Classification of Image and Audio Digits Dataset

Pavan Santosh Nippani Ravi
Computer Science Department
Texas A&M University
College Station, TX 77845
`pavan_santosh@tamu.edu`

Abstract

The following project is an implementation of deep learning neural network classifier model for multi modal data. The dataset consists of MNIST image and audio data represented as 1D numpy arrays. The data represents digits in the range of 0 to 9. The model uses two Convectional Neural Network Model which are trained on image and audio respectively and then they are late fused with an multi layer neural network in the end. After tuning and adding layers, the final model resulted in an accuracy score of 0.987.

1 Introduction

The project focuses on tackling the challenge of multimodal data classification, where each data point consists of both images and audio representations alongside corresponding labels denoting digits ranging from 0 to 9. Traditional classification models are often tailored to a single data modality, such as Convolutional Neural Networks (CNNs) for images, Recurrent Neural Networks (RNNs) for sequential data like audio, and transformers like BERT for text. However, the fusion of multiple modalities poses a unique set of challenges.

To address this, our approach leverages the strengths of these individual models to generate embeddings, or latent representations, for both image and audio data. Specifically, we employ CNNs to extract features from images and audio spectrograms, respectively. These embeddings are then concatenated to form a unified representation, which serves as input to a multi-layer neural network. This network is tasked with learning the intricate relationships between the embeddings and effectively classifying the digits.

By integrating information from both image and audio modalities, our model aims to capture richer and more nuanced patterns inherent in the data. Through the fusion of these modalities at the embedding level, we seek to enhance classification performance and enable the model to generalize better across diverse inputs.

This project not only addresses the practical challenge of multimodal data classification but also contributes to the broader understanding of how different data modalities can be effectively integrated to enhance machine learning tasks.

2 Model

The following subsection go through the different steps in building the model and how the parameters were chosen

2.1 Data Preprocessing

The data is available on the course Kaggle page here The dataset is a multimodal MNIST dataset. It contains images of written digits as well as the audio of spoken digits, paired with the corresponding label (0-9).

In the pre-processing phase of the model development, the first loaded the training data for both the image and audio modalities, leveraging the numpy format for efficient handling of numerical data. To ensure robust model performance and accurate evaluation both the datasets are divided into training, validation sets and test sets, adhering to a common practice of allocating 10% of the data for testing and of the remaning, 20% to validation set. This separation allows to monitor the model's performance on unseen data during training and fine-tune its parameters accordingly.

For the audio data, the data z score normalized. The image data underwent a series of preprocessing steps tailored to enhance its suitability for training our model. Each pixel value in the image dataset was converted to a floating-point number and normalized by dividing it with 255. This normalization process scales the pixel values to a range between 0 and 1, effectively standardizing the data and ensuring that the model's performance is not adversely affected by differences in pixel intensity across images. This normalization step is particularly crucial for deep learning models, as it helps stabilize the training process and improves convergence.

Additionally, as part of the image pre-processing pipeline, we resized the images from their original 1x784 dimension to a more compact and standardized 28x28 2D array format.

2.2 Model Design

CNNs (Convolutional Neural Networks) are well-suited for classifying both image and audio datasets due to their ability to capture spatial and temporal features hierarchically. In images, CNNs excel at identifying patterns through convolutions, pooling, and non-linear activations, effectively learning local and global features. Similarly, in audio data, CNNs can extract meaningful spectrotemporal representations, capturing both frequency and time-domain information. This makes CNNs particularly adept at discerning complex patterns in audio signals, such as phonemes or musical notes. Moreover, CNNs' parameter sharing and local connectivity properties enable them to handle high-dimensional input data efficiently, making them an ideal choice for tasks requiring feature learning from raw data. Overall, CNNs offer a versatile framework for extracting discriminative features from image and audio inputs, making them a powerful choice for multimodal classification tasks.

2.3 Model Training

Started of with four models. The model characteristics are represented in Tables 1,2,3,4 for each successive model layers were added with the power of two. Each of the four models was run for 10 epochs and the final test score was calculated from the test set. After 10 epochs, Model 2produced the best results with a test accuracy of 0.99, this set of layers was hence chosen for the final model.

2.4 Hyperparameter Tuning

Adam optimizer is used tune the image audio and the fusion model, The Adam optimizer adjusts hyperparameters during the training process through adaptive learning rates. The key hyperparameters that Adam tunes are the learning rate, momentum parameters, and an optional term called epsilon. Adam dynamically adjusts the learning rate for each parameter based on the estimates of the first and second moments of the gradients. The addition of epsilon ensures numerical stability, particularly when the second moment estimate approaches zero.

3 Results

The final accuray of the model after testing test set acheived an accuracy of 0.987 and with the kaggle leaderboard check it gave an accuracy of 0.99. Looking at the TSNE plot (Figure-2) of the embedding, the image model does a good job in splitting the dataset into right labels. Audio data has overlapping clusters and the borders are not as distinct as image.

Layer (type)	Output Shape	Param #	Connected to
conv1d_1_input (InputLayer)	[(None, 507, 1)]	0	[]
input_2 (InputLayer)	[(None, 28, 28, 1)]	0	[]
conv1d_1 (Conv1D)	(None, 506, 16)	48	conv1d_1_input[0][0]
conv2d_1 (Conv2D)	(None, 28, 28, 32)	320	input_2[0][0]
batch_normalization_4 (BatchNormalization)	(None, 506, 16)	64	conv1d_1[0][0]
batch_normalization_6 (BatchNormalization)	(None, 28, 28, 32)	128	conv2d_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 253, 16)	0	batch_normalization_4[0][0]
flatten_3 (Flatten)	(None, 25088)	0	batch_normalization_6[0][0]
flatten_2 (Flatten)	(None, 4048)	0	max_pooling1d_1[0][0]
dense_6 (Dense)	(None, 64)	1605696	flatten_3[0][0]
dense_4 (Dense)	(None, 64)	259136	flatten_2[0][0]
batch_normalization_7 (BatchNormalization)	(None, 64)	256	dense_6[0][0]
batch_normalization_5 (BatchNormalization)	(None, 64)	256	dense_4[0][0]
dropout_1 (Dropout)	(None, 64)	0	batch_normalization_7[0][0]
dense_5 (Dense)	(None, 10)	650	batch_normalization_5[0][0]
dense_7 (Dense)	(None, 10)	650	dropout_1[0][0]
concatenate (Concatenate)	(None, 20)	0	dense_5[0][0], dense_7[0][0]
dense_8 (Dense)	(None, 64)	1344	concatenate[0][0]
batch_normalization_8 (BatchNormalization)	(None, 64)	256	dense_8[0][0]
dense_9 (Dense)	(None, 10)	650	batch_normalization_8[0][0]

Table 1: Model 1

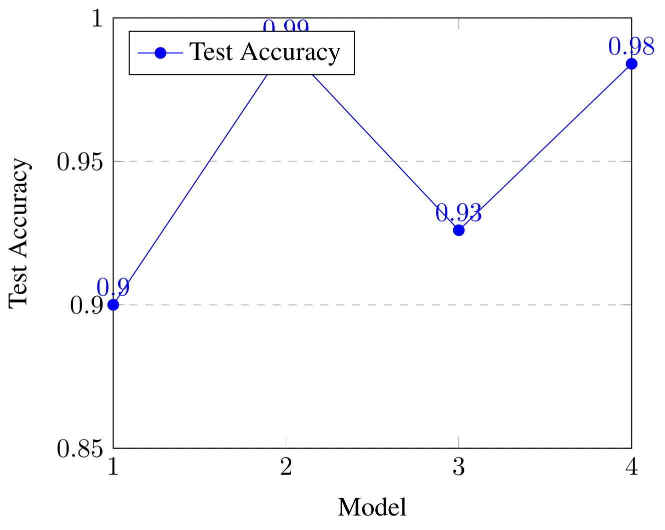

Figure 1: Model Accuracy after 10 epochs

4 Conclusion

The model performed well in terms of accuracy measurements. For each of the considered models, the layers were added without much consideration. Ideally all combination and values of layers should be trained and tested get the right number of layers for both CNN and the linear layer in the end. The CNN model worked well for the image data as seen from TSNE plt as it was able to split the classed. The audio CNN model did a decent job, not as good as image model.

Layer (type)	Output Shape	Param #	Connected to
conv1d_3_input (InputLayer)	[(None, 507, 1)]	0	[]
input_2 (InputLayer)	[(None, 28, 28, 1)]	0	[]
conv1d_3 (Conv1D)	(None, 506, 16)	48	conv1d_3_input[0][0]
conv2d_3 (Conv2D)	(None, 28, 28, 32)	320	input_2[0][0]
batch_normalization_4 (BatchNormalization)	(None, 506, 16)	64	conv1d_3[0][0]
batch_normalization_12 (BatchNormalization)	(None, 28, 28, 32)	128	conv2d_3[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 253, 16)	0	batch_normalization_4[0][0]
conv2d_4 (Conv2D)	(None, 28, 28, 32)	9248	batch_normalization_12[0][0]
conv1d_4 (Conv1D)	(None, 252, 32)	1056	max_pooling1d_3[0][0]
batch_normalization_13 (BatchNormalization)	(None, 28, 28, 32)	128	conv2d_4[0][0]
batch_normalization_5 (BatchNormalization)	(None, 252, 32)	128	conv1d_4[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 32)	0	batch_normalization_13[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 126, 32)	0	batch_normalization_5[0][0]
dropout_2 (Dropout)	(None, 14, 14, 32)	0	max_pooling2d_1[0][0]
conv1d_5 (Conv1D)	(None, 125, 64)	4160	max_pooling1d_4[0][0]
conv2d_5 (Conv2D)	(None, 14, 14, 64)	18496	dropout_2[0][0]
batch_normalization_6 (BatchNormalization)	(None, 125, 64)	256	conv1d_5[0][0]
batch_normalization_14 (BatchNormalization)	(None, 14, 14, 64)	256	conv2d_5[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 62, 64)	0	batch_normalization_6[0][0]
flatten_3 (Flatten)	(None, 12544)	0	batch_normalization_14[0][0]
flatten_1 (Flatten)	(None, 3968)	0	max_pooling1d_5[0][0]
dense_6 (Dense)	(None, 512)	6423040	flatten_3[0][0]
dense_2 (Dense)	(None, 64)	254016	flatten_1[0][0]
batch_normalization_15 (BatchNormalization)	(None, 512)	2048	dense_6[0][0]
batch_normalization_7 (BatchNormalization)	(None, 64)	256	dense_2[0][0]
dropout_3 (Dropout)	(None, 512)	0	batch_normalization_15[0][0]
dense_3 (Dense)	(None, 10)	650	batch_normalization_7[0][0]
dense_7 (Dense)	(None, 10)	5130	dropout_3[0][0]
concatenate (Concatenate)	(None, 20)	0	dense_3[0][0], dense_7[0][0]
dense_8 (Dense)	(None, 64)	1344	concatenate[0][0]
batch_normalization_16 (BatchNormalization)	(None, 64)	256	dense_8[0][0]
dense_9 (Dense)	(None, 128)	8320	batch_normalization_16[0][0]
batch_normalization_17 (BatchNormalization)	(None, 128)	512	dense_9[0][0]
dense_10 (Dense)	(None, 256)	33024	batch_normalization_17[0][0]
batch_normalization_18 (BatchNormalization)	(None, 256)	1024	dense_10[0][0]
dense_11 (Dense)	(None, 10)	2570	batch_normalization_18[0][0]

Table 2: Model 2

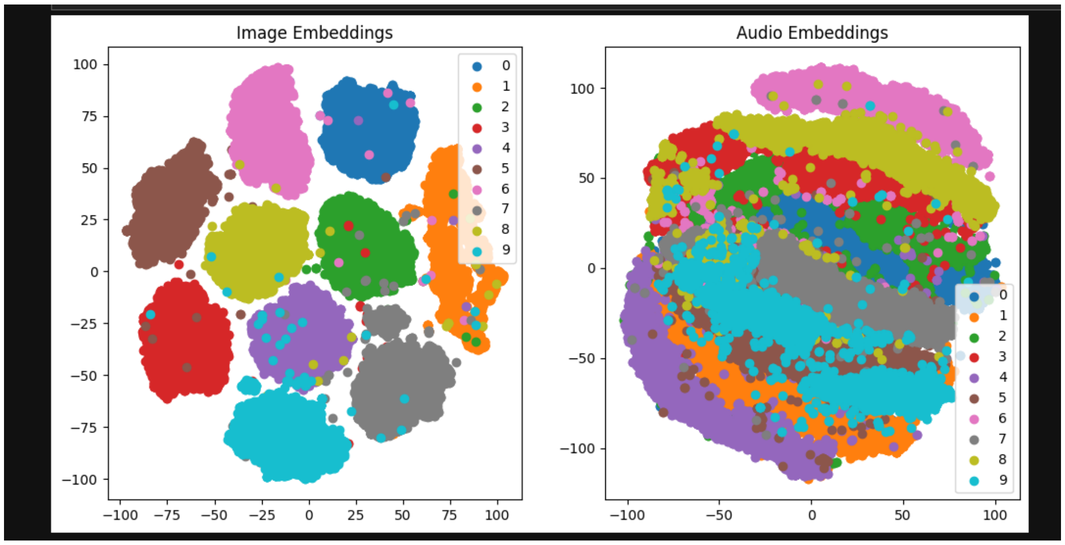

Figure 2: This is a caption for the image.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 28, 28, 1)]	0	[]
conv2d (Conv2D)	(None, 28, 28, 32)	320	input_1[0][0]
conv1d_input (InputLayer)	[(None, 507, 1)]	0	[]
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 32)	128	conv2d[0][0]
conv1d (Conv1D)	(None, 506, 16)	48	conv1d_input[0][0]
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9248	batch_normalization_3[0][0]
batch_normalization (BatchNormalization)	(None, 506, 16)	64	conv1d[0][0]
batch_normalization_4 (BatchNormalization)	(None, 28, 28, 32)	128	conv2d_1[0][0]
max_pooling1d (MaxPooling1D)	(None, 253, 16)	0	batch_normalization[0][0]
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0	batch_normalization_4[0][0]
conv1d_1 (Conv1D)	(None, 252, 32)	1056	max_pooling1d[0][0]
dropout (Dropout)	(None, 14, 14, 32)	0	max_pooling2d[0][0]
batch_normalization_1 (BatchNormalization)	(None, 252, 32)	128	conv1d_1[0][0]
flatten_1 (Flatten)	(None, 6272)	0	dropout[0][0]
flatten (Flatten)	(None, 8064)	0	batch_normalization_1[0][0]
dense_2 (Dense)	(None, 128)	802944	flatten_1[0][0]
dense (Dense)	(None, 64)	516160	flatten[0][0]
batch_normalization_5 (BatchNormalization)	(None, 128)	512	dense_2[0][0]
batch_normalization_2 (BatchNormalization)	(None, 64)	256	dense[0][0]
dropout_1 (Dropout)	(None, 128)	0	batch_normalization_5[0][0]
dense_1 (Dense)	(None, 10)	650	batch_normalization_2[0][0]
dense_3 (Dense)	(None, 10)	1290	dropout_1[0][0]
concatenate (Concatenate)	(None, 20)	0	dense_1[0][0], dense_3[0][0]
dense_4 (Dense)	(None, 64)	1344	concatenate[0][0]
batch_normalization_6 (BatchNormalization)	(None, 64)	256	dense_4[0][0]
dense_5 (Dense)	(None, 128)	8320	batch_normalization_6[0][0]
batch_normalization_7 (BatchNormalization)	(None, 128)	512	dense_5[0][0]
dense_6 (Dense)	(None, 10)	1290	batch_normalization_7[0][0]

Table 3: Model 3

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	320
conv1d_input (InputLayer)	(None, 507, 1)	0
batch_normalization_6 (BatchNormalization)	(None, 28, 28, 32)	128
conv1d (Conv1D)	(None, 506, 16)	48
conv2d_1 (Conv2D)	(None, 28, 28, 32)	9248
batch_normalization (BatchNormalization)	(None, 506, 16)	64
batch_normalization_7 (BatchNormalization)	(None, 28, 28, 32)	128
max_pooling1d (MaxPooling1D)	(None, 253, 16)	0
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
conv1d_1 (Conv1D)	(None, 252, 32)	1056
dropout (Dropout)	(None, 14, 14, 32)	0
batch_normalization_1 (BatchNormalization)	(None, 252, 32)	128
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
max_pooling1d_1 (MaxPooling1D)	(None, 126, 32)	0
batch_normalization_8 (BatchNormalization)	(None, 14, 14, 64)	256
conv1d_2 (Conv1D)	(None, 125, 64)	4160
conv2d_3 (Conv2D)	(None, 14, 14, 64)	36928
batch_normalization_2 (BatchNormalization)	(None, 125, 64)	256
batch_normalization_9 (BatchNormalization)	(None, 14, 14, 64)	256
max_pooling1d_2 (MaxPooling1D)	(None, 62, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 64)	0
conv1d_3 (Conv1D)	(None, 61, 128)	16512
dropout_1 (Dropout)	(None, 7, 7, 64)	0
batch_normalization_3 (BatchNormalization)	(None, 61, 128)	512
flatten_1 (Flatten)	(None, 3136)	0
max_pooling1d_3 (MaxPooling1D)	(None, 30, 128)	0
dense_3 (Dense)	(None, 512)	1606144
flatten (Flatten)	(None, 3840)	0
batch_normalization_10 (BatchNormalization)	(None, 512)	2048
dense (Dense)	(None, 64)	245824
dropout_2 (Dropout)	(None, 512)	0
batch_normalization_4 (BatchNormalization)	(None, 64)	256
dense_4 (Dense)	(None, 1024)	525312
dense_1 (Dense)	(None, 128)	8320
batch_normalization_11 (BatchNormalization)	(None, 1024)	4096
batch_normalization_5 (BatchNormalization)	(None, 128)	512
dropout_3 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	1290
dense_5 (Dense)	(None, 10)	10250
concatenate (Concatenate)	(None, 20)	0
dense_6 (Dense)	(None, 64)	1344
batch_normalization_12 (BatchNormalization)	(None, 64)	256
dense_7 (Dense)	(None, 128)	8320
batch_normalization_13 (BatchNormalization)	(None, 128)	512
dense_9 (Dense)	(None, 10)	1290

Table 4: Model 4