- MATH 509 Final Project -

Chengao Fu, Jacob Serpico, Tony Yuan

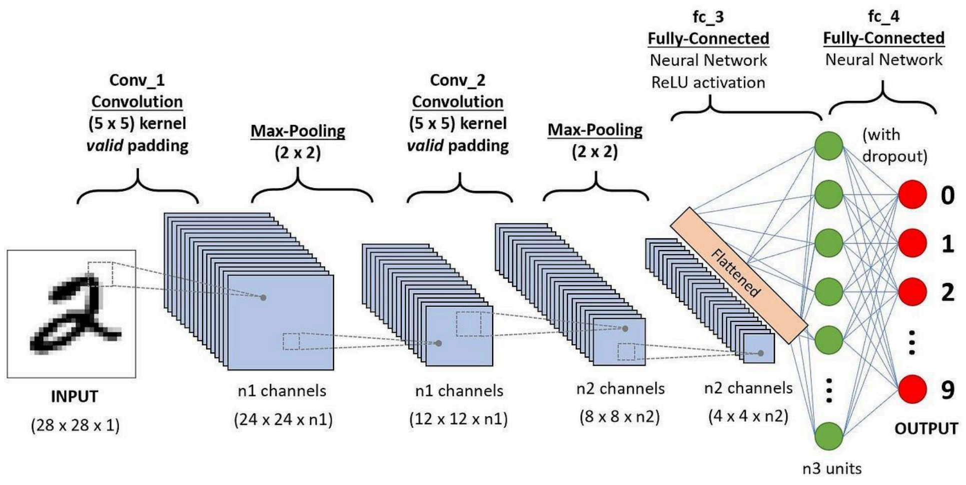
Introduction

Convolutional neural networks (CNNs) use three-dimensional data for image classification and object recognition tasks. More broadly, neural networks are a subset of machine learning, and are central to deep learning algorithms (1). IBM's Deep Blue achieved a brilliant win against then chess world champion Garry Kasparov in 1996, however, processes such as image classification or feature recognition were not reliable with computers until very recently. The latter are arguably simpler tasks for humans to perform than to become the world's best chess player. This fact brings forth a rather interesting question. What do we classify as intelligence? Although computers are now able to solve mathematical olympiad questions more efficiently than humans can, the idea that the solution itself or the way a computer solves the problem is indeed discussion-worthy. To avoid being too philosophical, for the purposes of this project we simply explore what the tasks are at hand and how we utilize machine-learning methods to solve them.

CNNs emerged from the study of the brain's visual cortex, and have been used in image recognition since the 1980s, but due to the substantial increase in computational power and amount of available training data, CNNs have now far surpassed humans on a number of complex visual tasks (2).

Definition 1.1: A *convolution* is an integral that expresses the amount of overlap of one function **g** as it is shifted over another function **f**. There are deep connections to Fourier analysis and is heavily used in signal processing. *Convolutional layers* use cross-correlations, which are very similar to convolutions (3).

Arguably the most important building block of a CNN is the *convolutional layer*, where neurons in the first convolutional layer are generally not connected to every pixel in this input image, but only to the pixels in their receptive fields, in contrast to other similar deep learning structures. Thus, the second layer is connected only to neurons within a small area of the previous first layer. As a result, the network concentrates on small, low-level features in the first hidden layer, then assembles these features into larger, higher-level features in the next hidden layer. For the following layers this process is repeated. Due to real images having this hierarchical structure, CNNs are a very strong tool for image processing. Here is an example of an arbitrary general CNN:



A CNN sequence to classify handwritten digits (4).

The remaining structural components of a CNN are explored in our own formulation and application to our project. The code for all of which is available in (5).

Data Source

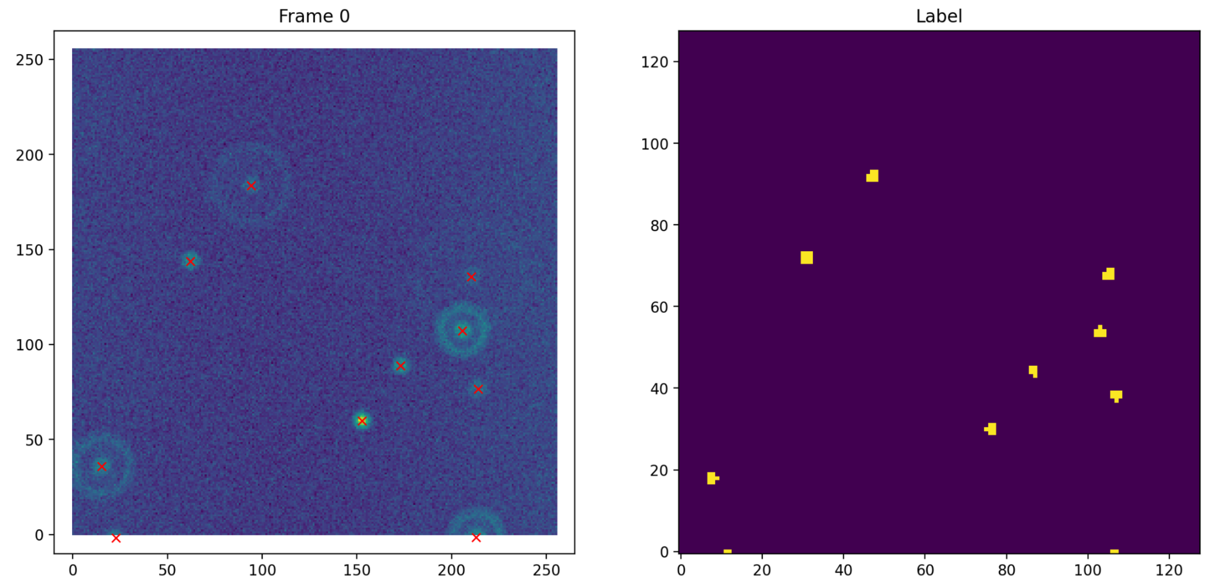
Our training video was generated through code, which is from Dr. Jay Newby's GitHub repository (6).

Objectives

We have three objectives for this project:

1. Use a CNN (Convolutional Neural Network) to identify particles in videos.
2. Use the result from step 1 to get the (x,y) particle positions in the videos.
3. Given a video and (x,y) particle positions, we aim to develop a neural network to estimate the z-position of each particle.

Objective 1



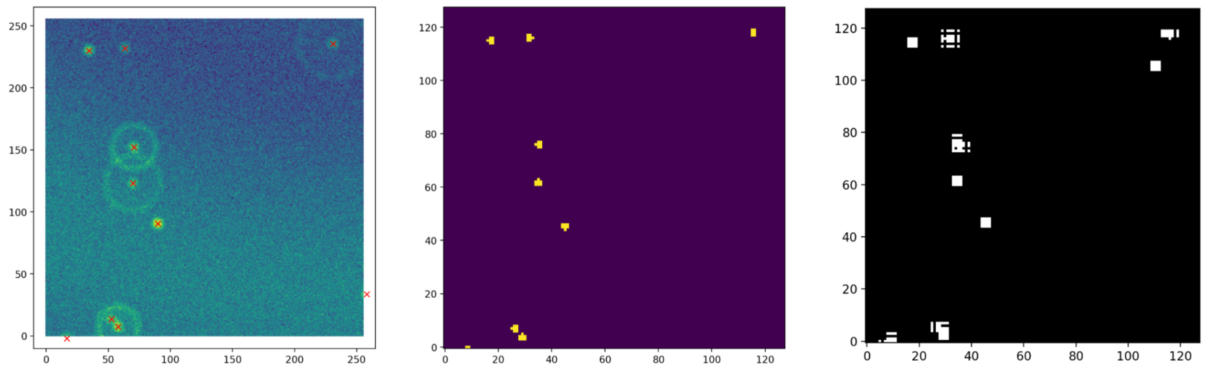
This is a frame from the video we got. The left side shows the image that we can get in reality, while the right side shows the labeled positions of the particles. Our goal is to train a neural network that can find the positions of the particles from the real images. We aim to train a model that, given an image on the left, can generate a result similar to the image on the right.

The image on the left is a 256x256 matrix, where each number represents a corresponding color of that point. The image on the right is a 128x128 matrix, where each data point is either (1, 0) or (0, 1). If the data is (0, 1), it indicates there is a particle, represented by the yellow in the picture. Therefore, the input to our neural network should be a 256x256 matrix, and the output should be a 128x128 matrix (x,y) where numbers represent the probability that a point might class one (1, 0) (Non particle) or class two (0,1) (particle). We set a threshold of 0.5, which means if the y is greater than 0.5 it will be labeled as a particle.

When we began training, we found that no matter what kind of neural network was used, the final output was always something near (1, 0), meaning that not a single point in the entire image was identified as containing a particle. We believe this is because in the right-hand image, the number of points with particles is too small compared to the entire image, less than 1%. This means that even if all of them were classified as not having particles, we could still achieve an accuracy rate of over 99%.

Therefore, we chose to modify the loss function to increase the penalty weight for incorrectly classifying points as not having particles. Specifically, according to the generation function, each particle point occupies approximately 6-7 pixels, and the entire image contains 16,384 pixels. Therefore, by increasing the weight for incorrectly classifying points by 300 times, we can make the ratio of pixels with particle points to those without particle points approximately equal. Consequently, in practice, we have increased the weight for incorrectly classifying points by 300 times.

Our model includes a series of convolutional layers, activation layers, pooling layers and batch normalization which is a common structure for a CNN model. Additionally, we have added some dropout layers to prevent the model from overfitting.



This is the result; the left image is what we can obtain in reality, the middle image shows the actual labeled positions of the particles, and the right image is where our CNN model prediction results.

Next, we need to use clustering methods to find the exact (x,y) positions of the particles based on these potential locations, which is the red crosses in the left image.

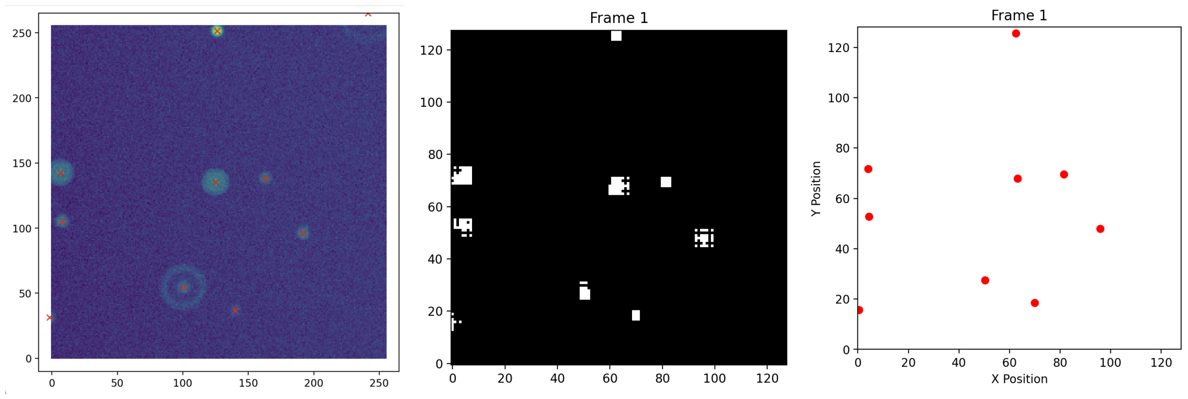
Objective 2

The task of identifying and tracking particle positions in sequential frames poses a significant challenge in the analysis of video data. Clustering algorithms offer a means to automate this process by grouping particle pixels based on spatial proximity. The two principal clustering methods under consideration are DBSCAN and K-means. Each method embodies a unique approach to partitioning the data points and is characterized by distinct advantages and limitations.

DBSCAN Clustering

DBSCAN is a non-parametric algorithm that excels in discovering clusters of arbitrary shapes by connecting regions of high density. Its primary parameters are the radius of the neighborhood (`eps`) and the minimum number of points required to form a dense region (`min_samples`). It begins with an arbitrary starting data point that has not been visited. If this point has sufficiently many points (minPts) within its neighborhood (defined by eps, the maximum distance between two points for one to be considered as in the neighborhood of the other), it forms a cluster. Neighboring points within eps are added to the cluster. This process is recursively repeated for each newly added point, effectively growing the cluster. Points not reachable from any other are marked as noise. Therefore, the flexibility of DBSCAN makes it particularly advantageous when the number of clusters is not known a priori and when the clusters have variable densities. However, selecting appropriate values for `eps` and `min_samples` is non-trivial and requires careful tuning, as they can significantly influence the resulting cluster formation. Moreover, DBSCAN may struggle with data that exhibits wide variations in density or with high-dimensional feature spaces.

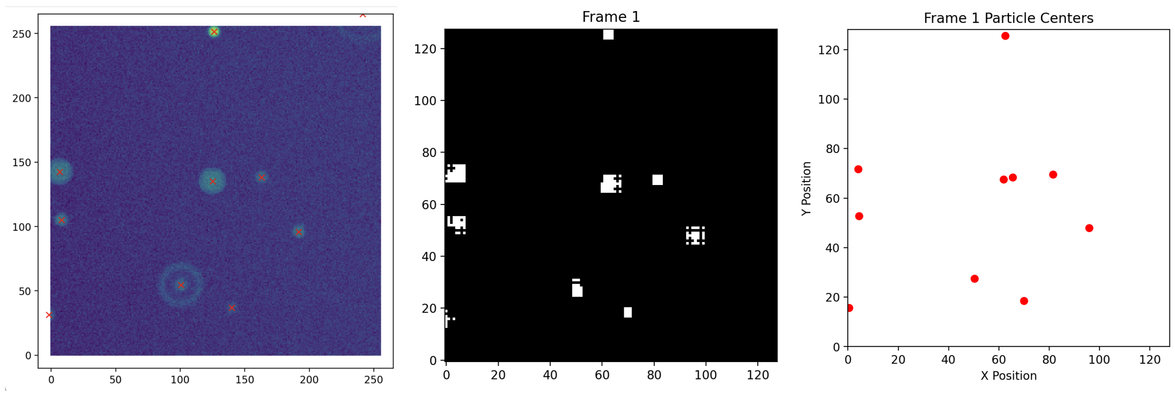
In our empirical findings, DBSCAN configured with parameters `eps=3` and `min_samples=3` yielded the most desirable outcomes. This configuration allowed for the differentiation of closely spaced particles, which became indistinguishable when `eps` was increased beyond 3. An increase in `min_samples` beyond 3 would likely result in a higher threshold for point inclusion in a cluster, potentially leading to the omission of valid particles as noise, especially when particles appear in proximity to one another.



This set of images demonstrates the accuracy of DBSCAN in identifying particle position coordinates.

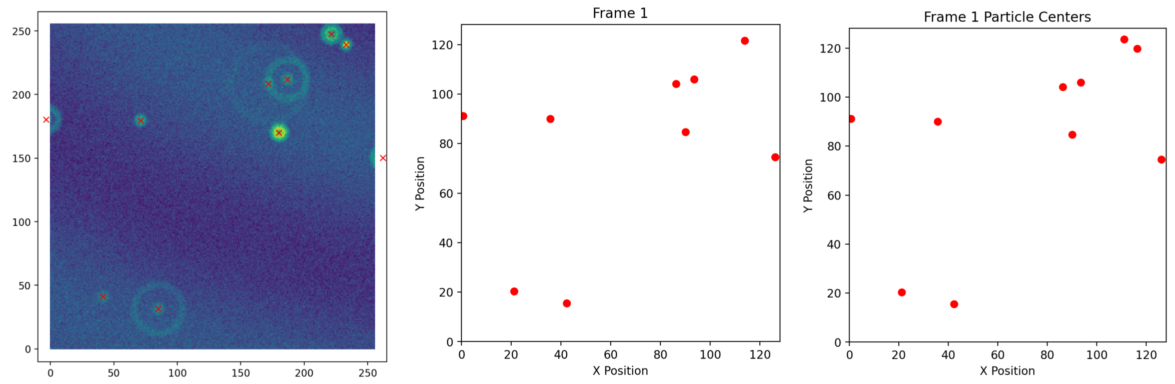
K-means Clustering

In contrast, K-means is a centroid-based algorithm that is simpler and more computationally efficient, especially for large data sets. It is initialized with a specified number of clusters (K) and iteratively refines the centroid locations to minimize within-cluster variance. Although the requirement to predefine the number of clusters may be considered a disadvantage, especially when this information is not readily available, K-means provides a straightforward clustering method that is easy to implement.



This set of images demonstrates the accuracy of K-means in identifying particle position coordinates.

In our project, the number of particles is fixed. The number of particles in the model generated by target one is equal to ten (k =10). The advantage of K-means in this case is that it allows simple and efficient identification of particle centers. This is indeed the case. Comparing the overall performance of DBSCAN and K-means, K-means is more accurate.



The graph group shows the real position of the particles, the position extracted by DBSCAN, and by the K-means algorithm respectively. As can be seen from this figure, in this study, k-means will help us extract more accurate particle positions when the number of clusters (k) is known.

<u>Comparative Evaluation</u>

Comparatively, DBSCAN is inherently suited for particle tracking as it dynamically adjusts to the number of particles and their movement across frames without the need for predefined cluster numbers. It accommodates spatial groupings of particles and can effectively distinguish between particles and background noise. K-means, while faster and more efficient for large-scale data, is limited by its assumption of cluster shapes and the requirement to set the number of clusters in advance. Therefore, when particle distributions are expected to be non-uniform, or when particle numbers are not consistent, DBSCAN is the preferable choice. Conversely, for uniform particle distributions with consistent particle numbers, K-means may offer a computationally less intensive alternative with satisfactory results.

Although DBSCAN performed well, K-means was arguably the more promising option for using a clustering algorithm to pinpoint the (x,y) particle positions. In our tests, K-means was able to help detect each particle's position with incredible accuracy, with the exception of a few anecdotal examples where a particle was partially located on the boundary of the region. However, a drawback is that you need to first specify how many particles there are in the domain for K-means to properly work. In a practical situation, there may be a significant number of particles to detect in a frame and can render manual counting tedious, if not impossible. A potential objective to pursue would be a way to automatically determine the number of particles in the domain to then feed to the K-means algorithm.

Considering how we have chosen to solve the problem using a clustering algorithm, the possibility of determining the number of particles prior automatically creates somewhat of a circular argument. We cannot detect the particles without the algorithm, but we cannot use the algorithm unless we know how many particles we have. Therefore, it is reasonable to suggest we attempt to find another way of classifying and locating the different particles, or improve our current usage of clustering algorithms such as DBSCAN.

Objective 3

For our third goal, we aimed to develop a neural network to estimate the z-position of a particle given a synthetic video dataset. Building upon our first objective, we incorporate modifications to the model's structure, parameters, and output. We begin by generating synthetic videos, simulating particles in a three-dimensional space. The particles appear as small circular regions with associated rings. To train our model, we extract patches in a neighborhood of the recorded coordinates of each particle. This technique aims to focus the model's attention on relevant features by reducing input dimensionality. The model itself in the code titled "title", linked "here".

We have two sets of convolutional layers each with 32 filters, followed by two with 64 filters and finally two with 128, all with a (3,3) kernel. These components are responsible for feature extraction. We included batch normalization after each convolution with the goal of increased learning stability and improved convergence speed. We incorporated Max Pooling with a (2,2) window to reduce spatial dimensions. Following the convolutional layers we added dropout to mitigate overfitting, dense layers to process extracted features to make predictions, and an output layer with a linear activation function to output our predicted z-coordinate. The model architecture was informed by "Hands-On Machine Learning with Scikit-Learn and TensorFlow". The model is tailored to address a rigorous task of spacial interpretation but visually less complex.

Discussion

A sample of the results we achieved are given as;

Actual Z: -12.790748596191406, **Predicted Z:** 1.2263526916503906

Actual Z: -26.982437133789062, **Predicted Z:** -32.5587158203125

Actual Z: 2.0883445739746094, **Predicted Z**: 2.986769437789917

Actual Z: 18.8074951171875, **Predicted Z:** 25.376373291015625

Actual Z: -10.088394165039062, **Predicted Z:** -13.981210708618164

Actual Z: -25.8587646484375, **Predicted Z:** -30.154020309448242

Actual Z: -16.196868896484375, **Predicted Z:** -25.22083854675293

Actual Z: 0.14545249938964844, **Predicted Z:** 1.5160222053527832

Actual Z: -8.696556091308594, **Predicted Z:** -14.095193862915039

Actual Z: 2.9379539489746094, **Predicted Z:** -26.33544921875

Additionally, there is a file in our GitHub repository with the full results. Below are the epoch results, and the plots of the validation and training loss;

```
Epoch 1/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 102ms/step - loss: 191.1093 -
val_loss: 9685.9863

Epoch 2/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - loss: 88.4490 -
val_loss: 18364.1406

Epoch 3/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step - loss: 55.1310 -
val_loss: 10204.7207

Epoch 4/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - loss: 41.5135 -
val_loss: 13834.3340

Epoch 5/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 111ms/step - loss: 42.0242 -
val_loss: 7055.5669

Epoch 6/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 116ms/step - loss: 32.1097 -
val_loss: 1989.5840

Epoch 7/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 101ms/step - loss: 28.8795 -
val_loss: 2742.1799

Epoch 8/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 99ms/step - loss: 44.4788 -
```

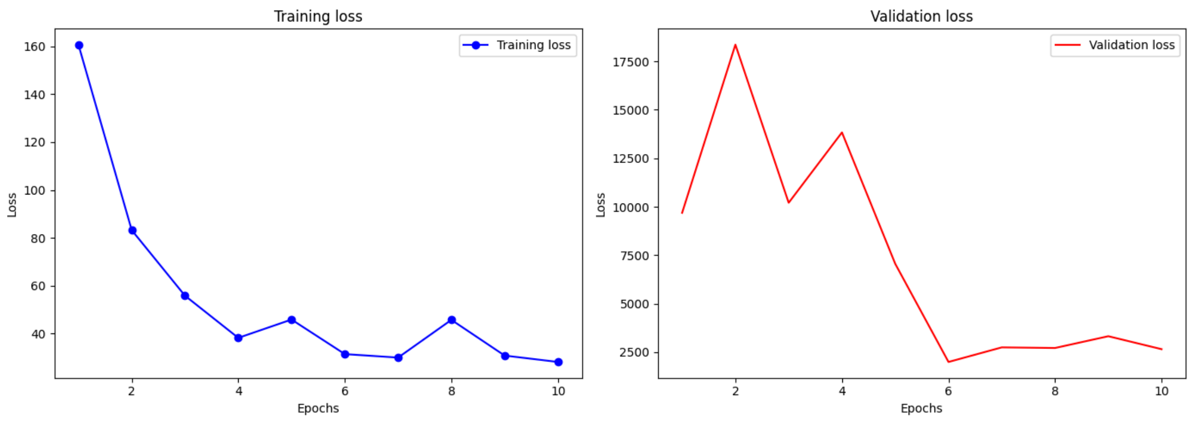
```
val_loss: 2710.1826

Epoch 9/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 97ms/step - loss: 33.2455 -
val_loss: 3317.6313

Epoch 10/10

13/13 ━━━━━━━━━━━━━━━━━━━━ 1s 98ms/step - loss: 23.9548 -
val_loss: 2648.8010
```



The task of extracting and interpreting spatial features from two-dimensional representations of three-dimensional movements is challenging. Despite the model being somewhat standard for broad image recognition tasks, there are a number of adjustments that we would explore if we were given more time.

Model Architecture: We could attempt to change the number of filters in the convolutional layers, size of the kernels, and the number of neurons in the dense layers.

Parameter Optimization: Incorporate techniques such as grid search or random search to explore different combinations of parameters such as learning rate, batch size, and epoch number.

Regularization: To combat overfitting, we could modify the dropout rates.

Visualization and Step Verification: Incorporate a variety of visualization tools and logging the training and validation metrics could provide insight into where we should make improvements to our model.

The main limitation for this project is time to allocate to actually working on it. Due to the sheer number of tasks to complete this semester across a number of dedications. Given more time and a higher level of competency with machine learning, specifically constructing and modifying deep learning networks, I believe I could produce accurate and meaningful results for the prediction of the z-coordinate.

Discussion

Convolutional neural networks (CNNs) are a powerful tool used in both industry and academia for a number of purposes, such as image processing, detection, and generation, video analytics, and natural language processing (7). In this project, we attempted to generate synthetic videos, detect particles, identify their positions, and further predict three-dimensional coordinates based on a two-dimensional image. We utilized a formulation of a standard model for image feature detection to begin modifying to fit our needs. We then attempted to use clustering algorithms in order to identify and predict the locations of particles. Finally, we began to modify our existing network in an attempt to predict the z-coordinate of each particle, where we unfortunately failed to maintain a high degree of accuracy due to the inherent challenge of the problem and time permitted to address it.

For further exploration, we would like to incorporate a method that can determine the number of particles in a given frame ahead of time, so that we do not need to manually specify this value to the K-means algorithm, or to try another method of detection and prediction. Additionally, for objective 3 we would like to tweak the model's architecture, parameters, and structure in order to create a better prediction of the z-coordinate based on a two-dimensional image.

References

(1) https://www.ibm.com/topics/convolutional-neural-networks

(2) https://powerunit-ju.com/wp-content/uploads/2021/04/Aurelien-Geron-Hands-On-Machine-Learning-with-Scikit-Learn-Keras-and-Tensorflow_-Concepts-Tools-and-Techniques-to-Build-Intelligent-Systems-OReilly-Media-2019.pdf

(3) https://en.wikipedia.org/wiki/Convolution

(4) https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

(5) https://github.com/jacobserpico/MATH-509-Final-Project

(6) https://github.com/newby-jay/SSC_Workshop_2021/tree/main/Project%20NeuralNet

(7) https://www.xenonstack.com/blog/convolutional-neural-network