feed-forward neural network model and convolutional neural network model. We tested our models with provided test data and submit the result to Kaggle competition. We conclude that within models we adopt, convolutional neural network results in the highest test accuracy.

VII. Statement of Contribution

Mellata contributed to the article by constructing the logistic regression model and comprehensive writing of the report. Chan's contributions are development of feed-forward neural

network without using libraries and comprehensive writing of the report. The contributions Cao made are the development of CNN and writing and reviewing of the report. We hereby state that all the work presented in this report is that of the authors.

REFERENCES

[1] LeCun, Yann. "The MNIST database of handwritten digits." http://yann. lecun. com/exdb/mnist/ (1998).

[2] Abraham et al. "Machine learning for neuroimaging with scikit-learn" 10.3389/fninf.2014.00014 (2014).

[3] Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." arXiv preprint arXiv:1603.04467 (2016).

[4] I, Joliffe. "Principal Component Analysis". Springer Verlag (1986).

[5] Litjens, Geert, et al. "A survey on deep learning in medical image analysis." arXiv preprint arXiv:1702.05747 (2017).

[6] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[7] Fischer, Asja, and Christian Igel. "An introduction to restricted Boltzmann machines." Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications (2012): 14-36.

[8] Vincent, Pascal, et al. "Extracting and composing robust features with denoising autoencoders." Proceedings of the 25th international conference on Machine learning. ACM, 2008.

[9] Sahiner, Berkman, et al. "Classification of mass and normal breast tissue: a convolution neural network classifier with spatial domain and texture images." IEEE transactions on Medical Imaging 15.5 (1996): 598-610.

[10] Hinton, Geoffrey E., Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.

[11] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[12] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

[13] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research 15.1 (2014): 1929-1958.

[14] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[15] Cohen, Gregory, et al. "EMNIST: an extension of MNIST to handwritten letters." arXiv preprint arXiv:1702.05373 (2017).

[16] Coates, Adam, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." Proceedings of the fourteenth international conference on artificial intelligence and statistics. 2011.

VIII. APPENDIX

A. Derivation of derivative of error with respect to the weights going into the output layer:

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta y_i}\frac{\delta y_i}{\delta s_i}\frac{\delta s_i}{\delta w_{ji}} \tag{7}$$

Examining each part in turn, we get

$$\frac{\delta E}{\delta y_i} = \frac{-t_i}{y_i} + \frac{1-t_i}{1-y_i} = \frac{y_i - t_i}{y_i(1-y_i)} \tag{8}$$

$$\frac{\delta y_i}{\delta s_i} = y_i(1-y_i) \tag{9}$$

$$\frac{\delta s_i}{\delta w_{ji}} = h_j \tag{10}$$

In which we attain:

$$\frac{\delta E}{\delta w_{ji}} = (y_i - t_i)h_j \tag{11}$$

B. Derivation of derivative of error with respect to weights going into hidden layer:

$$\frac{\delta E}{\delta s_j} = \sum_{i=1}^{n_{class}} \frac{\delta E}{\delta s_i}\frac{\delta s_i}{\delta h_j}\frac{\delta h_j}{\delta s_j} = \sum_{i=1}^{n_{class}} (y_i - t_i)(w_{ji})(h_j(1-h_j)) \tag{12}$$

and

$$\frac{\delta E}{\delta w_{kj}} = \frac{\delta E}{\delta s_j}\frac{\delta s_j}{\delta w_{kj}} \tag{13}$$

and so, we can conclude it equals

$$\sum_{i=1}^{n_{class}} (y_i - t_i)(w_{ji})(h_j(1-h_j))(x_k) \tag{14}$$