HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

FINAL REPORT

Handwritten digit recognition using ANN

Ngo Quang Minh
minh.mn205163@sis.hust.edu.vn

Pham Khanh Linh
linh.pk205186@sis.hust.edu.vn

Major: Global ICT

Supervisor: Dr. Nguyen Nhat Quang _____

Signature

School: School of Information and Communications Technology

HANOI, 06/2023

ACKNOWLEDGMENT

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project.

First and foremost, we would like to thank our project supervisor, Dr Nguyen Nhat Quang, for their invaluable guidance, support, and expertise throughout the project. Their extensive knowledge and guidance have played a crucial role in shaping our project and helping us overcome challenges. We would also like to acknowledge the efforts and contributions of our team members, Pham Khanh Linh and Ngo Quang Minh. Their dedication, hard work, and collaboration have been essential in carrying out the project tasks, conducting experiments, and analyzing the results. The synergy of our team has greatly enhanced the quality of our work.

ABSTRACT

Handwritten digit recognition is a prominent problem in the field of computer vision, with applications ranging from automated postal sorting to digitizing historical documents. This project focuses on developing an artificial neuron learning model using TensorFlow for accurate and robust digit recognition. The model is trained and evaluated on the MNIST dataset, which consists of 60,000 training images and 10,000 test images of handwritten digits. The neural network architecture includes layers such as flattening, dense layers with ReLU activation, dropout, and batch normalization. The Adam optimizer with a learning rate of 0.001 is employed to optimize the model's performance. The model is trained using the sparse categorical crossentropy loss function, which is suitable for multi-class classification tasks. The objective of this project is to achieve high accuracy in classifying handwritten digits, contributing to the advancement of optical character recognition systems. The report provides an overview of the project, including the model architecture, optimization techniques, evaluation metrics, and performance analysis.

Keyword: handwritten digit recognition, MNIST dataset, computer vision, artificial neuron network, HOG

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

1.1 Problem Statement

The problem of digit handwritten recognition has been a longstanding challenge in the field of computer vision and pattern recognition. Handwritten digits are inherently diverse and can vary in terms of writing styles, shapes, and sizes, making their accurate recognition a complex task. This problem finds applications in various domains, including postal services, check processing, and document digitization, where automated recognition of handwritten digits is crucial for efficient and reliable operations.

The main challenge in digit handwritten recognition arises from the inherent variability and ambiguity present in handwritten digits. Different individuals may have distinct writing styles, leading to variations in the appearance of the same digit. Additionally, variations in writing speed, stroke thickness, and pen pressure further contribute to the complexity of the problem. Furthermore, handwritten digits may suffer from issues such as overlapping, slanting, and noise, further complicating their recognition.

The objective of this project is to develop a robust and accurate digit handwritten recognition system that can effectively classify handwritten digits across a wide range of writing styles and variations. By addressing this problem, we aim to improve the efficiency of digit recognition tasks in various applications, enabling automation and reducing the reliance on manual efforts.

1.2 Organization of Report

In this report, we will divide it into seven chapters in which each chapter presents a specific field of the problem. This organization allows for a logical progression of ideas and facilitates a clear presentation of our methodology and findings.

Chapter 2: Dataset
In this chapter, we provide an overview of the dataset used in our project. We discuss the source of the dataset, its characteristics, and the preprocessing steps undertaken to prepare it for training our model. Additionally, we analyze the distribution of classes within the dataset and highlight any challenges or considerations related to data imbalance.

Chapter 3: HOG (Histogram of Oriented Gradients)
This chapter focuses on the HOG feature extraction technique, which is employed as a preprocessing step in our digit handwritten recognition system. We explain the

theoretical foundations of HOG and its relevance in capturing essential features from images. Furthermore, we describe the implementation details and parameter settings utilized for extracting HOG features from our dataset.

Chapter 4: Network Architecture

In this chapter, we delve into the network architecture employed for our digit handwritten recognition model. We discuss the rationale behind choosing a specific architecture and highlight the key components, such as flatten layer, dense layers, and dropout layers. Additionally, we elaborate on the activation functions and regularization techniques incorporated to enhance the model's performance.

Chapter 5: Numerical Results

This chapter presents the numerical results obtained from our digit handwritten recognition system. We discuss the evaluation metrics used to assess the model's performance, such as accuracy, precision, recall, and F1 score. We provide a comprehensive analysis of the results, including comparisons with baseline models and discussions on the model's strengths and limitations. Additionally, we present visualizations and examples to illustrate the model's ability to correctly classify handwritten digits.

Chapter 6: Conclusion

Chapter 7: Reference

CHAPTER 2. DATASET

2.1 MNIST Dataset

The MNIST (Modified National Institute of Standards and Technology) dataset is a popular benchmark dataset in the field of machine learning. The MNIST dataset consists of a collection of grayscale images of handwritten digits from 0 to 9. The images in MNIST are normalized and centered, with a fixed size of 28x28 pixels. The pixel values in the MNIST dataset range from 0 to 255, where 0 represents black and 255 represents white. By normalizing the pixel values to a range between 0 and 1, we can improve the training process and the convergence of the model.

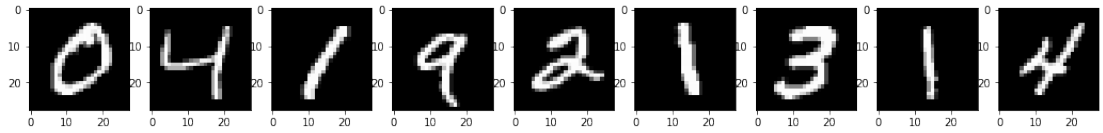


Figure 2.1: MNIST dataset

In total, the dataset contains 70,000 images which are divided into two main parts: a training set and a test set. The training set contains 60,000 images, while the test set contains 10,000 images. We take 10 percent of the images from the training set to construct the validation set which includes 6,000 images. Therefore, the original dataset will be divided to three parts:

- Training set: 54,000 examples (77,14%)
- Validation set: 6,000 examples (8,57%)
- Test set: 10,000 examples (14,29%)

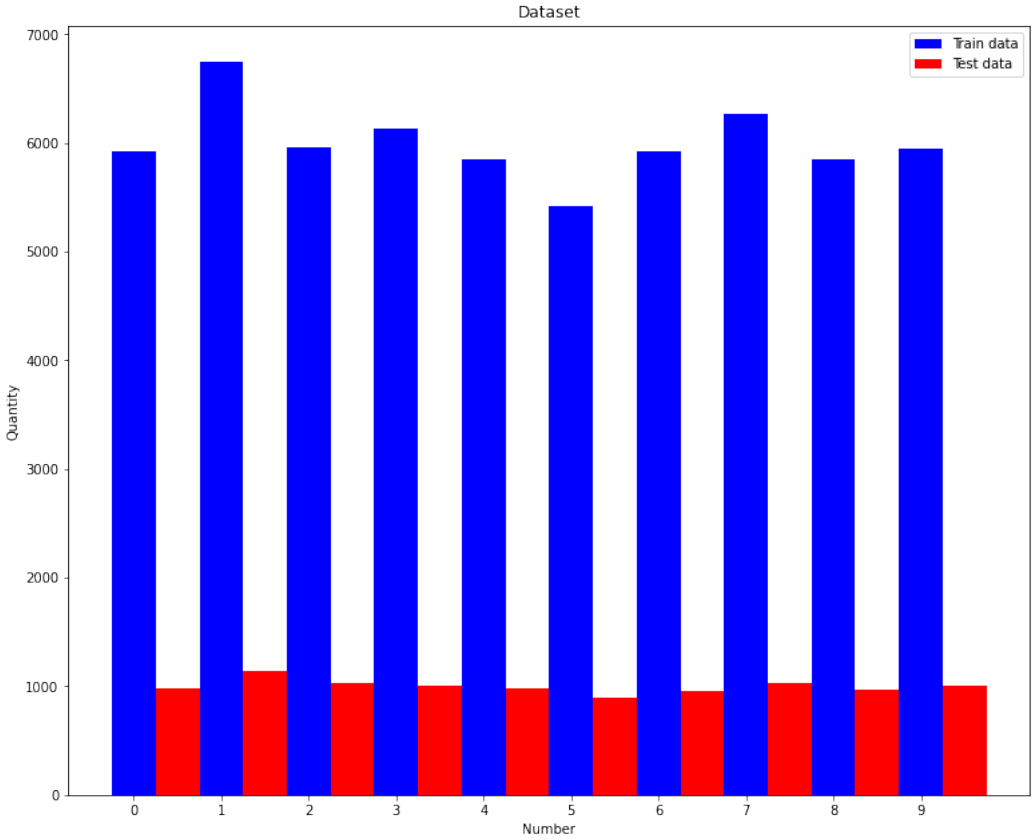


Figure 2.2: MNIST dataset distribution

The dataset is designed to have an equal distribution of examples across all classes, meaning that each digit class has approximately the same number of examples. This balance ensures that machine learning models trained on the MNIST dataset have an equal opportunity to learn and generalize across all digit classes.

2.2 Noise

Adding noise to the MNIST dataset can be a valuable technique to enhance the robustness and generalization of machine learning models trained on this dataset. By introducing random variations to the pixel values of the images, the model becomes more resilient to noise and better equipped to handle real-world scenarios.

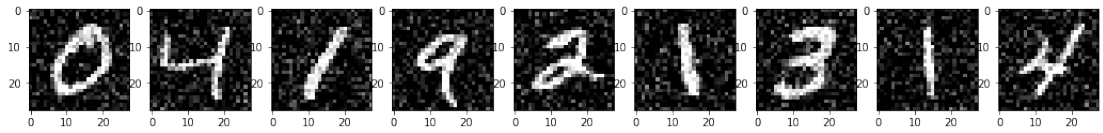


Figure 2.3: MNIST dataset with noise

For this project, our approach is to employ a Gaussian distribution to generate random noise values. By sampling from this distribution with a mean of zero and a standard deviation of 1, random values are obtained to add to the original pixel

values. We also introduces a noise factor to control the level of noise to the image while preserving the underlying structure and content.

CHAPTER 3. HISTOGRAM OF ORIENTED GRADIENTS

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection and recognition tasks. The technique counts occurrences of gradient orientation in localized portions of an image.

The image is divided into small connected regions called cells, and for the pixels within each cell, a histogram of gradient directions is compiled. The descriptor is the concatenation of these histograms. For improved accuracy, the local histograms can be contrast-normalized by calculating a measure of the intensity across a larger region of the image, called a block, and then using this value to normalize all cells within the block. This normalization results in better invariance to changes in illumination and shadowing. For illustration, we use $8x8$ pixels per cell and $2x2$ cells per block with 9 orientation bins for $128x64$ pixel image.

3.1 Gradient computation

The gradient is obtained by combining magnitude and angle from the image. First Gx and Gy is calculated for each pixel. First Gx and Gy is calculated using the formulae below for each pixel value where r, c refer to rows and columns respectively.

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \quad G_y(r,c) = I(r-1,c) - I(r+1,c) \tag{3.1}$$

This process will give us two new matrices – one storing gradients in the x-direction and the other storing gradients in the y direction. This is similar to using a *Sobel Kernel* of size 1. The magnitude would be higher when there is a sharp change in intensity, such as around the edges.

After calculating Gx and Gy, magnitude and angle of each pixel is calculated using the formulae mentioned below.

$$\text{Magnitude}(\mu) = \sqrt{G_x^2 + G_y^2} \quad Angle(\theta) = \left|\tan^{-1}\left(G_y/G_x\right)\right| \tag{3.2}$$

3.2 Orientation binning

After obtaining the gradient of each pixel, the gradient matrices (magnitude and angle matrix) are divided into $3x3$ pixels groups to form a cell. For each cell, a 9-point histogram is calculated. The histogram channels are evenly spread over 0 to 180 degrees. As a result, a 9-point histogram develops a histogram with 9 bins

and each bin has an angle range of 20 degrees.

Value									
Bins	0	20	40	60	80	100	120	140	160

Table 3.1: Representation of a 9 bin histogram

A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude.

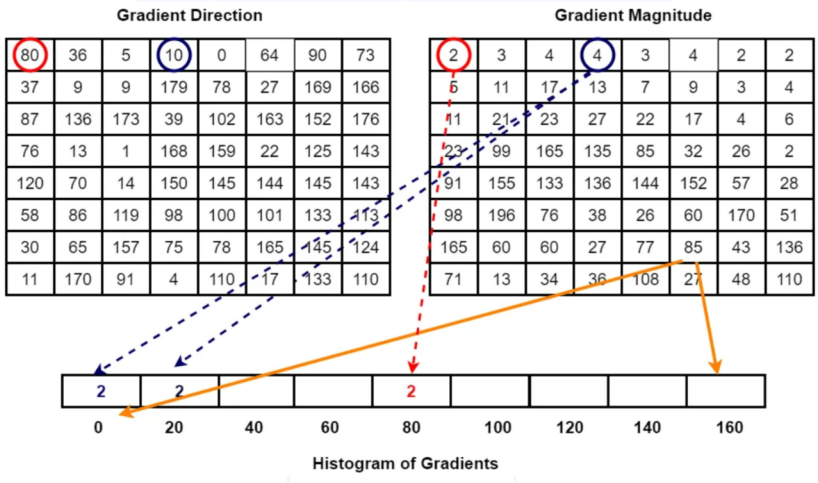


Figure 3.1: Illustration for calculation of 9 bin histograms

If the magnitude is between twos bins, we can calculate the value for the bins by the formulae below with y is the gradient magnitude, $x0$ is the value for bin $l-1$, $x1$ is the value for bin l and x is the considered gradient direction.

$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y \quad x_l = \frac{(x - x_0)}{x_1 - x_0} * y \tag{3.3}$$

3.3 Descriptor blocks

To account for changes in illumination and contrast, the gradient strengths must be locally normalized, which requires grouping the cells together into larger, spatially connected blocks. The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions.

Once histogram computation is over for all cells, 4 cells from the 9 point histogram matrix are grouped together to form a block of size $2x2$. This clubbing is done in an overlapping manner with a stride of 8 pixels. For all 4 cells in a block, we concatenate all the 9 point histograms for each constituent cell to form a 36 feature vector.

$$f_{bi} = [b_1, b_2, b_3, \ldots, b_{36}] \tag{3.4}$$

To normalize, the value of k is first calculated by the following formulae :

$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + \ldots\ldots + b_{36}^2}$$

(3.5)

$$f_{bi} = \left[\left(\frac{b_1}{k}\right), \left(\frac{b_2}{k}\right), \left(\frac{b_3}{k}\right), \ldots., \left(\frac{b_{36}}{k}\right)\right] \tag{3.6}$$

3.4 Block normalization

Block normalization is an important step in the Histogram of Oriented Gradients (HOG) algorithm. By performing block normalization, the HOG algorithm accounts for local variations and contrast differences within an image. This normalization helps ensure that the extracted HOG features are more robust and less sensitive to changes in lighting conditions or local contrast variations, leading to improved performance in object detection and recognition tasks. There are many normalization method, in this project we use the L2-norm. This technique normalizes the histograms within each block based on their overall magnitude, ensuring that the resulting histogram vector has a unit Euclidean length.

$$f_{bi} = \frac{f_{bi}}{\sqrt{\|f_{bi}\|^2 + \epsilon^2}} \tag{3.7}$$

Where ϵ is a small value added to the square of $f_b$in order to avoid zero division error.

3.5 Experiment

In our project, we have utilized the scikit-image library's feature extraction for the Histogram of Oriented Gradients (HOG) algorithm. The library allow us to customise the number of orientation bins, the number of pixels per cell, number of cells per block. The table presented below shows some of the results of our experiments conducted with various parameters in our project:

Orientation	Pixels per cell	Cells per block	Validation Accuracy
9	4x4	3x3	0.9482
9	4x4	2x2	0.9339
9	3x3	3x3	0.9677
8	3x3	3x3	0.9669
6	3x3	3x3	0.9786
4	3x3	3x3	**0.9838**
2	3x3	3x3	0.9573

Table 3.2: HOG Result

After careful considering, we deiced to take 4 orientaion bins, $3x3$ pixels per cell and $3x3$ cells per block. Therefore, each image of $28x28$ pixel will be extracted to 1764 features vector.

CHAPTER 4. NETWORK ARCHITECTURE

4.1 Layers

In our research, we conducted an exploration of various network architectures for Artificial Neural Networks. The accuracy of the validation set served as a reliable benchmark to gauge the effectiveness and suitability of each architecture. The tables below show the number of node in hidden layers and the accuracy on HOG model and Pixel model.

# Nodes in 1st Layer	128	256	512	1024	1024
# Nodes in 2nd Layer	128	128	256	512	1024
Number of parameters	243,722	486,026	1,037,578	2,337,290	2,867,210
Accuracy	0.9776	0.981	0.9813	**0.9838**	0.9827

Table 4.1: ANN Architecture for HOG model

# Nodes in 1st Layer	128	256	512	1024	1024
# Nodes in 2nd Layer	128	128	256	512	1024
Number of parameters	136,074	235,146	535,818	1,333,770	1,863,690
Accuracy	0.9652	0.9699	0.9742	**0.9761**	0.9743

Table 4.2: ANN Architecture for Pixel model

4.1.1 Input layer

The Flatten layer is typically placed at the beginning of the model, immediately following the input layer. Its primary purpose is to transform the 2D image data into a 1D array, which allows the network to process the individual pixel values as separate input features.

In our Pixel model, we use Flatten Layer as the input layer of the model which flattens the input images, which are 2D arrays of shape (28, 28), into a 1D array of shape (784,). It essentially reshapes the input data from a 2D grid to a 1D vector, allowing it to be fed into the subsequent dense layers. For the HOG model, the input shape will be the shape of input feature vector.

4.1.2 Dense Layer

Dense layers are defined using the ***'tf.keras.layers.Dense'*** class. Each dense layer consists of a set of neurons that are fully connected to the neurons in the previous layer. The output of a dense layer is determined by the weighted sum of the inputs passed through an activation function. The ReLU (Rectified Linear Unit)

activation function is applied to the output of this layer, which introduces non-linearity to the network. ReLU function returns 0 for negative inputs and the input value for positive inputs, effectively removing any negative values and keeping the positive values intact.

4.1.3 Dropout Layer

The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons.

Dropout layers are important in training model because they prevent overfitting on the training process. If they are not present, the first batch of training samples influences the learning in a disproportionately high manner. This, in turn, would prevent the learning of features that appear only in later samples or batches:

[H]

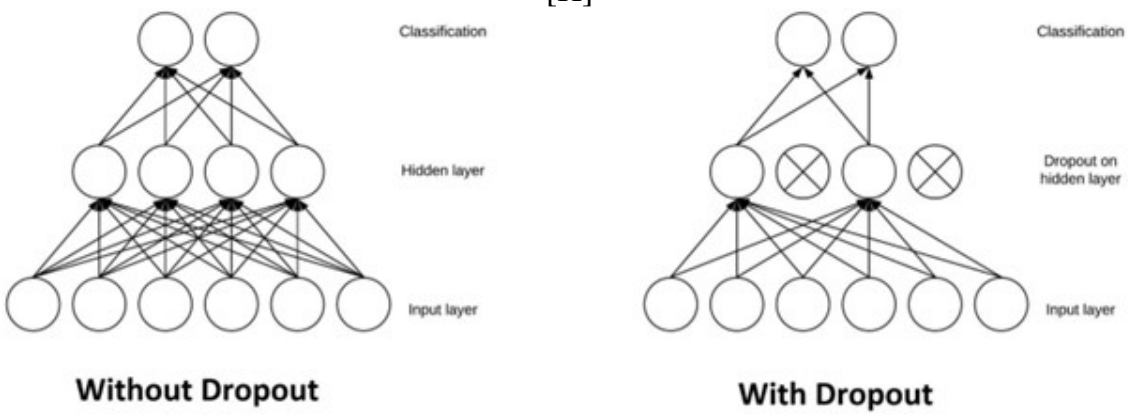


Figure 4.1: ANN with & without dropout

4.1.4 Batch Normalization

Batch normalization is a technique used to improve the training process and the overall performance of the model. It normalizes the inputs of a layer by subtracting the batch mean and dividing by the batch standard deviation, effectively normalizing the distribution of the layer's inputs.

The purpose of batch normalization is to address the internal covariate shift, which is the change in the distribution of the input values to each layer during training. We apply batch normalization before the activation function which allows the activation function to operate on the normalized inputs, which can have a positive impact on the model's performance.

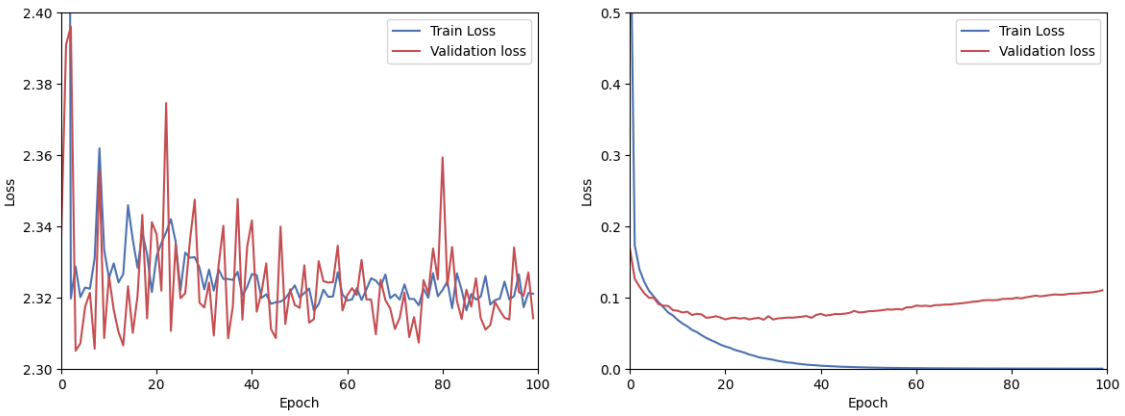
4.1.5 Output layer

This is the output layer of the model with 10 units, corresponding to the 10 possible classes (digits 0-9) in the MNIST dataset. The softmax activation function is applied to the output of this layer, which converts the raw predictions into probabilities. The values of the units in this layer represent the probabilities of the input image belonging to each class. The class with the highest probability is considered as the predicted class.

4.2 Optimizer

The Adam optimizer is a popular choice for training neural networks, known for its efficiency and effectiveness. It combines the advantages of two other optimization methods, Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), to provide robust and adaptive learning rate adjustments. The learning rate determines the step size taken during weight updates, affecting the convergence speed and final performance of the model.

Choosing an appropriate learning rate is crucial, as a value that is too high may lead to overshooting the optimal weights, while a value that is too low may result in slow convergence. It often requires experimentation and fine-tuning to find an optimal learning rate for a given task and model architecture.



(a) Learning rate of 1

(b) Learning rate of 0.001

Figure 4.2: Loss for different learning rate

The learning rate determines the step size taken during weight updates, affecting the convergence speed and final performance of the model. In our context, we apply the Adam optimizer with a learning rate of 0.001 suggests an initial choice for a moderately small learning rate, aiming for a balance between convergence speed and precision.

4.3 Loss Function

In the provided model, the loss function used is Sparse Categorical Crossentropy. The choice of an appropriate loss function is crucial for training a neural network as it determines how the model's performance is evaluated and optimized.

Sparse Categorical Crossentropy is commonly used in multi-class classification tasks where each input sample belongs to one specific class out of multiple classes. It is suitable for scenarios where the classes are mutually exclusive, meaning that each sample can only be assigned to a single class label.

The sparse categorical cross-entropy loss function compares the predicted probabilities (computed from logits) with the true target labels. It measures the dissimilarity between the predicted probabilities and the true labels, providing a measure of how well the model is performing in terms of classifying the input data.

$$\text{Sparse Categorical Crossentropy} = -\sum_{i=1}^{C} y_i \log(p_i)$$

where:

C represents the number of classes,

y_i is the true probability (0 or 1) for class i,

p_i is the predicted probability for class i.(4.1)

4.4 Activation Function

We also experiment the influence of various activation functions on the hidden layers of our neural network. Experimentally, it has been determined that the rectified linear unit (ReLU) activation function shows the best performance for our specific problem domain.

Activation function	Pixel model	HOG model
Relu	**0.9685**	**0.9842**
Sigmoid	0.9680	0.9810
Tanh	0.9680	0.9828
LeakyRelu	0.9681	0.9817

Table 4.3: Activation functions

4.4.1 Relu

Definition: The rectified linear activation function or ReLu is a non-linear function or piecewise linear function that will output the input directly if it is positive, otherwise, it will output of zero. It is the most commonly used activation function in Neuron Networks especially in Multilayer perceptrons.

Mathematically, it is expressed as:

$$f(x) = \max(0, x)$$

where x is the input to the function and $f(x)$ represents the output.

Advantages of ReLU

1. Non-linearity: ReLU introduces non-linearity to the network, allowing it to model complex relationships between input features. This non-linear activation helps the network learn and represent more intricate patterns in the data.
2. Sparsity: ReLU activation leads to sparsity in the network as it sets negative values to zero. This sparsity can help in reducing the overall computational complexity of the network and improve its efficiency.
3. Avoiding vanishing gradient: Unlike some other activation functions, ReLU does not suffer from the vanishing gradient problem. The gradient remains constant for positive values, allowing for efficient backpropagation during training and enabling the network to learn effectively.
4. Faster convergence: ReLU has been observed to promote faster convergence during the training process. The linear nature of ReLU for positive inputs allows the gradients to flow more freely, leading to quicker learning and faster convergence towards optimal solutions.

4.4.2 Softmax Activation Function

In the field of machine learning and neural networks, the choice of activation function is crucial for accurately representing and interpreting the output of a network. One commonly used activation function for multi-class classification problems is the softmax function. The softmax activation function is particularly well-suited for problems where the output needs to represent a probability distribution over multiple classes.

Softmax Activation Function

The softmax activation function takes a vector of real numbers as input and

transforms it into a probability distribution over the classes. Mathematically, the softmax function can be defined as follows:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}$$

where x_i is the input to the function, C is the total number of classes, and e represents the base of the natural logarithm.

Interpretation and Properties

The softmax function converts the input values into probabilities by exponentiating them and normalizing the result. The output of the softmax function can be interpreted as the likelihood or confidence score assigned to each class. The probabilities sum up to 1, ensuring that the output represents a valid probability distribution.

Advantages of Softmax

1. Probability Interpretation: The softmax function produces outputs that can be directly interpreted as class probabilities. This makes it suitable for multi-class classification tasks, where the goal is to assign a probability to each class.
2. Output Normalization: The softmax function normalizes the output probabilities, ensuring that they sum up to 1. This normalization property enables direct comparison and ranking of the predicted probabilities across different classes.
3. Gradient Properties: The softmax function has a well-defined and computationally efficient gradient, which is crucial for the training process. The gradient can be efficiently computed during backpropagation, enabling effective optimization of the network parameters.

Considerations

While the softmax activation function has several advantages, it is important to consider certain aspects when using it:

1. One-Hot Encoding: The softmax function requires the ground truth labels to be one-hot encoded, meaning that each class is represented by a binary vector where only one element is 1 and the rest are 0. This encoding scheme is necessary for the softmax function to compute accurate probabilities.
2. Sensitivity to Outliers: The softmax function can be sensitive to outliers or extreme input values. Large input values can result in very small or very large probabilities, which may affect the overall performance of the model. It is important to preprocess and normalize the input data appropriately to mitigate

this issue.

We conduct the experiment on the noisy dataset with noise factor of 0.2, batch size of 100 with 25 epochs and HOG model (the original pixel model show the same pattern).

5.1 Overfitting

Despite the model have given good result, the model we have trained fits too well to the training set. Our model has almost zero loss in the training set but a higher loss in the Validation set (0.1144). It then becomes difficult for the model to generalize to new examples that were not in the training set.

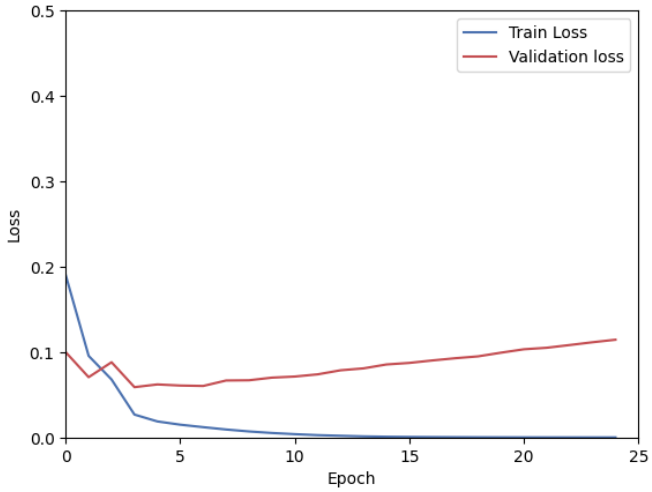
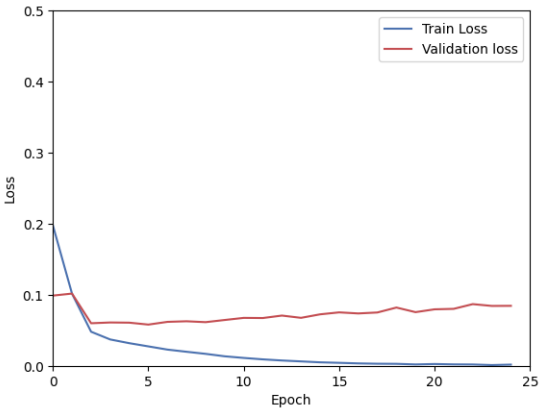


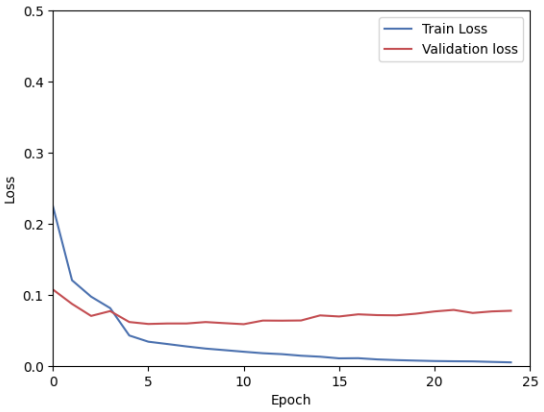
Figure 5.1: Overfit

5.1.1 Dropout

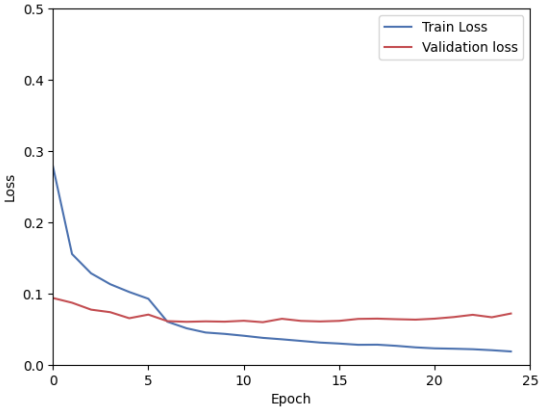
To address this problem, we have used Dropout layers. By ignoring a random subset of units in a layer while setting their weights to zero during that phase of training, dropout layers prevents the network from relying too heavily on specific neurons and encourages the learning of more robust and independent features and avoid overfitting. We have set the Dropout rate from 0.1 to 0.9.



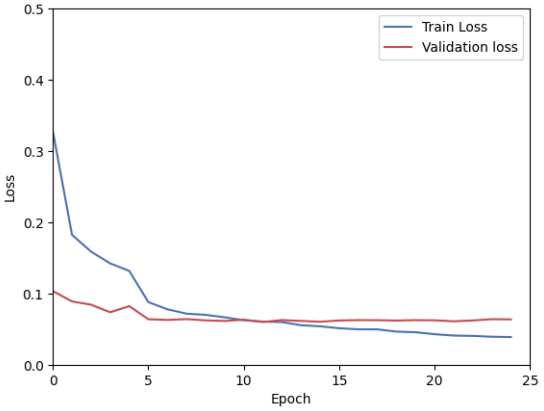
(a) Drop rate 0.1



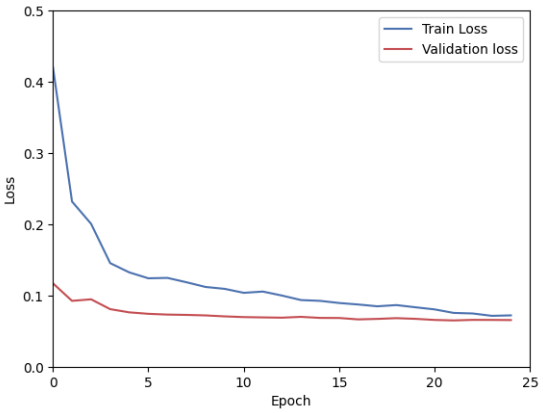
(b) Drop rate 0.3



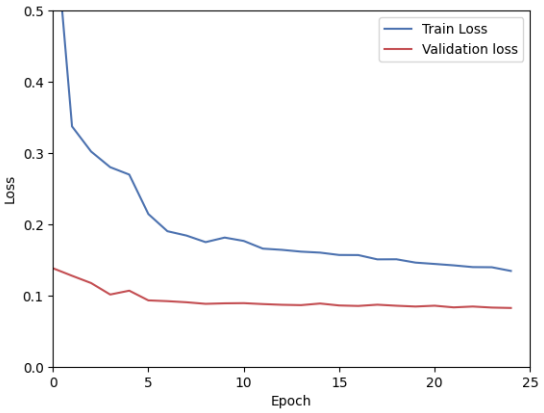
(c) Drop rate 0.5



(d) Drop rate 0.6



(e) Drop rate 0.7



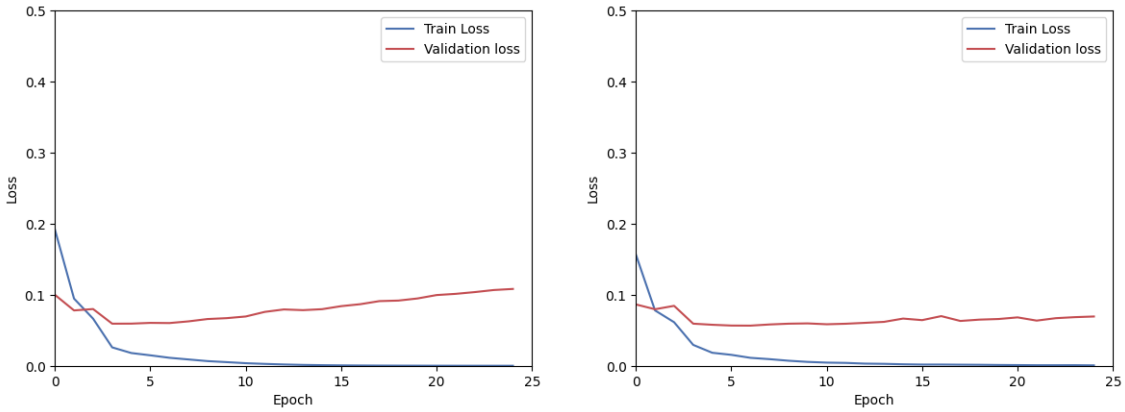
(f) Drop rate 0.8

Figure 5.2: Dropout on overfitting

As we can see from the figures above, incearsing the Dropout rate will help to reduce the overfitting. However, if the drop rate is too high, it will also increase the Loss and Validation Loss which we do not want. Because of that, we have to find the model which eliminate the overfitting and still have acceptable Loss. After consider carrefully, we have choose the Dropout rate from 0.5 to 0.7 with the Loss and Validation Loss rate under 0.07.

5.1.2 Batch Normalization

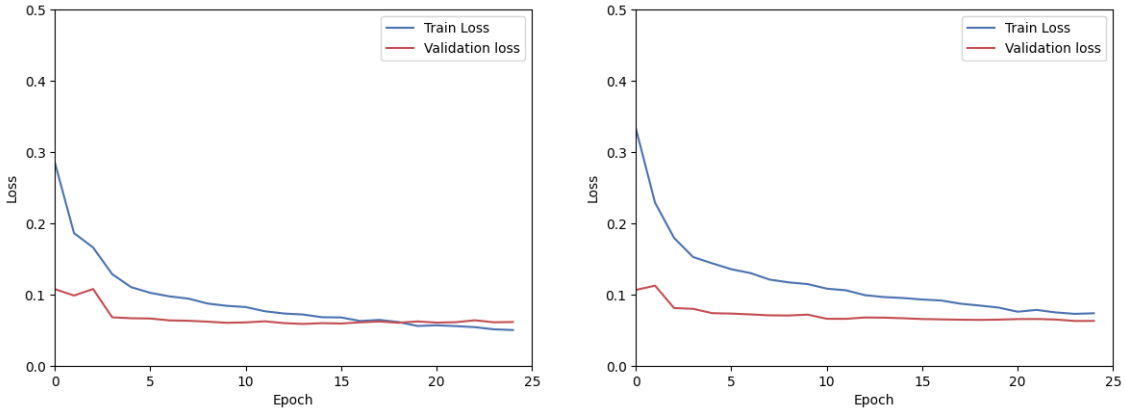
Batch normalization is also applied to our model to avoid overfitting.



(a) Without Batch Normalization (b) With Batch Normalization

Figure 5.3: Batch Normalization on overfitting

The figure demonstrate the validation loss with and without Batch Normalization, the better result belongs to the model with Batch Normalization.



(a) Batch Normalization and Drop rate 0.5 (b) Batch Normalization and Drop rate 0.6

Figure 5.4: Batch Normalization and Dropout on overfitting

We also try to apply both Batch Normalization and Dropout and found that drop rate 0.5 give best result with validation loss of 0.0613 and accuracy of 0.9837.

5.1.3 Early Stopping

On the other hand, we also use the Early Stopping method to reduce overfitting. Early stopping is a method that allows you to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset, in our model the stopping condition is the Validation

loss is no longer decreasing. Result in the validation loss of 0.06 and accuracy of 0.9862.

5.2 Final Result

In our experimental evaluation, we systematically varied the noise factor applied to the models, exploring its impact on the accuracy. Except for the input layer, all components of the model remained unchanged for both HOG model and Pixel model.

Noise factor	0	0.1	0.2	0.3	0.4
HOG	0.9913	0.9881	0.9862	0.9798	0.9552
Pixel	0.9861	0.9841	0.9774	0.9652	0.9507

Table 5.1: HOG model and Pixel model accuracy

The HOG model have better performance compare to Pixel model in every level of noise.

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.97	0.98	0.97	1010
4	0.97	0.97	0.97	982
5	0.99	0.97	0.98	892
6	0.98	0.98	0.98	958
7	0.97	0.97	0.97	1028
8	0.97	0.97	0.97	974
9	0.97	0.96	0.96	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

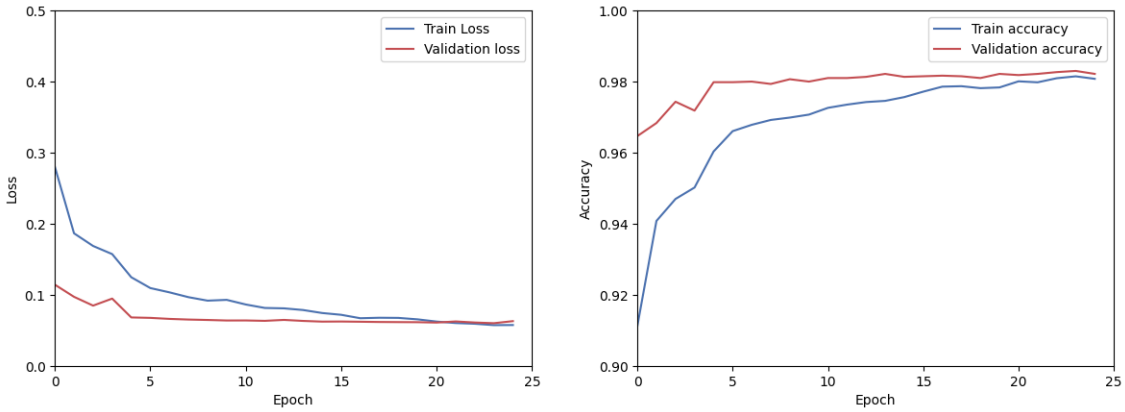
(a) Pixel model score

	precision	recall	f1-score	support
0	0.99	1.00	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.99	0.99	1032
3	0.99	0.99	0.99	1010
4	0.99	0.99	0.99	982
5	0.98	0.99	0.99	892
6	0.99	0.99	0.99	958
7	0.98	0.99	0.98	1028
8	0.98	0.98	0.98	974
9	0.99	0.97	0.98	1009
accuracy			0.99	10000
macro avg	0.99	0.99	0.99	10000
weighted avg	0.99	0.99	0.99	10000

(b) HOG model score

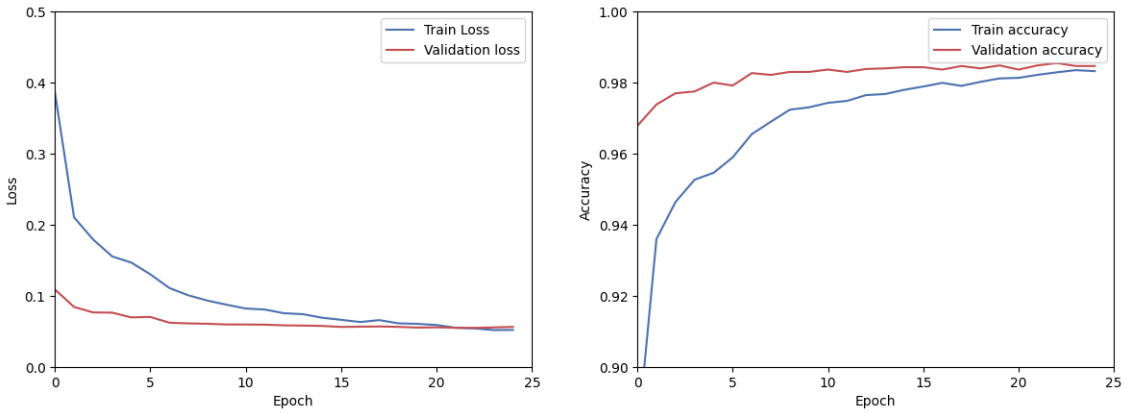
Figure 5.5: Evaluation on HOG and Pixel Model

The precision, accuracy, recall and f-1 score are high for both models (roughly over 98 percent) for all classes. However, the HOG model still have better overall perform compare to Pixel model.



(a) Loss (b) Accuracy

Figure 5.6: Loss and Accuracy for final HOG model noise factor of 0.2



(a) Loss (b) Accuracy

Figure 5.7: Loss and Accuracy for final Pixel model with noise factor of 0.2

Both models has effectively learned the underlying patterns in the training data, leading to stable and accurate predictions. As training progresses, the loss on train set and validation set decrease gradually without significant fluctuations.

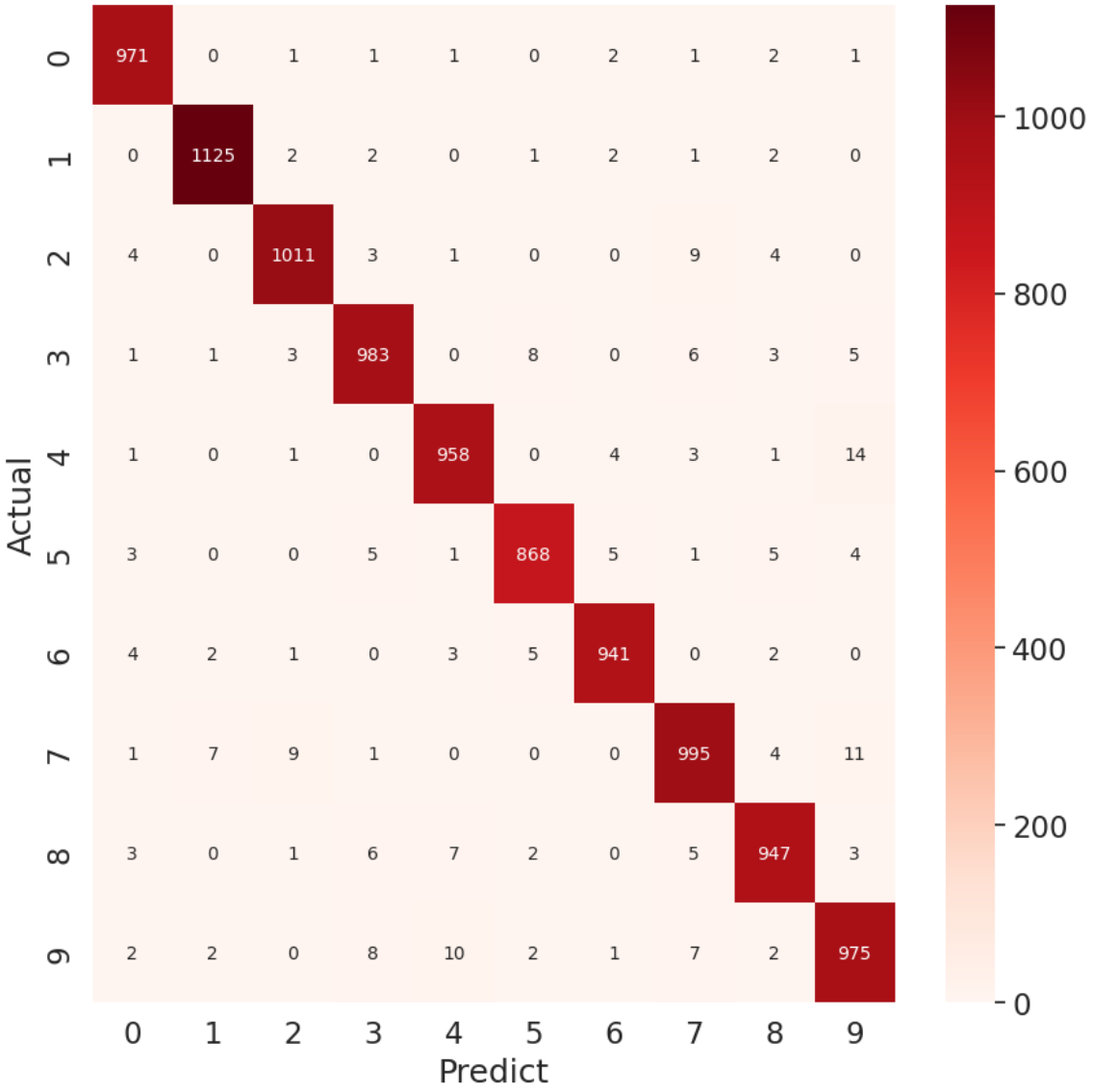


Figure 5.8: Confusion Matrix for Pixel model with noise factor of 0.2

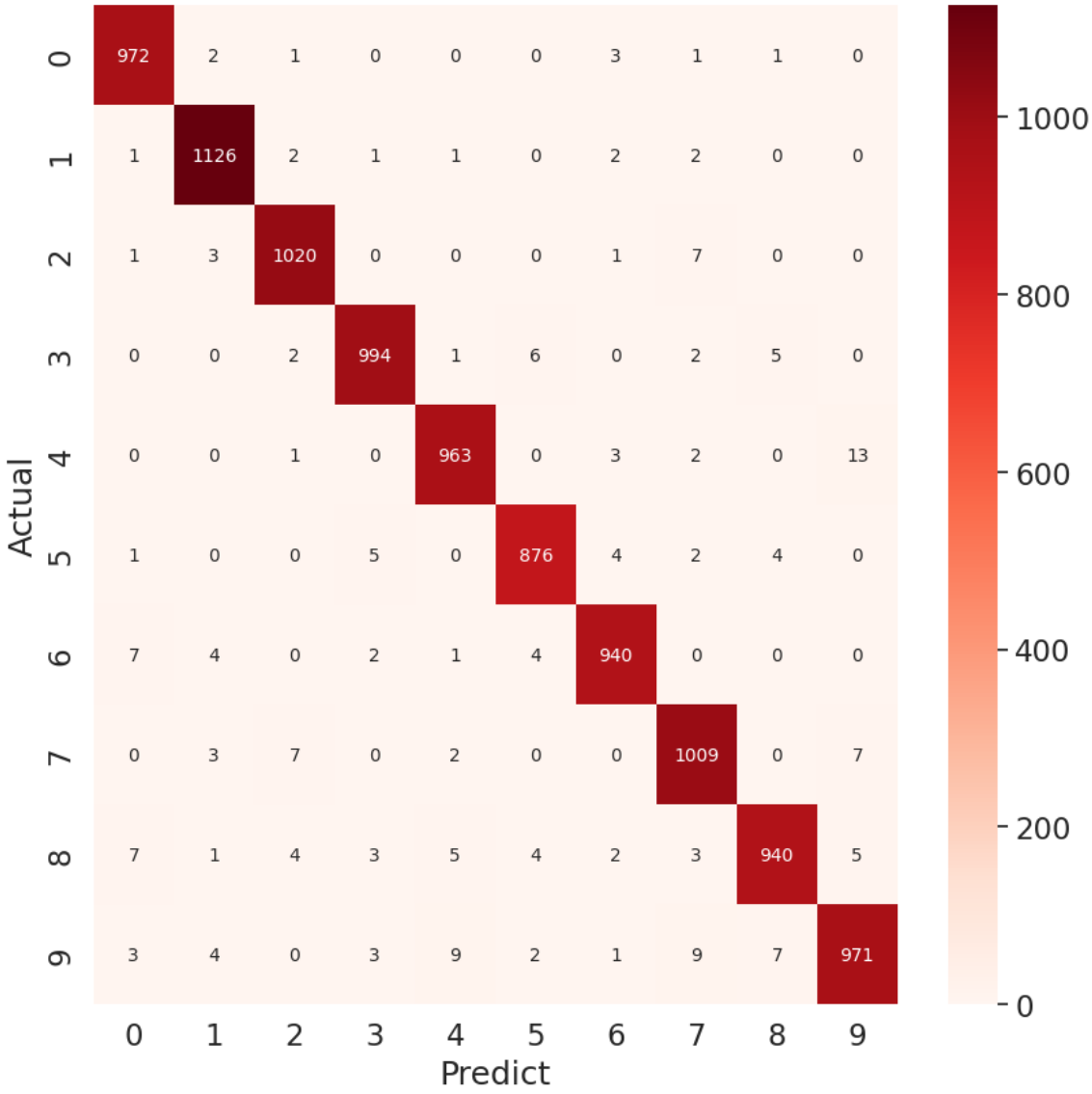


Figure 5.9: Confusion Matrix for HOG model with noise factor of 0.2

CHAPTER 6. CONCLUSIONS

6.1 Summary

In conclusion, this project focused on the problem of digit handwritten recognition and aimed to develop a robust and accurate system for classifying handwritten digits. Throughout the course of this project, we explored various techniques and methodologies, including dataset preparation, feature extraction using Histogram of Oriented Gradients (HOG), network architecture design, and numerical result analysis.

In summary, this project made significant progress in addressing the problem of digit handwritten recognition. We developed a robust and accurate system by leveraging techniques such as dataset preparation, feature extraction using HOG, and network architecture design. Our system achieved commendable accuracy in recognizing handwritten digits and demonstrated its potential for real-world applications.

The outcomes of this project contribute to the advancement of digit recognition technology and have implications in various domains, including postal services, document digitization, and check processing. The developed system can automate the process of digit recognition, improve operational efficiency, and reduce errors associated with manual efforts.

6.2 Suggestion for Future Works

Expanding the scope of our project holds promising prospects in recognizing a wider range of characters. By incorporating additional language scripts like Chinese, French and English our model would exhibit enhanced versatility in handwritten recognition. By embracing the challenge of recognizing multiple languages and diverse datasets, our project propels us towards more comprehensive and robust handwritten recognition systems

CHAPTER 7. REFERENCE

[1] Navneet Dalal and Bill Triggs, "Histograms of Oriented Gradients for Human Detection" https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

[2] Diederik P. Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization" https://arxiv.org/abs/1412.6980

[3] "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" https://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

[4] "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" https://arxiv.org/abs/1502.03167