Multi-Label Classification of Handwritten Digits

Brandon Nguyen

December 4, 2025

Abstract

The problem of classification in data can be broken into several subcategories. The most commonly studied problems in introductory machine learning courses are the binary and multi-class classification problems which involve dividing data points into two or more categories of data, or classes. However, the lesser studied subcategory within typical machine learning classes is the multi-label classification problem which involves the division of data points into classes like the aforementioned subcategories of classification, but also introduces the possibility that a data point can classified as a member of multiple classes simultaneously and not exclusively one. This project will explore several convolutional neural network architectures and how their structural differences impact performance on the task of multi-label classification.

1 Introduction

The MNIST dataset is a well, widely known dataset that consists of various handwritten digits from 0 to 9. One could consider the MNIST dataset as a benchmark dataset for those first learning how to work with fully-connected neural networks and convolutional neural networks. With that in mind, it is typically used in the task of multi-class classification. However, the MNIST dataset can be expanded for use the multi-label classification problem, where one simply concatenates several digits together to develop a multi-label classification problem where several digits can exist within one image. The goal of this project will be to correctly classify what digits are present within a given image from the DoubleMNIST dataset.

2 Method

This project takes on three different architectures for completing the task of multi-label classification. The first is a mini-AlexNet architecture, the second is a ResNet-like architecture, and the final is a fully convolutional neural network. The dataset that will be used in experimentation is the DoubleMNIST dataset which includes two digits per image and this factor highly influenced the decision to create two classification heads in each architecture.

2.1 Training Procedure

Training was conducted over a defined number of epochs using the standard supervised learning setup. The Adam optimizer was selected for the experiments because of its stability and fast convergence, with learning rate being the primary hyperparameter adjusted during testing. Loss and accuracy were recorded at each epoch for both training and testing sets to enable detailed performance tracking. In some experiments, the learning rate and batch size were systematically varied to observe their impact on convergence behavior. Learning rate scheduling was implemented to aid the models in converging to a more optimal minima through the PyTorch ReduceLROnPlateau module which was configured to reduce learning rate by a factor of 0.1 if validation loss did not increase after a gap of 1 epoch.

2.2 AlexNet-like Architecture

The first architecture to be discussed here is heavily inspired by the AlexNet architecture but is scaled down for the purposes of this project since images are merely 64x64, whereas the AlexNet architecture

was designed originally with a 227x227 image in mind.

Following the feature learning layers in the convolutional aspect of the network, the fully connected layers function to classify the digits based on the learned features of the network. To introduce multi-label capability to the network, two classification heads, one for each digit, are used in the end of the network to determine the ordered digits present in the image.
The AlexNet-like architecture consists of 5 convolutional layers followed by two fully connected layers that feed into two classification heads. The architecture is structured as follows:

- **Convolutional Layer 1**
  - Convolution with 1 input channel, 32 output channels, 5x5 kernel, and stride of 1
  - ReLU activation function
  - Max Pooling with 2x2 kernel and stride of 2
- **Convolutional Layer 2**
  - Convolution with 32 input channels, 64 output channels, 5x5 kernel, and stride of 1
  - ReLU activation function
  - Max Pooling with 2x2 kernel and stride of 2
- **Convolutional Layer 3**
  - Convolution with 64 input channels, 128 output channels, 3x3 kernel, and stride of 1
  - ReLU activation function
- **Convolutional Layer 4**
  - Convolution with 128 input channels, 128 output channels, 3x3 kernel, and stride of 1
  - ReLU activation function
- **Convolutional Layer 5**
  - Convolution with 128 input channels, 256 output channels, 3x3 kernel, and stride of 1
  - ReLU activation function
  - Max Pooling with 2x2 kernel and stride of 2
- **Fully Connected Layers**
  - Tensor is flattened
  - Dense Layer with 2,304 input units and 1024 output units
  - ReLU activation function
  - Dropout with rate of 50%
  - Dense Layer with 1,024 input units and 512 output units
  - ReLU activation function
  - Dropout with rate of 50%
- **Classification Heads**
  - **Digit 1 Head:** Dense Layer with 512 input units and 10 output units
  - **Digit 2 Head:** Dense Layer with 512 input units and 10 output units

2.3 ResNet-like Architecture

The second architecture to be implemented is derived from the ResNet18 architecture with some adjustments made to better suit the DoubleMNIST dataset. ResNet is a deep learning architecture that attempts to mitigate the vanishing gradient problem by reintroducing the "residual", or a layer output as part of the current layer output for forward propagation. The architecture consists of a single convolution layer followed by five residual block layers, an average pooling layer, and two fully connected layer classification heads. The code structure was derived from [1]. The architecture is structured as follows:

- **Convolutional Layer 1**
  - Convolution with 1 input channel, 64 output channels, 3x3 kernel, and stride of 1
  - Batch Normalization with 64 features
  - ReLU activation function
- **Residual Layer 1**
  - Block 1
    - Convolution with 64 input channels, 64 output channels, 3x3 kernel, stride of 1, and padding of 1.
    - Batch Normalization with 64 features
    - ReLU activation function
    - Convolution with 64 input channels, 64 output channels, 3x3 kernel, stride of 1, and padding of 1.
    - Batch Normalization with 64 features
    - Shortcut layer - adds residual from input of layer to current output tensor.
    - ReLU activation
  - Block 2
    - Convolution with 64 input channels, 64 output channels, 3x3 kernel, stride of 1, and padding of 1.
    - Batch Normalization with 64 features
    - ReLU activation function
    - Convolution with 64 input channels, 64 output channels, 3x3 kernel, stride of 1, and padding of 1.
    - Batch Normalization with 64 features
    - Shortcut layer - adds residual from input of block to current output tensor.
    - ReLU activation
- **Residual Layer 2**
  - Block 1
    - Convolution with 64 input channels, 128 output channels, 3x3 kernel, stride of 2, and padding of 1.
    - Batch Normalization with 128 features
    - ReLU activation function
    - Convolution with 128 input channels, 128 output channels, 3x3 kernel, stride of 1, and padding of 1.
    - Batch Normalization with 128 features
    - Shortcut layer - adds residual from input of layer to current output tensor.
    - ReLU activation
  - Block 2

        * Convolution with 128 input channels, 128 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 128 features
        * ReLU activation function
        * Convolution with 128 input channels, 128 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 128 features
        * Shortcut layer - adds residual from input of block to current output tensor.
        * ReLU activation

- **Residual Layer 3**
    - Block 1
        * Convolution with 128 input channels, 256 output channels, 3x3 kernel, stride of 2, and padding of 1.
        * Batch Normalization with 256 features
        * ReLU activation function
        * Convolution with 256 input channels, 256 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 256 features
        * Shortcut layer - adds residual from input of layer to current output tensor.
        * ReLU activation
    - Block 2
        * Convolution with 256 input channels, 256 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 128 features
        * ReLU activation function
        * Convolution with 256 input channels, 256 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 256 features
        * Shortcut layer - adds residual from input of block to current output tensor.
        * ReLU activation

- **Residual Layer 4**
    - Block 1
        * Convolution with 256 input channels, 512 output channels, 3x3 kernel, stride of 2, and padding of 1.
        * Batch Normalization with 512 features
        * ReLU activation function
        * Convolution with 512 input channels, 512 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 512 features
        * Shortcut layer - adds residual from input of layer to current output tensor.
        * ReLU activation
    - Block 2
        * Convolution with 512 input channels, 512 output channels, 3x3 kernel, stride of 1, and padding of 1.
        * Batch Normalization with 512 features
        * ReLU activation function
        * Convolution with 512 input channels, 512 output channels, 3x3 kernel, stride of 1, and padding of 1.

        * Batch Normalization with 512 features
        * Shortcut layer - adds residual from input of block to current output tensor.
        * ReLU activation

- **Pooling**
    - Global Average Pooling with output (1,1) for each channel
    - Tensor is flattened
- **Classification Heads**
    - **Digit 1 Head:** Dense Layer with 512 input units and 10 output units
    - **Digit 2 Head:** Dense Layer with 512 input units and 10 output units

2.4 Fully Convolutional Neural Network

The final architecture to be implemented is an improved iteration of the AlexNet-like architecture with the main architectural change moving from fully connected dense layers for the output classification heads to fully convolutional layers instead.
This architecture also introduces batch normalization in the convolutional layers aid in regularization of the parameters in the convolutional, feature learning part of the network. In short, the architecture consists of 5 convolutional layers, a fully convolutional layer, and two convolutional classification heads. The architecture is structured as follows:

- **Convolutional Layer 1**
    - Convolution with 1 input channel, 32 output channels, 5x5 kernel, and stride of 1
    - Batch Normalization with 32 features
    - ReLU activation function
    - Max Pooling with 2x2 kernel and stride of 2
- **Convolutional Layer 2**
    - Convolution with 32 input channels, 64 output channels, 5x5 kernel, and stride of 1
    - Batch Normalization with 64 features
    - ReLU activation function
    - Max Pooling with 2x2 kernel and stride of 2
- **Convolutional Layer 3**
    - Convolution with 64 input channels, 128 output channels, 3x3 kernel, and stride of 1
    - Batch Normalization with 128 features
    - ReLU activation function
- **Convolutional Layer 4**
    - Convolution with 128 input channels, 128 output channels, 3x3 kernel, and stride of 1
    - Batch Normalization with 128 features
    - ReLU activation function
- **Convolutional Layer 5**
    - Convolution with 128 input channels, 256 output channels, 3x3 kernel, and stride of 1
    - Batch Normalization with 256 features
    - ReLU activation function

  - Max Pooling with 2x2 kernel and stride of 2

- **Fully Convolutional Layers**
  - Convolution with 256 input channels, 512 output channels, 3x3 kernel, and stride of 1
  - Batch Normalization with 512 features
  - ReLU activation function

- **Classification Heads**
  - **Digit 1 Head:** Convolution Layer with 512 input channels, 10 output channels, and kernel size of 1.
  - **Digit 2 Head:** Convolution Layer with 512 input channels, 10 output channels, and kernel size of 1.

3 Experiments

3.1 Dataset

The dataset that will be used for experimentation is the DoubleMNIST dataset. This dataset was generated from the MNIST dataset using the generator Python script from the MultiDigitMNIST repository. [2]

When running the script as instructed within the repository, 1000 samples are generated per class with two digits existing within each image. That makes for 100 different combinations of digits and 100,000 total samples generated. The samples are split into training, validation, and testing sets with a proportion of 64%, 16%, 20% of the original dataset, respectively.

For all dataset splits, the images are transformed into a single grayscale channel and normalized using a mean of 0.1307 and standard deviation of 0.3081, which are the standard values used for MNIST normalization.

However, the training set receives some additional transforms to allow the models to better learn patterns and generalize instead of memorizing the training samples.

These training dataset transformations include:

- Random Rotation by 15 degrees
- Random Affine with translation by (0.1, 0.1), scaling by (0.9, 1.1), and shearing by 10.
- Random application of Random Erasing of probability 0.5 and scale (0.02, 0.1) with probability 0.5.

One important factor to note when using the DoubleMNIST or any multi-digit MNIST dataset is that the ImageFolder module from PyTorch parses the image folder names for each class as a string. However, to separate the digits as separate classification targets, i.e. class '13' separated as predictions '1' and '3', the string must be converted to an integer which can be done by applying a target transform that converts all the string targets to integers. The label can then later be separated during training and inference by using integer division and the modulo operator to obtain the tens and ones place digit, respectively.

3.2 Evaluation metrics

The various architectures will be evaluated using combined cross entropy loss of the two classifier heads and joint-digit classification accuracy (both digits must be classified correctly to be considered a true positive). Although labels for each image is by default a number between 00 and 99, the label can be separated into two digits by taking the integer division of the label and the modulo of the label for the digit in the tens and the ones place, respectively. Classification accuracy is defined as the number of correctly classified images divided by the total number of images observed and will be reported as a percentage.

The results to be discussed in the next section demonstrate training results on 64,000 samples and validation (mid-training testing) on 16,000 samples. Testing is performed on the remaining 20,000 samples at the end of training (after 30 epochs).

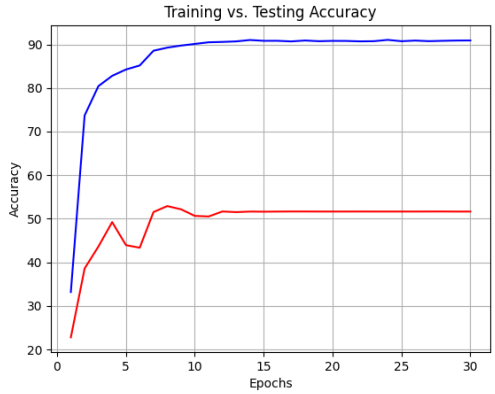
3.3 Results

The following Table 1 summarizes the results of training for the various models across 30 epochs for each learning rate they were trained on. The bolded values represent the best values across all models and their learning rate variations.

Model	Final Train Acc./Loss	Best Val. Acc./Loss	Testing Acc.	Testing Loss
AlexNet (lr = 1e-3)	90.92% / 0.307	52.91% / 1.793	51.45%	1.999
AlexNet (lr = 5e-4)	94.81% / 0.173	73.07% / 0.923	66.96%	1.341
ResNet (lr = 5e-4)	**98.54% / 0.0442**	94.66% / 0.195	93.64%	0.239
ResNet (lr = 2.5e-4)	98.47% / 0.0443	**96.62% / 0.139**	**96.20%**	**0.163**
Fully Conv. Net (lr = 1e-3)	96.20% / 0.117	89.13% / 0.373	83.13%	0.584
Fully Conv. Net (lr = 5e-4)	96.25% / 0.115	88.46% / 0.397	85.10%	0.507

Table 1: Model Performance Across Architectures

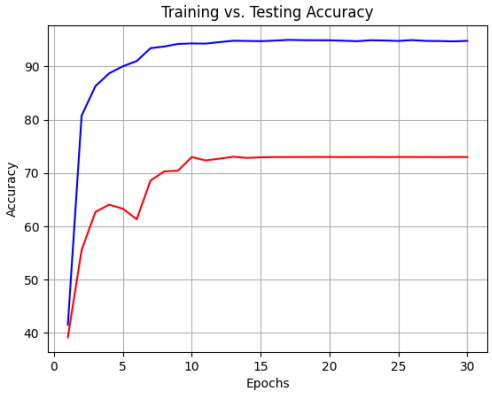
The following graphs in Figure 1 and Figure 2 display the accuracy and cross entropy loss curves for the AlexNet-like Network across 30 epochs.



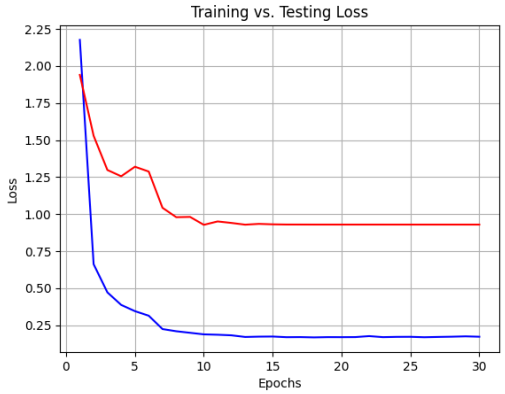
(a) Training (Blue) and Testing (Red) Accuracy

(b) Training (Blue) and Testing (Red) Loss

Figure 1: AlexNet Performance vs. Epochs with Learning Rate = 1e-3



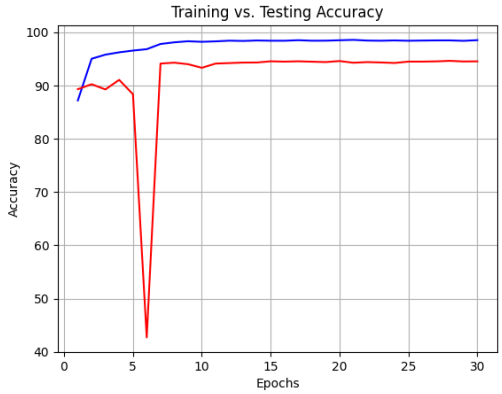
(a) Training (Blue) and Testing (Red) Accuracy



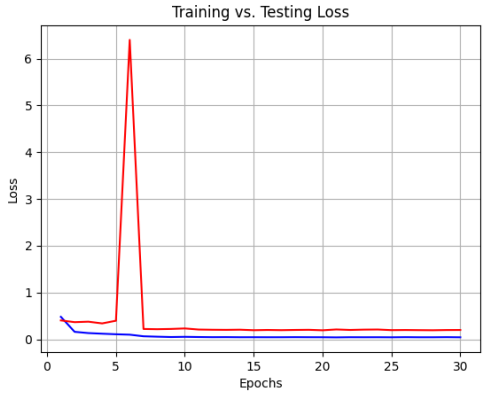
(b) Training (Blue) and Testing (Red) Loss

Figure 2: AlexNet Performance vs. Epochs with Learning Rate = 5e-4

The following graphs in Figure 3 and Figure 4 display the accuracy and cross entropy loss curves for the ResNet-like Network across 30 epochs.

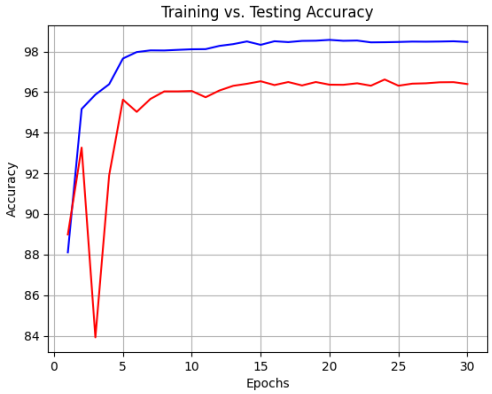


(a) Training (Blue) and Testing (Red) Accuracy



(b) Training (Blue) and Testing (Red) Loss

Figure 3: ResNet Performance vs. Epochs with Learning Rate = 5e-4

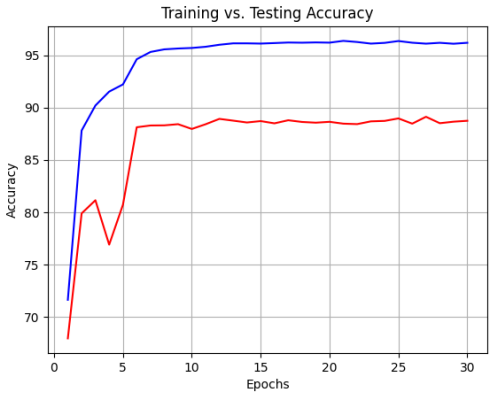


(a) Training (Blue) and Testing (Red) Accuracy

(b) Training (Blue) and Testing (Red) Loss

Figure 4: ResNet Performance vs. Epochs with Learning Rate = 2.5e-4

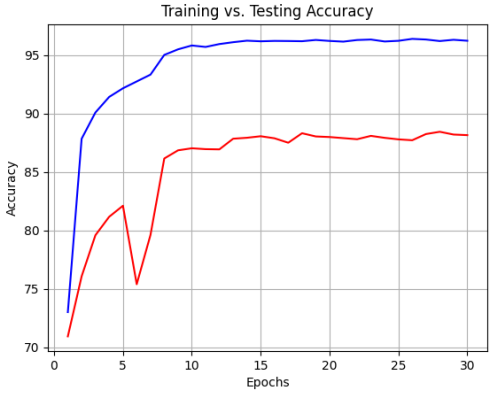
The following graphs in Figure 5 and Figure 6 display the accuracy and cross entropy loss curves for the Fully Convolutional Network across 30 epochs.



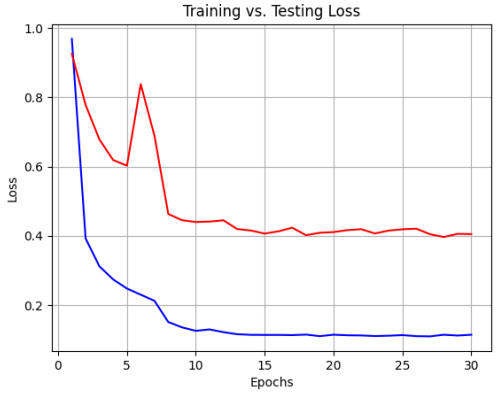
(a) Training (Blue) and Testing (Red) Accuracy

(b) Training (Blue) and Testing (Red) Loss

Figure 5: FCN Performance vs. Epochs with Learning Rate = 1e-3



(a) Training (Blue) and Testing (Red) Accuracy



(b) Training (Blue) and Testing (Red) Loss

Figure 6: FCN Performance vs. Epochs with Learning Rate = 5e-4

3.4 Analysis and discussions

3.4.1 Analysis of AlexNet performance

Based on the values shown in Table 1, AlexNet performed the worst of the three model types explored in this project with its best validation accuracy being 73.07% and testing accuracy being 66.96% when trained with a learning rate of 5e-4. The learning rate of 5e-4 was heuristically found to be consistently better than the learning rate of 1e-3 seeing a 21% difference in its best validation accuracy and 15% difference in testing accuracy when compared to the results of training with a learning rate of 1e-3. This result is likely due to the AlexNet architecture being a relative deep network and the large amount of fully-connected hidden units that are present in the fully-connected layers.

Another important observation to note is that across both trainings of the AlexNet model, final training accuracy is 90% and 94% for learning rates 1e-3 and 5e-4, respectively. However, the accuracy tapers off by 40% and 28% for the same learning rates, respectively. Although decreasing the learning rate reduced underfitting, this performance difference between training and testing demonstrates poor generalization and may be due to the model beginning to memorize the training data instead of continuing to learn useful features.

3.4.2 Analysis of ResNet performance

The ResNet18 inspired architecture performed the best of the three models explored in this project according to the values shown in Table 1. The incorporation of residual connections in the network improved gradient flow, enabling the model to train deeper feature representations without suffering from the vanishing gradient problem. This is reflected in both its high validation accuracy and top-performing test accuracy of 96.20%.

A substantial performance increase can be seen in lowering the learning rate from 5e-4 to 2.5e-4 with a 3% difference between the testing accuracy and 2% in peak validation accuracy. This suggests that while model capacity was high enough to avoid underfitting a more conservative learning rate allowed the model to converge to a more optimal minimum.

3.4.3 Analysis of Fully Convolutional Network performance

The fully convolutional model performed substantially better than the baseline AlexNet-like architecture, suggesting that removing the fully connected layers and replacing them with fully convolutional layers, effectively lowering parameter counts, reduced overfitting and improved spatial feature retention, as expected. The values in Table 1 show a flat 32% and 19% increase in testing accuracy and a flat 37% and 25% increase in peak validation accuracy when training with a learning rate of 1e-3 and 5e-4, respectively.

An interesting observation to note is that even across different learning rates for all model architectures, the accuracy and loss curves show similar peaks and dips around the same epochs and share similar curves. Take for example, the dip / spike in accuracy / loss that both occur at the 4-6 epoch range in Figure 5 and Figure 6.

3.4.4 Challenges in Development

Some challenges encountered during the development of the implementations for this project include mismatched labeling, failure to set the model to evaluation mode, and learning rate issues.

Mismatched labeling and forgetting to configure the model for evaluation mode during inference led to serious degradations in validation and testing performance. The problem of mismatched labeling was due to how the ImageFolder PyTorch module parses the dataset folder, which essentially stored the labels as strings instead of integers which the digit decoupler required. This issue was resolved by applying a target transform to the ImageFolder dataset object that converted the strings to integer values to be used as labels for training and inference. Forgetting to set the model to evaluation mode led to obvious performance issues because the model would then still attempt to update parameters and drop parameters where batch normalization and dropout layers were active during inferences.

Learning rate was a parameter that had to be heuristically tuned for. Training initially began with a learning rate of 1e-3, which showed mediocre results, especially with the ResNet-like architecture, leading to the decision to experiment with lower learning rates. Even with a learning rate of 5e-4 as a starting point, all models were struggling to converge to an optimal minima, so plateau-based learning rate scheduling was implemented aid in convergence of the models.

4 Conclusion

This project explored three neural network architectures for multi-label handwritten digit classification using the DoubleMNIST dataset. The AlexNet-like baseline, while capable of achieving reasonably high training accuracy, consistently overfit and underperformed on validation and test sets. The fully convolutional netowrk improved on this baseline by reducing parameter count and better preserving spatial information.

However, the ResNet-like architecture demonstrated the best overall performance, achieving a test accuracy of 96.20% with the lowest overall loss. Its residual blocks enabled stable gradient propagation and efficient training, allowing it to learn more discriminative multi-label features.

These results indicate that for multi-digit recognition tasks, deeper networks with skip connections offer clear advantages. Future work may explore stronger data augmentation, larger multi-label MNIST variants (e.g., TripleMNIST) or a multi-EMNIST variant, or attention-based architectures such as Vision Transformers to further enhance multi-label classification performance.

References

[1] G. Ayush, Mishra. Resnet18 from scratch using pytorch. [Online]. Available: https://www.geeksforgeeks.org/deep-learning/resnet18-from-scratch-using-pytorch/

[2] S.-H. Sun, "Multi-digit mnist for few-shot learning," 2019. [Online]. Available: https://github.com/shaohua0116/MultiDigitMNIST