Delft University of Technology

EE4C12 Machine Learning for Electrical Engineering
Project Report

Image Classification Exploiting Sparsity

Authors:

Nikos Fotopoulos (No. 6079040)
n.fotopoulos-1@student.tudelft.nl
Junchi Liu (No. 5983533)
j.liu-61@student.tudelft.nl

March 20, 2024

Summary

This project has two primary objectives. The first goal is to develop a neural network capable of accurately classifying digit images from the MNIST dataset. For the second goal, we apply a compressed sensing technique to reduce the size of the MNIST dataset, and then reconstruct it using LASSO regression. Subsequently, we evaluate the model's performance on this reconstructed dataset to assess the quality of our compression. To determine the optimal values for the regularization parameter λ and the size of the compressed image vector m, we conduct a comprehensive grid search. This allows us to find the best values that not only provide a high-quality reconstruction, but also significantly reduce the dataset's size. Ultimately, we select m = 300 and $\lambda = 0.1$, achieving an impressive 98.08% accuracy on the test set of the original images and a 93.62% accuracy on the test set of the reconstructed images.

1 Data Pre-processing Activities

1.1 Detailed Pipeline

For the image pre-processing, we first scale the intensity of the 28 x 28 grayscale images from the range [0,255] to [0,1] by dividing with 255. We then flatten each one of them into 784×1 vectors. The MNIST dataset consists of 70000 images, so we end up with a 70000×784 matrix X. Using the compressed sensing method described in Section 1.2, we turn these original image vectors into compressed vectors of smaller dimension, and then we use LASSO regression to restructure them. At the same time, we build a neural network model and train it on the original image vectors, and evaluate its performance on the reconstructed set.

The workflow of data pre-processing is shown in Figure 1:

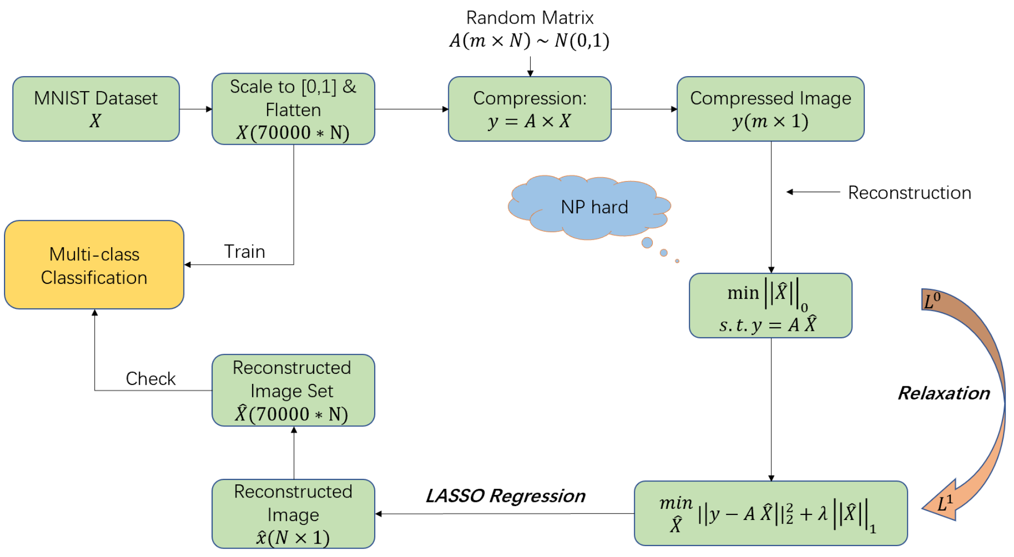

Random Matrix
$A(m \times N) \sim N(0,1)$
MNIST Dataset X
Scale to [0,1] & Flatten $X(70000 * N)$
Compression: $y = A \times X$
Compressed Image $y(m \times 1)$
Reconstruction
NP hard
Train
Multi-class Classification
$\min \|\hat{X}\|_0$ $s.t.\, y = A\hat{X}$
L^0
Relaxation
L^1
Check
Reconstructed Image Set $\hat{X}(70000 * N)$
Reconstructed Image $\hat{x}(N \times 1)$
LASSO Regression
$\min_{\hat{X}} \|y - A\hat{X}\|_2^2 + \lambda \|\hat{X}\|_1$

Figure 1: Workflow for data pre-processing

1.2 Compressed Sensing

For each $N \times 1 (N = 784)$ image vector $x, i = 1, 2, ..., 70000$, we left-multiply by a random matrix $A(m \times N)$ to get the compressed image vector y. We select A to follow the Gaussian distribution with zero mean and unit variance. Because Gaussian-distributed random matrices possess strong randomness, the construction of matrix A is not bound by specific patterns or structures, aiding in obtaining meaningful information with a reduced number of measurements.

The parameter m is determined by ourselves. It represents the amount of compression that is imposed on the data, and it fits the condition $1 < m < N$. The resulting vector y is a representation of x in fewer dimensions. However, trying to recover x from y is not straightforward, because

$$y = A \times x \tag{1}$$

is an underdetermined linear system of equations, having infinitely many solutions.

The least-squares solution to such problems is to minimize the L^2 norm, i.e. to minimize the amount of energy in the system. Mathematically, this is typically simple, involving a matrix multiplication with the pseudo-inverse of the sampled basis. However, we also have an additional piece of information: x is sparse. Solving the least squares indeed gives a solution that minimizes the the mean squared error, but we can not ensure that this solution is also sparse.

To enforce the sparsity constraint when solving for the underdetermined system of linear equations, one can minimize the number of nonzero components of the solution. This particular function, which quantifies the number of non-zero components within a vector, is referred to as the L^0 norm:

$$min||x||_0 \qquad s.t. \quad y = A \times x \tag{2}$$

Due to the NP-hard nature of the problem, solving it is not a straightforward task. Instead, we have to try to transform it into another, more relaxed form. Compressed sensing theory ensures that, in many instances, the L^1 norm can be used as a good approximation for the L^0 norm. This equivalence finding enables the solution of the L^1 problem, which is more manageable than the L^0 problem. Expressing the task of identifying the candidate with the smallest L^1 norm can be done quite straightforwardly, and there are already efficient solution methods available for it. Therefore problem in equation 2 can be transformed to the following:

$$min||x||_1 \qquad s.t. \quad y = A \times x \tag{3}$$

which in penalised form can be written as:

$$\min_x ||y - A \times x||_2^2 + \lambda||x||_1 \tag{4}$$

Thus, we can solve this problem by setting an appropriate parameter λ and using LASSO Regression. This way get the reconstructed image vector $\hat{x}$.

1.3 Lasso Regression

LASSO regression is one of many ways to solve the compressed sensing problem. The LASSO cost function consists of a weighted sum of error in reconstruction (sum of squares) and L^1 norm of the reconstructed signal (sum of absolute values). The LASSO regression model uses shrinkage, which refers to a process in which data values are pulled closer to the mean. It encourages simple, sparse models (i.e. models with fewer parameters).

2 Multi-Class Classification

In the given dataset, we have ten labels from 0 to 9, each corresponding to the digit depicted in each image. As such, this problem is a multi-class classification problem. Deep learning is widely used in image recognition due to its advantages, such as strong feature extraction ability and high recognition accuracy. Therefore, we developed a neural network model for this task.

2.1 Creating data set for training

For model optimization, we split the original image data set into training, validation, and test sub-sets. Their proportions are 70%, 10% and 20% respectively. It is worth noting here that we only used the original image set for training and not the reconstructed one. We only evaluate the model on the reconstructed image set to see how good our compression is.

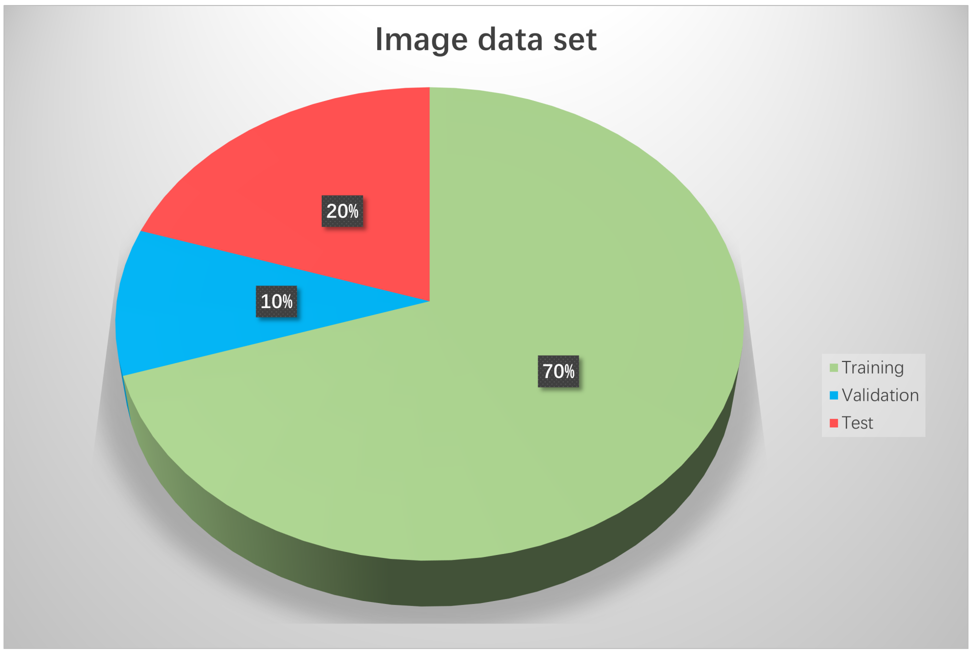


Figure 2: Schematic diagram of splitting data

The representation of labels (digits format, 0 to 9) is not suitable for the neural network prediction layer that outputs probabilities per class. A more suitable format is called a one-hot vector, a 10-dim vector with all elements 0, except for the index of the digit class. For example, if the label is 2, the equivalent one-hot vector is [0,0,1,0,0,0,0,0,0,0]. Thus, we converted each label set to a one-hot vector.

2.2 Developing the Neural Network Model

We define a function to implement our model. It has 4 input variables:

- 'input_dim': The dimension of the input data, which corresponds to the number of features. In MNIST set, it is $N = \mathbf{784}$
- 'num_labels': The number of categories or classes in the classification problem, i.e., **10** categories the input data should be classified into (0-9).
- 'hidden_layer_size': The number of neurons in the hidden layer of the model, which determines the network's complexity. In this problem, we chose to have only one hidden layer, because the classification task is not very difficult. We assign **256** to hidden layer size. 256 units are chosen since 128, 512 and 1,024 units have a higher validation loss.
- 'dropout=**0.45**': The dropout regularization probability, used to prevent overfitting.

In this function, we created a Sequential model, which is a simple neural network model in Keras where layers are stacked sequentially. Next, we defined three fully connected layers (Dense Layers):

- The first Dense layer specifies an input dimension of "*input_dim*" and an output dimension of "*hidden_layer_size*". It is used to accept the input data and map it to the hidden layer.
- The second Dense layer still has an output dimension of "*hidden_layer_size*" and is used to build connections between the hidden layers.
- The third Dense layer has an output dimension of "*num_labels*" and is used to map the output of the hidden layer to the final classification labels.

In summary, this code defines a feedforward neural network with two hidden layers for solving this classification problem. The model summary is shown in Figure 3:

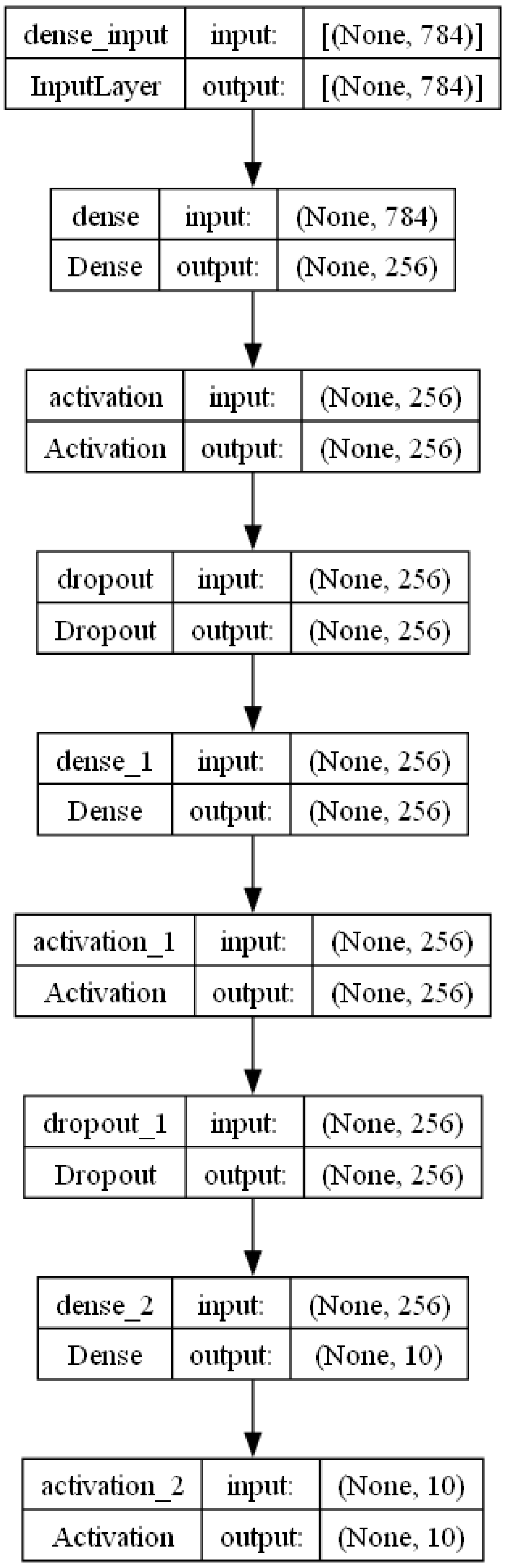

dense_input | input: | [(None, 784)]
InputLayer | output: | [(None, 784)]
dense | input: | (None, 784)
Dense | output: | (None, 256)
activation | input: | (None, 256)
Activation | output: | (None, 256)
dropout | input: | (None, 256)
Dropout | output: | (None, 256)
dense_1 | input: | (None, 256)
Dense | output: | (None, 256)
activation_1 | input: | (None, 256)
Activation | output: | (None, 256)
dropout_1 | input: | (None, 256)
Dropout | output: | (None, 256)
dense_2 | input: | (None, 256)
Dense | output: | (None, 10)
activation_2 | input: | (None, 10)
Activation | output: | (None, 10)

Figure 3: Neural Network Model

After that, we built our model with batch size = 128 and number of epochs is set to 40. However, an early stopping mechanism was implemented, so that the training stops if the validation loss does decrease after 3 epochs. This resulted in training lasting less than 40 epochs.

The compilation of the model can be done as follows:

```
batch_size = 128
num_epochs = 40
```

```
model = ClassificationNN(input_dim=N, num_labels=num_labels, hidden_layer_size=256, dropout
    ↪ =0.45)
model.compile(loss='categorical_crossentropy',
                    optimizer='adam',
                    metrics=['accuracy'])
```

1. **Loss function**

   In this example, we use categorical crossentropy as the loss function. It is defined as the negative of the sum of the product of the target and the logarithm of the prediction.

   For classification by category, categorical crossentropy or mean squared error is a good choice after the softmax activation layer. For binary classification problems, the binary crossentropy loss function is normally used after a sigmoid activation layer.

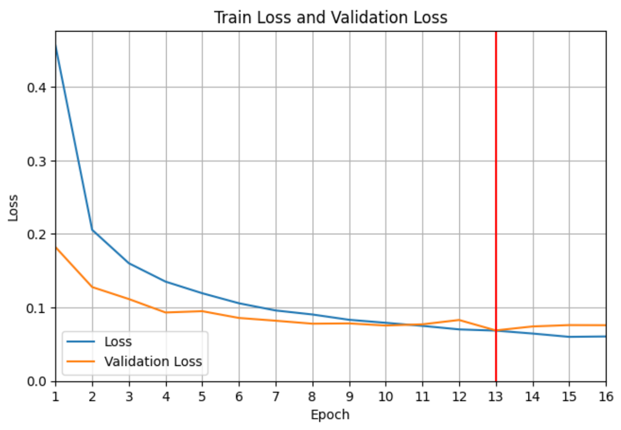
2. **Optimization**

   In Keras, there are several choices for optimizers. The most commonly used optimizers are; Stochastic Gradient Descent (SGD), Adaptive Moments (Adam) and Root Mean Squared Propagation (RMSprop). Each optimizer features tunable parameters like learning rate, momentum, and decay.

   Adam and RMSprop are variations of SGD with adaptive learning rates. In the proposed classifier network, Adam is used since it is often used in literature.

2.3 Results on the Validation Set

After successful training, the model achieved a validation loss of 0.069 and a validation accuracy of 97.80%. The model was trained for 16 epochs, but at epoch 13 the lowest validation loss was observed, so we keep the weights of the model at that instance. In Figure 4 we can see the training and validation loss/accuracy of the model as a function of the epoch. In other words, we can see how the training progresses.



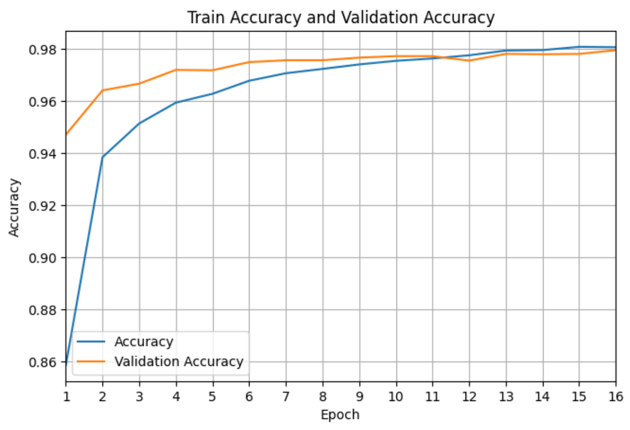


Figure 4: Training and Validation Loss (left) and Training and Validation Accuracy (right).

2.4 Grid Search

It was stated previously that parameters m and λ are set by ourselves. Their value affects the performance of the model on the reconstructed dataset. Therefore, to make the best possible reconstruction, we do grid search for m and λ. We choose the parameters that give us the lowest error on the validation set of the reconstructed images.

In practice, we set grid search parameters $m = [200, 300, 400, 500, 600]$ and $\lambda = [0.1, 0.2, 0.5, 1]$. Different from the way we split the original image set, here for the reconstructed image set, we consider it as a "test" set and see how well our compression works. As such, we split it into 80% validation data and 20% test data. We also make sure that the test data of the reconstructed set corresponds to the same digits of the test data of the original image set.

In this part we have measured several targets to evaluate the model:

- Mean Squared Error (MSE) of the reconstruction

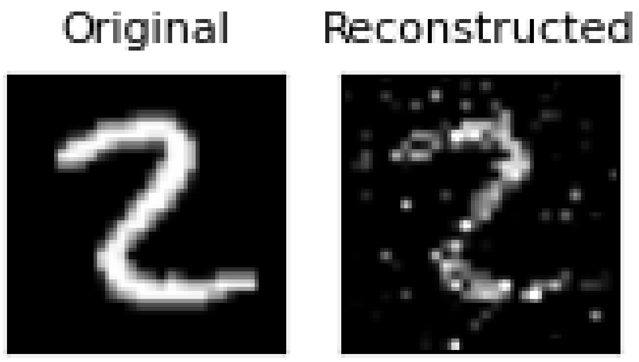


Figure 5: 76th image of both data sets.

- Validation Loss & Accuracy
- Visualization of same order image in both data set

An example of the visualization is shown in Figure 5. Table 1 shows the results for each combination of m, λ. We see that a higher value of m leads to a smaller loss and a higher accuracy. However, the higher the value of m the lighter the compression of the images. We also notice that when $\lambda = 0.1$, better results are achieved no matter the value of m.

As such, we consider m = 300 and $\lambda = 0.1$ as the optimal values. Compressing the images using these parameters, the model achieves a low validation loss 0.171 and a high validation accuracy (94.81%) on the reconstructed images. When compressed, the size of the data is reduced by $1 - (300/784) = 61.73\%$.

Parameters		**Results**		
m	**λ**	**Reconstruction MSE**	**Val loss on reconstructed**	**Val acc on reconstructed**
200	0.1	0.092	0.886	71.61%
200	0.2	0.093	1.015	67.91%
200	0.5	0.095	1.103	65.94%
200	1.0	0.100	1.427	53.35%
300	0.1	0.051	0.171	94.81%
300	0.2	0.061	0.267	92.50%
300	0.5	0.080	0.550	86.11%
300	1.0	0.097	1.574	51.61%
400	0.1	0.026	0.075	97.86%
400	0.2	0.035	0.122	96.68%
400	0.5	0.067	0.398	90.08%
400	1.0	0.096	1.305	65.63%
500	0.1	0.011	0.056	98.42%
500	0.2	0.024	0.081	97.95%
500	0.5	0.059	0.279	94.22%
500	1.0	0.097	1.404	62.94%
600	0.1	0.006	0.049	98.60%
600	0.2	0.018	0.065	98.32%
600	0.5	0.058	0.236	95.79%
600	1.0	0.098	1.369	66.36%

Table 1: Grid search for best values of m and λ

2.5 Comparing results on the test set

Loss And Accuracy

At this point we evaluate the performance (loss, accuracy) of our final model on

Performance on Test Set	**Loss**	**Accuracy**
Original Images	0.068	98.08%
Reconstructed Images	0.192	93.62%

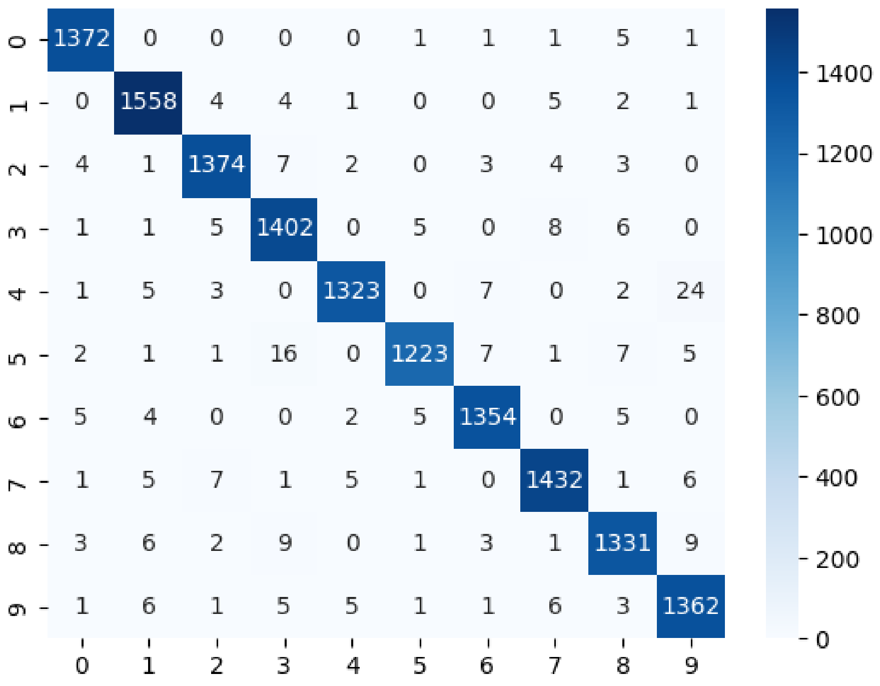
Table 2: Model performance on the test sets of the original and reconstructed images.

1. the test set of the original images
2. the test set of the reconstructed images

Both of these test sets correspond to the same digits and labels. The results are shown in Table 2. A relatively small difference is observed between the accuracy and loss of the two sets, thereby signifying that the images have been appropriately reconstructed from their compressed format.

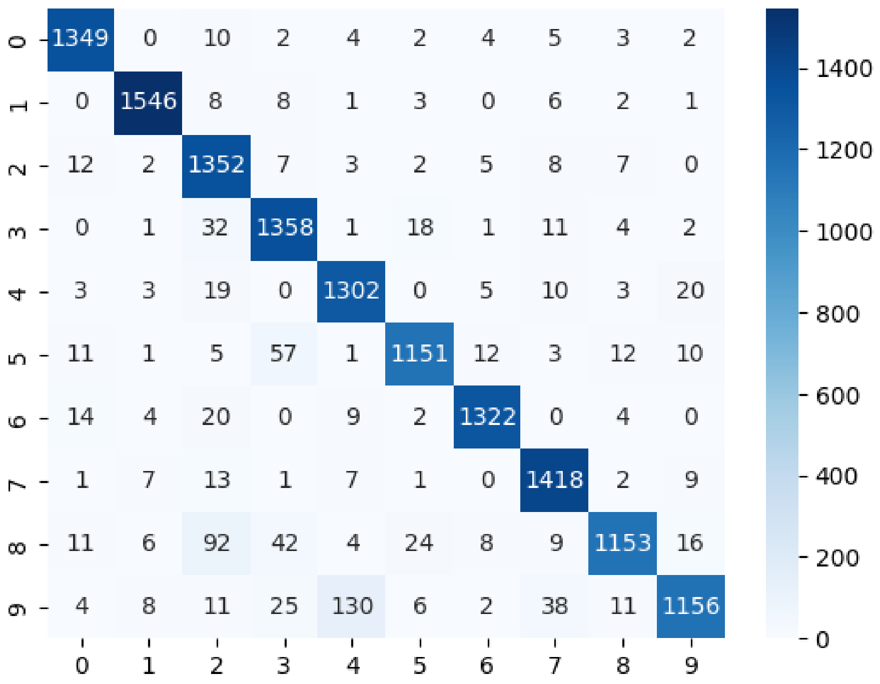
Confusion Matrix

A confusion matrix is a valuable tool in evaluating the performance of a multi-class classification model. It provides a detailed breakdown of how well the model has classified instances for each class. The confusion matrix for the test set of the original images is shown in Figure 6 and for the reconstructed images in Figure 7.



	0	1	2	3	4	5	6	7	8	9
0	1372	0	0	0	0	1	1	1	5	1
1	0	1558	4	4	1	0	0	5	2	1
2	4	1	1374	7	2	0	3	4	3	0
3	1	1	5	1402	0	5	0	8	6	0
4	1	5	3	0	1323	0	7	0	2	24
5	2	1	1	16	0	1223	7	1	7	5
6	5	4	0	0	2	5	1354	0	5	0
7	1	5	7	1	5	1	0	1432	1	6
8	3	6	2	9	0	1	3	1	1331	9
9	1	6	1	5	5	1	1	6	3	1362

Figure 6: Confusion Matrix of the original image test set.



	0	1	2	3	4	5	6	7	8	9
0	1349	0	10	2	4	2	4	5	3	2
1	0	1546	8	8	1	3	0	6	2	1
2	12	2	1352	7	3	2	5	8	7	0
3	0	1	32	1358	1	18	1	11	4	2
4	3	3	19	0	1302	0	5	10	3	20
5	11	1	5	57	1	1151	12	3	12	10
6	14	4	20	0	9	2	1322	0	4	0
7	1	7	13	1	7	1	0	1418	2	9
8	11	6	92	42	4	24	8	9	1153	16
9	4	8	11	25	130	6	2	38	11	1156

Figure 7: Confusion Matrix of the reconstructed image test set.

From these confusion matrices, we can conclude that it is our model performs greatly on both image sets because most of the values are distributed on the diagonal in the matrix (true positives). For the reconstructed image confusion matrix, elements $(8, 2)$ and $(9, 4)$ have a high misclassification rate, at 92 and 130 respectively. However, elements of the diagonal in the matrix differ by an order of magnitude compared to those not on the diagonal, so this is not a significant issue.

Conclusion

Using compressed sensing and the LASSO regression model, the optimal values for the two hyperparameters, m and λ, are found to be 300 and 0.1, respectively. This configuration results in a Mean Squared Error (MSE) of 0.052, signifying not only efficiency in compression and storage but also high accuracy. Additionally, under these settings, we constructed a 3-layer neural network classification model using Keras with a batch size of 128 and trained it using an early stopping mechanism. The model was initially trained on the original image data and subsequently employed to classify numbers in a compressed image dataset, achieving an accuracy of 93.62% and a validation loss of 0.068. In conclusion, we have successfully created an appropriate compressed image dataset, as well as an accurate neural network model for classification.