Particle Detection in microscopy videos based on CNN

Jialin He, Yuanzhe Zhang, Dennis Kwong

April 16, 2024

Abstract

This is the final report for our MATH 509 group project. When given a microscopic video, true particles appear white while empty spaces appear black. Through thresholding in OpenCV, it is possible to roughly locate the locations of the particles. However, such approach does not suffice due to varying particle size, the existence of "false" particles, and does not give insights on particle movement tracking. Our group have thus used a machine learning approach - CNN to deal with the task, in addition to some other code snippets that could deal with the above issues.

Contents

1 Introduction

With better technology, microscope has higher magnifications and thus are able to record the movement of microscopic-sized particles. However, even with highly advanced microscopes, the issue of poor signal-to-noise ratios (i.e. high noise) could not be alleviated. Moreover, how to post-process microscopic videos and output the positions of particles on screen remains a hot topic in the machine learning and applied science community.

Our project is divided into two parts. First, before analyzing any microscopic videos, we have to find ways to turn mp4 videos into an array of data. This is done by OpenCV. In addition, after extracting the videos, we have attempted to perform basic analysis on the videos (i.e. part of Project Salmonella).

Second, we have generated training data via a python class. Since there are only limited number of microscope recordings, it is not realistic to train the neural network with those recordings. The python class generates video that roughly resembles a microscopic recording, with parameters including noise level, standard deviation of Brownian motion and particle size. The training data is then fed to train the CNN, which will later be used for particle detection (i.e. part of Project NeuralNet).

2 Data processing

In Project NeuralNet, no data was provided. In Project Salmonella, two types of data - tracks and videos were given. 'Videos' data are mp4 videos captured by a microscope. The videos are 10 seconds long. 'Tracks' data are csv files containing the (x,y) coordinates of each particle in the videos at frame $t \in \{0,1,\ldots,159\}$.

particle	x	y	frame t
0	427.158	69.356	0
0	427.128	69.407	1
0	427.823	69.605	2
⋮	⋮	⋮	⋮

Table 1: Data structure of 'Tracks' dataset

2.1 OpenCV: A thresholding approach & limitations

Code for this part can be found in `ProjectNeuralNet/opencv_experiment.ipynb`.

Microscopy videos are stored in mp4 format, in which its pixel data could be translated and stored in an array by the help of OpenCV. By the *cvtColor* method, a colorized video is turned interpreted as a grayscale video, and then converted into a matrix of values ranging from 0 to 255, representing each pixel's light intensity (0 being black, 255 being white).

OpenCV also provides a *threshold* method. It stores all pixels with white intensity higher than the threshold to be 1, and others to be 0.

$$\text{output pixel value} = \begin{cases} 1 & \text{if pixel intensity} \geq \text{threshold} \\ 0 & \text{if pixel intensity} < \text{threshold} \end{cases}$$

Intuitively, in the video, particles appear white while empty space appears black. So upon setting a proper threshold, it is theoretically possible to detect which part of the image corresponds to particles, while which part is not. Upon testing this idea on a microscopic video, a major issue could be spotted.

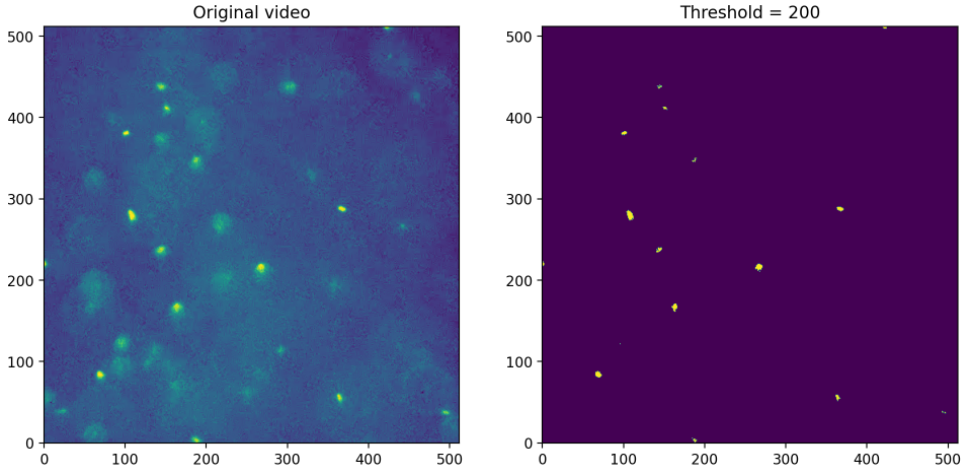

Figure 1: Frame $t = 0$, Threshold $= 200$

The above shows the problem of over-filtering. Setting the threshold too high eliminates potential white spots, which could represent particles.

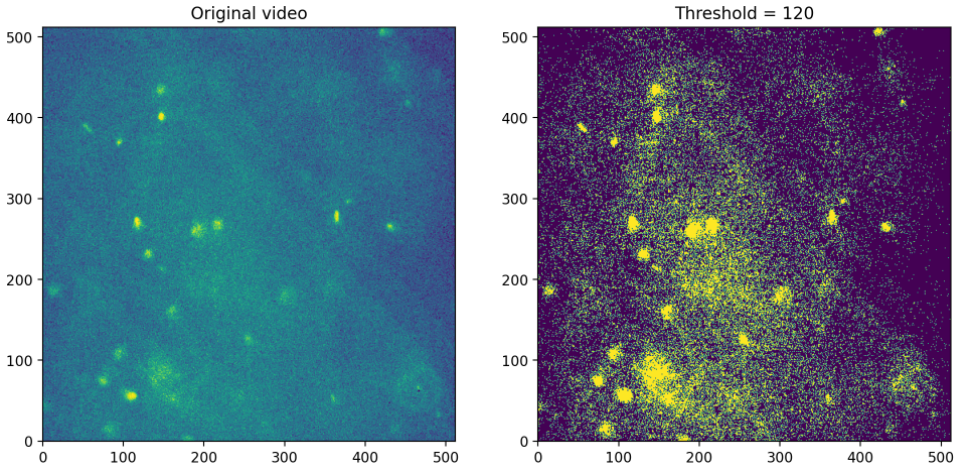

Figure 2: Frame $t = 135$, Threshold $= 120$

On the other hand, setting the threshold too low would cause the map to be "contaminated" by noise. In frames with very high noise, the noise exceeds the threshold value, and thus gets a

value of 1 in the map. This experiment gives us insights on why more advanced tools, such as neural networks, should be used instead.

2.2 Interpretation of videos

Code for this part can be found in `ProjectSalmonella/ProjectSalmonella.ipynb`, Goal 1.

We were provided with 27 recorded videos in Project Salmonella. Before diving into the Neural Network part of this project, we tried to get some intuitions as to how to analyze particle paths. With the given data, we are able to estimate the average speed of the particles. Although this is not directly related to the CNN part of the project, we figured it would be helpful to think of particle tracking in a particle velocity point of view. This could be a potential direction for future discoveries.

The first method of estimating particle velocity is to simply take the mean of the velocities in each subsequent frames. It was given that $dx = dy = 0.156\mu m$, suppose the particle moved for (x,y) pixels for each direction between two frames, we would have distance travelled d to be

$$d = \sqrt{(x \cdot dx)^2 + (y \cdot dx)^2}$$

We take the mean of all distance travelled between frames, divided by $dt = 0.0667$ to get an estimated swim speed of $7.22\mu m/s$.

3 Mathematical Model

3.1 Generating Training Data

To apply the CNN model for particle detection, we will be using a Python class to generate training data. Without prior knowledge of the way the particles move, it is assumed that particles move in a random Brownian motion. Besides, to simulate the background noises in the microscopic videos, a standard deviation parameter of background noise is added to control the strength of the noise.

To reduce the trouble of not knowing how many particles are on the screen, a parameter 'Nparticles' is used to define the number of particles. Each particle have different sizes and z-value, which sometimes cause there to be a ring surrounding the particle. This can also be adjusted. (see Table 2)

Variable	Description
Nt	Number of frames (video)
a	Spot radius scale factor (1.5-4)
kappa	Noise level (0.1)
Nparticles	Number of particles in the video
Ibacklevel	Intensity level of the random background relative to maximum (0-1)
sigma_motion	Standard deviation of random Brownian motion per video frame

Table 2: Input parameters

The method '_sample_motion' specifies how the Brownian motion works.

1. Define the lower and upper boundaries of meshgrid:

$$b_{\text{lower}} = [-10, -10, -30], b_{\text{upper}} = [Nx + 10, Ny + 10, 30]$$

2. Initial positions determined by uniform distribution $U(0,1)$: $X_0 = b_{\text{lower}} + (b_{\text{upper}} - b_{\text{lower}}) \times U$ for all 3 coordinates.

3. Normal distribution with s.d. σ_{motion} is used to determine dX, which is the normal increment changes in particle positions.

4. With different dX per iteration in step 3, X would store the final position, which is obtained after an unbounded Brownian motion has been performed.

5. Reflected Brownian Motion: If the particle goes outside of the boundary, it is reflected back into the defined lower and upper bounds by

$$X = |X - b_{\text{lower}}| + b_{\text{lower}} \text{ and } X = |b_{\text{upper}} - X| + b_{\text{upper}}$$

3.2 CNN layers

Code for this part can be found in `ProjectNeuralNet/Training_Data/finalized_code.ipynb`.

Neural networks are mathematical models that mimics how biological neural networks, such as the human brain works. A particular subclass of Neural Network, Convolutional Neural Network (CNN), was chosen due to its popularity and ability for downsampling, feature extraction and classification.

The CNN architecture we have chosen mainly relies on 3 types of layers: Convolutional layer, Pooling layer and Fully Connected layer.

Layer (type)	Output Shape	Param #
conv2D	(256, 256, 32)	320
MaxPooling2D	(128, 128, 32)	0
conv2D	(128, 128, 64)	18,496
conv2D	(256, 256, 32)	1,154
dense	(128, 128, 128)	384
dense	(128, 128, 2)	258

Table 3: Layers of CNN

3.2.1 Convolutional Layer

Convolutional layer is the bread-and-butter layer for any CNN architectures. Its purpose is to perform feature extraction and localization. In each frame, an image of 256×256 is fed into the network. It would take too many hidden parameters to do a dense layer directly from the bitmap image, since the complexity is unreasonably high. Besides, there is not much disadvantageous by using a 'blurred' image. In our attempts, we used multiple convolutional layers, and each of them has a size 3×3 filters.

3.2.2 Pooling Layer

We added a 2×2 max pooling layer to compress the size of frame from 256×256 to 128×128. A max pooling layer simply picks the largest value within a 2×2 grid as the new output. We expect the largest values picked would retain the most prominent output, and thus would not result in major data loss. It also has a 'stride' parameter which specifies the offset of squares to be picked.

3.2.3 Fully Connected Layer

Fully connected layers are added at the end of the architecture. It is used mainly for classification and output. We have used ReLU function and Softmax function for the last two dense layers respectively.

4 Solution of the problem

Code for this part can be found in `ProjectNeuralNet/Training_Data/finalized_code.ipynb`.

To reiterate, we have chosen to use a CNN model to identify the positions of particles. Below states comprehensively how the CNN model is trained.

4.1 Training of CNN-model

First, we constructed CNN model with the aforementioned architecture, then we use the *model.fit* function in TensorFlow to train it. Here are the parameters for the training.

Items	**Choices**
Optimizer	Adam
Loss function	Sparse categorical cross-entropy loss
Epoch	50
Final loss	0.0079
Final accuracy	0.9967

Table 4: CNN Training Details

4.2 Extraction of particle positions

4.2.1 Filtering processing for CNN output

The output of our CNN model is a 2-dimensional image of probabilities. However, the point of interest are the potential locations of particles. Thus, to determine which area of the bitmap we are interested in, we naturally think about filtering. This is because after CNN processing, the bitmap has a high difference in brightness value between the bright points and the dark spots (empty space).

Notice that although we are using thresholding as well (which was mentioned to be incapable of handling noise and weak signals), our thresholding is not done on the original video frames, but on the output of the CNN model.

To make our filter more accurate, we first normalized the probability image. The highest probability within all pixels might not be 1, we would expect it to be somewhere around 0.9. We would then use this probability to normalize the probability of every pixel such that it forces the highest probability in the bitmap to be 1.

$$\text{Normalized Probability} = \frac{\text{Pixel Probability}}{\text{Highest Probability in Bitmap}}$$

Then, we will use a threshold function, where every points with probability greater than or equal to 0.7 will round up to a 1, and others regarded as 0. This results in a matrix with entries 0 and 1, allowing easier decision-making.

$$M_{i,j} = \begin{cases} 1 & \text{if normalized probability of pixel } (i,j) \geq 0.7 \\ 0 & \text{else} \end{cases}$$

The threshold value 0.7 was a logical guess. For future projects, it could be an idea to use grid search to tune such hyperparameters. But for our purpose, 0.7 had given high accuracy scores, and thus we would be using this value throughout the project.

4.2.2 Selection of half-isolated points (Overview)

From the previous filtering step, we have simplified a bitmap of probabilities to a 0-1 bitmap. However, if we simply treat points with label 1 as particles, we would have had way more than 10 particles, which is the pre-defined number of particles on screen.

The reason for the excess amount of 1s is because of the size of particles. A particle need not be exactly 1 pixel large. Therefore, we have to find some features and get an idea of how to determine what "isolated patches" and "center of patches" are.

- **Feature 1 (Isolated points)**: Suppose the current pixel i has label '1', it is isolated if there are no other pixels of label '1's next to it.
- **Feature 2 (Neighbours)**: 2 points x,y are neighbours if dist(x,y) $< r$, where r is a predefined radius value.
- **Feature 3 (path-connected points)**: 2 points x,y are path-connected if there exists a path of pixels, all with label '1', connecting the 2 points.

If we could remove all path-connected points and neighbourhoods, this could greatly reduce the list of potential particles (optimally 10 so that it matches the given data).

4.2.3 Selection of half-isolated points (Removing Path-connected Points)

In this subsection, we explain our algorithm to remove *path-connected points* first.

Initial Algorithm:

1. Set list particle = []
2. Enumerate all pairs of points labelled 1, denoted as i, j:
   - if particle = [], append i into it.
   - otherwise, between path of i, j, if there exists a 0 in all paths (i.e. i, j are isolated points), then append i to the list.

In short, we try to identify points that are isolated, and then ignore points that are connected to points in the *particle* array. This process reduces local patches into a singular point.

The algorithm certainly makes sense, but the complexity is very high. To iterate between pairs of points is $O(n^2)$, but to iterate all paths between them would further increase the complexity. For example, imagine a 3×3 rectangle where point i, j are at two opposite corners, then imagine a 15×15 rectangle with similar setting. The number of paths between i and j increases geometrically.

Therefore, we will modify part of the iteration part. We will be searching *along the boundary of the rectangle* between i and j to reduce to 2 routes from i to j. If 0 exists in any of the paths, they are half-isolated (for now, which will be further filtered in the next step).

Definition (set of half-isolated points): A set A stores half-isolated points if for any two points $i \neq j \in A$, draw a rectangle between them (or a straight line if they have identical x or y values), and there exists a 0-indexed point in both paths.

Updated Algorithm:

1. Set list particle = []
2. Enumerate all pairs of points labelled 1, denoted as i, j:
   - if particle = [], append i into it.
   - otherwise, Within the **2 boundary paths** from i to j, if there exists a 0 in both paths, then append i to the list.

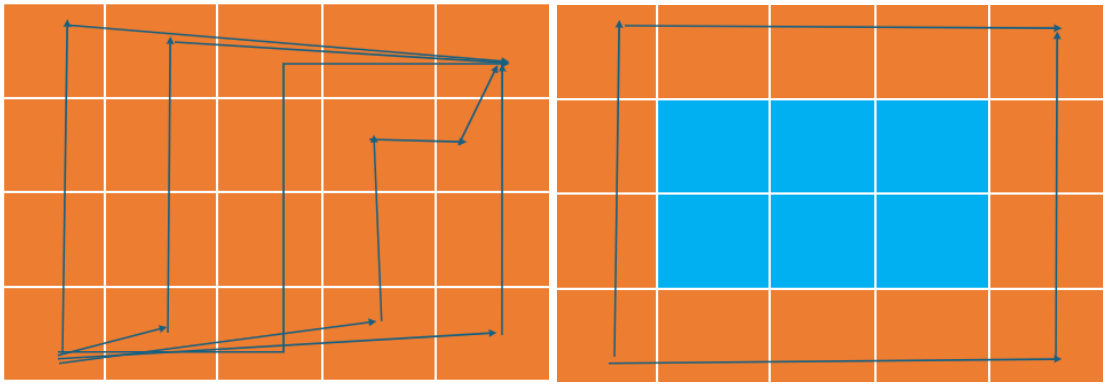

Figure 3: Original Algorithm: search along all paths (left) and
Updated algorithm: Search along the boundary (right)

This method neglects some of the connected points, but will be effective in filtering some of them out. The complexity would definitely be $O(n^2)$, while the previous could be way over $O(n^3)$ due to the path-finding part.

4.2.4 Selection of isolated points (Neighbourhoods)

This part is quite intuitive. We will be filtering points according to the Manhattan distance. We have a set a filtering radius $r = 10$, which means points that are at most 10-pixels away from each other will be removed.

1. Set radius $r = 10$.
2. Enumerate for each point i in the half-isolated list of points A.
    - Pick another point j (which was not chosen as i before).
    - If dist$(i,j) < 10$, remove point j.

where the distance function is just:

$$\text{dist}(i,j) = |i_x - j_x| + |i_y - j_y|$$

After this step, all neighbouring particles would been filtered out, with points remaining in the list representing a representative coordinate for the whole patch of white spots.

5 Results interpretation

5.1 Effect display

First, we take one set of generated tracks as our sample to show the result of the described algorithm. We first pass the input image to the CNN to get an output.

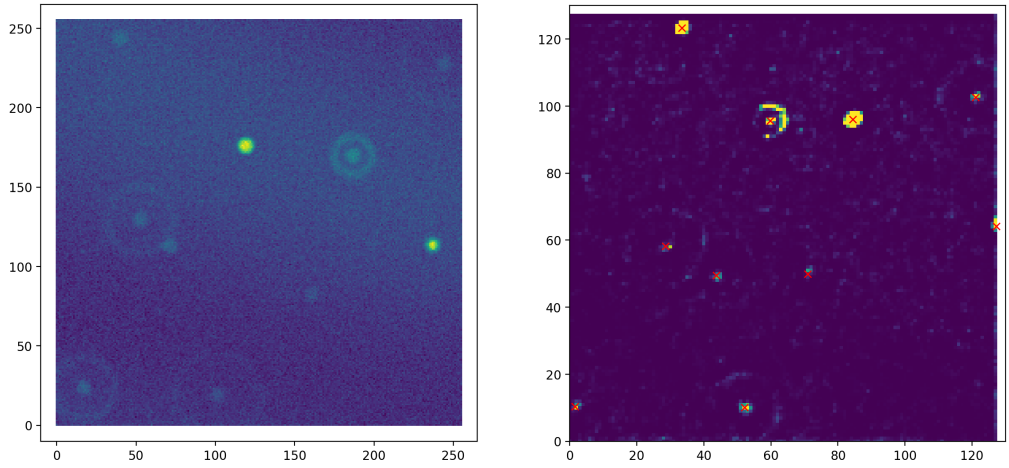

Figure 4: Input image (left), CNN model output (right)

The red crosses in the image represent the positions of 10 simulated particles. The CNN output has a more yellowish color near the red crosses, which means the probability values assigned are high. This shows that the CNN is effective in capturing the potential particle positions.

Second, we normalize the probabilities, and use a 0.7 threshold to produce a 0,1-bitmap. This simplifies the image.

After thresholding, the points with high probabilities are mostly kept. However, you would also notice that some circles around the center of the particle remains (figure 5, left). Those are not new particles but indeed part of a particle that spans more than 1 pixel.

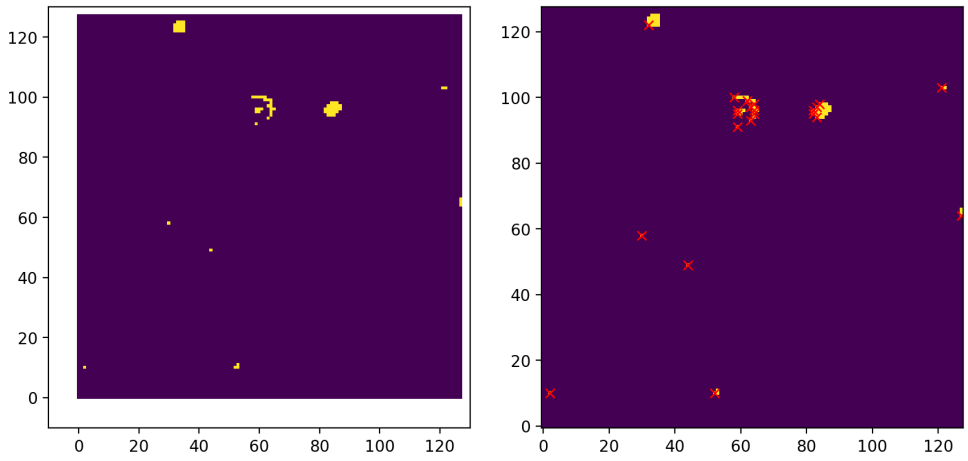

Figure 5: Image after 0.7 threshold (left), Half-isolated points (right)

We enumerate all pairs of points, and keep the half-isolated points only (figure 5, right). Some of the neighbourhood points remains due to the pairs being path-disconnected, but it reduces the number of points in the set with low complexity. For a high number of frames to process, this is a sacrifice we need to take to reduce the complexity of our algorithm.

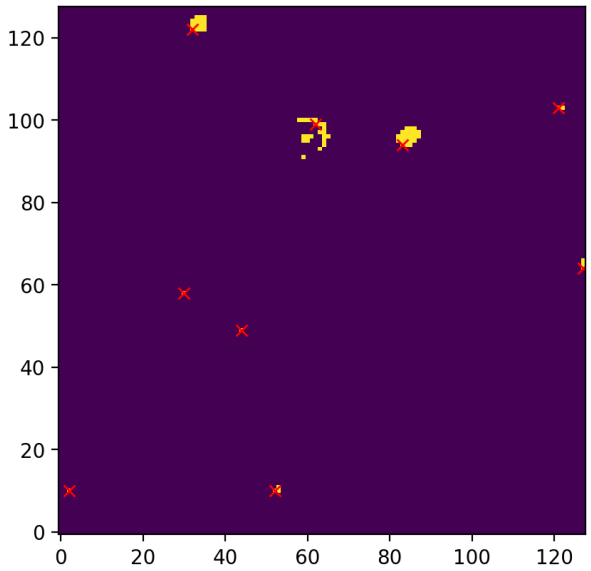

Figure 6: Finalized Particles after distance filtering

Finally, we use distance filtering to get rid of neighbourhood points. The red crosses in the image corresponds to the deduced particle locations, which visually looks quite accurate. However, we do need some proper validation metrics to make persuasive claims that the model is accurate.

5.2 Validation Score

We use an F1-score system to evaluate our model. Calculating F1-score requires us to first accumulate a count of True Positives, True Negatives, False Positives and False Negatives. We denote these values by TP, TN, FP, FN.

- **TP**: Label 'True', recognized as 'True'
- **FP**: Label 'False', recognized as 'True'
- **FN**: Label 'True', recognized as 'False'
- **TN**: Label 'False', recognized as 'False'

We want TP and TN to be high, while FP and FN to be minimal. Using the 4 values, we could then obtain 3 important metrics: True positive rate (TPR), False positive rate (FPR) and Accuracy.

- $\text{TPR} = \frac{\text{TP}}{\text{TP+FN}}$
- $\text{FPR} = \frac{\text{FP}}{\text{FP+TN}}$
- $\text{Acc} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}}$

We have fed our model with a test set, and got the following metric:

Score	**Value**
TPR	0.9188
FPR	0.0609
Acc	0.8580

Table 5: Validation Score

The metrics has shown that our detection algorithm did a decent job.

6 Critique of the model

6.1 About Goal 3

We have similar issues with another group - not being able to get something useful in Goal 3. We will discuss some difficulties when dealing with goal 3.

> **Goal 3**: Given a video and particle positions (from the training data generator) develop a neural network to estimate the z-position of each particle

In the video, it is easy to interpret what the (x,y)-coordinates of the particles are. Wherever the white spots are, those are the potential particle locations. z-coordinates on the other hand are not easy to interpret. From speculation, it is possible that the *size of the ring* around the particle and *particle size* could relate to the corresponding z-position.

If such speculation holds true, then there would be difficulties defining the convolution layers. In our model, we have used conv2D layers with 3×3 filters. If we increase the filter size, we might accidentally removed 'rings around particle' due to averaging. If we decrease the filter size, we are no longer doing CNN.

Therefore, we tried to simply define a neural network with plenty of dense layers, but the output was strange.

Layer (type)	**Output Shape**	**Param #**
input	(10,3)	/
dense	(10,128)	512
flatten	(1280)	0
dense	(128)	163968
dense	(64)	8256
dense	(1)	65

Table 6: Goal 3 trial

6.2 Future developments

We have actually tried another Neural Network architecture. Instead of just CNN and dense layers, it makes sense to use some sort of Recurrent neural network (RNN), such as Long-Short Term Memory (LSTM). RNN is widely used for processing sequential data, and the video we are working with just happens to be a sequence of bitmaps. LSTM in particular excels in capturing long-term dependencies, maintaining long-term memory while able to selectively forget information over time as well.

In our first attempt, we added a bidirectional LSTM layer to the neural network. The output did not turn out to be valid, could be because of reshaping issues, but we are certain that more could be done in this direction.

Layer (type)	Output Shape	Param #
conv2D	(256, 256, 32)	320
MaxPooling2D	(128, 128, 32)	0
conv2D	(128, 128, 64)	18,496
TimeDistributed	(128, 8192)	0
Bidirectional	(128, 128)	4,227,584
dense	(128, 128)	16512
dense	(128, 128)	16512

Table 7: Layers of CNN

Bibliography

Newby, J. M., Schaefer, A., Lee, P. T., M. Gregory Forest, & Lai, S. K. (2018). Convolutional neural networks automate detection for tracking of submicron-scale particles in 2D and 3D. 115(36), 9026–9031. Retrieved from https://doi.org/10.1073/pnas.1804420115

CS231n Convolutional Neural Networks for Visual Recognition. (n.d.). Retrieved from https://cs231n.github.io/