# MNIST classification using Multimodal Deep Fusion Model

**Ankit Basu**
Computer Science Department
Texas A&M University
College Station, TX 77845
ankitbasu@tamu.edu

## Abstract

In this homework, we have implemented a novel deep learning model for multi-modal classification of handwritten digits and spoken digits, leveraging convolutional neural networks (CNNs) for image processing and Long Short-Term Memory (LSTM) networks for sequence understanding. Our model integrates both image and audio inputs, applying data augmentation techniques to increase generalization and achieve better performance. We demonstrate an accuracy of 99% on the test dataset. The model is an end-to-end solution for multi-modal classification tasks, with potential applications in diverse domains such as speech recognition and image classification.

## 1 Introduction

Multi-modal classification, the task of categorizing data with multiple modalities, poses unique challenges in machine learning due to the diverse nature of input sources. In this homework, we address the problem of multi-modal classification by designing a deep learning model capable of effectively processing both image and audio inputs. Our model aims to classify handwritten digits and spoken digits simultaneously, leveraging convolutional neural networks (CNNs) for image processing and Long Short-Term Memory (LSTM) networks for sequence understanding of embedded data. By integrating these two modalities, we enable the model to make accurate predictions across diverse data sources.

## 2 Method

For the image input preprocessing, the initial array of size $(N \times 784)$, where each row represents a flattened $(28 \times 28)$ image, is transformed to a numpy array of size $[N \times 1 \times 28 \times 28]$. The numbers denote the batch size ($B$), the number of channels ($C$), the height ($H$), and the width ($W$) of each image tensor respectively. This transformation reshapes the flattened images into their original $(28 \times 28)$ dimensions while also adding a single channel, as commonly required by the network.

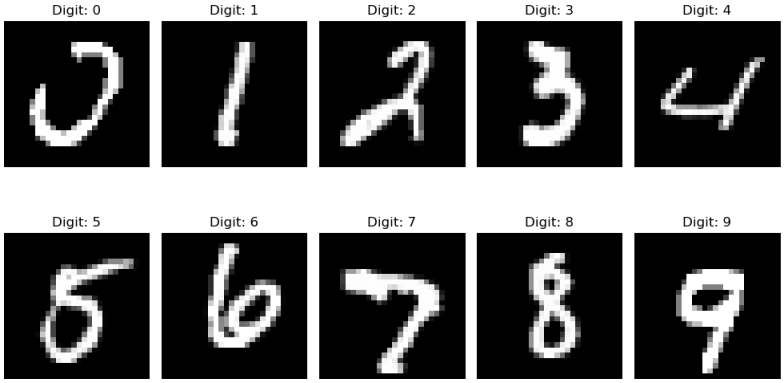
Next, the image data is processed by converting each image into a PIL (Python Imaging Library) Image object. This conversion facilitates subsequent augmentation techniques such as Random Flips, Random Rotation, and Random Cropping. These augmentation techniques help augment the dataset by introducing variations in the images, to enhance the model's ability to generalize to unseen data. These transformations are performed on the PIL Image objects, with the augmented images having dimensions $[B \times H \times W \times C]$.

Once the augmentation is applied, the augmented PIL Images are then converted back into tensors using the ToTensor() method. This method normalizes the pixel values of the images to the range [0, 1] and returns a tensor of shape $[B \times C \times H \times W]$, suitable for feeding into the convolution layers.

For the audio input pre-processing, the data remains unchanged in terms of its dimensions ($N \times 507$). Each row of the input array represents an audio sample with 507 features. The data is directly converted into a tensor without any reshaping, as it is already in a suitable format for further processing. This tensor is then fed into the model for subsequent processing and classification.

### 2.1 Data Pre-processing

For the image input pre-processing, the initial array of size $(N \times 784)$ is reshaped into a numpy array of size $[N \times 1 \times 28 \times 28]$, where each image is represented as a $(28 \times 28)$ matrix with a single channel. This reshaping is crucial to keep the structure of the images required for convolutional layers' processing.




Next, the image data is transformed into PIL (Python Imaging Library) Image object, resulting in dimensions $[B \times H \times W \times C]$. This step facilitates the application of augmentation techniques such as Random Flips, Random Rotation, and Random Cropping, which introduce variations into the dataset, thereby enhancing model generalization.

After applying these augmentations, the PIL Image objects are converted back into tensors using the ToTensor() method. This method normalizes the pixel values of the images to the range [0, 1] and returns a tensor of shape $[B \times C \times H \times W]$, suitable for input into the Conv2D and MaxPool layers.

On the other hand, for the audio input preprocessing, the data remains intact with dimensions $(N \times 507)$. Each row represents an audio sample with 507 features. This data is directly converted into a tensor without any reshaping, as it is already in a suitable format for further processing. The tensor is then fed into the model for subsequent processing and classification.

### 2.2 Model Design

The model comprises three main components:

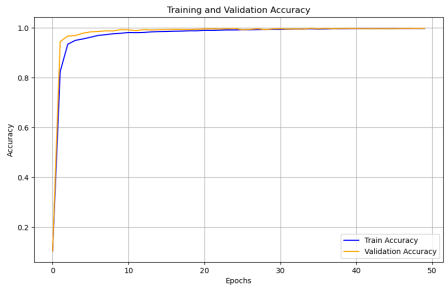
Image Encoder: This component utilizes convolution layers with max-pooling to extract features from $(28 \times 28)$ input images. The input images pass through two blocks of paired CNN layers: the first block uses a kernel size of 5 to capture broad spatial features, while the second block employs a kernel size of 3 to filter finer details. Max-pooling operations are applied after each block to downsample the feature maps.

Spoken Input Encoder: This component employs linear layers with dropout regularization to process the spoken input data. Sequentially applied linear layers with rectified linear unit (ReLU) activations reduce dimensionality and extract relevant features. The embeddings from this component are concatenated with the flattened embeddings we got from the image processing component.

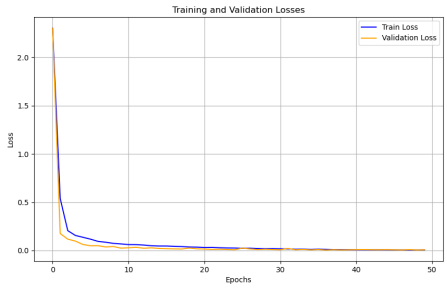
Final Layer (with LSTM): The final layer utilizes Long Short-Term Memory (LSTM) architecture to analyze the concatenated embeddings and understand data patterns. The LSTM dynamically adjusts its internal state based on the sequential input, capturing long-range dependencies and contextual information.

### 2.3 Model Training

During training, we see a quick boost in accuracy and there was a dip in loss as well. As part of the training model, I have used 50 epochs to train with 10% validation and 10% testing data holdout.

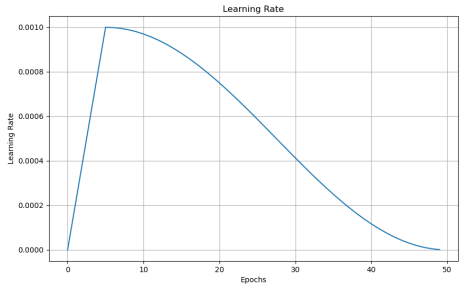



The model also sees a steady dip in the loss value as well after a few initial epochs.




### 2.4 Hyperparameter Tuning

The biggest impact on the model was the learning rate. The warmup and cosine reduction of the learning rate helped the model converge faster and reduce loss in the initial few epochs. To achieve this, we have used a custom scheduler.



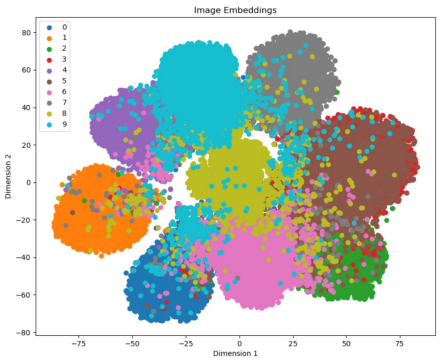

In addition to this, we are using a decay of $1e-2$ in the AdamW optimizer to apply stochastic gradient descent with L2 regularization that penalizes large weights in the network. As part of the learning, AdamW performed better than other optimizers.

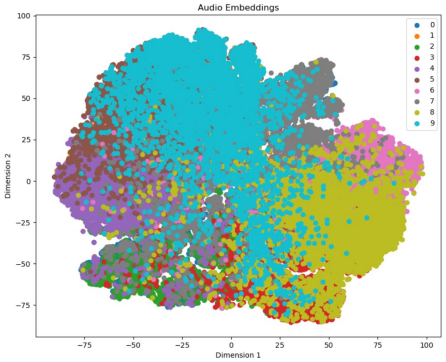
Also, using a batch size of 64 instead of 128 helped the model converge faster and evade getting stuck into the local-minima problem.

Hidden channels refers to the dimensionality of the hidden state of the LSTM cell. A higher hidden dimension helps learn sequential distinctive features, for our implementation, we have used 256 however, 128 also worked equally nicely (given the data is pretty simple).

Embed channels refer to the dimensionality of the input data at each time step, in our case, it is 128.

### 2.5 Embedding Analysis




From the images, we can see that Image Embeddings have worked much better in separating the clusters than it has done in the audio embeddings. The inter-separability of Image Embedding clusters has helped the KMeans to group the nearby data points better than overlapping cluster shapes of Audio Embedding Clusters.

## 3 Results

Using the model, I could achieve 99% accuracy on the test set while my training and validation loss were at 0.0820 and 0.0409 respectively. This model has secured 0.996 in the Kaggle competition.

Below is the classification report generated on the held-out test data for evaluation.

Test Overall Accuracy: 0.99

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 619 |
| 1 | 1.00 | 1.00 | 1.00 | 676 |
| 2 | 0.98 | 0.99 | 0.98 | 600 |
| 3 | 0.99 | 0.98 | 0.98 | 642 |
| 4 | 0.99 | 1.00 | 0.99 | 551 |
| 5 | 0.99 | 1.00 | 0.99 | 524 |
| 6 | 1.00 | 0.98 | 0.99 | 616 |
| 7 | 0.99 | 0.99 | 0.99 | 617 |
| 8 | 0.98 | 0.99 | 0.99 | 568 |
| 9 | 0.99 | 0.99 | 0.99 | 587 |
| accuracy | | | 0.99 | 6000 |
| macro avg | 0.99 | 0.99 | 0.99 | 6000 |
| weighted avg | 0.99 | 0.99 | 0.99 | 6000 |

We can see the model accurately classifies different labels. Based on the F1 scores, the best-performing classes are 0 and 1 at 100%, and the least-performing classes are 2 and 3 at 98%.

## 4 Conclusion

In this homework, we have presented a comprehensive approach to multi-modal classification of handwritten and spoken digits, leveraging Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM).

Our model demonstrates a great accuracy of 99% on the test dataset, showcasing its effectiveness in handling diverse data modalities. Furthermore, the exploration of additional data augmentation techniques, hyperparameter tuning, and optimization strategies has contributed to enhancing the model's performance and generalization capabilities.

To improve the robustness of the model further, we can incorporate additional augmentation techniques. In addition, for sequence understanding, self-attention based transformers could be a great choice.