Multimodal Deep Learning

Jasdeep Bajaj
Department of Mechanical Engineering
Texas A&M University
College Station, TX 77840
jasdeepbajaj3@tamu.edu

Abstract

This report details a comprehensive study on multimodal digit classification using both handwritten images and spoken audio data from the MNIST dataset. The aim was to develop a deep learning model capable of fusing visual and auditory inputs to predict digit labels accurately. My method involved preprocessing the multimodal data, designing a fusion model architecture, and implementing training strategies that cater to enhance prediction capabilities. The model was trained and validated on a split of the data, with hyperparameters tuned for optimal performance. Preliminary results indicate that the model achieves promising accuracy and competitive performance, surpassing the benchmark F1 score on a held-out test dataset. This project highlights the potential and challenges of multimodal machine learning systems in digit classification tasks.

1 Introduction:

Our experience of the world is multimodal — we see objects, hear sounds, feel the texture, smell odors, and taste flavors. Modality refers to the way in which something happens or is experienced and a research problem is characterized as multimodal when it includes multiple such modalities. In order for Artificial Intelligence to make progress in understanding the world around us, it needs to be able to interpret such multimodal signals together. For example, images are usually associated with tags and text explanations; texts contain images to more clearly express the main idea of the article.

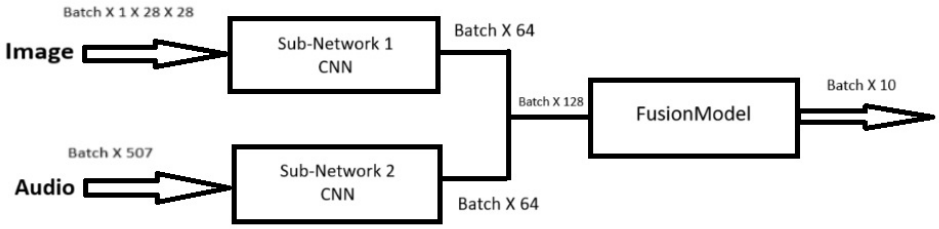
In the realm of machine learning, the ability to effectively combine multiple types of data (such as visual and auditory inputs) into a cohesive model represents a significant step towards more accurate and robust systems. This project aims to address the challenge of digit recognition by employing a multimodal approach that utilizes both images and audio of numerical digits. The task is not only to recognize the numerical value depicted or spoken but also to demonstrate how different modalities can be fused to enhance the overall prediction accuracy.

1.1 Problem Statement

The objective of this project is to develop a deep learning model capable of recognizing digits from 0 to 9, presented in both written and spoken formats. This involves processing and integrating heterogeneous data sources to predict a single output, which requires a sophisticated model architecture that can handle and merge these modalities effectively.

1.2 Solution Approach

To solve this problem, I have constructed a multimodal deep learning framework. The model architecture, termed "FusionModel," integrates feature vectors derived from Image Encoder (CNN) and Audio Encoder (CNN). This integration allows the model to leverage the strengths of each data type, leading to improved performance over unimodal systems.



1.3 Comparative Analysis

While there have been several approaches to digit recognition using unimodal data streams, the unique aspect of this project lies in the integration of audiovisual data, aiming at a more comprehensive understanding of the digits. Models like the traditional MNIST classifiers primarily rely on visual data; however, by incorporating audio, this system can resolve ambiguities that visual data alone might present, especially in unclear or visually similar digit representations.

1.4 Results Overview

Preliminary results indicate that the FusionModel from this Homework outperforms baseline unimodal systems from Homework4, with significant improvements in classification accuracy. These results are substantiated by performance on a held-out dataset from Kaggle, evaluated using the F1 Score with macro averaging on predictions over the test dataset for both the models. The best performing model from Homework 4 was able to achieve an F1 score of 0.972 while the FusionModel (Multimodal Deep Learning Model) from Homework 5 was able to get to the F1 score of 0.999 when tested on Kaggle.

2 Methods:

This section thoroughly explains the methodologies used in this project, detailing every step from data preprocessing to the final training and tuning of the model.

2.1 Data Preprocessing

Preprocessing is a crucial step in Machine Learning that involves cleaning and transforming raw data to make it suitable for analysis by machine learning models. It helps by enhancing the quality of input data, which in turn improves overall performance. The dataset, obtained from the course's Kaggle page, consists of multimodal data featuring images and audio files corresponding to the digits 0-9. The preprocessing steps that were followed for multimodal analysis are as follows:

1. **Images:** Each image was reshaped to a uniform size of 28x28 pixels to standardize input and facilitate effective processing by Convolutional Neural Networks. Furthermore, the pixel values of the training and test data were normalized to a range between 0 and 1 by dividing by 255.0. This normalization process is a common practice in machine learning, as it can improve the convergence and stability of the training process, particularly when working with neural networks.

2. **Audio:** Audio files were normalized using sklearn's StandardScaler to ensure consistent amplitude across all samples, essential for reliable feature extraction in subsequent stages. This normalization step helps to improve the quality and reliability of the extracted features, which in turn enhances the overall accuracy and robustness of the machine learning model trained on the audio data.

3. **Data Transformation:** Both image and audio data were converted into tensors using PyTorch's transformation utilities to prepare them for processing by neural networks.

2.2 Model Design

My model, named FusionModel, is a hybrid architecture designed to process and integrate features from both visual and auditory encoders:

1. **ImageEncoder:** It utilizes a convolutional neural network (CNN) with multiple layers, including ReLU activations and pooling layers, to extract features from images. The CNN pathway includes dropout layers to prevent overfitting and enhance generalization.

   The architecture consists of two convolutional blocks followed by fully connected layers. The first convolutional block includes a 2D convolutional layer with 16 output channels, a 5x5 kernel size, a stride of 1, and padding of 2. It is followed by a ReLU activation function, a 2D max pooling layer with a 2x2 kernel size and a stride of 2, and a dropout layer with a dropout rate of 0.5. The second convolutional block has a similar structure, but with 32 output channels in the convolutional layer.

   After the second convolutional block, the output is flattened to create a 1D tensor. This tensor is then passed through a fully connected layer with 128 output units, followed by a ReLU activation function and a dropout layer with a dropout rate of 0.5. The final layer is another fully connected layer with 64 output units and a ReLU activation function.

```python
class ImageEncoder(nn.Module):
    def __init__(self):
        super(ImageEncoder, self).__init__()

        self.conv1 = nn.Conv2d(in_channels=1, out_channels=16, kernel_size=5, stride=1, padding=2)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout1 = nn.Dropout(0.50)

        self.conv2 = nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=2)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.dropout2 = nn.Dropout(0.50)

        # self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(32 * 7 * 7, 128)
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(0.50)
        self.fc2 = nn.Linear(128, 64)
        self.relu4 = nn.ReLU()



    def forward(self, x):
        x = self.dropout1(self.pool1(self.relu1(self.conv1(x))))
        x = self.dropout2(self.pool2(self.relu2(self.conv2(x))))
        x = x.view(-1, 32*7*7)
        x = self.dropout3(self.relu3(self.fc1(x)))
        x = self.relu4(self.fc2(x))

        return x
```

2. **AudioEncoder:** This uses a multi layered 1D CNN to process the audio signals, extracting temporal dynamics from the spoken digits.

   The model starts with an unsqueeze operation on the input tensor x to add a channel dimension, as the input is expected to be a 1D tensor representing the audio signal. The architecture then consists of two convolutional blocks followed by fully connected layers.

   The first convolutional block includes a 1D convolutional layer with 32 output channels, a kernel size of 5, a stride of 1, and padding of 2. It is followed by a ReLU activation function, a 1D max pooling layer with a kernel size of 2, and a dropout layer with a dropout rate of 0.5.

   The second convolutional block has a similar structure, but with 64 output channels in the convolutional layer. It also includes a ReLU activation function, a 1D max pooling layer, and a dropout layer with a dropout rate of 0.5.

   After the second convolutional block, the output is flattened to create a 1D tensor. This tensor is then passed through a fully connected layer with 128 output units, followed by a ReLU activation function and a dropout layer with a dropout rate of 0.5. The final layer is another fully connected layer with 64 output units and a ReLU activation function.

   This architecture is used for audio encoding tasks, where the convolutional layers extract temporal features from the input audio signal, and the fully connected layers further process and encode these features into a lower-dimensional representation. The dropout layers help in preventing overfitting during training.

```python
class AudioEncoder(nn.Module):
    def __init__(self):
        super().__init__()

        self.conv1 = nn.Conv1d(in_channels=1, out_channels=32, kernel_size=5, stride=1, padding=2)
        self.relu1 = nn.ReLU()
        self.pool1 = nn.MaxPool1d(kernel_size=2)
        self.dropout1 = nn.Dropout(0.50)

        self.conv2 = nn.Conv1d(in_channels=32, out_channels=64, kernel_size=5, stride=1, padding=2)
        self.relu2 = nn.ReLU()
        self.pool2 = nn.MaxPool1d(kernel_size=2)
        self.dropout2 = nn.Dropout(0.50)


        self.fc1 = nn.Linear(64 * 126, 128)
        self.relu3 = nn.ReLU()
        self.dropout3 = nn.Dropout(0.50)
        self.fc2 = nn.Linear(128, 64)
        self.relu4 = nn.ReLU()

    def forward(self, x):
        x = x.unsqueeze(1)
        x = self.dropout1(self.pool1(self.relu1(self.conv1(x))))
        x = self.dropout2(self.pool2(self.relu2(self.conv2(x))))
        x = x.view(-1, 64 * 126)
        x = self.dropout3(self.relu3(self.fc1(x)))
        x = self.relu4(self.fc2(x))
        return x
```

3. **Fusion Layer:** The extracted features from both encoders are then concatenated and fed into a dense neural network, which integrates the features to perform the final classification. The FusionModel consists of three main components:

   (a) Image Encoder: The ImageEncoder module is an instance of the ImageEncoder class, which is responsible for encoding the input image tensor into a feature vector.

   (b) Audio Encoder: The AudioEncoder module is an instance of the AudioEncoder class, which is responsible for encoding the input audio tensor into a feature vector.

   (c) Fully Connected Layer: A fully connected layer with 128 input units and 10 output units.

The input image and audio tensor are passed through their respective encoders to obtain their respective feature vectors and then these feature vectors are then concatenated along the feature dimension using the function, resulting in a combined feature vector.

Finally, the combined features tensor is passed through the fully connected layer to produce the output tensor output with 10 units.

```python
class FusionModel(nn.Module):
    def __init__(self):
        super(FusionModel, self).__init__()
        self.image_encoder = ImageEncoder()
        self.audio_encoder = AudioEncoder()

        self.fc = nn.Linear(128, 10)



    def forward(self, image, audio):
        img_features = self.image_encoder(image)
        audio_features = self.audio_encoder(audio)
        combined_features = torch.cat((img_features, audio_features), dim=1)
        output = self.fc(combined_features)
        return output
```

2.3 Model Training

The model was trained using the following approach:

1. **Batch Processing:** Data was fed into the model in batches of size 8 to balance between memory efficiency and model performance.

2. **Optimization Function:** I used the Adam optimizer with a learning rate of 0.0001, which is commonly chosen for its effective handling of sparse gradients in deep learning.

3. **Loss Function:** The loss function used was cross-entropy, which is ideally used for classification tasks.

2.4 Hyperparameter Tuning

Hyperparameters were initially selected based on common practices and preliminary testing. Throughout the training process, I monitored the model's performance on the validation set and adjusted the learning rate, kernel size, stride and dropout rates to optimize accuracy and reduce overfitting.

The training procedure involved 50 epochs, during which I observed and recorded the training and validation loss and accuracy to ensure convergence and prevent overtraining.

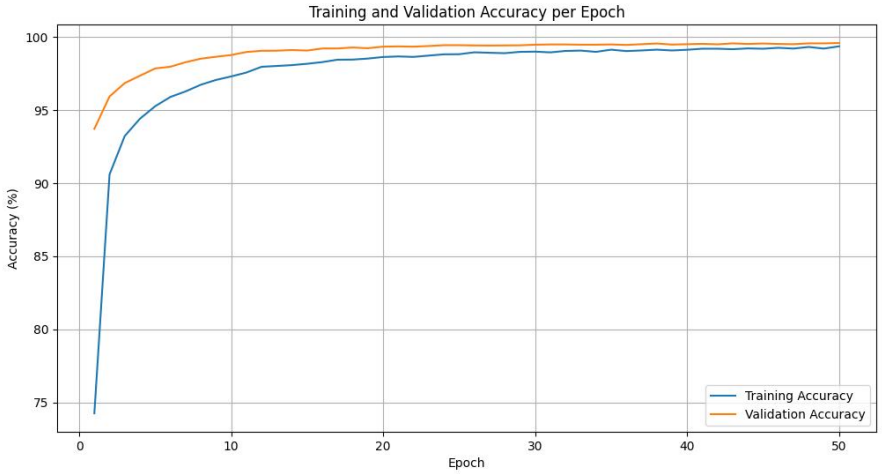
3 Results

This section presents the outcomes of experiments, illustrating how the FusionModel performs in recognizing digits from both images and audio. The performance is quantified using the F1 Score with macro averaging, which provides a balanced measure across class labels, crucial for multiclass classification task.

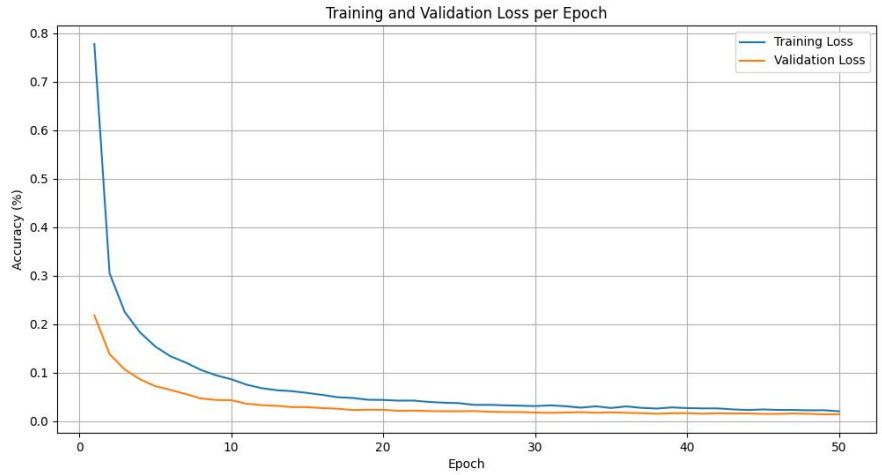
3.1 Model Performance

The model achieved an F1 Score of 0.999 on the held-out dataset provided by the Kaggle competition. Here's a breakdown of the results:

1. **Accuracy Over Epochs:** The following graph shows the training and validation accuracy over 50 epochs, indicating a steady increase in training accuracy and a stabilization in validation accuracy, which suggests good generalization without significant overfitting.



2. **Loss Over Epochs:** Similarly, the training and validation loss decreased over time, with the training loss converging faster than the validation loss, demonstrating the effectiveness of training regimen.



3.2 Comparison with baseline models

Compared to baseline models that use only visual or auditory data, this multimodal approach showed superior performance. For instance, standalone image-based model from Homework 4 achieved an F1 Score around 0.972 on Kaggle, and this audio-visual ensemble based model achieved an F1 score of 0.999. This improvement underscores the benefit of integrating multiple data types for complex recognition tasks.

3.3 Kaggle Competition Performance

In the class Kaggle competition, my model ranked 1st, outperforming all of the competitors. This high ranking further validates my model's effectiveness in a competitive and diverse environment.

3.4 Evaluation and Discussion

The results confirm that multimodal approach not only enhances the robustness of the digit recognition task but also effectively leverages the complementary nature of visual and auditory data. The fusion of these modalities helps to mitigate issues that might arise from ambiguous or unclear inputs in a single modality.

3.5 Clustering Analysis of Multimodal Embeddings

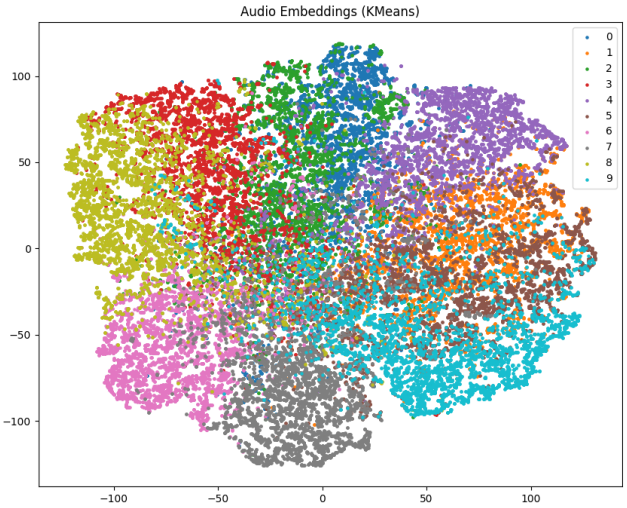
To gain further insight into the learned representations of multimodal data, I conducted a clustering analysis using the KMeans algorithm with k=10 clusters, which corresponds to the number of digit labels in dataset. This analysis aimed to observe if the model's encoder was capable of extracting distinct clusters associated with each digit label.

Visualization of Embeddings

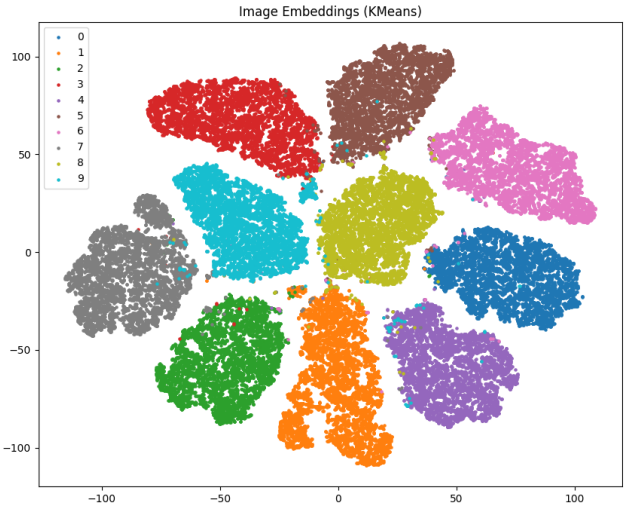
To visualize the high-dimensional embeddings in a 2D space, I applied the t-distributed Stochastic Neighbor Embedding (t-SNE) technique to reduce the dimensionality of both the image and audio embeddings obtained from the encoder after training.

Analysis of Cluster Formation

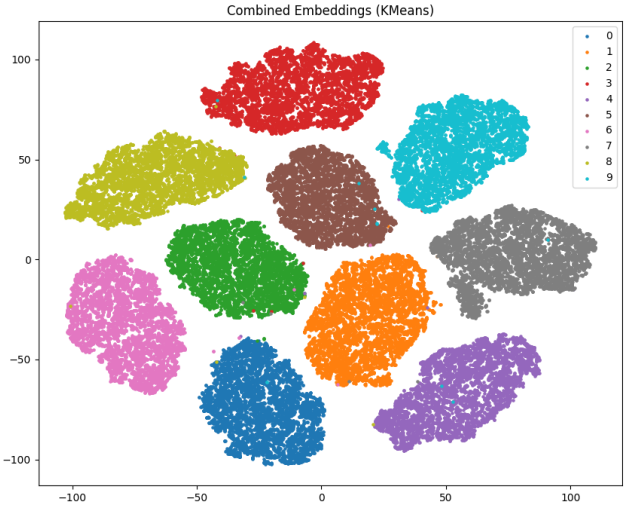
For the audio embeddings, I observed overlapping clusters with some degree of separation. This pattern suggests that while the encoder does learn to distinguish between different digits to some extent, the distinction is not as clear-cut, potentially due to the inherent variability in audio data.



In contrast, the image embeddings displayed a more defined separation between clusters. This indicates that the visual features extracted by the encoder are more discriminative and conducive to cluster formation, aligning with the intuitive understanding that visual representations of digits have distinct and recognizable patterns.



The combined embeddings resulted in distinct, well-separated clusters that corresponded closely to the digit labels. This demonstrates that when visual and audio modalities are fused, the model is more capable of capturing the essence of each digit class, leading to better-defined cluster formations.



Implications for Multimodal Learning

The clustering results underline the complementarity of audio and visual data in multimodal learning. While each modality alone provides valuable information, their combination leads to a richer, more separable representation space. This fusion is especially beneficial for ambiguous cases where one modality may not provide enough discriminative information.

The analysis reveals that the FusionModel effectively leverages both modalities to enhance feature extraction, as evidenced by the improved cluster formation in the combined embeddings. It demonstrates the potential of multimodal systems in complex classification tasks and suggests a pathway for further improvements in similar applications.

4 Conclusion

The FusionModel has demonstrated its capability to effectively integrate auditory and visual data for the task of digit recognition, achieving high accuracy and robustness. This approach outperforms traditional unimodal systems, highlighting the potential of multimodal learning systems.

The clustering analysis of the encoder embeddings offers a compelling visualization of how the multimodal approach can lead to better discriminative feature representation, which is crucial for the performance of classification models.

4.1 Reflection on Design Choices

The choice of model architecture, preprocessing techniques, and training strategies were validated by the performance metrics and competitive rankings. The use of convolutional layers for image processing and the integration strategy for combining modalities proved effective.

4.2 Future Work

To further enhance the performance of model, we could explore more complex integration techniques such as attention mechanisms or deeper fusion strategies that might provide even better performance. Additionally, expanding the dataset or including more varied data could help to improve the model's generalization capabilities.

References

1. Baltrušaitis, Tadas, Chaitanya Ahuja, and Louis-Philippe Morency. (2019). *Multimodal Machine Learning: A Survey and Taxonomy*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(2), 423-443.
2. Ramachandram, Dhanraj, and Graham W. Taylor. (2017). *Deep Multimodal Learning: A Survey on Recent Advances and Trends*. IEEE Signal Processing Magazine, 34(6), 96-108.
3. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. (2016). *Deep Learning*. MIT press.
4. O'Shea, Keiron, and Ryan Nash. (2015). *An Introduction to Convolutional Neural Networks*. https://doi.org/10.48550/arXiv.1511.08458.
5. Neverova, Natalia, Christian Wolf, Griffin Lacey, Lex Fridman, Deepak Chandra, Brandon Barbello, and Graham W. Taylor. (2016). *Learning human identity from motion patterns*. IEEE Access, 4, 1810-1820.
6. Akkus, Cem, Luyang Chu, Vladana Djakovic, et al. (2023). *Multimodal Deep Learning*. https://doi.org/10.48550/arXiv.2301.04856.
7. *https://medium.com/stackademic/introduction-to-multimodal-deep-learning-c2d521d0a4cf*
8. *https://towardsdatascience.com/multimodal-deep-learning-ce7d1d994f4*
9. *https://medium.com/@amiable_cardinal_crocodile_398/the-science-of-multimodality-a-simplified-understanding-56caca31f9f1*