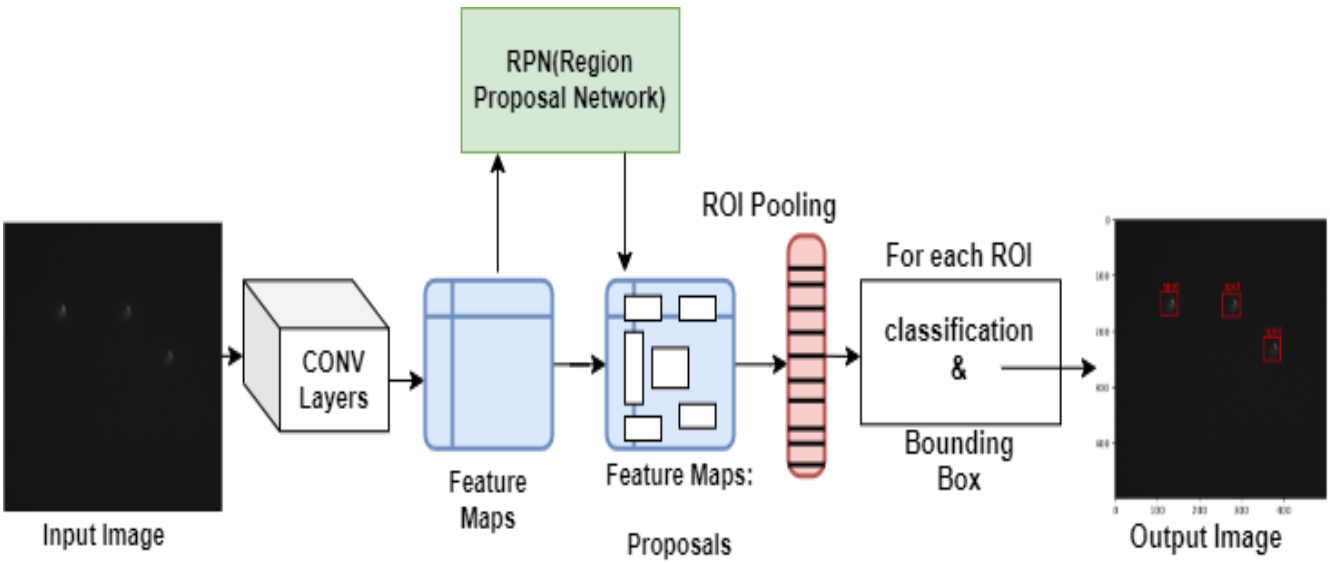
Investigating Chromatin Substance using Deep Learning

Abstract: Chromatin is a structure present within the cell nucleus. The reason behind studying chromatin structure is to analyze both efficacy and risk of cancer drugs. When a section of DNA is being expressed or repaired, chromatin assumes to be a **more spread-out state**.

Our research goal is to accurately identify the center position of chromatin structure using Deep Learning models.

Introduction:

In our previous research, we used an object detection model called Faster-RCNN model to identify chromatin spirals, and subsequently derived spiral centers. Below is Faster RCNN architecture we used to identify the chromatin centers in our previous phrase:



Phrase1: Predicting Chromatin centers using FRCNN model.

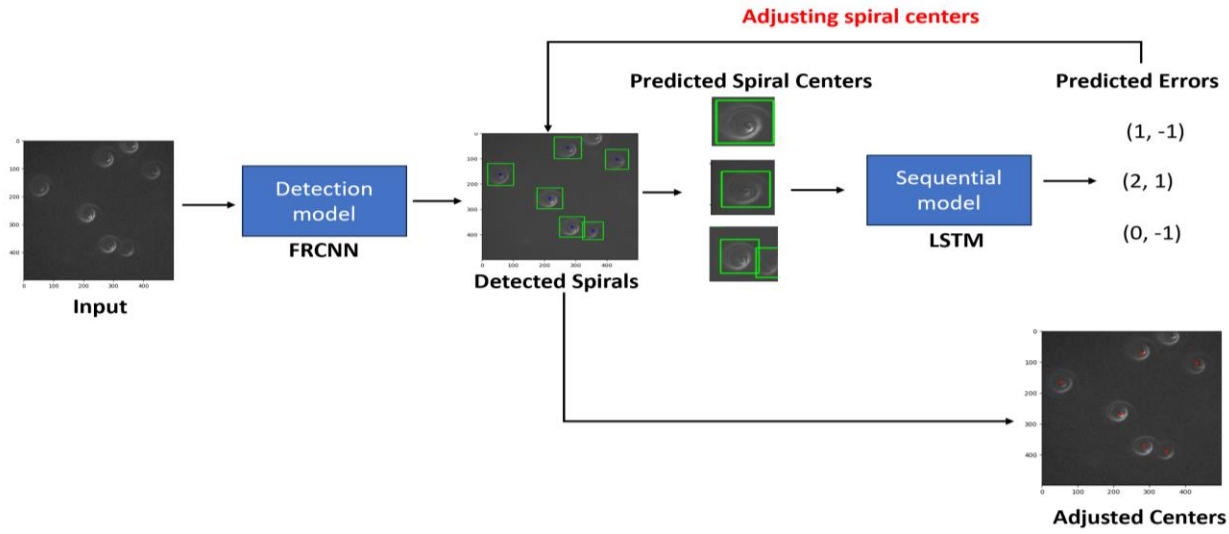
While our results from using the FRCNN model were satisfactory, we noticed that there were still some small pixel value differences in between the centers predicted by the model and the actual centers (NOSPP centers). To bridge this gap and

minimize errors, we proposed a solution: predicting the errors using Sequential model, and then adjusting the coordinates accordingly.

In our current proposal, we aim to modify these centers (point coordinates) by incorporating error adjustments. To achieve this, we utilized a Deep Learning approach called Long Short-Term Memory (LSTM) model to predict errors associated with the chromatin patterns.

Initially, we trained a deep learning model to learn knowledge of the error patterns and during testing, if we pass a spiral the model must detect any errors. The predicted errors are again passed back to adjust the predicted center point coordinates by FRCNN model. Our plan is to update the predicted center coordinates based on the predicted errors.

Proposed Architecture:



Explanation:

1. **Initial Spiral Detection with Faster-RCNN:**

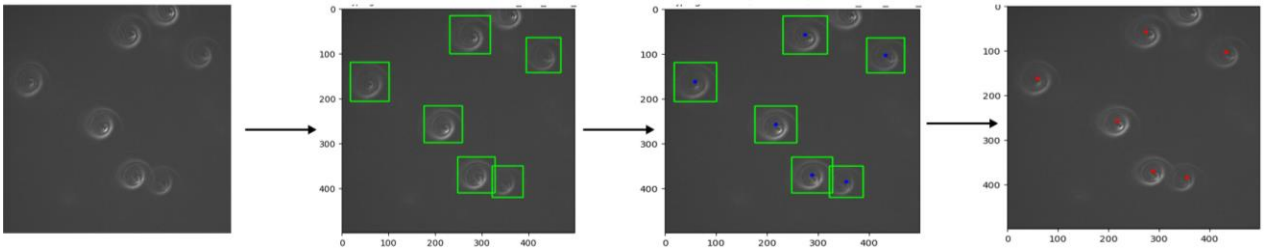
The first step involves using the Faster-RCNN (Region-Based Convolutional Neural Network) model for identifying chromatin spirals within images. Faster-RCNN is an

object detection algorithm that can locate objects within an image and classify them into predefined categories.

2. **Identification of Chromatin Centers:**

After detecting the chromatin spirals using the Faster-RCNN model, the centers of these spirals are determined by deriving the center of their bounding boxes.

```
for box in bndboxes:
        box = tuple(box)
        xmin = int(box[0])
        xmax = int(box[2])
        ymin = int(box[1])
        ymax = int(box[3])
predicted_center = (xmin + (xmax-xmin)//2, ymin + (ymax-ymin)//2)
```



The centers predicted by the Faster-RCNN model may have some errors compared to the actual centers of the chromatin spirals.

3. **Error Prediction with LSTM:**

To address these errors, an LSTM (Long Short-Term Memory) model is employed. The LSTM model is trained to learn the patterns of errors associated with the predicted chromatin centers.

During training, the LSTM model learns to predict the errors that may occur in the predicted chromatin centers.

4. **Error Adjustment and Refinement:**

During the testing phase, when a new chromatin spiral is detected and its center is predicted by the Faster-RCNN model, these predicted centers are adjusted based on the errors predicted by the LSTM model.

The predicted errors from the LSTM model are used to refine the coordinates of the predicted chromatin centers obtained from the Faster-RCNN model.

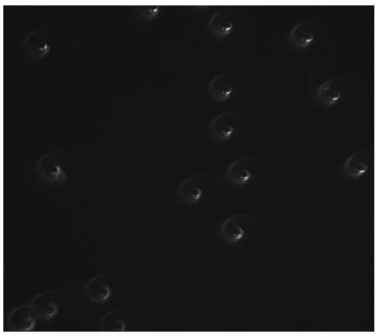
By adjusting the predicted centers based on the predicted errors, the aim is to minimize the errors between the predicted centers and the actual centers of the chromatin spirals.

Data Collection:

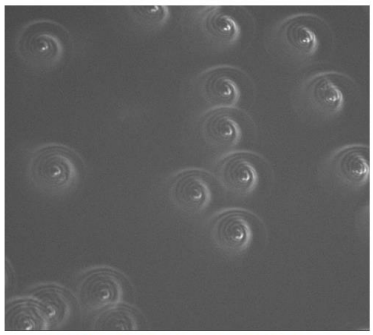
Initially we've received the dataset from Wake Forest School of Medicine consisting of approximately 160 TIFF images. Each TIFF image contains a sequence of either 61 or 122 frames capturing chromatin movement.

The dataset includes images captured both with and without cell movement. Images labeled as "NOSPP" are captured when cells are stationary and are considered as ground truth.

On the other hand, images labeled as "SPP" are captured during when cells are in movement and are the primary focus of our analysis.



11.11.21_SPP_Fov8_1_NDTiffStack_frame_35



11.11.21_NOSPP_Fov8_1_NDTiffStack_frame_35

After predicted the centers using FRCNN model; we created a dataset by hand that includes predicted spirals and their related errors. We performed rotations to spiral objects (45 degrees and +45 degrees) to improve the dataset size, we totally gathered around 150 bead sequences(objects) for the purpose of training and testing. Below is the sample single file; each file is the sequence of spiral objects. The path attribute contains the spiral images that are related with the training process. During training, the model takes the spiral images together with the corresponding error values for the x and y coordinates.

	images	slice	box	path	predictedx	predictedy	gtx	gty	errorx	errory
2	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	0	0	data/image	167	394	162	395	-5	1
3	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	1	2	data/image	167	395	162	395	-5	0
4	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	2	1	data/image	167	395	162	395	-5	0
5	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	3	0	data/image	167	394	162	395	-5	1
6	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	4	0	data/image	167	395	162	395	-5	0
7	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	5	1	data/image	167	393	162	395	-5	2
8	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	6	1	data/image	167	395	162	395	-5	0
9	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	7	0	data/image	168	395	162	395	-6	0
10	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	8	1	data/image	167	395	162	395	-5	0
11	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	9	2	data/image	169	393	162	395	-7	2
12	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	10	2	data/image	167	391	162	395	-5	4
13	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	11	2	data/image	165	391	162	395	-3	4
14	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	12	2	data/image	167	392	162	395	-5	3
15	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	13	2	data/image	167	391	162	395	-5	4
16	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	14	2	data/image	166	391	162	395	-4	4
17	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	15	1	data/image	166	392	162	395	-4	3
18	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	16	1	data/image	166	391	162	395	-4	4
19	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	17	0	data/image	166	390	162	395	-4	5
20	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	18	0	data/image	165	389	162	395	-3	6
21	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	19	0	data/image	165	390	162	395	-3	5
22	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	20	0	data/image	165	389	162	395	-3	6
23	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	21	0	data/image	164	389	162	395	-2	6
24	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	22	0	data/image	164	390	162	395	-2	5
25	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	23	1	data/image	164	389	162	395	-2	6
26	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	24	0	data/image	164	388	162	395	-2	7
27	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	25	0	data/image	164	389	162	395	-2	6
28	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	26	0	data/image	164	389	162	395	-2	6
29	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	27	0	data/image	163	390	162	395	-1	5
30	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	28	0	data/image	164	389	162	395	-2	6
31	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	29	0	data/image	165	390	162	395	-3	5
32	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	30	0	data/image	166	392	162	395	-4	3
33	1.4.22_SPP_Fov1_1_NDTiffStack (1)_frame_	31	0	data/image	165	392	162	395	-3	3

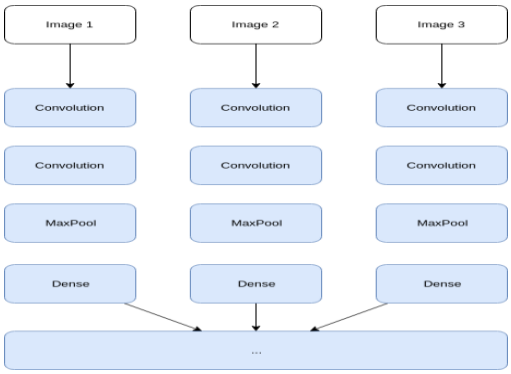
Sample file contains sequence of beads.

Sequence Error prediction Using CNN-LSTM Network:

A CNN-LSTM network use convolutional and LSTM layers to learn from the training data. This hybrid architecture is commonly used when we want to deal with sequence of images or video frames.

i) Convolutional Neural Networks (CNNs):

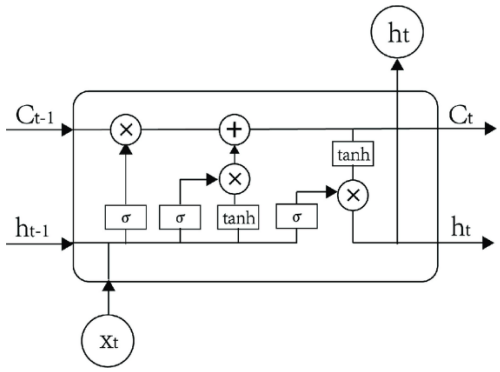
CNNs are excellent at capturing spatial patterns and features from input data, such as images. They consist of convolutional layers followed by pooling layers, which help in extracting hierarchical features from the input data while reducing the spatial dimensions. In the context of the CNN-LSTM architecture, CNNs are typically used as feature extractors from each element (e.g., frame) of the input sequence.



Time Distributed CNN

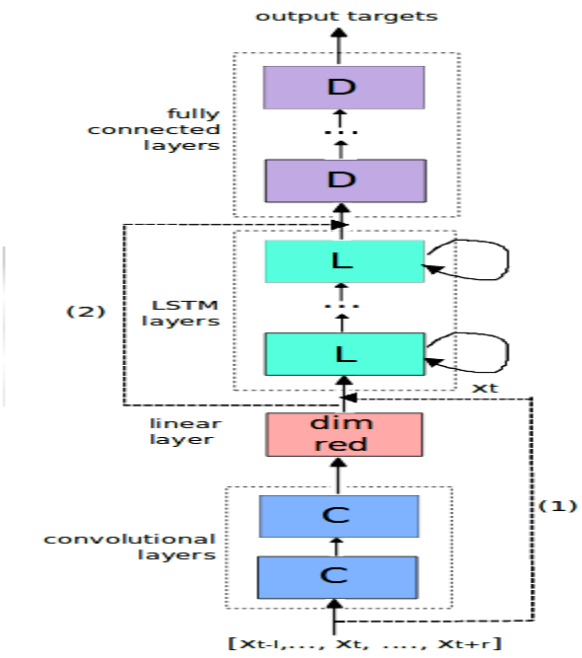
ii) Long Short-Term Memory (LSTM) Networks:

LSTM networks are a type of recurrent neural network (RNN) designed to capture temporal dependencies within sequential data. They are composed of memory cells that can maintain information over time and selectively update or forget it based on the input data.



iii) Integration of CNN and LSTM:

After passing each sequence of images through the CNN layers, the output features from each time step are typically flattened to create a one-dimensional feature vector for each time step. This step is often performed using a Flatten layer.

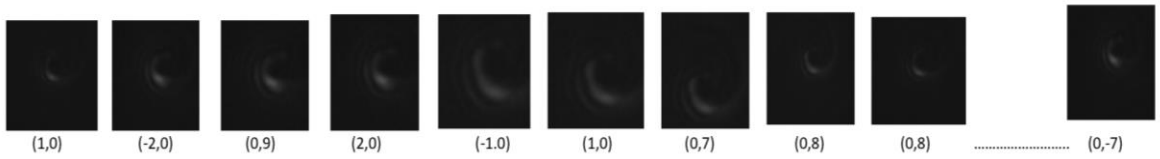


Architecture of CNN-LSTM

The flattened feature vectors from each time step are then passed to the LSTM layers. The LSTM layers process these feature vectors sequentially, capturing temporal dependencies and patterns within the sequence.

Implementation:

After gathering the data i.e. spiral images and related errors, we used simple 3 CNN layers (Conv2D, MaxPooling2D, and Flatten) and 2 LSTM layers in the model. Each sequence of spiral images is passed into CNN model to extract features after that flatten the features into one dimensional vector.



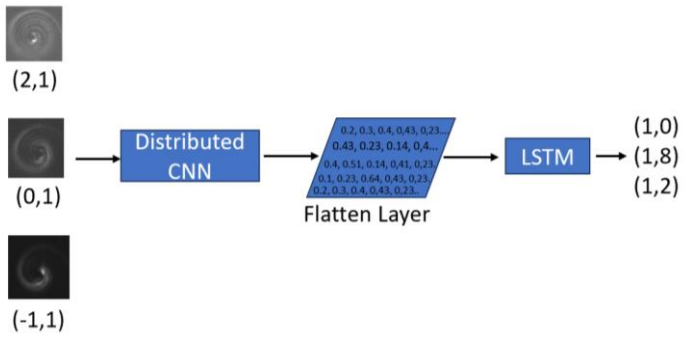
Spirals + Errors

In our dataset, the length of each sequence varies because of the cells are moving. **Consequently, in some frames, the object may not be present**, resulting in shorter sequences as time progresses. The challenge arises when dealing with variable-length sequences, as RNNs typically expect inputs of fixed dimensions. They can still handle variable-length sequences with the help of techniques like padding, masking, or dynamic sequence length handling. Padding involves adding filler values (e.g., zeros) to sequences to make them uniform in length**. Masking allows the model to ignore padded or irrelevant timesteps during training.**

```
Number of sequences before padding:
Train sequences: 99
Validation sequences: 21

Sample sequence lengths before padding:
Train sequence lengths: [61, 61, 61, 62, 61, 61, 61, 61, 61, 61, 61, 61, 61, 58, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 62, 61, 61, 64,
61, 62, 61, 61, 61, 62, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 61, 122, 122, 122, 122, 122, 122, 122, 122, 122, 120, 122, 122, 122, 81, 122, 122, 122, 122, 106, 122, 122, 122, 122, 122, 122, 122, 12
2, 122, 122, 98, 122, 104, 98, 122, 121, 107]
Validation sequence lengths: [122, 122, 122, 61, 61, 61, 61, 61, 61, 61, 61, 88, 115, 122, 117, 122, 122, 119, 114, 122, 119]

Number of sequences after padding:
Train sequences padded: 99
Validation sequences padded: 21

Sample sequence lengths after padding:
Train sequence lengths after padding: [122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 1
22, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 1
22, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122]
Validation sequence lengths after padding: [122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122, 122]
```

Dealing with variable length sequences

CNN model is having feature extraction part while LSTM model learns sequential pattern between feature along errors.



Feature Extraction with CNNs: Each image in the sequence is first passed through a CNN. The CNN acts as a feature extractor, pulling out relevant spatial features from each frame.

Sequence Learning with LSTMs: The sequence of feature-rich representations (output from the CNNs) is then fed into an LSTM. The LSTM processes these features sequentially and learns the temporal dynamics and dependencies between frames. It can predict future states of the sequence or identify anomalies/errors based on learned patterns.

Model Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistributed)	(None, None, 38, 38, 64)	1,792
time_distributed_1 (TimeDistributed)	(None, None, 36, 36, 32)	18,464
time_distributed_2 (TimeDistributed)	(None, None, 18, 18, 32)	0
time_distributed_3 (TimeDistributed)	(None, None, 10368)	0
masking (Masking)	(None, None, 10368)	0
lstm (LSTM)	(None, None, 64)	2,670,848
dropout (Dropout)	(None, None, 64)	0
lstm_1 (LSTM)	(None, None, 32)	12,416
time_distributed_4 (TimeDistributed)	(None, None, 2)	66

Total params: 8,110,760 (30.94 MB)
Trainable params: 2,703,586 (10.31 MB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 5,407,174 (20.63 MB)

i) 2D Convolution Operation

Convolutional Layers: The first part of your CNN comprises two convolutional layers. The first layer uses 64 filters, and the second layer uses 32 filters. Both have a kernel size of (3, 3). These layers are used to extract spatial features from the input images or frames.

64 Filters: The first convolution layer applies 64 different filters to the input. Each filter detects different features, such as edges, shapes, or textures**.**

32 Filters: The output from the first layer is then processed by the second convolution layer with 32 filters. This layer further refines the features extracted by the first layer, focusing on more specific patterns in the data.

ii) Max Pooling

Pooling Layer: After the convolutional layers, a max pooling layer with a pool size of (2, 2) is applied. This operation reduces the spatial dimensions of the feature maps by half, decreasing the number of computational resources needed and helping to prevent overfitting by abstracting the features.

iii) Flattening

Flatten: The output from the pooling layer consists of reduced spatial feature maps for each frame. These maps are then flattened into a 1D array. Flattening is necessary to transform the 2D feature maps into a format suitable for feeding into the subsequent LSTM layers. It lines up the features into a vector that can represent the input for sequence processing.

iv) Masking Layer

Masking: A masking layer is applied next, specifically targeting sequences with a value of 0.0. This step is crucial for dealing with variable-length sequences in batch data, as it tells the network to ignore any timestep in the sequence that contains only zeros. This prevents the network from learning from or making decisions based on these 'empty' or irrelevant timesteps.

v) LSTM Layers

LSTM Processing: The sequence of 1D arrays (flattened feature vectors from each frame) is then processed by two LSTM layers. The first LSTM layer has 64 units, and

the second has 32 units. These layers are responsible for capturing temporal dependencies—learning from the order and timing of the features across the sequence.

64 Units: The first LSTM layer processes the input sequence, maintaining an internal state that tracks temporal information across the frames.

32 Units: The output of the first LSTM layer is further processed by a second LSTM layer with 32 units, refining the temporal information and preparing it for final prediction.

vi) Fully Connected Dense Layer and Output

Dense Layer and Output: Finally, the output from the last LSTM layer is fed into a fully connected dense layer. This layer integrates the learned features and temporal patterns to make a final prediction. The network outputs two values for each timestep, errorx and errory, which might represent the magnitude and direction of errors or any other relevant metrics depending on the specific application.

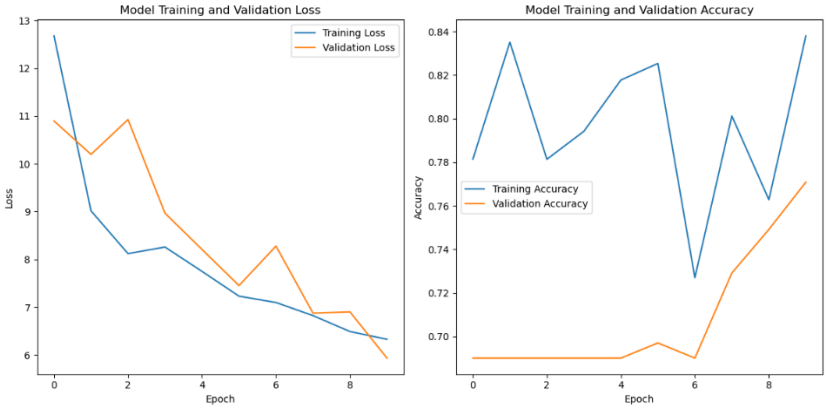
Evaluation metrics:

We performed 10-fold cross validation to evaluate the average results.

Cross Validation 1: (average performance)

Training	**99 bead sequences**
Validation	**1.27.22 & 1.4.22(18 bead sequences)**
Testing	**11.18.21(12 bead sequences)**

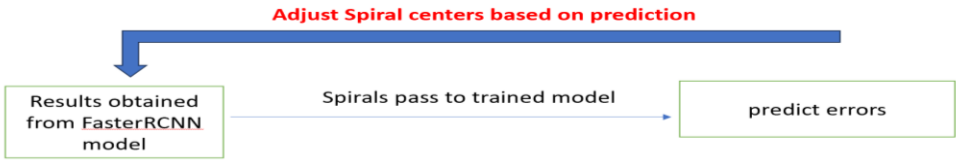
Training Loss and Accuracy curves:



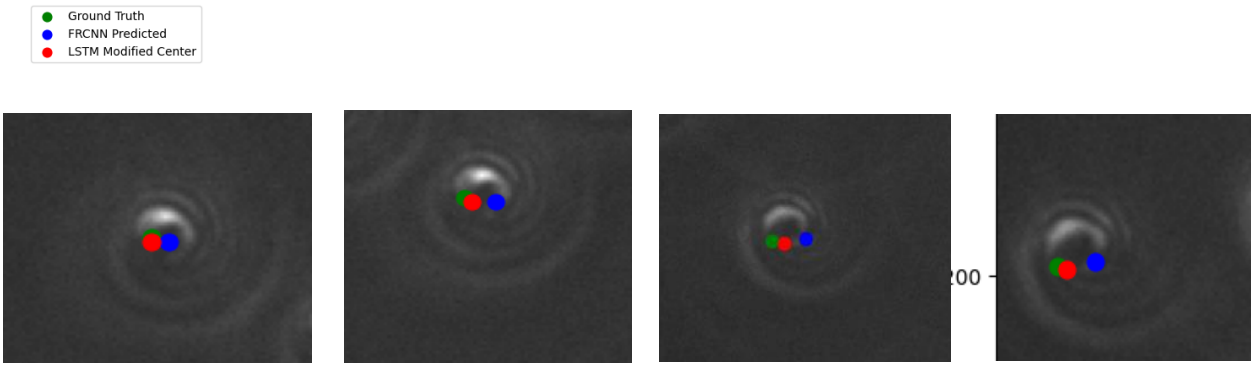
Testing Results:

Files	Average X error	Modified Average X error	Average Y error	Modified Average Y error
11.18.1	6.2	4.2	4.1	0.6
11.18.21.2	6.7	4.8	3.2	1.9
11.18.21.3	7.0	3.2	3.6	0.8
11.18.21.4	6.3	3.5	4.4	0.7
11.18.21.5	5.1	1.9	4.2	0.5
11.18.21.8	6.0	2.7	3.5	0.6
11.18.21.10	3.0	1.3	5.6	1.8
11.18.21.12	4.6	1.5	1.5	1.9
11.18.21.11	6.0	2.4	2.4	1.7

Comparison results on spirals belongs to test data (11.18.21):



GREEN: NOSPP Centers, BLUE: FRCNN predicted centers, RED: LSTM after adjustment centers



11.18.21_SPP_Fov1_1_NDTiffStack_frame_121.jpeg (Frame 121)

Cross Validation: bad results compared to all other folds.

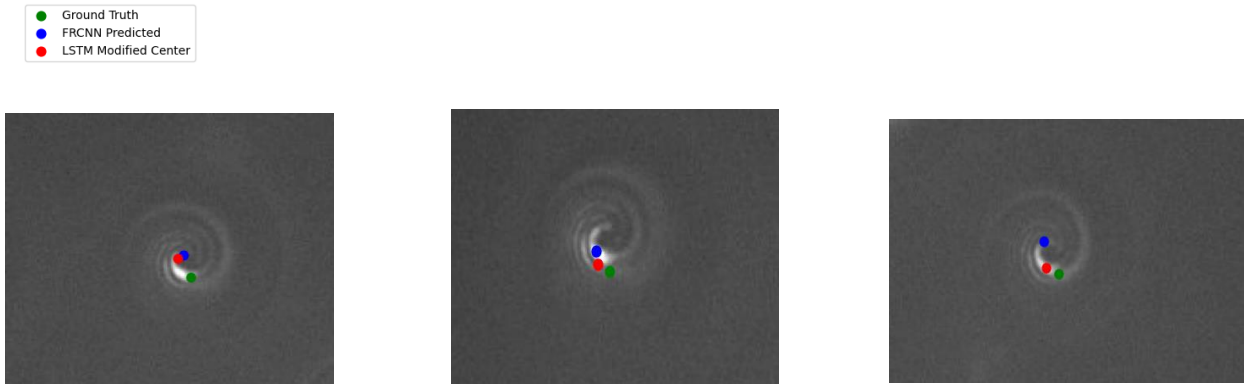
Training	**103 bead sequences**
Validation	**11.16.21(18 bead sequences)**
Testing	**2.3.22(8 bead sequences)**

Loss and Accuracy graphs:

Testing Results:

Files	Average X error	Modified Average X error	Average Y error	Modified Average Y error
2.3.22.1	7.2	5.7	14.0	12.6
2.3.22.2	8.5	6.8	14.8	13.5
2.3.22.3	8.9	6.8	14.0	12.5
2.3.22+45.3	10.1	9.8	7.8	6.0
2.3.22+45.2	5.1	1.9	4.2	0.5
2.3.22-45.1	5.4	4.1	9.6	8.2
2.3.22-45.2	4.4	3.3	9.9	8.3
2.3.22+45.1	9.1	8.1	8.8	7.6

Comparison results:

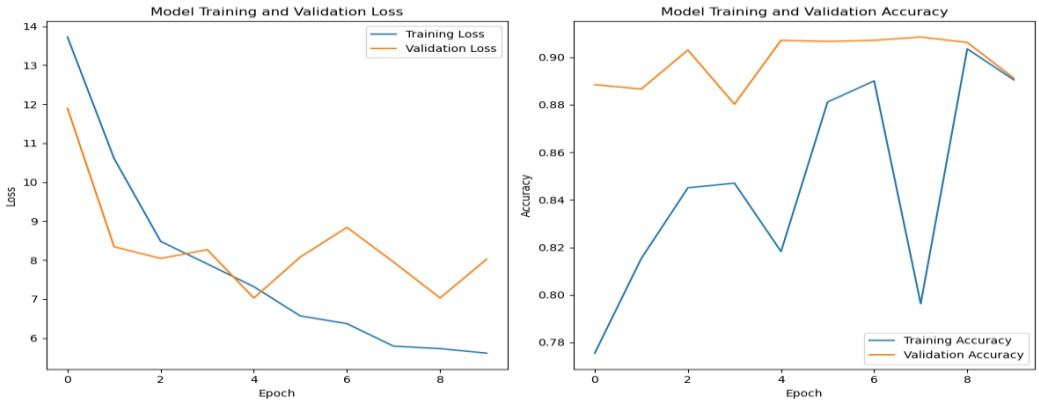


2.3.22_SPP_Fov1_1_NDTiffStack61.jpg(frame :61)

Cross Validation 3: (best performance)

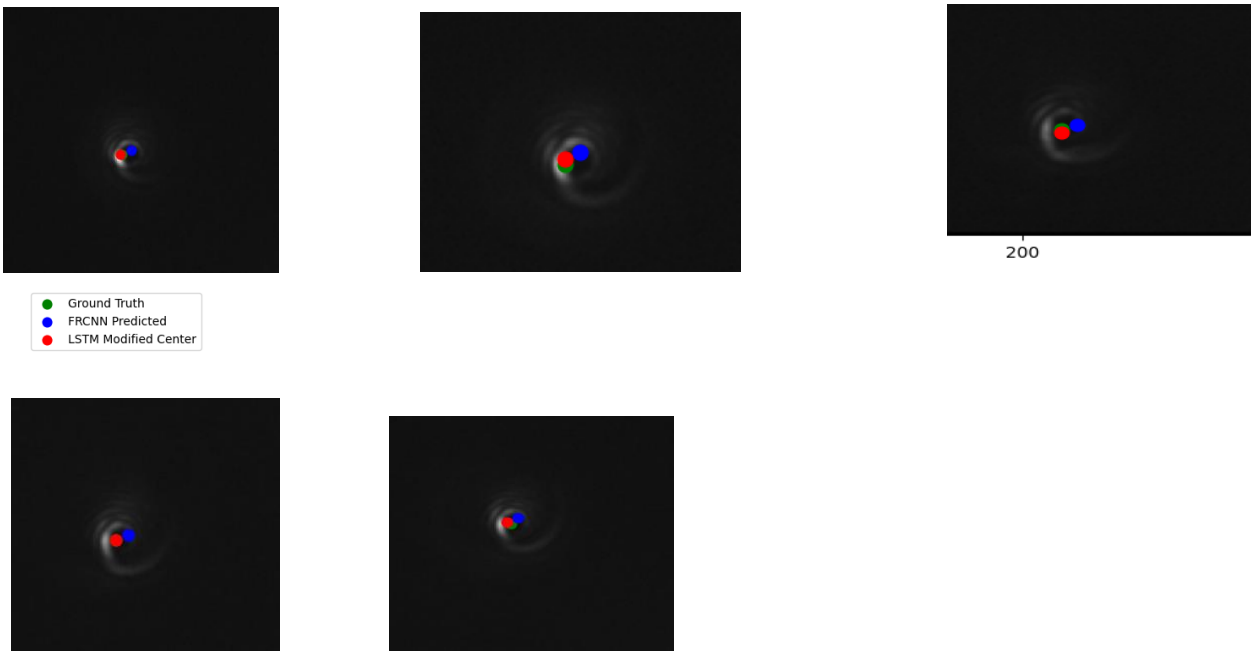
Training	**97 bead sequences**
Validation	**1.27.22 & 1.4.22(18 bead sequences)**
Testing	**1.20.22(15 bead sequences)**

Loss and Accuracy graphs:



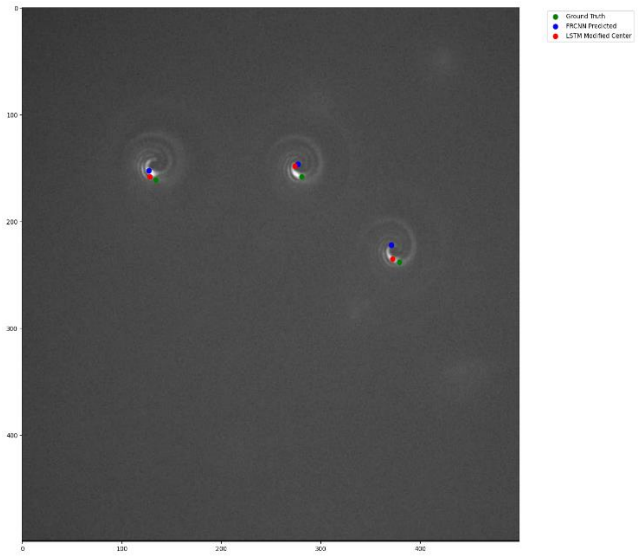
Testing Results:

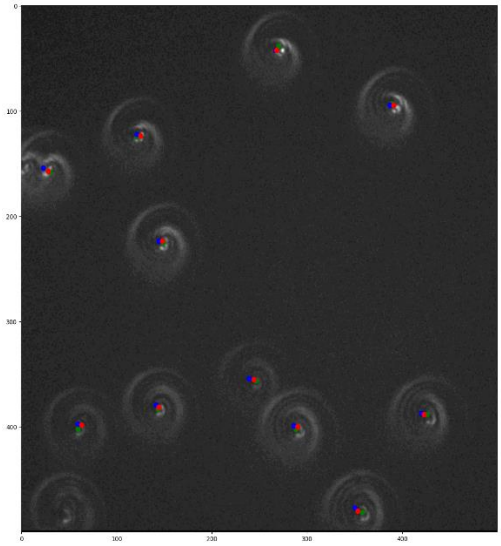
Files	Average X error	Modified Average X error	Average Y error	Modified Average Y error
1.20.22.6	2.4	2.0	4.5	1.5
1.20.22.4	2.5	1.4	3.8	1.0
1.20.22.3	5.3	1.7	5.4	2.6
1.20.22.2	2.6	1.4	4.4	1.5
1.20.22.1	4.8	1.6	3.7	1.0
1.20.22+45.1	2.8	1.2	7.2	3.6
1.20.22+45.2	2.5	1.5	7.8	4.2
1.20.22+45.3	2.6	1.9	5.8	2.4



1.20.22_SPP_Fov1_1_NDTiffStack_frame_37.jpeg (frame 37)

Results





Conclusion:

Adjusting the predicted centers of cells is our primary objective, through our experimentation with sequential models, particularly utilizing recurrent neural networks like LSTM we achieved a reasonable results.

References:

https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/

https://datascience.stackexchange.com/questions/58781/how-to-use-timedistributed-fo-cnnlstm

https://stackoverflow.com/questions/68915795/why-does-my-keras-timedistributed-cnn-lstm-model-expect-an-incomplete-shape

https://levelup.gitconnected.com/hands-on-practice-with-time-distributed-layers-using-tensorflow-c776a5d78e7e