# 3D Convolutional Neural Networks

Sam Berglin
berglin@wisc.edu

Zheming Lian
zlian4@wisc.edu

Jiahui Jiang
jjiang86@wisc.edu

## Abstract

*3-dimensional (3D) data is prevalent in video, vehicle, and medical imaging technologies. Deep learning developed to leverage this 3D data could have far reaching applications, ranging from spatially aware self driving cars to advanced medicine. In this project, we implement 3D Convolutional Neural Networks (CNNs) for classification of an augmented 3D MNIST based dataset. We implement three architectures: a "Traditional" one, "All-Convolutional" one, and "Connectome" one. We developed the Traditional and All-Convolutional architectures from validation on our dataset while we implemented the Connectome architecture according to modern deep learning research. We compare our results to three baseline machine learning methods. All CNNs perform better than baseline machine learning methods. The Traditional architecture is the strongest network, but the All-Convolutional architecture has comparable accuracy. While the Connectome architecture is the weakest among the deep learning methods, it still outperforms the baseline classifiers despite its origin in MRI data.*

*Key words: 3D Convolutional Neural Networks; 3D image classification; Deep Learning*

## 1. Introduction

We implement a 3D convolutional neural network (CNN) for 3D image classification. We discussed 2D CNNs during lecture, but only briefly mentioned 3D CNNs. We recognize that since we live in a technologically advancing 3D world, 3D data will only grow in importance.

Convolutions are filters with learnable parameters that are used to extract low-dimensional features from an input data. 3D CNNs are quite similar to 2D CNNs, except that 3D CNNs apply a 3D filter to the dataset which slides in 3 directions $(x, y, z)$ to calculate the low level feature representations, as in Figure 1. Their output shape is still a 3D volume space, such as cube. Note that there is no dimensional limit to a convolution. Thus, we can apply these convolutions to 3D images with additional color channels.

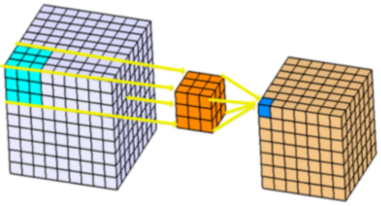


Figure 1. A 3D convolution. Figure from [1].

3D CNNs are commonly used for devices or sensors operating in 3D environments [14] [9]. They are also prevalent in the medical domain where 3D images are acquired through radiographs or magnetic resonance imaging (MRI) [15] [5].

As technology continues to advance at an exponential rate, generation of 3D, rather than 2D, data will rise in tandem. Augmenting deep learning models to leverage this added dimension of information could have far reaching benefits, especially in the fields of medical imaging and computer vision. Knowledge of these models will likely prove useful in our careers in academia or industry.

Since we are applying these networks for educational purposes, we will be using simpler datasets than medical images or temporal video. We consider the 3D MNIST dataset [11]. This dataset contains 3D points clouds generated from the original MNIST images [4]. Each 3D image can take one of 10 classes: $(0, 1, ..., 9)$.

We implement various 3D CNNs architectures and compare their effectiveness. We hope to achieve reasonable accuracies on this 10 class problem, but definitely outperform baseline classification methods. We also aim to develop a thorough understanding of a 3D CNNs as a product of this project.

## 2. Related Work

One of the earliest and most prevalent applications of a 3D CNN involved recognition of human actions in surveillance videos [9]. They treat a continuous motion across 2D frames as a 3D image and classify this concatenated input. This differs from our scenario, as we wish to classify truly 3D data rather than sequential 2D data.

VoxNet [14] is another 3D CNN that is applied to truly 3D data. However, they classify 3D images for real-time object recognition. An important product of their work is the speed of the network; it labels "hundreds of instances per second."

Another network relevant to our study is what we term the "Connectome" architecture [12]. This architecture classified resting-state MRI (rs-fMRI) scans. These scans "hold the potential to serve as a diagnostic or prognostic tool for a wide variety of conditions, such as autism, Alzheimers disease, and stroke" [12]. Their simple CNN architecture (described in Figure 2) takes full advantage of the spatial structure of rs-fMRI data and reports state-of-the-art accuracies on rs-fMRI-based discrimination of autism patients and healthy controls. We compare our tuned architecture against this as a standard deep learning method.

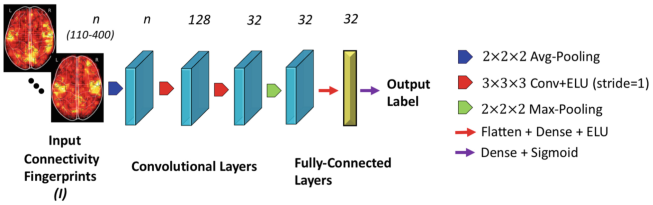


Figure 2. The Connectome architecture [12]. Number of channels is denoted above. The depicted input is for the original paper and we alter it for our dataset.

Our 3D MNIST dataset was created for education purposes. However, prior classification accuracies on this dataset were found in a blog post after all experiments were performed [2]. They report a 77.1% test accuracy. We did not use the blog post for architectural inspiration, but use this as a baseline accuracy. To the best of our knowledge, this is the only other reported classification accuracy on this dataset.

# 3. Proposed Method

We briefly describe the specific, but simple, differences between a 2D CNNs and a 3D CNNs. This includes 3D pooling, dropout, and BatchNorm. Then, we propose the two architectures we developed for our tasks and further explain the Connectome architecture.

## 3.1. 3D Adjustments

Clearly the largest difference between a 2D and 3D CNN involves the convolutions. 2D convolutions pan the 2D kernel in two dimensions, while 3D convolutions pan a 3D kernel in all three dimensions. However, other subtle differences exist.

**3D Pooling** 3D pooling is the most natural extension. The pooling function (a maximum or an average, for example) is applied in a 3D volume instead of a 2D window. This is shown in Figure 3. 3D pooling facilitates a robust model in terms of local invariance, just like pooling for a 2D CNN.

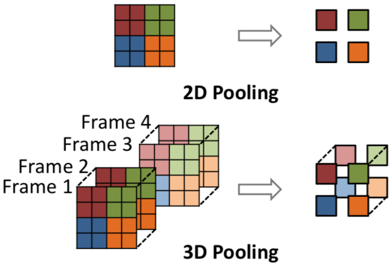


Figure 3. Diagram of 3D pooling with size $2 \times 2 \times 2$ from [6].

**3D Dropout** 3D dropout, like 2D dropout, removes entire feature maps. Only in this case, the dropout is applied to the 3D feature maps instead of the 2D feature maps. 3D dropout prevents overfitting and and forces the network to consider more connections.

**3D BatchNorm** Batch normalization is applied over a mini-batch of 3D inputs with added channels. The normalization for an arbitrary unit can still be described as

$$y = \frac{x - \mathrm{E}(x)}{\sqrt{\mathrm{Var}(x) + \varepsilon}} \times \gamma + \beta \qquad (1)$$

where $\gamma$ and $\beta$ are learnable parameters. The expectation is still taken over the mini-batch. The only difference is that we are normalizing a higher dimensional feature map. Specifically, instead of normalizing a batch of 3D inputs (2D images $\times$ number of channels), we normalize a batch of 4D inputs comprised of 3D images with additional channels.

## 3.2. Traditional Architecture

Our first architecture, which we will call the "Traditional" architecture, is a standard 3D convolutional network composed of several convolutional layers followed by fully connected layers. Its specifics are described in Figure 4.

## 3.3. All-convolutional Architecture

Our second architecture is "All-Convolutional." It is a 3D convolutional network solely composed of convolutional layers. Based on the traditional architecture, the all-convolutional architecture is implemented by replacing maxpooling layers and fully connected layers with 3D convolutional layers.

## 3.4. Connectome Architecture

The Connectome architecture is described in Figure 2. It is somewhat similar to our Traditional architecture, as

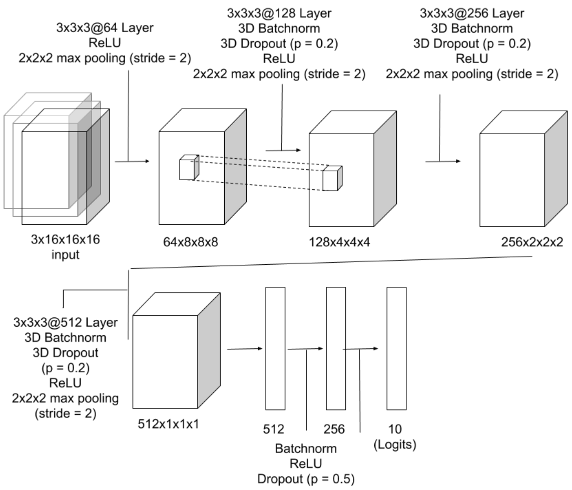


Figure 4. Our traditional architecture.

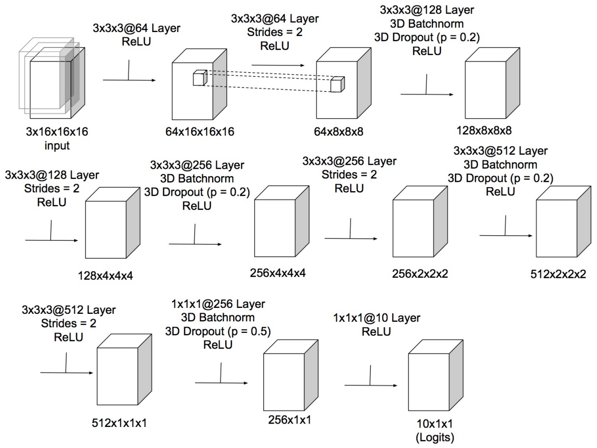


Figure 5. All-convolutional architecture.

it contains two convolutional layers followed by fully connected layers (along with pooling). However, it has fewer units and thus parameters. We use this method as a standard 3D CNN.

### 3.5. Parameters

We summarize the parameter counts between our three deep learning architecture in Table 1.

| Architecture | Parameter Count |
|---|---|
| Traditional | 4,787,210 |
| All-Convolutional | 14,190,302 |
| Connectome | 150,186 |

Table 1. Parameter counts for our three deep learning architectures.

### 3.6. Baseline Machine Learning Methods

To get an idea of some baseline accuracies outside of deep learning, we employ a few simple machine learning algorithms on our dataset and compare the accuracies with our CNNs. We apply regularized logistic regression, random forest, and k Nearest Neighbors (kNN) to our dataset. For all three methods, we use the scikit-learn Python package [16].

**Logistic Regression** We start with a simple yet powerful classifier: regularized logistic regression. We use $L_2$ regularization to handle to the high dimensionality of our 3D data.

**Random Forest** Next we move on to Random Forest. Random forest is an ensemble learning method for classification, regression, and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. We use the default number of trees $(10)$. The performance of decision trees can possibly be improved with hyper-parameter tuning with a grid-search. But for our purposes, we stick to default parameters for an estimate of the classifier performance.

**k Nearest Neighbors (kNN)** kNN is a non-parametric method used for classification and regression. In kNN classification, an object is classified by a majority vote of its "k nearest" neighbors. For example, if $k = 1$, the object is simply assigned to the class of that single nearest neighbor. We tune for $k$ on the validation set and get $k = 5$. An important note is that although the training time for kNN is very short (no parameters to fit), testing takes relatively long since each training example must be compared to each test example to determine the distances.

## 4. Experiments

For all methods, we used separate training, validation, and test sets. Specifically, we use the training dataset to train, the validation dataset to check the quality of the hyperparameters, and the test set once at the end to assess accuracy. All accuracies are reported on the test set. We used a learning rate of $0.001$, a batch size of 128, and 30 epochs for all deep learning models.

### 4.1. Dataset

The dataset we are using is a 3D MNIST dataset from Kaggle [11]. This dataset contains 3D points clouds generated from the original MNIST images with three color channels. Each input can take one of 10 classes $(0, 1, ..., 9)$.

### 4.2. Data Augmentation

The original dataset has 5,000 training instances and 1,000 test instances. Since we are dealing with a large number of parameters, the size of the dataset could become a constraint of model performance. We accordingly create our own 3D point cloud data based on the original instances. We rotate instances by random degrees on three axis $(x, y, z)$. For rotations on x-axis and y-axis, the degrees range from 0 to 90 degrees. For rotations on z-axis, the degrees range from 0 to 180 degrees. Then, the uniform noise is added to the rotated point cloud.

In this way, 20,000 training instances and 2,000 test instances are generated and then added to the original dataset. The experiments of this project is based on this augmented dataset with 25,000 training instances and 3,000 test instances. When we initialize a dataset object, this augmented test set is randomly split into two evenly sized sets: 1,500 for validation and 1,500 for testing. This split of 20,000 training, 1,500 validation, and 1,500 test are the final splits we used. Drawing validation from the original test set is acceptable since there is no difference in the distribution between the original train and original test data; we simply draw from the original "test" set to preserve sufficient data for the final training set.

### 4.3. Software

Google Cloud Platform "Colab" was the GPU resource for this project. Models were implemented in Python with the PyTorch package. We also used the scikit-learn package to implement the baseline machine learning methods.

## 5. Results and Discussion

Applying the baseline and deep learning methods yield Tables 2 and 3, respectively.

| Model | Accuracy | Training Time |
|---|---|---|
| Logistic Regression | 52.6 | 6.44 |
| Random Forest | 52.1 | 0.11 |
| KNN | 55.3 | 0.30 |

Table 2. Baseline accuracies (%) on test set. Training time is in minutes.

| Architecture | Accuracy | Training Time |
|---|---|---|
| Traditional | 79.4 | 13.6 |
| All-Convolutional | 77.0 | 27.9 |
| Connectome | 70.2 | 8.6 |

Table 3. Deep learning accuracies (%) on test set. Training time is in minutes.

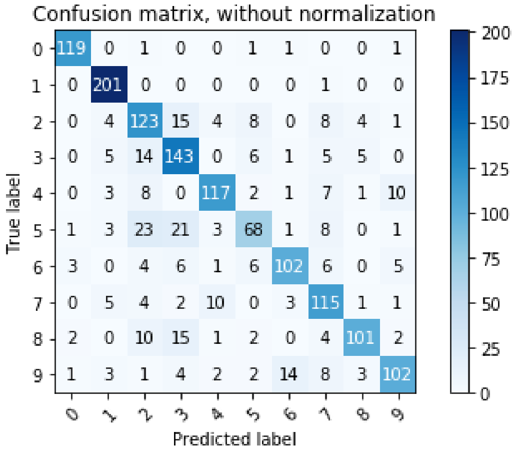
All architectures produce accuracies significantly higher than all three baseline methods. Only the Traditional and one mildly outperforms the reported blog post accuracy of 77.1%. We see that the Traditional architecture has the highest accuracy, while having less parameters than the All-Convolutional network. Perhaps the additional parameters of All-Convolutional network are unnecessary, which leads to the Traditional network having a higher generalization performance. Meanwhile, the parameters of the Connectome network may not provide sufficient capacity to model the classification task; this would explain why the Connectome network has the lowest accuracy.

The Connectome architecture still yields stronger accuracies than traditional machine learning methods. It also is comprised of relatively few parameters. Since we know the architecture was originally designed for MRI data [12], our results suggest this architecture is resilient to the dataset. Future work considering the Connectomes network could continue to suggest it is a strong, general purpose 3D CNN.

During tuning on the validation set, the largest gains came from adding BatchNorm and Dropout. We suspect these regularization techniques forced the network to learn more robust features. A small accuracy boost also resulted from adding image augmentation. With less than 30,000 training instances, this dataset is somewhat small in the context of deep learning. We suspect adding more data would continue to improve generalization performance.

Training time for all deep learning models was relatively brief, all being less than half an hour. We found that training beyond the specified hyperparameters caused overfitting. This is likely a result of the simple architectures we used compared to the large, sophisticated networks that can take hours or days to train.

We also include the confusion matrix in Figure 6 for the strongest classifier, which is our Traditional architecture.



|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 119 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 201 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 4 | 123 | 15 | 4 | 8 | 0 | 8 | 4 | 1 |
| 3 | 0 | 5 | 14 | 143 | 0 | 6 | 1 | 5 | 5 | 0 |
| 4 | 0 | 3 | 8 | 0 | 117 | 2 | 1 | 7 | 1 | 10 |
| 5 | 1 | 3 | 23 | 21 | 3 | 68 | 1 | 8 | 0 | 1 |
| 6 | 3 | 0 | 4 | 6 | 1 | 6 | 102 | 6 | 0 | 5 |
| 7 | 0 | 5 | 4 | 2 | 10 | 0 | 3 | 115 | 1 | 1 |
| 8 | 2 | 0 | 10 | 15 | 1 | 2 | 0 | 4 | 101 | 2 |
| 9 | 1 | 3 | 1 | 4 | 2 | 2 | 14 | 8 | 3 | 102 |

Figure 6. Confusion matrix for the Traditional architecture. The boxes indicate cumulative counts (i.e. without normalization).

The two most common (True, Predicted) confusion pairs are $(5, 2)$ and $(5, 3)$ which occur 23 and 21 times, respectively. Interestingly, these relationships are not symmetric. That is, $(2, 5)$ only occurs eight and $(3, 5)$ only occurs six times. Perhaps this is an indication that our model has overfitting, since an easily confused pair should be mistaken equally often both ways by a human. A less likely explanation is that the dataset simply contains many, for example, 5s that look like 2s compared to 2s that look like 5s.

Our method has several weaknesses. First off, the effect of the amount of training data was not investigated at all. The true accuracy limit of about $80\%$ could be attributed to a lack of data. However, the original MNIST dataset has $70,000$ examples. More 3D data could be generated from these points to ascertain if the limit is truly due to the architecture or the data.

Furthermore, we applied fairly simple CNN architectures since we were new to 3D CNNs. We loosely based our Traditional network from the AlexNet architecture [13]. But since this architecture was originally for 2D images, it may not be entirely appropriate.

# 6. Conclusions

In conclusion, traditional architecture produces the strongest accuracy with a brief training time. We significantly outperformed the baseline machine learning methods as planned, boosting accuracy by over $20\%$ with our custom architectures. We also achieved reasonable classification accuracies, as desired from the outset of the project.

Once again, we discovered work on this same dataset with a 3D CNN in a blog post [2] while drafting this report. While the purpose of the post was to introduce 3D CNNs, our Traditional network did outperform the one in the post that had an accuracy of $77.1\%$. To the best of our knowledge, this is the only other accuracy that has been reported on this dataset.

Most importantly, we developed the thorough understanding of CNNs we desired, especially in 2D and 3D networks. We also developed familiarity with the PyTorch package in the process of this work. PyTorch will be our framework of choice for future problems concerning deep learning.

## 6.1. Future Work

Future extensions of this work span two areas: improving the network and improving data augmentation.

We applied fairly simple CNN architectures in this project. We loosely based our architecture from the AlexNet architecture [13]. However, augmenting a ResNet [8] or VGG-16 architecture [18] for 3D inputs could improve generalization accuracy.

Another interesting avenue is to use a generative adversarial network (GAN) to augment the training data. Perhaps training a GAN to produce realistic images and then adding those images to the training set would improve accuracy beyond standard image augmentation. This idea has been explored for lung nodule detection [7].

Future work regarding the Connectomes architecture could prove fruitful as well. It still drastically outperformed baseline classification methods on this dataset, despite its origin in MRI data [12]. This network could be applied to other datasets to test its status as a robust, general-purpose 3D CNN.

# 7. Acknowledgements

Our most important acknowledgement is for professor Sebastian Raschka. His advice in implementing the All-Convolutional Architecture was invaluable. We further thank Professor Raschka for demonstrating the structure of a custom data loader in PyTorch. We also developed our train-validate-test structure from his base code in Stat 479 Homework 4.

Various other sources were helpful for this project. We thank Jian Shijian for code demonstrating how to load this unique 3D dataset [17]. We also thank Dahuin Jung for an introductory 3D CNN PyTorch implementation [10]. The functions of performing rotation and adding noise are credited to David de la Iglesia Castro [3]. The developers of scikit-learn provided helpful example code for implementing our confusion matrices [16].

# 8. Contributions

Sam Berglin implemented the Traditional and Connectome architectures, as well as the custom data loaders and train-validation-test scheme. Jiahui Jiang implemented the All-Convolutional network and aided data loading along with the Traditional network. She also created visualizations of the misclassified testing instances for the presentation. Zheming Lian implemented data augmentation, all baseline classifiers, and helped with the All-Convolutional architecture. All authors contributed equally to writing this report.

# References

[1] 3d convolutions: Understanding and implementation.

[2] S. Aggarwal. 3d-mnist image classification, 2018.

[3] D. de la Iglesia Castro. Augmenting data, 2017.

[4] L. Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[5] Q. Dou, H. Chen, L. Yu, L. Zhao, J. Qin, D. Wang, V. C. Mok, L. Shi, and P.-A. Heng. Automatic detection of cerebral microbleeds from mr images via 3d convolutional neural networks. *IEEE transactions on medical imaging*, 35(5):1182–1195, 2016.

[6] H. Fan, H.-C. Ng, S. Liu, Z. Que, X. Niu, and W. Luk. Reconfigurable acceleration of 3d-cnns for human action recognition with block floating-point representation. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 287–2877. IEEE, 2018.

[7] C. Gao, S. Clark, J. Furst, and D. Raicu. Augmenting lidc dataset using 3d generative adversarial networks to improve lung nodule detection. In *Medical Imaging 2019: Computer-Aided Diagnosis*, volume 10950, page 109501K. International Society for Optics and Photonics, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[9] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[10] D. Jung. Simple 3d convolutional network in pytorch, 2017.

[11] Kaggle. 3d mnist, 2017.

[12] M. Khosla, K. Jamison, A. Kuceyeski, and M. R. Sabuncu. 3d convolutional neural networks for classification of functional connectomes. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 137–145. Springer, 2018.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[14] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[15] A. Payan and G. Montana. Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks. *arXiv preprint arXiv:1502.02506*, 2015.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] J. Shijian. Loading 3d mnist, 2018.

[18] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.