LDA and CNN on 3D MNIST Dataset

Anonymous ACL submission

Abstract

This project investigates the classification of 3D representations of handwritten digits using a modified version of the MNIST dataset. By transforming 2D images into voxelized 3D point clouds, this research introduces new challenges and opportunities in digit classification and visualization. Three models are compared: a baseline Linear Discriminant Analysis (LDA), a 3D Convolutional Neural Network (CNN) with batch normalization and ReLU activation, and a modified 3D CNN utilizing group normalization and LeakyReLU activation. The experiments were conducted on a dataset comprising 10,000 training samples and 2,000 test samples, with consistent data preprocessing across models.

The 3D CNN models significantly outperform the LDA baseline, showcasing the effectiveness of deep learning in extracting features from high-dimensional voxelized data. Furthermore, the modified 3D CNN demonstrates enhanced performance compared to the standard 3D CNN, indicating that architectural adjustments, such as alternative normalization and activation functions, can lead to improved results. Evaluation metrics, including accuracy, precision, recall, and F1-score, were used to validate the models' performance. This study contributes to the growing field of 3D computer vision by offering insights into the comparative strengths of traditional and deep learning approaches for 3D data classification, while emphasizing the potential of advanced CNN architectures.

1 Introduction

This project aims to develop a machine learning pipeline for classifying 3D representations of handwritten digits based on a modified version of the MNIST dataset (Castro De La Iglesia, 2019). Transformation of the data set into 3D point clouds introduces a novel aspect to the classic 2D data set. Point clouds enhance the classification challenge and offer a rich visualization potential. I explore the effectiveness of different machine learning and deep learning models in classifying voxelized data, as well as experiment with 3D visualizations to enable better interpretability and presentation of classification results. The baseline model chosen is Linear Discriminant Analysis (LDA) and it will be compared to a 3D Convolutional Neural Network (CNN) with activation function ReLU (Rectified Linear Unit). This project aims to create a robust model for 3D digit classification and utilize computer vision techniques to produce intuitive 3D visualizations of each digit. Overall, this report is a comparative analysis and evaluation of LDA and CNN on their ability to classify voxelized 3D handwritten digits.

2 Background

2.1 Description of the 3D MNIST Dataset

The 3D MNIST dataset used for this project is an augmented version of the original MNIST handwritten digits dataset (Castro De La Iglesia, 2019). The original MNIST (Modified National Institute of Standards and Technology) dataset is a large collection of images of handwritten single digits that is commonly used for training various image processing systems (LeCun et al., n.d.). While the original MNIST has a training set of 60,000 samples, and a test set of 10,000 samples, this project utilizes just 10,000 samples for training, and 2000 samples for testing.

This 3D MNIST dataset was created by Devid De La Iglesia Castro on Kaggle, an online programming platform for data science and machine learning enthusiasts (Castro De La Iglesia, 2019). Castro generated 3D point clouds from the original images and then voxelized each into boxes of 16*16*16 voxels. The data is then stored in an HDF5 format with four main arrays: X_train (10,000 training samples with each digit represented as a 4096-dimensional vector from the voxelized 3D point cloud), y_train (10,000 corresponding labels), X_test (2,000 test samples), and y_test (2,000 corresponding labels). In addition, random rotations and noise were added to the data set. In the interest of applying deep learning on 3D point clouds, the 3D MNIST dataset is a great starting point to try out different learning techniques because it requires minimal pre-processing and formatting.

2.2 Understanding Point Clouds and Voxels

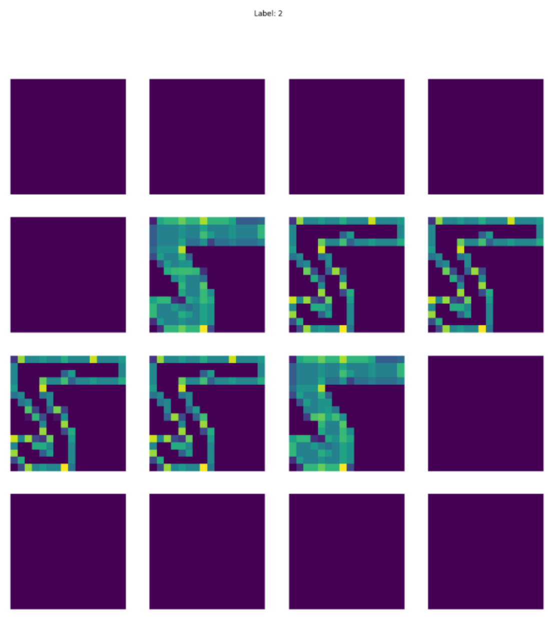


Figure 1: Voxels of the digit ‘2’ sliced along the z-axis coded with Matplotlib

To better understand how to utilize the 3D MNIST dataset, we must also detail its datatype. To create the 3D dataset, Castro generated point clouds from the 2D MNIST images. It consists of data points throughout the volume of the 3D digits shape with x, y, and z coordinates. To the human eye and brain, we can classify the number from looking at a point cloud of the object, but the computer cannot. As a whole, the points make up the shape and volume of an object, but individually, the points simply exist in a space with no order, color, and continuity between each other. At this stage, point clouds are not applicable to deep neural networks. Transforming raw data into actionable insights requires some level of feature engineering. Specifically, there needs to be a conversion, or extraction, of its features before it can be fed into a deep neural network.

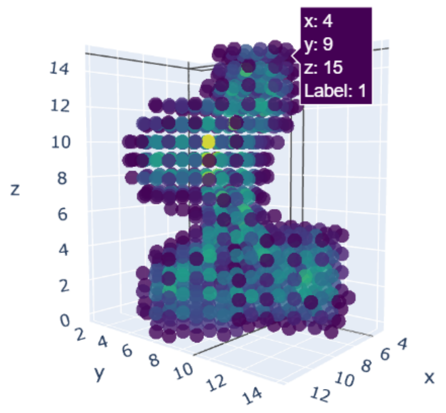


Figure 2: Visualization of a voxelized point cloud of the digit ‘1’ coded with Plotly

There are architectures such as PointNet that have been invented specifically for processing point clouds for deep learning.Another method of feature extraction is voxelization. Voxelization is the method that Castro has applied to the 3D point cloud handwritten digits to make it applicable to a 3D CNN.This process transforms the sparse and unordered nature of point clouds into a structured grid format, similar to the way pixels are organized in 2D images for 2D CNNs. By dividing the 3D space into a regular voxel grid, each voxel can be assigned a value representing the density or average of points within it, enabling convolutional layers to effectively extract spatial features in three dimensions.

In this dataset, each point cloud is split into a grid of 16 x 16 x 16 voxels. Voxels are cube shaped spaces. Then, the value of each voxel is computed by the average of the point cloud point positions inside of it, which effectively establishes a mathematical relationship between the points. If there exist no points, then the voxel is considered empty. The figure below is a visualization of the number ‘1’ as a 16 x 16 x 16 voxelized point cloud.

2.3 Understanding Convolutional Neural Network (CNN)

CNNs are a specialized type of deep learning model designed for structured data like images. (Chen et al., 2021) They utilize convolutional layers to automatically extract hierarchical features directly from raw input data. As shown in Figure 3 the CNN consists of convolutional layers, activation function, pooling, and fully connected layers to

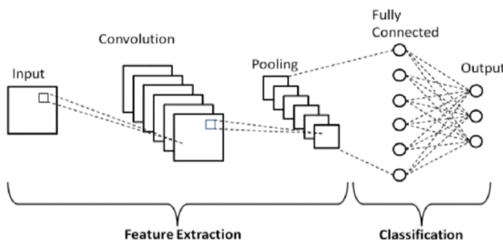


Figure 3: Basic CNN diagram

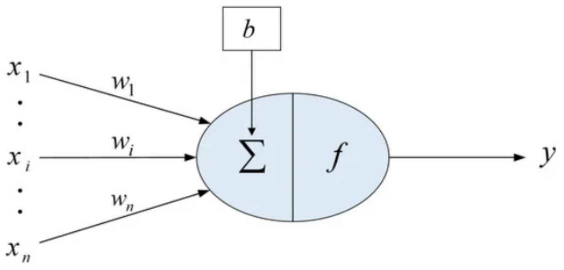


Figure 4: Deep learning neuron model where xi is the input signal, n is the number of signals, the weight value of the input signal is wi, bias is b and output of neurons is y. Note: Taken from article *Review of Image Classification Algorithms Based on Convolutional Neural Networks*

help with output generation (Newatia, 2020).

To extract even further, the principle operation of deep learning models is taking multiple input values and performing mathematical transformation to compute a single output value; The mathematical transformation is

$$f\left(b+\sum_{i=1}^{n}(x_i \times w_i)\right) \tag{1}$$

(Chen et al., 2021) The activation functions for this project are specified later in the "Approach" section. With this, a biological neuron is often used as an analogy for deep learning neural networks (Figure 4). Many neurons can be layered together as shown in Figure 5 to form a multilayer perceptron. This functionality of transforming many inputs to one output unit through hidden layers is a key component of understanding CNNs.

A convolutional layer performs a mathematical operation called convolution, which involves a sliding window function on a pixel matrix of an image (Keita, 2023). This sliding function is known as the kernel or filter. The filter applied to subregions of the image performs element-wise multiplication and sums the results to produce a new value (Keita, 2023). This value is then placed in a feature map,

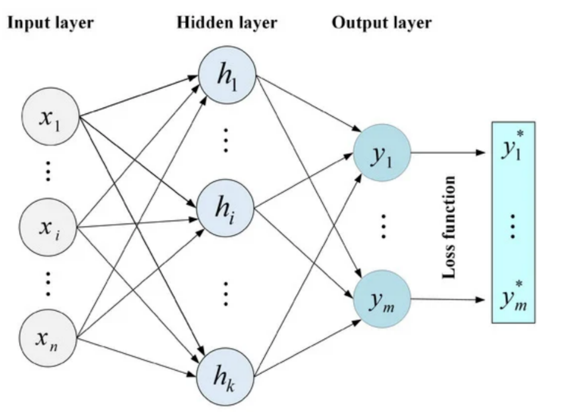


Figure 5: Multilayer perceptron. Note: Taken from article *Review of Image Classification Algorithms Based on Convolutional Neural Networks* (Chen et al., 2021)

which highlights the presence of certain patterns, such as edges, textures, or shapes, within the image. Multiple filters are used in a convolutional layer, each designed to detect different features. For instance, one filter may be adept at identifying horizontal edges, while another may excel at detecting vertical edges or specific textures. By stacking multiple convolutional layers, CNNs can learn hierarchical representations of the input data, where lower layers capture simple features and higher layers capture more complex patterns.

Typically, after the convolutional layer, the pooling layer is next (Chen et al., 2021). The pooling layer performs down-sampling which reduces the dimensionality of the input data while preserving the most critical features. This step enhances computational efficiency because of the dimensionality reduction, and makes the output feature map more resistant to error and distortion.

As described with the neuron analogy, the inputs must undergo a mathematical transformation. Thus, after each convolution operation, the activation function is applied. The specific activation functions utilized for this project are specified later in the section "Approach". After multiple convolutional and pooling layers, the output is flattened into a one-dimensional vector. Fully connected layers use this vector to integrate all learned features and output class probabilities. This layer acts as the classifier, with neurons representing potential class memberships.

While CNNs are highly effective for 2D data such as images, many real-world applications involve volumetric data or sequences that contain spatial and temporal information. In this case, a 3D MNIST dataset. To handle this type of data, researchers extend the principles of CNNs into the third dimension. 3D CNNs are a natural extension of 2D CNNs. Instead of 2D kernels sliding over height and width dimensions, 3D kernels operate across height, width, and depth. This allows the model to capture volumetric features

2.4 Technology and Programming Libraries Used for this Project

For data processing, I leveraged NumPy for efficient numerical operations and h5py to handle the HDF5 file type of the dataset. For 3D visualization, I used Matplotlib for static plots and Plotly for creating interactive visualizations. In the machine learning domain, I used PyTorch to develop and train 3D convolutional neural networks (3D CNNs) and utilized Scikit-learn for linear discriminant analysis (LDA), one-hot encoding, and computing evaluation metrics.

Since this project involves vectors of 4096-dimension and a deep learning neural network, it is more efficient to harness GPU processing power as the execution duration would greatly decrease. So, all development and testing were conducted on Kaggle. Kaggle offers free remote access to their T4 x 2 GPU and P100 GPU limited to 30 hours per week, as well as a cloud-based environment for running Jupyter Notebooks. The cloud-based environment ensures seamless integration of these libraries and tools.

3 Approach

3.1 Visualization

Before beginning to train any model, it was important to visualize the given dataset as there are many variables to consider, especially when working with unfamiliar datasets. Visualization enables discernment of inherent characteristics and nuances that might not be apparent from numerical data inspection. By examining visual plots, researchers can identify critical features such as variations in digit representations, which may significantly influence both model architecture and preprocessing strategies.

In the provided implementation, the visualize_voxels function facilitates this process by randomly selecting three indices to generate interactive voxel plots (Azizov, 2022). Interactive 3D plots are especially advantageous because they al-

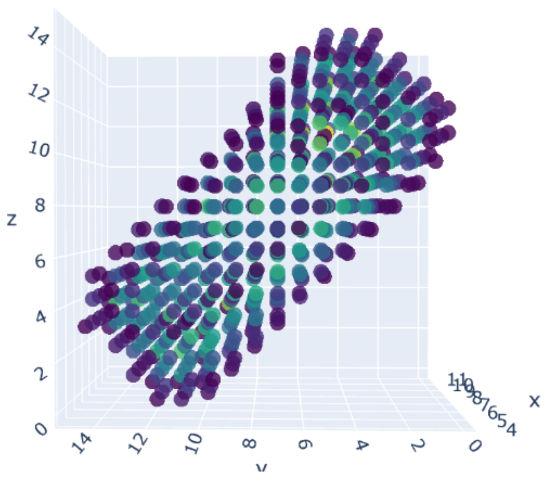


Figure 6: Another variation of the digit ‘1’

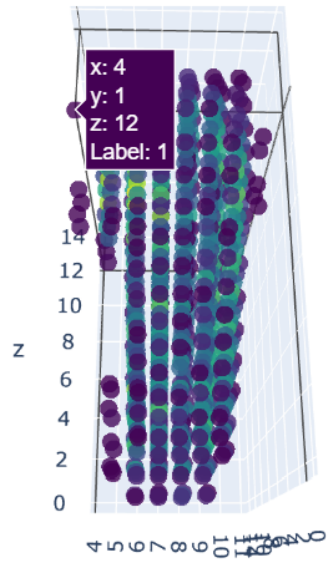


Figure 7: Figure 6 rotated to the left shows that it is not completely flat because noise and rotation has been added to the dataset

low rotation, offering a comprehensive 360-degree view of the digit structures. These initial visualizations, as shown in Figure 6, depict the digit ’1’. However, a comparison between Figures 6 and 7 reveals the dataset’s inclusion of diverse handwriting styles for the same digit. This variability shows that the model must classify with finer granularity, as it must learn to recognize multiple representations of the same numeral. While there is the option to remove these variations to simplify the classification task, doing so would compromise the dataset’s authenticity and fail to reflect real-world scenarios where such diversity is prevalent.

3.2 Baseline Model: Linear Discriminant Analysis (LDA)

As a baseline, LDA has been implemented and evaluated to show the difference between a model meant for image classification (CNN) and a model not designed for image classification.

3.3 Better Fit Model: Convolutional Neural Network (CNN)

A better fit model for this classification task is a convolutional neural network (CNN) designed for 3D data classification. The network comprises of multiple 3D convolutional layers, batch normalization, ReLU activations, max-pooling operations, and a fully connected layer. At its core, the model consists of a sequence of five convolutional layers, each equipped with specific enhancements to improve learning and generalization, followed by a fully connected (dense) layer for classification.

The model begins with an input layer that accepts 3D tensors (voxel data). The 3D tensor input is typically represented as

$$\mathbf{X} \in \mathbb{R}^{D\times H\times W\times C} \tag{2}$$

(OpenAI, 2024) where D, H, W denote the depth, height, and width of the input volume, and C is the number of channels (1 for grayscale in this case). The first set of layers applies 3D convolution using a filter

$$\mathbf{W} \in \mathbb{R}^{k_d\times k_h\times k_w\times C_{\text{in}}\times C_{\text{out}}} \tag{3}$$

(OpenAI, 2024) to extract spatial and structural features across three dimensions. Here,

$$k_d, k_h, k_w$$

are kernel dimensions, and

$$C_{\text{in}}, C_{\text{out}} \tag{4}$$

(OpenAI, 2024) are the input and output channels. Each convolution layer uses a 3×3×3 kernel, effectively scanning small cubes of data to detect localized patterns. The convolution operation for an output activation at location (d, h, w, c) can be expressed as:

$$y_{d,h,w,c} = \sum_{i=1}^{k_d}\sum_{j=1}^{k_h}\sum_{k=1}^{k_w}\sum_{l=1}^{C_{\text{in}}} W[i,j,k,l,c]\cdot r + b_c \tag{5}$$

(OpenAI, 2024) where

$$r = X[d+i-1, h+j-1, w+k-1, l]$$

$$b_c = \text{ bias for the c-th output channel.}$$

Following each convolution operation, batch normalization is applied.

$$\hat{x} = \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}}, \quad y = \gamma\hat{x}+\beta, \tag{6}$$

(OpenAI, 2024) where

$$\mu \text{ and } \sigma^2$$

are the mean and variance of the batch, and

$$\gamma, \beta$$

are learnable parameters. This normalization stabilizes training by normalizing the activations and reducing internal covariate shifts, enabling faster convergence. Subsequently, a ReLU (Rectified Linear Unit) activation function is employed to introduce non-linearity, allowing the network to learn complex patterns. After the second set of convolutional layers, a 3D max-pooling operation is performed. The pooling operation selects the maximum value in a local window:

$$y_{d,h,w,c} = \max_{i,j,k} X[d+i, h+j, w+k, c], \tag{7}$$

(OpenAI, 2024) where i, j, k are the pooling window dimensions. Additional convolutional blocks refine the extracted features further. These blocks follow the same pattern of convolution, batch normalization, and ReLU activation, progressively transforming the data into a more abstract representation. A final convolutional layer reduces the number of channels to 16. This dimensionality reduction simplifies the feature representation while retaining sufficient information for the classification task.

Once the 3D convolutional layers extract and distill the features, the data is passed to a flattening layer, converting the multi-dimensional tensor into a one-dimensional vector suitable for a fully connected (dense) layer. This vector serves as input to the dense layer, which maps the extracted features to the desired number of output classes (in this case, 10). A dropout layer with a 20

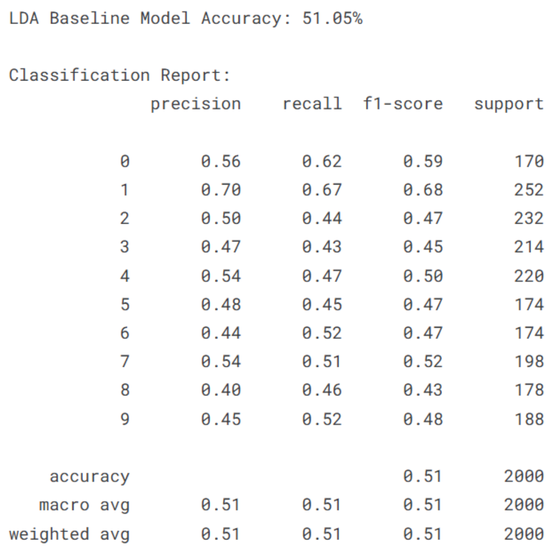
4 Experiments, Classification Reports, and Results

As aforementioned, the LDA model is a non-image classification model to compare with CNN. On top of the batch normalization and ReLU activation function 3D CNN model, there is an additional 3D CNN model with some modifications. This modified 3D CNN utilizes group normalization and LeakyReLU as its activation function instead. The results of LDA compared to either 3D CNN are unsurprising. On the other hand, the modified 3D CNN show promising results.

For all three models, the test and train dataset remained the same. The only difference is that in 3D CNN there are initialized variables needed, such as learning rate=0.001, epochs=30, batch size=32, number of threads/processes for data loading to batches=2, and weight decay=0.0001.

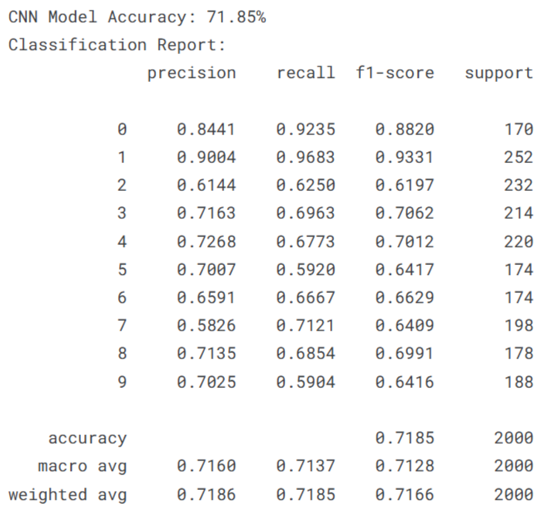
The relevant evaluation metrics for these experiments are accuracy, precision, recall, and f1-score.

4.1 Accuracy and Classification Reports



```
LDA Baseline Model Accuracy: 51.05%

Classification Report:
              precision    recall  f1-score   support

           0       0.56      0.62      0.59       170
           1       0.70      0.67      0.68       252
           2       0.50      0.44      0.47       232
           3       0.47      0.43      0.45       214
           4       0.54      0.47      0.50       220
           5       0.48      0.45      0.47       174
           6       0.44      0.52      0.47       174
           7       0.54      0.51      0.52       198
           8       0.40      0.46      0.43       178
           9       0.45      0.52      0.48       188

    accuracy                           0.51      2000
   macro avg       0.51      0.51      0.51      2000
weighted avg       0.51      0.51      0.51      2000
```

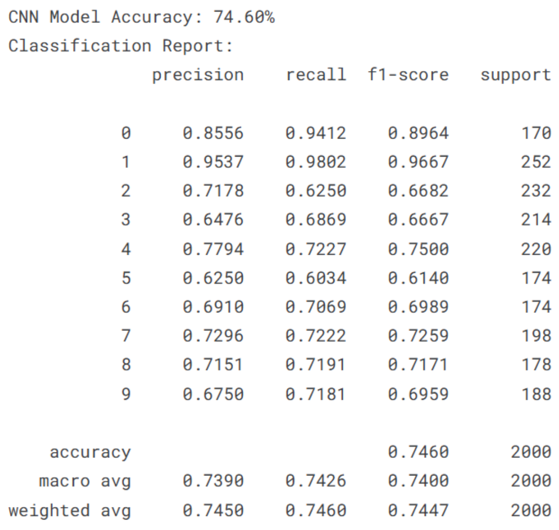
Figure 8: LDA model accuracy and classification report



```
CNN Model Accuracy: 71.85%
Classification Report:
              precision    recall  f1-score   support

           0     0.8441    0.9235    0.8820       170
           1     0.9004    0.9683    0.9331       252
           2     0.6144    0.6250    0.6197       232
           3     0.7163    0.6963    0.7062       214
           4     0.7268    0.6773    0.7012       220
           5     0.7007    0.5920    0.6417       174
           6     0.6591    0.6667    0.6629       174
           7     0.5826    0.7121    0.6409       198
           8     0.7135    0.6854    0.6991       178
           9     0.7025    0.5904    0.6416       188

    accuracy                         0.7185      2000
   macro avg     0.7160    0.7137    0.7128      2000
weighted avg     0.7186    0.7185    0.7166      2000
```

Figure 9: 3D CNN model accuracy and classification report

The modified CNN report shows that with the exact same configurations, the overall accuracy of the test set has improved.

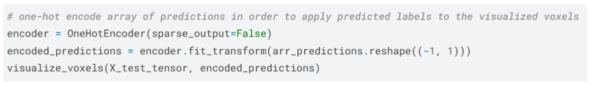


```
CNN Model Accuracy: 74.60%
Classification Report:
              precision    recall  f1-score   support

           0     0.8556    0.9412    0.8964       170
           1     0.9537    0.9802    0.9667       252
           2     0.7178    0.6250    0.6682       232
           3     0.6476    0.6869    0.6667       214
           4     0.7794    0.7227    0.7500       220
           5     0.6250    0.6034    0.6140       174
           6     0.6910    0.7069    0.6989       174
           7     0.7296    0.7222    0.7259       198
           8     0.7151    0.7191    0.7171       178
           9     0.6750    0.7181    0.6959       188

    accuracy                         0.7460      2000
   macro avg     0.7390    0.7426    0.7400      2000
weighted avg     0.7450    0.7460    0.7447      2000
```

Figure 10: MODIFIED 3D CNN model accuracy and classification report

4.2 Testing Labels on Plotted Voxels

To show qualitative results of the CNNs, I called the ’visualize_voxels’ function once again. Except, this time the parameter consists of the predicted labels to label the outputted plots.



```
# one-hot encode array of predictions in order to apply predicted labels to the visualized voxels
encoder = OneHotEncoder(sparse_output=False)
encoded_predictions = encoder.fit_transform(arr_predictions.reshape((-1, 1)))
visualize_voxels(X_test_tensor, encoded_predictions)
```

Figure 11: Calling visualize_voxel() after model epochs have cemepleted

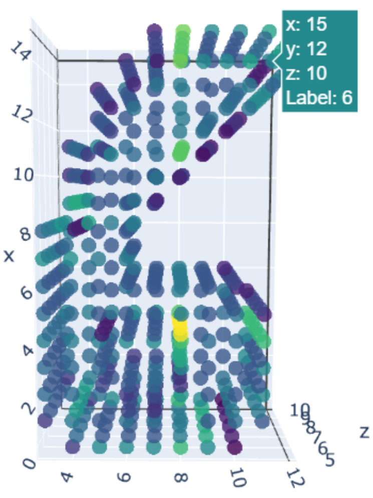


Figure 12: Voxels of the digit '6' labeled correctly by the predicted labels of the trained and fitted CNN model

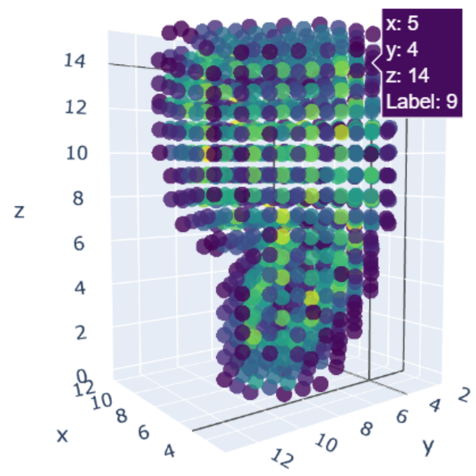


Figure 13: Voxels of the digit '9' labeled correctly by the predicted labels of the trained and fitted MODIFIED CNN model

4.3 Graph of Accuracy Throughout Epochs

These graphs include loss of the training and testing epochs, but it will not be considered an evaluation metric. While loss functions are crucial for training machine learning models, they aren't always the best evaluation metric because they can be scale-dependent and hard to interpret. Loss values can vary widely with different datasets, and low loss values may result from over-fitting, failing to generalize to new data. Additionally, loss functions might not align with the ultimate goals of a task, where application-specific metrics like accuracy, precision, or recall provide more meaningful insights into a model's performance. Instead, the relevant information on Figure 14 and Figure 15 show an increase in test accuracy and then an unfortunate plateau, which means that the model could be over-fitting and further optimizations need to be made.

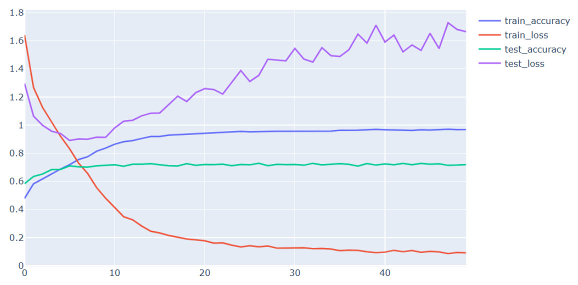


Figure 14: Graph of CNN Accuracy and Loss

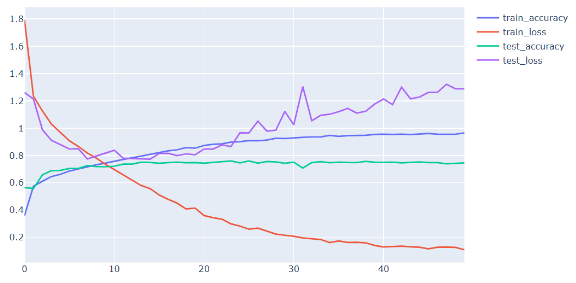


Figure 15: Graph of MODIFIED CNN Accuracy and Loss

5 Conclusion

This study presents a comparative evaluation of three models—LDA, a baseline 3D CNN with batch normalization and ReLU activation, and a modified 3D CNN employing group normalization and LeakyReLU activation—for the classification of voxelized 3D representations of handwritten digits. The results highlight the limitations of LDA in handling complex, high-dimensional voxelized data, as expected. The standard 3D CNN demonstrates significant improvement, leveraging deep learning's ability to extract hierarchical features from 3D data. Notably, the modified 3D CNN shows promising results, suggesting that alternative normalization and activation strategies can enhance performance further.

By maintaining consistent training and testing datasets and evaluating using accuracy, precision, recall, and F1-score, the experiments ensure robust and reliable comparisons. The findings underline the potential of advanced 3D CNN architectures in handling voxelized datasets, paving the way for further exploration of optimized configurations for similar 3D classification tasks.

Limitations

ACL 2023 requires all submissions to have a section titled "Limitations", for discussing the limitations of the paper as a complement to the discussion of strengths in the main text. This section should occur after the conclusion, but before the references. It will not count towards the page limit. The discussion of limitations is mandatory. Papers without a limitation section will be desk-rejected without review.

While we are open to different types of limitations, just mentioning that a set of results have been shown for English only probably does not reflect what we expect. Mentioning that the method works mostly for languages with limited morphology, like English, is a much better alternative. In addition, limitations such as low scalability to long text, the requirement of large GPU resources, or other things that inspire crucial further investigation are welcome.

Ethics Statement

Scientific work published at ACL 2023 must comply with the ACL Ethics Policy.[1] We encourage all authors to include an explicit ethics statement on the broader impact of the work, or other ethical considerations after the conclusion but before the references. The ethics statement will not count toward the page limit (8 pages for long, 4 pages for short papers).

Acknowledgements

This document has been adapted by Jordan Boyd-Graber, Naoaki Okazaki, Anna Rogers from the style files used for earlier ACL, EMNLP and NAACL proceedings, including those for EACL 2023 by Isabelle Augenstein and Andreas Vlachos, EMNLP 2022 by Yue Zhang, Ryan Cotterell and Lea Frermann, ACL 2020 by Steven Bethard, Ryan Cotterell and Rui Yan, ACL 2019 by Douwe Kiela and Ivan Vulić, NAACL 2019 by Stephanie Lukin and Alla Roskovskaya, ACL 2018 by Shay Cohen, Kevin Gimpel, and Wei Lu, NAACL 2018 by Margaret Mitchell and Stephanie Lukin, BibTEX suggestions for (NA)ACL 2017/2018 from Jason Eisner, ACL 2017 by Dan Gildea and Min-Yen Kan, NAACL 2017 by Margaret Mitchell, ACL 2012 by Maggie Li and Michael White, ACL 2010 by Jing-Shin Chang and Philipp Koehn, ACL 2008 by Johanna D. Moore, Simone Teufel, James Allan, and Sadaoki Furui, ACL 2005 by Hwee Tou Ng and Kemal Oflazer, ACL 2002 by Eugene Charniak and Dekang Lin, and earlier ACL and EACL formats written by several people, including John Chen, Henry S. Thompson and Donald Walker. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence* and the *Conference on Computer Vision and Pattern Recognition*.

[1] https://www.aclweb.org/portal/content/acl-code-ethics

References

S. Azizov. 2022. 3d-mnist | basic cnn | adorable visualisations. https://kaggle.com/code/michaelcripman/3d-mnist-basic-cnn-adorable-visualisations.

D. Castro De La Iglesia. 2019. 3d mnist. https://www.kaggle.com/datasets/daavoo/3d-mnist.

L. Chen, S. Li, Q. Bai, J. Yang, S. Jiang, and Y. Miao. 2021. Review of image classification algorithms based on convolutional neural networks. *Remote Sensing*, 13(22):Article 22.

Z. Keita. 2023. An introduction to convolutional neural networks: A comprehensive guide to cnns in deep learning. https://www.datacamp.com/tutorial/introduction-to-convolutional-neural-networks-cnns.

Y. LeCun, C. Cortes, and C. J. C. Burges. n.d. Mnist handwritten digit database. https://yann.lecun.com/exdb/mnist/. Retrieved December 12, 2024.

A. Newatia. 2020. Convolutional neural networks. https://theneuralnetworkblog.wordpress.com/2020/06/15/convolutional-neural-networks/. Accessed December 12, 2024.

OpenAI. 2024. Chatgpt (dec 2024 version). https://chat.openai.com. Accessed December 12, 2024.

A Example Appendix

This is a section in the appendix.