Full Code

December 8, 2022

[13]:
```
%pylab inline
%config InlineBackend.figure_format = 'retina'
from ipywidgets import interact
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import keras #
from keras.models import Model, Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, CuDNNLSTM, Flatten
import math
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout, LSTM, Input,␣
↪LeakyReLU, ConvLSTM2D, Conv2D
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error
import numpy as np
from numpy import pi
import tensorflow as tf
import tensorflow.python.keras.backend as K
```

```
Populating the interactive namespace from numpy and matplotlib
```

[2]:
```
class Particle_Tracking_Training_Data(tf.Module):
    def __init__(self, Nt, rings=True):
        self.Nt = int(Nt)
        self.Ny = self.Nx = 256
        self.d = 3
        ximg = [[[i, j] for i in np.arange(self.Ny)]
            for j in np.arange(self.Nx)]
        self.ximg = np.float32(ximg)

        x = np.arange(self.Nx) - self.Nx//2
        y = np.arange(self.Ny) - self.Ny//2
```

```
        X0, Y0 = np.meshgrid(x, y)
        self.X = np.float32(X0)
        self.Y = np.float32(Y0)

        if rings:
            self.ring_indicator = 1.
        else:
            self.ring_indicator = 0.

        self._gen_video = tf.function(
            input_signature=(
                tf.TensorSpec(
                    shape=[self.Ny, self.Nx, self.Nt, None], dtype=tf.float32),
                tf.TensorSpec(shape=[self.Nt, None], dtype=tf.float32),
                tf.TensorSpec(shape=[], dtype=tf.float32),
                tf.TensorSpec(shape=[], dtype=tf.float32),
                tf.TensorSpec(shape=[], dtype=tf.float32),)
        )(self._gen_video)

        self._gen_labels = tf.function(
            input_signature=(
                tf.TensorSpec(
                    shape=[self.Ny, self.Nx, self.Nt, None], dtype=tf.float32),)
        )(self._gen_labels)

    def __call__(self, kappa, a, IbackLevel, Nparticles, sigma_motion):
        ## random brownian motion paths
        ## Nt, Nparticles, 3
        xi = self._sample_motion(Nparticles, sigma_motion)

        #### translate track positions to img coords
        ## Ny, Nx, Nt, Np, 2
        XALL = (self.ximg[:, :, None, None, :]
                - xi[None, None, :, :, :2])
        ## Ny, Nx, Nt, Np
        r = tf.math.sqrt(XALL[..., 0]**2 + XALL[..., 1]**2)
        z = xi[..., 2]

        ### generate video
        I = self._gen_video(r, z, kappa, a, IbackLevel)

        ### generate labels
        labels = self._gen_labels(r)

        return I, labels, xi

    @staticmethod
```

```
def rand(n):
    return tf.random.uniform([n], dtype=tf.float32)

@tf.function(
    input_signature=(
        tf.TensorSpec(shape=[], dtype=tf.int32),
        tf.TensorSpec(shape=[], dtype=tf.float32),))
def _sample_motion(self, Nparticles, sigma_motion):
    #### boundaries
    b_lower = tf.constant(
        [-10, -10, -30.], tf.float32)
    b_upper = tf.constant(
        [self.Nx+10, self.Ny+10, 30.], tf.float32)
    #### uniform random initial possitions
    U = tf.random.uniform(
        [1, Nparticles, self.d],
        dtype=tf.float32)
    X0 = b_lower + (b_upper - b_lower)*U
    #### normal increments
    dX = tf.random.normal(
        [self.Nt, Nparticles, self.d],
        stddev=sigma_motion,
        dtype=tf.float32)
    #### unbounded Brownian motion
    X = X0 + tf.math.cumsum(dX, axis=0)
    #### reflected brownian motion
    ## note that this is imperfect,
    ## if increments are very large it wont work
    X = tf.math.abs(X - b_lower) + b_lower
    X = -tf.math.abs(b_upper - X) + b_upper
    return X

def _gen_video(self, r, z, kappa, a, IbackLevel):
    uw = (0.5 + self.rand(1))/2.
    un = tf.floor(3*self.rand(1))
    uampRing = 0.2 + 0.8*self.rand(1)
    ufade = 15 + 10*self.rand(1)
    rmax = ufade*(un/uw)**(2./3.)
    ufadeMax = 0.85
    fade = (1. - ufadeMax*tf.abs(tf.tanh(z/ufade)))
    core = tf.exp(-(r**2/(8.*a))**2)
    ring = fade*(tf.exp(-(r - z)**4/(a)**4)
            + 0.5*uampRing*tf.cast(r<z, tf.float32))
    I = tf.transpose(
        tf.reduce_sum(
            fade*(core + self.ring_indicator*ring),
            axis=3),
```

```
            [2, 0, 1]) # Nt, Ny, Nx
        I += IbackLevel*tf.sin(
            self.rand(1)*6*pi/512*tf.sqrt(
                self.rand(1)*(self.X - self.rand(1)*512)**2
                    + self.rand(1)*(self.Y - self.rand(1)*512)**2))
        I += tf.random.normal(
            [self.Nt, self.Ny, self.Nx],
            stddev=kappa,
            dtype=tf.float32)
        Imin = tf.reduce_min(I)
        Imax = tf.reduce_max(I)
        I = (I - Imin)/(Imax - Imin)
        I = tf.round(I*tf.maximum(256., tf.round(2**16*self.rand(1))))
        return I

    def _gen_labels(self, r):
        R_detect = 3.
        ## (Ny, Nx, Nt)
        detectors = tf.reduce_sum(
            tf.cast(r[::2, ::2, :, :] < R_detect, tf.int32),
            axis=3)
        ## (Nt, Ny, Nx)
        P = tf.transpose(
            tf.cast(detectors > 0, tf.int32), [2, 0, 1])
        ## (Nt, Ny, Nx, 2)
        labels = tf.stack([1-P, P], 3)
        return labels
```

[3]:
```
Nt = 300 ## number of frames for each video
kappa = 0.1 ## standard deviation of background noise added to image
a = 3. ## scale factor for the size of particle spots (not true size of␣
 ↪particles)
IbackLevel = 0.1 ## relative intensity of randomly generated background pattern;
 ↪ in (0, 1)
Nparticles = 3 ## the number of particles (more => slower)
sigma_motion = 2 ## the standard deviation for particle brownian motion; should␣
 ↪be in (0, 10)


pt = Particle_Tracking_Training_Data(Nt) ## create object instance

vid, labels, tracks = pt(kappa, a, IbackLevel, Nparticles, sigma_motion)
```

0.1 Generating the Data

[4]:
```
@interact(t=(0, Nt-1, 1))
def plotfn(t=0):
    fig = figure(1, [14, 7])
    imshow(vid[t], origin='lower')
    #xlim(-10, 265)
    #ylim(-10, 265)
```

```
interactive(children=(IntSlider(value=0, description='t', max=299), Output()),␣
↪_dom_classes=('widget-interact'…
```

[5]:
```
def generate_data(size, pt, kappa, a, IbackLevel, Nparticles, sigma_motion):
    all_vid, all_labels, all_tracks = [],[],[]
    for i in range(size):
        vid, labels, tracks = pt(kappa, a, IbackLevel, Nparticles, sigma_motion)
        all_vid.append(vid[:,::2,::2]) # downsample video to Ntx128x128
        all_labels.append(labels)
        all_tracks.append(tracks)
    all_vid = tf.convert_to_tensor(all_vid)
    all_labels = tf.convert_to_tensor(all_labels)
    all_tracks = tf.convert_to_tensor(all_tracks)

    all_vid = tf.expand_dims(all_vid, 4)
    all_labels = tf.squeeze(all_labels)
    return all_vid, all_labels, all_tracks
```

[ ]:

[ ]:

[ ]:

1 Detecting the Particles

[67]:
```
train_size = 20
val_size = 3
Nt = 10
Nparticles_det = 10
pt = Particle_Tracking_Training_Data(Nt)

train_vid, train_labels, train_tracks = generate_data(train_size, pt, kappa, a,␣
↪IbackLevel, Nparticles_det, sigma_motion)
val_vid, val_labels, val_tracks = generate_data(val_size, pt, kappa, a,␣
↪IbackLevel, Nparticles_det, sigma_motion)
```

1.0.1 CNN model

[68]:
```
#https://stackoverflow.com/questions/51793737/
↪custom-loss-function-for-u-net-in-keras-using-class-weights-class-weight-not

def weightedLoss(originalLossFunc, weightsList):

    def lossFunc(true, pred):
        axis = -1 #if channels last
        #axis=  1 #if channels first

        #argmax returns the index of the element with the greatest value
        #done in the class axis, it returns the class index
        classSelectors = K.argmax(true, axis=axis)
            #if your loss is sparse, use only true as classSelectors

        #considering weights are ordered by class, for each class
        #true(1) if the class index is equal to the weight index
        classSelectors = tf.cast(classSelectors, tf.int32)
        classSelectors = [K.equal(i, classSelectors) for i in␣
↪range(len(weightsList))]

        #casting boolean to float for calculations
        #each tensor in the list contains 1 where ground true class is equal to␣
↪its index
        #if you sum all these, you will get a tensor full of ones.
        classSelectors = [K.cast(x, K.floatx()) for x in classSelectors]

        #for each of the selections above, multiply their respective weight
        weights = [sel * w for sel,w in zip(classSelectors, weightsList)]

        #sums all the selections
        #result is a tensor with the respective weight for each element in␣
↪predictions
        weightMultiplier = weights[0]
        for i in range(1, len(weights)):
            weightMultiplier = weightMultiplier + weights[i]


        #make sure your originalLossFunc only collapses the class axis
        #you need the other axes intact to multiply the weights tensor
        loss = originalLossFunc(true,pred)
        loss = loss * weightMultiplier

        return loss
    return lossFunc
```

[69]:
```
precision_f = tf.keras.metrics.Precision(class_id=1)
recall_f = tf.keras.metrics.Recall(class_id=1)

def f1_score(y_true, y_pred):
    precision = precision_f(y_true, y_pred)
    recall = recall_f(y_true, y_pred)
    f1_val = 2*(precision*recall)/(precision+recall)
    return f1_val
```

[70]:
```
train_vid_cnn = tf.reshape(train_vid, (train_size*Nt, 128, 128, 1))
train_labels_cnn = tf.reshape(train_labels, (train_size*Nt, 128, 128, 2))
val_vid_cnn = tf.reshape(val_vid, (val_size*Nt, 128, 128, 1))
val_labels_cnn = tf.reshape(val_labels, (val_size*Nt, 128, 128, 2))
```

[71]:
```
loss_list1 = list()

class SaveBatchLoss1(tf.keras.callbacks.Callback):
    def on_train_batch_end(self, batch, logs=None):
        loss_list1.append(logs['loss'])
```

[72]:
```
batch_size = 10
epochs = 50

model1 = Sequential()
model1.add(Input((128,128,1)))
model1.add(Conv2D(12, 3, padding='same', activation=LeakyReLU(alpha=0.01),␣
 ↪strides=(1,1)))
model1.add(Conv2D(32, 3, padding='same', activation=LeakyReLU(alpha=0.01),␣
 ↪strides=(1,1)))
model1.add(Conv2D(2, 3, padding='same', strides=(1,1)))
model1.add(Activation('softmax'))

model1.compile(optimizer='adam',
             loss=weightedLoss(keras.losses.categorical_crossentropy, [0.2, 0.
 ↪8]),
             metrics=['accuracy', tf.keras.metrics.
 ↪Precision(name='precision_1', class_id=1),
                      tf.keras.metrics.Recall(name='recall_1', class_id=1),␣
 ↪f1_score])

num_batches = int(train_size/batch_size)
model1.fit(train_vid_cnn, train_labels_cnn, steps_per_epoch=num_batches,␣
 ↪epochs=epochs, verbose=1,
          validation_data=(val_vid_cnn, val_labels_cnn),␣
 ↪callbacks=SaveBatchLoss1())
```

```
Epoch 1/50
```

```
2/2 [==============================] - 7s 4s/step - loss: 8.6499 - accuracy:
0.9531 - precision_1: 0.0032 - recall_1: 0.0375 - f1_score: 0.0060 - val_loss:
6.9045 - val_accuracy: 0.9922 - val_precision_1: 0.0015 - val_recall_1: 0.0017 -
val_f1_score: 0.0057
Epoch 2/50
2/2 [==============================] - 5s 3s/step - loss: 8.9291 - accuracy:
0.9934 - precision_1: 0.0045 - recall_1: 0.0037 - f1_score: 0.0056 - val_loss:
5.1317 - val_accuracy: 0.9933 - val_precision_1: 0.0020 - val_recall_1: 0.0017 -
val_f1_score: 0.0055
Epoch 3/50
2/2 [==============================] - 5s 3s/step - loss: 6.0940 - accuracy:
0.9929 - precision_1: 0.0065 - recall_1: 0.0062 - f1_score: 0.0055 - val_loss:
2.0264 - val_accuracy: 0.9810 - val_precision_1: 0.0257 - val_recall_1: 0.1148 -
val_f1_score: 0.0072
Epoch 4/50
2/2 [==============================] - 5s 3s/step - loss: 3.8997 - accuracy:
0.9358 - precision_1: 0.0172 - recall_1: 0.2952 - f1_score: 0.0155 - val_loss:
1.9006 - val_accuracy: 0.9421 - val_precision_1: 0.0274 - val_recall_1: 0.4323 -
val_f1_score: 0.0219
Epoch 5/50
2/2 [==============================] - 5s 3s/step - loss: 2.3909 - accuracy:
0.9663 - precision_1: 0.0309 - recall_1: 0.2704 - f1_score: 0.0274 - val_loss:
1.5458 - val_accuracy: 0.9926 - val_precision_1: 0.0938 - val_recall_1: 0.1198 -
val_f1_score: 0.0292
Epoch 6/50
2/2 [==============================] - 5s 3s/step - loss: 2.6187 - accuracy:
0.9941 - precision_1: 0.1002 - recall_1: 0.0757 - f1_score: 0.0308 - val_loss:
1.6484 - val_accuracy: 0.9951 - val_precision_1: 0.1734 - val_recall_1: 0.0890 -
val_f1_score: 0.0316
Epoch 7/50
2/2 [==============================] - 5s 3s/step - loss: 2.5580 - accuracy:
0.9954 - precision_1: 0.1926 - recall_1: 0.0801 - f1_score: 0.0331 - val_loss:
1.0222 - val_accuracy: 0.9939 - val_precision_1: 0.2099 - val_recall_1: 0.2447 -
val_f1_score: 0.0350
Epoch 8/50
2/2 [==============================] - 5s 3s/step - loss: 1.6063 - accuracy:
0.9920 - precision_1: 0.1454 - recall_1: 0.2437 - f1_score: 0.0397 - val_loss:
0.8481 - val_accuracy: 0.9695 - val_precision_1: 0.0727 - val_recall_1: 0.6288 -
val_f1_score: 0.0443
Epoch 9/50
2/2 [==============================] - 5s 3s/step - loss: 1.5747 - accuracy:
0.9614 - precision_1: 0.0543 - recall_1: 0.5815 - f1_score: 0.0518 - val_loss:
1.0891 - val_accuracy: 0.9540 - val_precision_1: 0.0577 - val_recall_1: 0.7609 -
val_f1_score: 0.0554
Epoch 10/50
2/2 [==============================] - 5s 3s/step - loss: 1.3431 - accuracy:
0.9688 - precision_1: 0.0674 - recall_1: 0.5869 - f1_score: 0.0613 - val_loss:
0.5417 - val_accuracy: 0.9889 - val_precision_1: 0.1763 - val_recall_1: 0.5633 -
```

```
val_f1_score: 0.0649
Epoch 11/50
2/2 [==============================] - 5s 3s/step - loss: 1.1557 - accuracy:
0.9918 - precision_1: 0.1929 - recall_1: 0.3859 - f1_score: 0.0702 - val_loss:
0.6029 - val_accuracy: 0.9944 - val_precision_1: 0.3093 - val_recall_1: 0.4429 -
val_f1_score: 0.0730
Epoch 12/50
2/2 [==============================] - 5s 3s/step - loss: 1.2881 - accuracy:
0.9947 - precision_1: 0.2948 - recall_1: 0.3189 - f1_score: 0.0772 - val_loss:
0.5032 - val_accuracy: 0.9936 - val_precision_1: 0.2885 - val_recall_1: 0.5190 -
val_f1_score: 0.0801
Epoch 13/50
2/2 [==============================] - 5s 3s/step - loss: 1.0422 - accuracy:
0.9930 - precision_1: 0.2394 - recall_1: 0.4213 - f1_score: 0.0851 - val_loss:
0.4527 - val_accuracy: 0.9854 - val_precision_1: 0.1609 - val_recall_1: 0.7133 -
val_f1_score: 0.0886
Epoch 14/50
2/2 [==============================] - 5s 3s/step - loss: 0.8922 - accuracy:
0.9832 - precision_1: 0.1266 - recall_1: 0.6107 - f1_score: 0.0942 - val_loss:
0.6655 - val_accuracy: 0.9707 - val_precision_1: 0.0951 - val_recall_1: 0.8275 -
val_f1_score: 0.0970
Epoch 15/50
2/2 [==============================] - 5s 3s/step - loss: 0.9626 - accuracy:
0.9760 - precision_1: 0.1000 - recall_1: 0.6966 - f1_score: 0.1014 - val_loss:
0.4469 - val_accuracy: 0.9832 - val_precision_1: 0.1495 - val_recall_1: 0.7760 -
val_f1_score: 0.1042
Epoch 16/50
2/2 [==============================] - 5s 3s/step - loss: 0.7692 - accuracy:
0.9876 - precision_1: 0.1673 - recall_1: 0.6008 - f1_score: 0.1091 - val_loss:
0.3495 - val_accuracy: 0.9920 - val_precision_1: 0.2650 - val_recall_1: 0.6792 -
val_f1_score: 0.1120
Epoch 17/50
2/2 [==============================] - 5s 3s/step - loss: 0.7896 - accuracy:
0.9933 - precision_1: 0.2760 - recall_1: 0.5096 - f1_score: 0.1166 - val_loss:
0.3375 - val_accuracy: 0.9935 - val_precision_1: 0.3136 - val_recall_1: 0.6568 -
val_f1_score: 0.1194
Epoch 18/50
2/2 [==============================] - 5s 3s/step - loss: 0.7653 - accuracy:
0.9937 - precision_1: 0.2941 - recall_1: 0.5141 - f1_score: 0.1237 - val_loss:
0.3224 - val_accuracy: 0.9909 - val_precision_1: 0.2463 - val_recall_1: 0.7273 -
val_f1_score: 0.1265
Epoch 19/50
2/2 [==============================] - 5s 3s/step - loss: 0.6671 - accuracy:
0.9907 - precision_1: 0.2188 - recall_1: 0.6008 - f1_score: 0.1311 - val_loss:
0.3833 - val_accuracy: 0.9848 - val_precision_1: 0.1683 - val_recall_1: 0.8091 -
val_f1_score: 0.1335
Epoch 20/50
2/2 [==============================] - 5s 3s/step - loss: 0.6722 - accuracy:
```

```
0.9854 - precision_1: 0.1571 - recall_1: 0.6863 - f1_score: 0.1375 - val_loss:
0.3908 - val_accuracy: 0.9836 - val_precision_1: 0.1594 - val_recall_1: 0.8225 -
val_f1_score: 0.1396
Epoch 21/50
2/2 [==============================] - 5s 3s/step - loss: 0.6342 - accuracy:
0.9867 - precision_1: 0.1698 - recall_1: 0.6805 - f1_score: 0.1432 - val_loss:
0.2988 - val_accuracy: 0.9898 - val_precision_1: 0.2319 - val_recall_1: 0.7766 -
val_f1_score: 0.1455
Epoch 22/50
2/2 [==============================] - 5s 3s/step - loss: 0.5927 - accuracy:
0.9917 - precision_1: 0.2462 - recall_1: 0.6112 - f1_score: 0.1494 - val_loss:
0.2667 - val_accuracy: 0.9929 - val_precision_1: 0.3030 - val_recall_1: 0.7279 -
val_f1_score: 0.1517
Epoch 23/50
2/2 [==============================] - 5s 3s/step - loss: 0.5902 - accuracy:
0.9936 - precision_1: 0.3012 - recall_1: 0.5759 - f1_score: 0.1555 - val_loss:
0.2563 - val_accuracy: 0.9925 - val_precision_1: 0.2923 - val_recall_1: 0.7486 -
val_f1_score: 0.1577
Epoch 24/50
2/2 [==============================] - 5s 3s/step - loss: 0.5484 - accuracy:
0.9925 - precision_1: 0.2695 - recall_1: 0.6142 - f1_score: 0.1614 - val_loss:
0.2753 - val_accuracy: 0.9891 - val_precision_1: 0.2242 - val_recall_1: 0.8091 -
val_f1_score: 0.1635
Epoch 25/50
2/2 [==============================] - 5s 3s/step - loss: 0.5287 - accuracy:
0.9892 - precision_1: 0.2052 - recall_1: 0.6801 - f1_score: 0.1668 - val_loss:
0.2966 - val_accuracy: 0.9866 - val_precision_1: 0.1921 - val_recall_1: 0.8354 -
val_f1_score: 0.1686
Epoch 26/50
2/2 [==============================] - 6s 3s/step - loss: 0.5161 - accuracy:
0.9885 - precision_1: 0.1969 - recall_1: 0.6982 - f1_score: 0.1716 - val_loss:
0.2487 - val_accuracy: 0.9899 - val_precision_1: 0.2388 - val_recall_1: 0.8085 -
val_f1_score: 0.1735
Epoch 27/50
2/2 [==============================] - 6s 3s/step - loss: 0.4849 - accuracy:
0.9915 - precision_1: 0.2484 - recall_1: 0.6564 - f1_score: 0.1766 - val_loss:
0.2190 - val_accuracy: 0.9926 - val_precision_1: 0.2998 - val_recall_1: 0.7738 -
val_f1_score: 0.1786
Epoch 28/50
2/2 [==============================] - 6s 3s/step - loss: 0.4783 - accuracy:
0.9932 - precision_1: 0.2954 - recall_1: 0.6222 - f1_score: 0.1818 - val_loss:
0.2121 - val_accuracy: 0.9926 - val_precision_1: 0.3006 - val_recall_1: 0.7772 -
val_f1_score: 0.1836
Epoch 29/50
2/2 [==============================] - 6s 3s/step - loss: 0.4584 - accuracy:
0.9927 - precision_1: 0.2813 - recall_1: 0.6419 - f1_score: 0.1867 - val_loss:
0.2227 - val_accuracy: 0.9905 - val_precision_1: 0.2518 - val_recall_1: 0.8135 -
val_f1_score: 0.1884
```

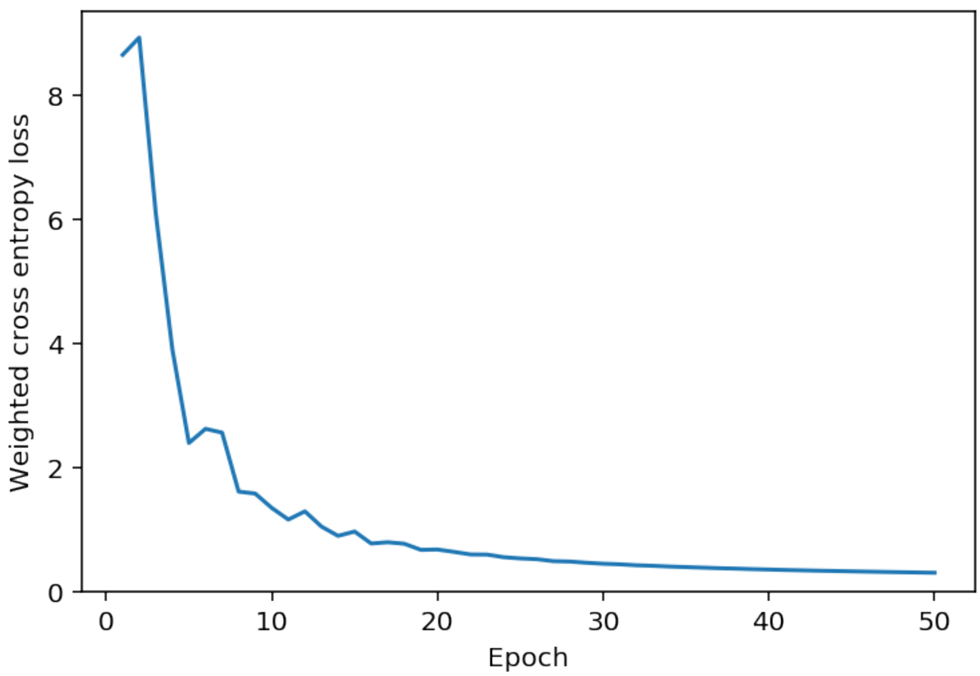
```
Epoch 30/50
2/2 [==============================] - 6s 3s/step - loss: 0.4432 - accuracy:
0.9907 - precision_1: 0.2355 - recall_1: 0.6816 - f1_score: 0.1912 - val_loss:
0.2332 - val_accuracy: 0.9890 - val_precision_1: 0.2257 - val_recall_1: 0.8337 -
val_f1_score: 0.1927
Epoch 31/50
2/2 [==============================] - 6s 3s/step - loss: 0.4338 - accuracy:
0.9901 - precision_1: 0.2249 - recall_1: 0.6956 - f1_score: 0.1953 - val_loss:
0.2102 - val_accuracy: 0.9908 - val_precision_1: 0.2570 - val_recall_1: 0.8147 -
val_f1_score: 0.1968
Epoch 32/50
2/2 [==============================] - 6s 3s/step - loss: 0.4185 - accuracy:
0.9919 - precision_1: 0.2629 - recall_1: 0.6706 - f1_score: 0.1995 - val_loss:
0.1907 - val_accuracy: 0.9926 - val_precision_1: 0.3023 - val_recall_1: 0.7889 -
val_f1_score: 0.2011
Epoch 33/50
2/2 [==============================] - 6s 3s/step - loss: 0.4099 - accuracy:
0.9931 - precision_1: 0.2963 - recall_1: 0.6463 - f1_score: 0.2037 - val_loss:
0.1881 - val_accuracy: 0.9924 - val_precision_1: 0.2947 - val_recall_1: 0.7923 -
val_f1_score: 0.2053
Epoch 34/50
2/2 [==============================] - 5s 3s/step - loss: 0.3976 - accuracy:
0.9925 - precision_1: 0.2787 - recall_1: 0.6626 - f1_score: 0.2078 - val_loss:
0.1965 - val_accuracy: 0.9909 - val_precision_1: 0.2590 - val_recall_1: 0.8152 -
val_f1_score: 0.2092
Epoch 35/50
2/2 [==============================] - 6s 3s/step - loss: 0.3889 - accuracy:
0.9913 - precision_1: 0.2499 - recall_1: 0.6905 - f1_score: 0.2116 - val_loss:
0.1955 - val_accuracy: 0.9906 - val_precision_1: 0.2538 - val_recall_1: 0.8191 -
val_f1_score: 0.2129
Epoch 36/50
2/2 [==============================] - 6s 3s/step - loss: 0.3800 - accuracy:
0.9914 - precision_1: 0.2537 - recall_1: 0.6898 - f1_score: 0.2150 - val_loss:
0.1811 - val_accuracy: 0.9919 - val_precision_1: 0.2818 - val_recall_1: 0.8001 -
val_f1_score: 0.2164
Epoch 37/50
2/2 [==============================] - 5s 3s/step - loss: 0.3710 - accuracy:
0.9926 - precision_1: 0.2829 - recall_1: 0.6693 - f1_score: 0.2187 - val_loss:
0.1726 - val_accuracy: 0.9926 - val_precision_1: 0.3026 - val_recall_1: 0.7934 -
val_f1_score: 0.2200
Epoch 38/50
2/2 [==============================] - 5s 3s/step - loss: 0.3646 - accuracy:
0.9930 - precision_1: 0.2956 - recall_1: 0.6629 - f1_score: 0.2223 - val_loss:
0.1731 - val_accuracy: 0.9921 - val_precision_1: 0.2877 - val_recall_1: 0.7984 -
val_f1_score: 0.2236
Epoch 39/50
2/2 [==============================] - 6s 3s/step - loss: 0.3559 - accuracy:
0.9924 - precision_1: 0.2800 - recall_1: 0.6792 - f1_score: 0.2257 - val_loss:
```

```
0.1767 - val_accuracy: 0.9913 - val_precision_1: 0.2696 - val_recall_1: 0.8163 -
val_f1_score: 0.2269
Epoch 40/50
2/2 [==============================] - 5s 3s/step - loss: 0.3496 - accuracy:
0.9919 - precision_1: 0.2661 - recall_1: 0.6932 - f1_score: 0.2289 - val_loss:
0.1723 - val_accuracy: 0.9915 - val_precision_1: 0.2744 - val_recall_1: 0.8158 -
val_f1_score: 0.2301
Epoch 41/50
2/2 [==============================] - 5s 3s/step - loss: 0.3424 - accuracy:
0.9923 - precision_1: 0.2771 - recall_1: 0.6889 - f1_score: 0.2320 - val_loss:
0.1637 - val_accuracy: 0.9923 - val_precision_1: 0.2959 - val_recall_1: 0.8091 -
val_f1_score: 0.2332
Epoch 42/50
2/2 [==============================] - 5s 3s/step - loss: 0.3365 - accuracy:
0.9929 - precision_1: 0.2968 - recall_1: 0.6778 - f1_score: 0.2353 - val_loss:
0.1605 - val_accuracy: 0.9925 - val_precision_1: 0.3006 - val_recall_1: 0.8102 -
val_f1_score: 0.2364
Epoch 43/50
2/2 [==============================] - 5s 3s/step - loss: 0.3306 - accuracy:
0.9929 - precision_1: 0.2952 - recall_1: 0.6828 - f1_score: 0.2383 - val_loss:
0.1636 - val_accuracy: 0.9919 - val_precision_1: 0.2861 - val_recall_1: 0.8247 -
val_f1_score: 0.2395
Epoch 44/50
2/2 [==============================] - 5s 3s/step - loss: 0.3255 - accuracy:
0.9924 - precision_1: 0.2810 - recall_1: 0.6973 - f1_score: 0.2414 - val_loss:
0.1625 - val_accuracy: 0.9918 - val_precision_1: 0.2853 - val_recall_1: 0.8309 -
val_f1_score: 0.2424
Epoch 45/50
2/2 [==============================] - 6s 3s/step - loss: 0.3204 - accuracy:
0.9925 - precision_1: 0.2861 - recall_1: 0.6979 - f1_score: 0.2442 - val_loss:
0.1549 - val_accuracy: 0.9925 - val_precision_1: 0.3033 - val_recall_1: 0.8264 -
val_f1_score: 0.2453
Epoch 46/50
2/2 [==============================] - 5s 3s/step - loss: 0.3153 - accuracy:
0.9930 - precision_1: 0.3021 - recall_1: 0.6892 - f1_score: 0.2471 - val_loss:
0.1518 - val_accuracy: 0.9925 - val_precision_1: 0.3055 - val_recall_1: 0.8287 -
val_f1_score: 0.2482
Epoch 47/50
2/2 [==============================] - 5s 3s/step - loss: 0.3106 - accuracy:
0.9931 - precision_1: 0.3037 - recall_1: 0.6918 - f1_score: 0.2499 - val_loss:
0.1523 - val_accuracy: 0.9923 - val_precision_1: 0.2980 - val_recall_1: 0.8343 -
val_f1_score: 0.2510
Epoch 48/50
2/2 [==============================] - 5s 3s/step - loss: 0.3063 - accuracy:
0.9927 - precision_1: 0.2929 - recall_1: 0.7028 - f1_score: 0.2527 - val_loss:
0.1520 - val_accuracy: 0.9921 - val_precision_1: 0.2944 - val_recall_1: 0.8371 -
val_f1_score: 0.2536
Epoch 49/50
```

```
2/2 [==============================] - 6s 3s/step - loss: 0.3021 - accuracy:
0.9928 - precision_1: 0.2950 - recall_1: 0.7042 - f1_score: 0.2553 - val_loss:
0.1459 - val_accuracy: 0.9927 - val_precision_1: 0.3102 - val_recall_1: 0.8337 -
val_f1_score: 0.2563
Epoch 50/50
2/2 [==============================] - 5s 3s/step - loss: 0.2980 - accuracy:
0.9931 - precision_1: 0.3063 - recall_1: 0.6978 - f1_score: 0.2579 - val_loss:
0.1438 - val_accuracy: 0.9927 - val_precision_1: 0.3124 - val_recall_1: 0.8343 -
val_f1_score: 0.2589
```

[72]: <keras.callbacks.History at 0x1e09af2bbc8>

[73]:
```
plt.plot(range(1,epochs+1), loss_list1[num_batches-1::num_batches])
plt.xlabel('Epoch')
plt.ylabel('Weighted cross entropy loss')
plt.ylim(0)
plt.show()
```



[74]:
```
model1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 128, 128, 12)      120
```

```
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 128, 128, 32)      3488
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 128, 128, 2)       578
_________________________________________________________________
activation_2 (Activation)    (None, 128, 128, 2)       0
=================================================================
Total params: 4,186
Trainable params: 4,186
Non-trainable params: 0
_________________________________________________________________
```

1.0.2 CNN-LSTM

[88]:
```
loss_list2 = list()

class SaveBatchLoss2(tf.keras.callbacks.Callback):
    def on_train_batch_end(self, batch, logs=None):
        loss_list2.append(logs['loss'])
```

[89]:
```
# check which percentage is particles
num_part = tf.math.reduce_sum(train_labels[:,:,:,:,1])
print(1-num_part/(10*10*128*128))
```

```
tf.Tensor(0.9926837158203125, shape=(), dtype=float64)
```

[90]:
```
batch_size = 2
epochs = 50

model2 = Sequential()
model2.add(Input((Nt,128,128,1)))
model2.add(ConvLSTM2D(12, kernel_size=(3, 3), padding='same', strides=(1,1),
                     return_sequences=True,  data_format='channels_last'))
model2.add(ConvLSTM2D(16, kernel_size=(3, 3), padding='same', strides=(1,1),
                     return_sequences=True,  data_format='channels_last'))
model2.add(ConvLSTM2D(2, kernel_size=(3, 3), padding='same', strides=(1,1),
                     return_sequences=True,  data_format='channels_last',␣
 ↪activation="softmax"))

model2.compile(optimizer='adam',
            loss=weightedLoss(keras.losses.categorical_crossentropy, [0.005, 0.
 ↪995]),
            metrics=['accuracy', tf.keras.metrics.Precision(name='precision_1',␣
 ↪class_id=1),
                     tf.keras.metrics.Recall(name='recall_1', class_id=1),␣
 ↪f1_score])
```

```
num_batches = int(train_size/batch_size)
model2.fit(train_vid, train_labels, steps_per_epoch=num_batches, epochs=epochs,␣
 ↪verbose=1,
          validation_data=(val_vid, val_labels), callbacks=SaveBatchLoss2())
```

```
Epoch 1/50
WARNING:tensorflow:Model was constructed with shape (None, 20, 128, 128, 1) for
input KerasTensor(type_spec=TensorSpec(shape=(None, 20, 128, 128, 1),
dtype=tf.float32, name='input_6'), name='input_6', description="created by layer
'input_6'"), but it was called on an input with incompatible shape (2, 10, 128,
128, 1).
WARNING:tensorflow:Model was constructed with shape (None, 20, 128, 128, 1) for
input KerasTensor(type_spec=TensorSpec(shape=(None, 20, 128, 128, 1),
dtype=tf.float32, name='input_6'), name='input_6', description="created by layer
'input_6'"), but it was called on an input with incompatible shape (2, 10, 128,
128, 1).
10/10 [==============================] - ETA: 0s - loss: 0.0058 - accuracy:
0.9801 - precision_1: 0.0000e+00 - recall_1: 0.0000e+00 - f1_score:
0.2583WARNING:tensorflow:Model was constructed with shape (None, 20, 128, 128,
1) for input KerasTensor(type_spec=TensorSpec(shape=(None, 20, 128, 128, 1),
dtype=tf.float32, name='input_6'), name='input_6', description="created by layer
'input_6'"), but it was called on an input with incompatible shape (None, 10,
128, 128, 1).
10/10 [==============================] - 40s 4s/step - loss: 0.0058 - accuracy:
0.9801 - precision_1: 0.0000e+00 - recall_1: 0.0000e+00 - f1_score: 0.2583 -
val_loss: 0.0056 - val_accuracy: 0.9857 - val_precision_1: 0.0000e+00 -
val_recall_1: 0.0000e+00 - val_f1_score: 0.2577
Epoch 2/50
10/10 [==============================] - 36s 4s/step - loss: 0.0055 - accuracy:
0.9830 - precision_1: 0.0000e+00 - recall_1: 0.0000e+00 - f1_score: 0.2572 -
val_loss: 0.0050 - val_accuracy: 0.9780 - val_precision_1: 0.0000e+00 -
val_recall_1: 0.0000e+00 - val_f1_score: 0.2566
Epoch 3/50
10/10 [==============================] - 36s 4s/step - loss: 0.0047 - accuracy:
0.9552 - precision_1: 0.0000e+00 - recall_1: 0.0000e+00 - f1_score: 0.2560 -
val_loss: 0.0038 - val_accuracy: 0.9493 - val_precision_1: 0.0000e+00 -
val_recall_1: 0.0000e+00 - val_f1_score: 0.2554
Epoch 4/50
10/10 [==============================] - 36s 4s/step - loss: 0.0039 - accuracy:
0.9480 - precision_1: 0.4286 - recall_1: 2.5027e-04 - f1_score: 0.2549 -
val_loss: 0.0033 - val_accuracy: 0.9336 - val_precision_1: 1.0000 -
val_recall_1: 5.5991e-04 - val_f1_score: 0.2543
Epoch 5/50
10/10 [==============================] - 36s 4s/step - loss: 0.0032 - accuracy:
0.9518 - precision_1: 0.0000e+00 - recall_1: 0.0000e+00 - f1_score: 0.2538 -
val_loss: 0.0027 - val_accuracy: 0.9351 - val_precision_1: 1.0000 -
val_recall_1: 0.0011 - val_f1_score: 0.2532
```

```
Epoch 6/50
10/10 [==============================] - 41s 4s/step - loss: 0.0031 - accuracy:
0.9530 - precision_1: 0.6829 - recall_1: 0.0047 - f1_score: 0.2526 - val_loss:
0.0030 - val_accuracy: 0.9507 - val_precision_1: 0.7500 - val_recall_1: 0.0202 -
val_f1_score: 0.2521
Epoch 7/50
10/10 [==============================] - 41s 4s/step - loss: 0.0033 - accuracy:
0.9537 - precision_1: 0.6109 - recall_1: 0.0113 - f1_score: 0.2516 - val_loss:
0.0032 - val_accuracy: 0.8821 - val_precision_1: 0.6518 - val_recall_1: 0.0409 -
val_f1_score: 0.2511
Epoch 8/50
10/10 [==============================] - 36s 4s/step - loss: 0.0031 - accuracy:
0.9299 - precision_1: 0.5165 - recall_1: 0.0652 - f1_score: 0.2507 - val_loss:
0.0025 - val_accuracy: 0.9539 - val_precision_1: 0.5075 - val_recall_1: 0.1333 -
val_f1_score: 0.2505
Epoch 9/50
10/10 [==============================] - 36s 4s/step - loss: 0.0029 - accuracy:
0.9490 - precision_1: 0.4700 - recall_1: 0.1175 - f1_score: 0.2503 - val_loss:
0.0023 - val_accuracy: 0.9535 - val_precision_1: 0.5000 - val_recall_1: 0.1568 -
val_f1_score: 0.2502
Epoch 10/50
10/10 [==============================] - 36s 4s/step - loss: 0.0026 - accuracy:
0.9465 - precision_1: 0.5326 - recall_1: 0.1281 - f1_score: 0.2501 - val_loss:
0.0021 - val_accuracy: 0.9560 - val_precision_1: 0.5800 - val_recall_1: 0.1299 -
val_f1_score: 0.2500
Epoch 11/50
10/10 [==============================] - 36s 4s/step - loss: 0.0022 - accuracy:
0.9458 - precision_1: 0.5736 - recall_1: 0.1271 - f1_score: 0.2498 - val_loss:
0.0019 - val_accuracy: 0.9473 - val_precision_1: 0.5991 - val_recall_1: 0.1557 -
val_f1_score: 0.2498
Epoch 12/50
10/10 [==============================] - 35s 4s/step - loss: 0.0021 - accuracy:
0.9525 - precision_1: 0.5947 - recall_1: 0.1184 - f1_score: 0.2496 - val_loss:
0.0018 - val_accuracy: 0.9465 - val_precision_1: 0.5565 - val_recall_1: 0.1405 -
val_f1_score: 0.2495
Epoch 13/50
10/10 [==============================] - 33s 3s/step - loss: 0.0019 - accuracy:
0.9542 - precision_1: 0.5711 - recall_1: 0.1226 - f1_score: 0.2494 - val_loss:
0.0016 - val_accuracy: 0.9616 - val_precision_1: 0.5773 - val_recall_1: 0.1109 -
val_f1_score: 0.2492
Epoch 14/50
10/10 [==============================] - 34s 3s/step - loss: 0.0018 - accuracy:
0.9549 - precision_1: 0.5655 - recall_1: 0.0925 - f1_score: 0.2491 - val_loss:
0.0015 - val_accuracy: 0.9658 - val_precision_1: 0.6000 - val_recall_1: 0.0739 -
val_f1_score: 0.2488
Epoch 15/50
10/10 [==============================] - 34s 3s/step - loss: 0.0017 - accuracy:
0.9497 - precision_1: 0.6232 - recall_1: 0.0817 - f1_score: 0.2485 - val_loss:
```

```
0.0014 - val_accuracy: 0.9589 - val_precision_1: 0.5851 - val_recall_1: 0.0616 -
val_f1_score: 0.2483
Epoch 16/50
10/10 [==============================] - 36s 4s/step - loss: 0.0016 - accuracy:
0.9504 - precision_1: 0.6567 - recall_1: 0.0732 - f1_score: 0.2480 - val_loss:
0.0013 - val_accuracy: 0.9644 - val_precision_1: 0.6373 - val_recall_1: 0.0728 -
val_f1_score: 0.2477
Epoch 17/50
10/10 [==============================] - 35s 4s/step - loss: 0.0015 - accuracy:
0.9547 - precision_1: 0.6742 - recall_1: 0.0767 - f1_score: 0.2475 - val_loss:
0.0013 - val_accuracy: 0.9678 - val_precision_1: 0.6386 - val_recall_1: 0.0722 -
val_f1_score: 0.2472
Epoch 18/50
10/10 [==============================] - 34s 3s/step - loss: 0.0016 - accuracy:
0.9607 - precision_1: 0.6859 - recall_1: 0.0902 - f1_score: 0.2470 - val_loss:
0.0013 - val_accuracy: 0.9667 - val_precision_1: 0.6694 - val_recall_1: 0.0918 -
val_f1_score: 0.2468
Epoch 19/50
10/10 [==============================] - 34s 3s/step - loss: 0.0014 - accuracy:
0.9574 - precision_1: 0.6756 - recall_1: 0.0860 - f1_score: 0.2466 - val_loss:
0.0012 - val_accuracy: 0.9650 - val_precision_1: 0.6579 - val_recall_1: 0.0700 -
val_f1_score: 0.2463
Epoch 20/50
10/10 [==============================] - 35s 3s/step - loss: 0.0014 - accuracy:
0.9578 - precision_1: 0.6717 - recall_1: 0.0669 - f1_score: 0.2461 - val_loss:
0.0012 - val_accuracy: 0.9643 - val_precision_1: 0.6420 - val_recall_1: 0.0633 -
val_f1_score: 0.2458
Epoch 21/50
10/10 [==============================] - 36s 4s/step - loss: 0.0013 - accuracy:
0.9643 - precision_1: 0.6550 - recall_1: 0.0672 - f1_score: 0.2455 - val_loss:
0.0012 - val_accuracy: 0.9583 - val_precision_1: 0.5927 - val_recall_1: 0.0823 -
val_f1_score: 0.2452
Epoch 22/50
10/10 [==============================] - 35s 3s/step - loss: 0.0013 - accuracy:
0.9650 - precision_1: 0.6960 - recall_1: 0.1003 - f1_score: 0.2450 - val_loss:
0.0011 - val_accuracy: 0.9586 - val_precision_1: 0.6986 - val_recall_1: 0.1103 -
val_f1_score: 0.2449
Epoch 23/50
10/10 [==============================] - 36s 4s/step - loss: 0.0013 - accuracy:
0.9597 - precision_1: 0.7115 - recall_1: 0.0979 - f1_score: 0.2447 - val_loss:
0.0011 - val_accuracy: 0.9698 - val_precision_1: 0.6991 - val_recall_1: 0.0845 -
val_f1_score: 0.2446
Epoch 24/50
10/10 [==============================] - 36s 4s/step - loss: 0.0012 - accuracy:
0.9688 - precision_1: 0.7190 - recall_1: 0.0939 - f1_score: 0.2444 - val_loss:
9.8384e-04 - val_accuracy: 0.9756 - val_precision_1: 0.6840 - val_recall_1:
0.0885 - val_f1_score: 0.2442
Epoch 25/50
```

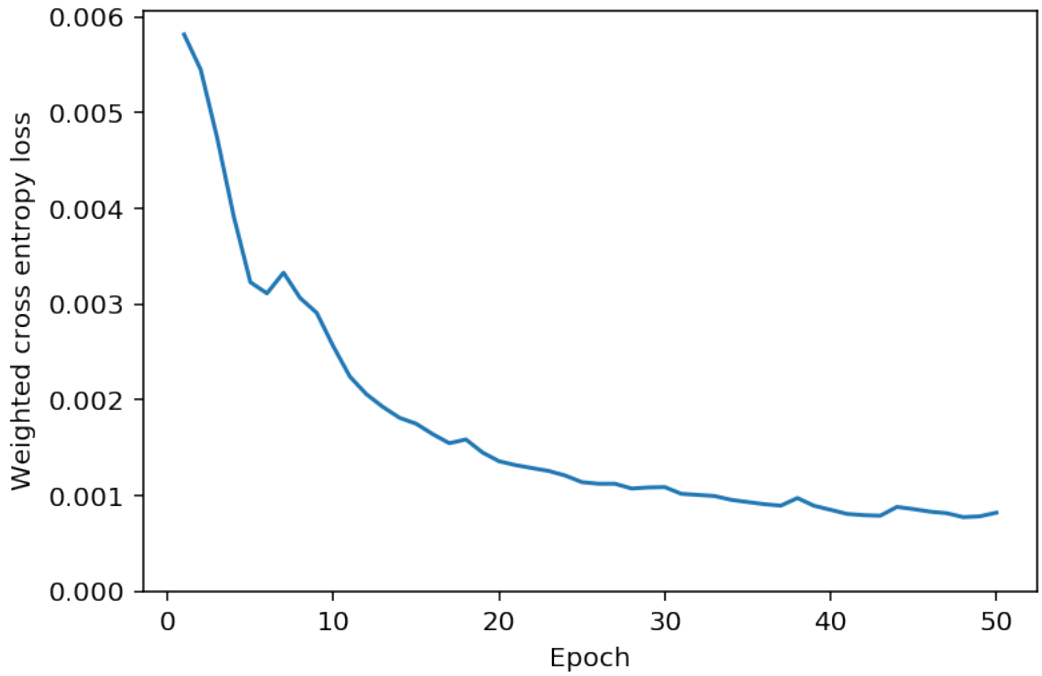
```
10/10 [==============================] - 37s 4s/step - loss: 0.0011 - accuracy:
0.9691 - precision_1: 0.7038 - recall_1: 0.0975 - f1_score: 0.2440 - val_loss:
9.7861e-04 - val_accuracy: 0.9729 - val_precision_1: 0.6640 - val_recall_1:
0.0941 - val_f1_score: 0.2439
Epoch 26/50
10/10 [==============================] - 36s 4s/step - loss: 0.0011 - accuracy:
0.9650 - precision_1: 0.7073 - recall_1: 0.1080 - f1_score: 0.2437 - val_loss:
9.9471e-04 - val_accuracy: 0.9648 - val_precision_1: 0.6690 - val_recall_1:
0.1053 - val_f1_score: 0.2436
Epoch 27/50
10/10 [==============================] - 37s 4s/step - loss: 0.0011 - accuracy:
0.9694 - precision_1: 0.7228 - recall_1: 0.1009 - f1_score: 0.2434 - val_loss:
9.4111e-04 - val_accuracy: 0.9697 - val_precision_1: 0.7111 - val_recall_1:
0.1075 - val_f1_score: 0.2433
Epoch 28/50
10/10 [==============================] - 36s 4s/step - loss: 0.0011 - accuracy:
0.9689 - precision_1: 0.7158 - recall_1: 0.1065 - f1_score: 0.2432 - val_loss:
8.7831e-04 - val_accuracy: 0.9710 - val_precision_1: 0.7257 - val_recall_1:
0.1170 - val_f1_score: 0.2430
Epoch 29/50
10/10 [==============================] - 37s 4s/step - loss: 0.0011 - accuracy:
0.9659 - precision_1: 0.7252 - recall_1: 0.1068 - f1_score: 0.2429 - val_loss:
9.4258e-04 - val_accuracy: 0.9627 - val_precision_1: 0.7068 - val_recall_1:
0.0985 - val_f1_score: 0.2428
Epoch 30/50
10/10 [==============================] - 37s 4s/step - loss: 0.0011 - accuracy:
0.9639 - precision_1: 0.7334 - recall_1: 0.1014 - f1_score: 0.2426 - val_loss:
0.0010 - val_accuracy: 0.9594 - val_precision_1: 0.6990 - val_recall_1: 0.1170 -
val_f1_score: 0.2425
Epoch 31/50
10/10 [==============================] - 36s 4s/step - loss: 0.0010 - accuracy:
0.9695 - precision_1: 0.7252 - recall_1: 0.1054 - f1_score: 0.2424 - val_loss:
9.2951e-04 - val_accuracy: 0.9635 - val_precision_1: 0.7323 - val_recall_1:
0.1041 - val_f1_score: 0.2422
Epoch 32/50
10/10 [==============================] - 37s 4s/step - loss: 0.0010 - accuracy:
0.9685 - precision_1: 0.7216 - recall_1: 0.0891 - f1_score: 0.2421 - val_loss:
9.3777e-04 - val_accuracy: 0.9599 - val_precision_1: 0.7174 - val_recall_1:
0.1109 - val_f1_score: 0.2419
Epoch 33/50
10/10 [==============================] - 37s 4s/step - loss: 9.9412e-04 -
accuracy: 0.9690 - precision_1: 0.7394 - recall_1: 0.1044 - f1_score: 0.2418 -
val_loss: 8.3993e-04 - val_accuracy: 0.9690 - val_precision_1: 0.7113 -
val_recall_1: 0.1159 - val_f1_score: 0.2416
Epoch 34/50
10/10 [==============================] - 37s 4s/step - loss: 9.5458e-04 -
accuracy: 0.9705 - precision_1: 0.7168 - recall_1: 0.1058 - f1_score: 0.2415 -
val_loss: 8.7521e-04 - val_accuracy: 0.9667 - val_precision_1: 0.7051 -
```

```
val_recall_1: 0.1165 - val_f1_score: 0.2414
Epoch 35/50
10/10 [==============================] - 37s 4s/step - loss: 9.3119e-04 -
accuracy: 0.9752 - precision_1: 0.7557 - recall_1: 0.1071 - f1_score: 0.2413 -
val_loss: 8.1733e-04 - val_accuracy: 0.9709 - val_precision_1: 0.7382 -
val_recall_1: 0.1137 - val_f1_score: 0.2412
Epoch 36/50
10/10 [==============================] - 37s 4s/step - loss: 9.0908e-04 -
accuracy: 0.9746 - precision_1: 0.7546 - recall_1: 0.1100 - f1_score: 0.2410 -
val_loss: 7.8211e-04 - val_accuracy: 0.9718 - val_precision_1: 0.7668 -
val_recall_1: 0.1344 - val_f1_score: 0.2410
Epoch 37/50
10/10 [==============================] - 37s 4s/step - loss: 8.9394e-04 -
accuracy: 0.9720 - precision_1: 0.7650 - recall_1: 0.1445 - f1_score: 0.2409 -
val_loss: 7.6925e-04 - val_accuracy: 0.9758 - val_precision_1: 0.8787 -
val_recall_1: 0.1501 - val_f1_score: 0.2410
Epoch 38/50
10/10 [==============================] - 37s 4s/step - loss: 9.7315e-04 -
accuracy: 0.9709 - precision_1: 0.8095 - recall_1: 0.1543 - f1_score: 0.2410 -
val_loss: 7.6086e-04 - val_accuracy: 0.9720 - val_precision_1: 0.8386 -
val_recall_1: 0.2066 - val_f1_score: 0.2411
Epoch 39/50
10/10 [==============================] - 38s 4s/step - loss: 8.9251e-04 -
accuracy: 0.9707 - precision_1: 0.7681 - recall_1: 0.1931 - f1_score: 0.2412 -
val_loss: 7.1644e-04 - val_accuracy: 0.9733 - val_precision_1: 0.8194 -
val_recall_1: 0.2133 - val_f1_score: 0.2414
Epoch 40/50
10/10 [==============================] - 37s 4s/step - loss: 8.5011e-04 -
accuracy: 0.9738 - precision_1: 0.7396 - recall_1: 0.1881 - f1_score: 0.2416 -
val_loss: 6.8593e-04 - val_accuracy: 0.9740 - val_precision_1: 0.7744 -
val_recall_1: 0.1999 - val_f1_score: 0.2417
Epoch 41/50
10/10 [==============================] - 36s 4s/step - loss: 8.0731e-04 -
accuracy: 0.9746 - precision_1: 0.7439 - recall_1: 0.1766 - f1_score: 0.2418 -
val_loss: 6.7548e-04 - val_accuracy: 0.9723 - val_precision_1: 0.7719 -
val_recall_1: 0.2027 - val_f1_score: 0.2420
Epoch 42/50
10/10 [==============================] - 36s 4s/step - loss: 7.9422e-04 -
accuracy: 0.9749 - precision_1: 0.7557 - recall_1: 0.1791 - f1_score: 0.2421 -
val_loss: 6.9754e-04 - val_accuracy: 0.9687 - val_precision_1: 0.7884 -
val_recall_1: 0.2357 - val_f1_score: 0.2422
Epoch 43/50
10/10 [==============================] - 36s 4s/step - loss: 7.8828e-04 -
accuracy: 0.9755 - precision_1: 0.7532 - recall_1: 0.1828 - f1_score: 0.2424 -
val_loss: 8.0602e-04 - val_accuracy: 0.9582 - val_precision_1: 0.7554 -
val_recall_1: 0.2335 - val_f1_score: 0.2425
Epoch 44/50
10/10 [==============================] - 36s 4s/step - loss: 8.8035e-04 -
```

```
accuracy: 0.9685 - precision_1: 0.7438 - recall_1: 0.1708 - f1_score: 0.2426 -
val_loss: 8.5405e-04 - val_accuracy: 0.9522 - val_precision_1: 0.7214 -
val_recall_1: 0.2363 - val_f1_score: 0.2427
Epoch 45/50
10/10 [==============================] - 36s 4s/step - loss: 8.5774e-04 -
accuracy: 0.9701 - precision_1: 0.7268 - recall_1: 0.1795 - f1_score: 0.2428 -
val_loss: 8.5326e-04 - val_accuracy: 0.9521 - val_precision_1: 0.7012 -
val_recall_1: 0.2615 - val_f1_score: 0.2430
Epoch 46/50
10/10 [==============================] - 36s 4s/step - loss: 8.3066e-04 -
accuracy: 0.9733 - precision_1: 0.7092 - recall_1: 0.1916 - f1_score: 0.2431 -
val_loss: 7.8960e-04 - val_accuracy: 0.9589 - val_precision_1: 0.6987 -
val_recall_1: 0.2324 - val_f1_score: 0.2433
Epoch 47/50
10/10 [==============================] - 37s 4s/step - loss: 8.1517e-04 -
accuracy: 0.9743 - precision_1: 0.7005 - recall_1: 0.1906 - f1_score: 0.2434 -
val_loss: 6.9579e-04 - val_accuracy: 0.9704 - val_precision_1: 0.7350 -
val_recall_1: 0.2531 - val_f1_score: 0.2436
Epoch 48/50
10/10 [==============================] - 36s 4s/step - loss: 7.7358e-04 -
accuracy: 0.9748 - precision_1: 0.6978 - recall_1: 0.1942 - f1_score: 0.2438 -
val_loss: 7.0020e-04 - val_accuracy: 0.9679 - val_precision_1: 0.7355 -
val_recall_1: 0.2912 - val_f1_score: 0.2440
Epoch 49/50
10/10 [==============================] - 37s 4s/step - loss: 7.8194e-04 -
accuracy: 0.9747 - precision_1: 0.7088 - recall_1: 0.2547 - f1_score: 0.2443 -
val_loss: 7.7402e-04 - val_accuracy: 0.9606 - val_precision_1: 0.7345 -
val_recall_1: 0.3639 - val_f1_score: 0.2447
Epoch 50/50
10/10 [==============================] - 40s 4s/step - loss: 8.2024e-04 -
accuracy: 0.9730 - precision_1: 0.7444 - recall_1: 0.2573 - f1_score: 0.2450 -
val_loss: 8.3197e-04 - val_accuracy: 0.9537 - val_precision_1: 0.7039 -
val_recall_1: 0.3701 - val_f1_score: 0.2454
```

[90]:
```
<keras.callbacks.History at 0x1e0992282c8>
```

[92]:
```python
plt.plot(range(1,epochs+1), loss_list2[num_batches-1::num_batches])
plt.xlabel('Epoch')
plt.ylabel('Weighted cross entropy loss')
plt.ylim(0)
plt.show()
```



[93]:
```
model2.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d_6 (ConvLSTM2D)  (None, 20, 128, 128, 12)  5664
_________________________________________________________________
conv_lst_m2d_7 (ConvLSTM2D)  (None, 20, 128, 128, 16)  16192
_________________________________________________________________
conv_lst_m2d_8 (ConvLSTM2D)  (None, 20, 128, 128, 2)   1304
=================================================================
Total params: 23,160
Trainable params: 23,160
Non-trainable params: 0
_________________________________________________________________
```

1.0.3 Visualizing the results

[94]:
```
Nt = 20
pt_test = Particle_Tracking_Training_Data(Nt)
vid, labels, tracks = pt_test(kappa, a, IbackLevel, Nparticles_det,␣
 ↪sigma_motion)
input_vid = tf.expand_dims(vid[:,::2,::2], 3)
```

[95]:
```
# get model predictions
#pred = model1.predict(input_vid)
pred = model2.predict(tf.expand_dims(input_vid,0))[0]
```

[100]:
```
# convert predictions to binary output
threshold = 0.4
pred_bin = pred.copy()
pred_bin[pred_bin <= threshold] = 0
pred_bin[pred_bin > threshold] = 1
```

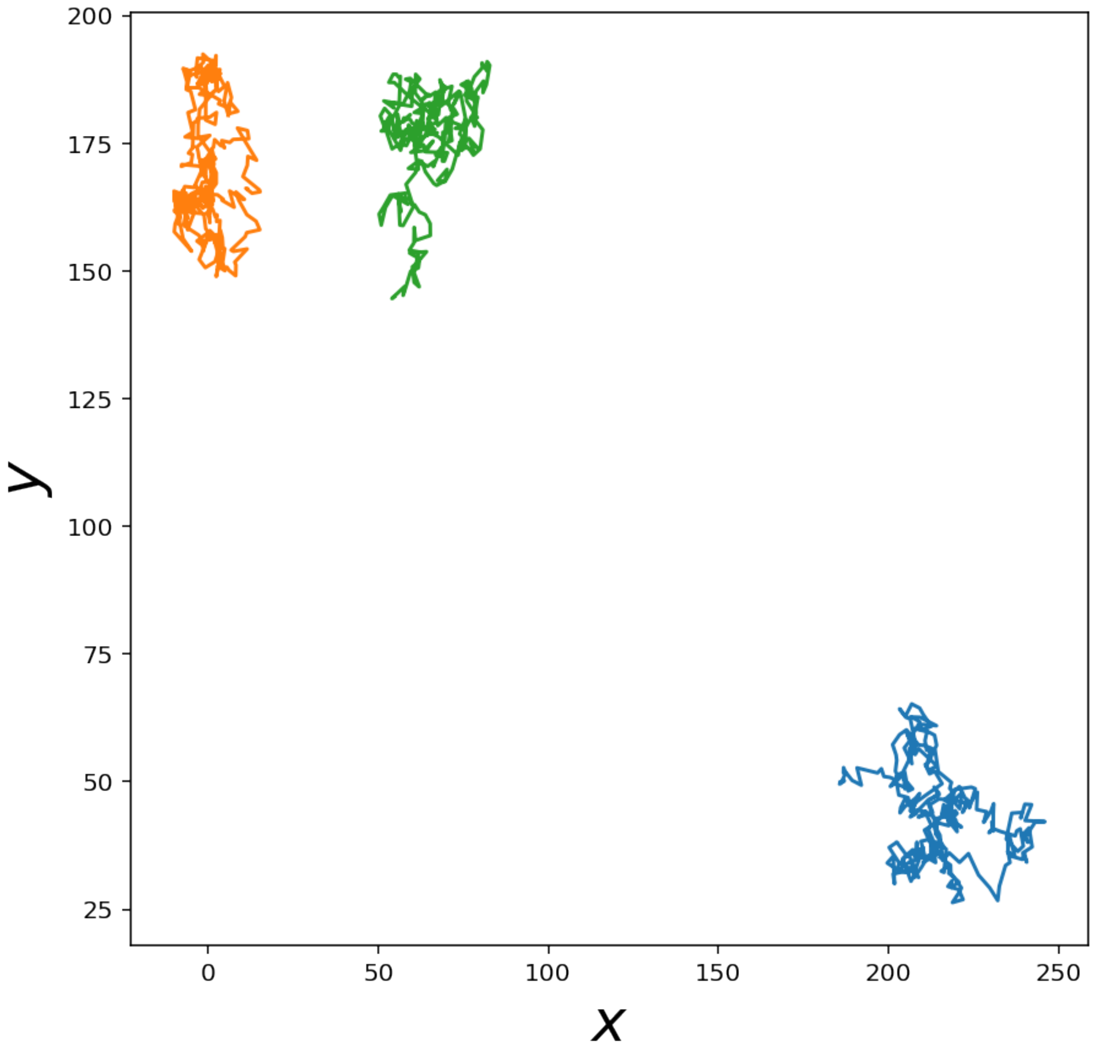
[101]:
```
@interact(t=(0, Nt-1, 1))
def plotfn(t=0, show_tracks=True):
    fig = figure(1, [14, 7])
    fig.add_subplot(131)
    imshow(vid[t], origin='lower')
    #if show_tracks:
    #    plot(tracks[t, :, 0], tracks[t, :, 1], 'rx')
    xlim(0, 255)
    ylim(0, 255)
    plt.title('Input image', fontsize=25)

    fig.add_subplot(132)
    imshow(vid[t], origin='lower')
    imshow(pred_bin[t, ..., 1], origin='lower')
    plt.title('Predicted particles', fontsize=25)

    fig.add_subplot(133)
    imshow(vid[t], origin='lower')
    imshow(labels[t, ..., 1], origin='lower')
    plt.title('Ground truth', fontsize=25)
```

```
interactive(children=(IntSlider(value=0, description='t', max=19),␣
↪Checkbox(value=True, description='show_trac…
```

[ ]:
```
```

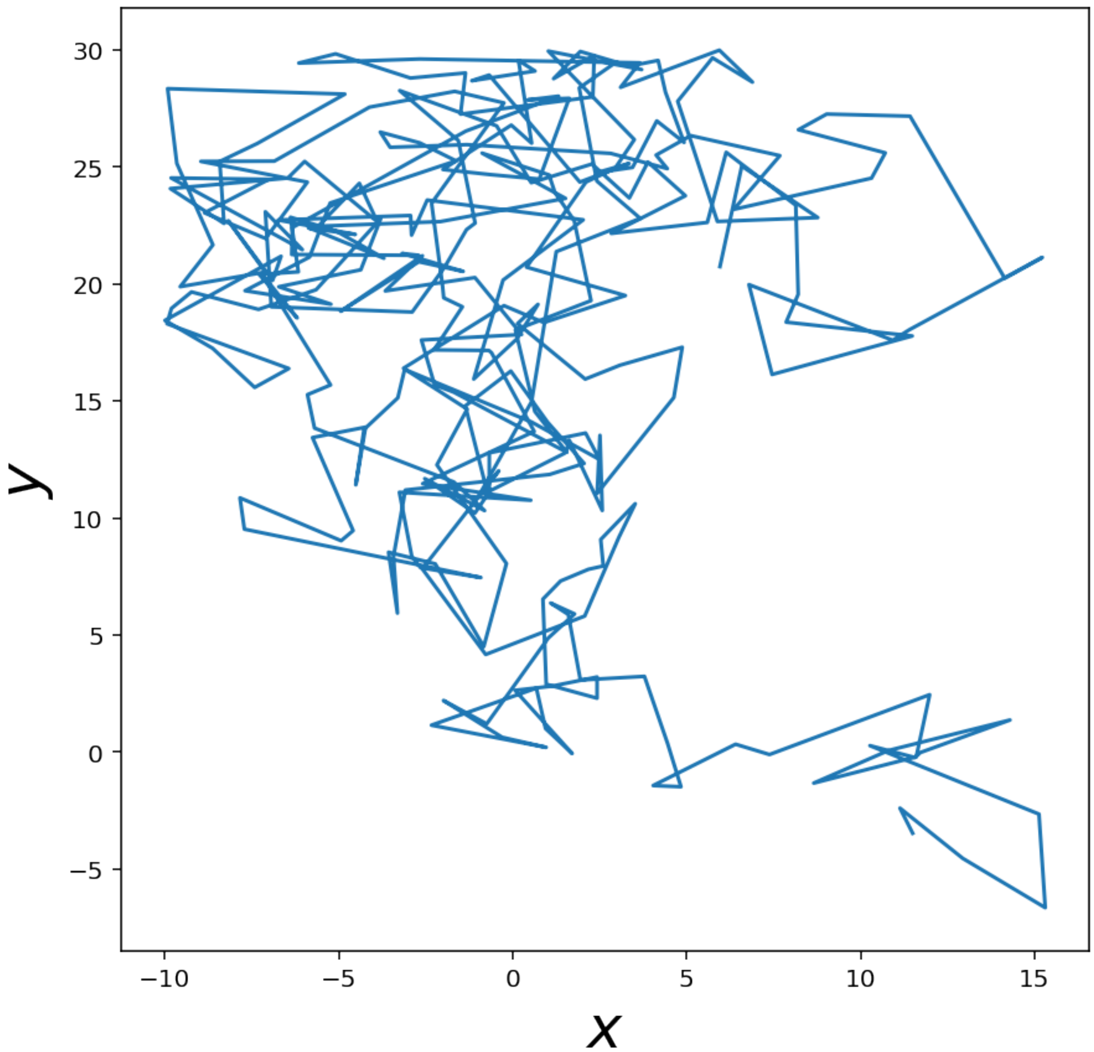
2 Tracking the Particles

[9]:
```
figure(1, [7, 7])
plot(tracks[..., 0], tracks[..., 1])
#xlim(-12, 300)
#ylim(-12, 300)
xlabel(r'$x$', fontsize=24)
ylabel(r'$y$', fontsize=24);
```



[ ]:

[11]:
```
figure(1, [7, 7])
plot(tracks[:,1,0], tracks[:,1,2])
#xlim(-12, 300)
#ylim(-12, 300)
xlabel(r'$x$', fontsize=24)
ylabel(r'$y$', fontsize=24);
```




```
[115]: X=np.array(tracks[:,0,0])
       X_scaled=(X-np.mean(X))/np.std(X)
       Y=np.array(tracks[:,0,1])
       Y_scaled=(Y-np.mean(Y))/np.std(Y)
```

```
[169]: def get_data(data):
           a_train=[]
           b_train=[]
           for i in range(10,len(data)):
               a_train.append(data[i-10:i])
               b_train.append(data[i])
           a_train, b_train = np.array(a_train), np.array(b_train)
           return(a_train, b_train)
```

[170]:
```
X_train, y_train=get_data(X_scaled[:250])
X_test, y_test=get_data(X_scaled[250:])
Xy_train, yy_train=get_data(Y_scaled[:250])
Xy_test, yy_test=get_data(Y_scaled[250:])
```

[ ]:
```

```

[ ]:
```

```

[247]:
```
X_train = []
y_train = []
X_train_scaled=X_scaled[:250]
#train_data=train_data.numpy()
for i in range(30,len(X_scaled)):
    X_train.append(X_scaled[i-30:i])
    y_train.append(X_scaled[i])
X_train, y_train = np.array(X_train), np.array(y_train)
```

[248]:
```
Xy_train = []
yy_train = []
Xy_train_scaled=Y_scaled[:250]
#train_data=train_data.numpy()
```

[ ]:
```
X_test = []
y_test = []

for i in range(30,len(X_scaled)):
    X_train.append(X_scaled[i-30:i])
    y_train.append(X_scaled[i])
X_train, y_train = np.array(X_train), np.array(y_train)

X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
```

[ ]:
```

```

[ ]:
```

```

[ ]:
```

```

[210]:
```

```

[ ]:
```

```

[ ]:
```

```

[76]:
```
opt=tf.keras.optimizers.RMSprop(
    learning_rate=0.001
)
model = Sequential()
model.add(CuDNNLSTM(units=32,return_sequences=True, input_shape=(X_train.
 ↪shape[1],1)))
#model.add(CuDNNLSTM(units=256,return_sequences=True))
#model.add(CuDNNLSTM(units=128,return_sequences=True))
#model.add(Dropout(0.2))
model.add(CuDNNLSTM(units=16,return_sequences=True))
#model.add(CuDNNLSTM(units=32,return_sequences=True))
model.add(Dropout(0.2))
model.add(CuDNNLSTM(units=8,return_sequences=True))
#model.add(CuDNNLSTM(units=8,return_sequences=True))
model.add(CuDNNLSTM(units=4,return_sequences=True))
model.add(Flatten())
model.add(Dense(1))
model.compile(optimizer=opt,loss='mean_squared_error',metrics=["mae"])
```

[171]:
```
model=Sequential()
model.add(CuDNNLSTM(units=128,return_sequences=True, input_shape=(X_train.
 ↪shape[1],1)) )
model.add(Dropout(0.2))
model.add(CuDNNLSTM(units=64,return_sequences=True) )
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(units=1))
# Compiling the RNN
model.compile(optimizer='rmsprop',loss='mean_squared_error')
# Fitting to the training set
#model.fit(X_train,y_train,epochs=50,batch_size=16)
```

[172]:
```
model.fit(Xy_train,yy_train,epochs=50,batch_size=16,validation_data=(Xy_test[:
 ↪2], yy_test[:2]))
y_pred = model.predict(Xy_test)
```

```
Epoch 1/50
15/15 [==============================] - 3s 51ms/step - loss: 0.3942 - val_loss:
0.0273
Epoch 2/50
15/15 [==============================] - 0s 17ms/step - loss: 0.2671 - val_loss:
0.0733
Epoch 3/50
15/15 [==============================] - 0s 14ms/step - loss: 0.2214 - val_loss:
0.0130
Epoch 4/50
15/15 [==============================] - 0s 15ms/step - loss: 0.2123 - val_loss:
```

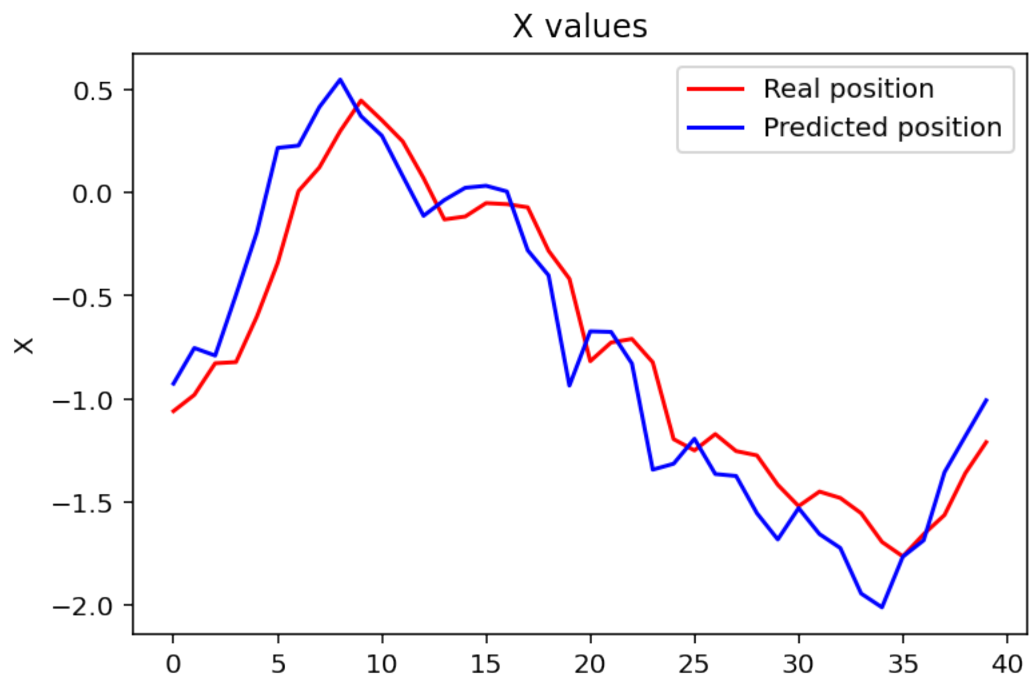
```
0.0633
Epoch 5/50
15/15 [==============================] - 0s 14ms/step - loss: 0.2002 - val_loss:
0.0469
Epoch 6/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1866 - val_loss:
0.0533
Epoch 7/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1691 - val_loss:
0.1987
Epoch 8/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1565 - val_loss:
0.1699
Epoch 9/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1450 - val_loss:
0.0362
Epoch 10/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1434 - val_loss:
0.0080
Epoch 11/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1394 - val_loss:
0.0292
Epoch 12/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1349 - val_loss:
0.0361
Epoch 13/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1274 - val_loss:
0.0794
Epoch 14/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1211 - val_loss:
0.0272
Epoch 15/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1308 - val_loss:
0.1241
Epoch 16/50
15/15 [==============================] - 0s 15ms/step - loss: 0.1114 - val_loss:
0.1256
Epoch 17/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1246 - val_loss:
0.0750
Epoch 18/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1083 - val_loss:
0.0807
Epoch 19/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1047 - val_loss:
0.3538
Epoch 20/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1097 - val_loss:
```

```
0.1290
Epoch 21/50
15/15 [==============================] - 0s 14ms/step - loss: 0.1005 - val_loss:
0.0386
Epoch 22/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0917 - val_loss:
0.0923
Epoch 23/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0891 - val_loss:
0.1447
Epoch 24/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0866 - val_loss:
0.0242
Epoch 25/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0836 - val_loss:
0.0344
Epoch 26/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0791 - val_loss:
0.0398
Epoch 27/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0812 - val_loss:
0.0130
Epoch 28/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0772 - val_loss:
0.0553
Epoch 29/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0874 - val_loss:
0.0650
Epoch 30/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0785 - val_loss:
0.0113
Epoch 31/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0745 - val_loss:
0.0046
Epoch 32/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0760 - val_loss:
0.0064
Epoch 33/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0692 - val_loss:
0.0457
Epoch 34/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0792 - val_loss:
0.0433
Epoch 35/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0727 - val_loss:
0.0077
Epoch 36/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0660 - val_loss:
```

```
0.0548
Epoch 37/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0626 - val_loss:
0.1822
Epoch 38/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0756 - val_loss:
0.0087
Epoch 39/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0612 - val_loss:
0.0471
Epoch 40/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0723 - val_loss:
0.0122
Epoch 41/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0667 - val_loss:
0.0166
Epoch 42/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0701 - val_loss:
0.0294
Epoch 43/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0719 - val_loss:
0.0903
Epoch 44/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0723 - val_loss:
0.0355
Epoch 45/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0615 - val_loss:
0.0617
Epoch 46/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0592 - val_loss:
0.0164
Epoch 47/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0651 - val_loss:
0.0809
Epoch 48/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0608 - val_loss:
0.0430
Epoch 49/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0639 - val_loss:
0.0327
Epoch 50/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0631 - val_loss:
0.0348
```

[173]:
```
plt.plot(y_pred, color='red',label='Real position')
plt.plot(yy_test, color='blue',label='Predicted position');
plt.title('X values')
```

```
#plt.xlabel('Date')
plt.ylabel('X')
plt.legend()
plt.show()

MSE1=mean_squared_error(y_pred, yy_test, squared=False)
print("MSE = ", MSE1)
```




```
MSE =  0.23956507
```

[174]:
```
model.fit(X_train,y_train,epochs=50,batch_size=16, validation_data=(X_test[:2],
  y_test[:2]))
x_pred = model.predict(X_test)
```

```
Epoch 1/50
15/15 [==============================] - 0s 20ms/step - loss: 0.0490 - val_loss:
0.0503
Epoch 2/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0478 - val_loss:
0.0663
Epoch 3/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0443 - val_loss:
0.0869
Epoch 4/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0480 - val_loss:
```

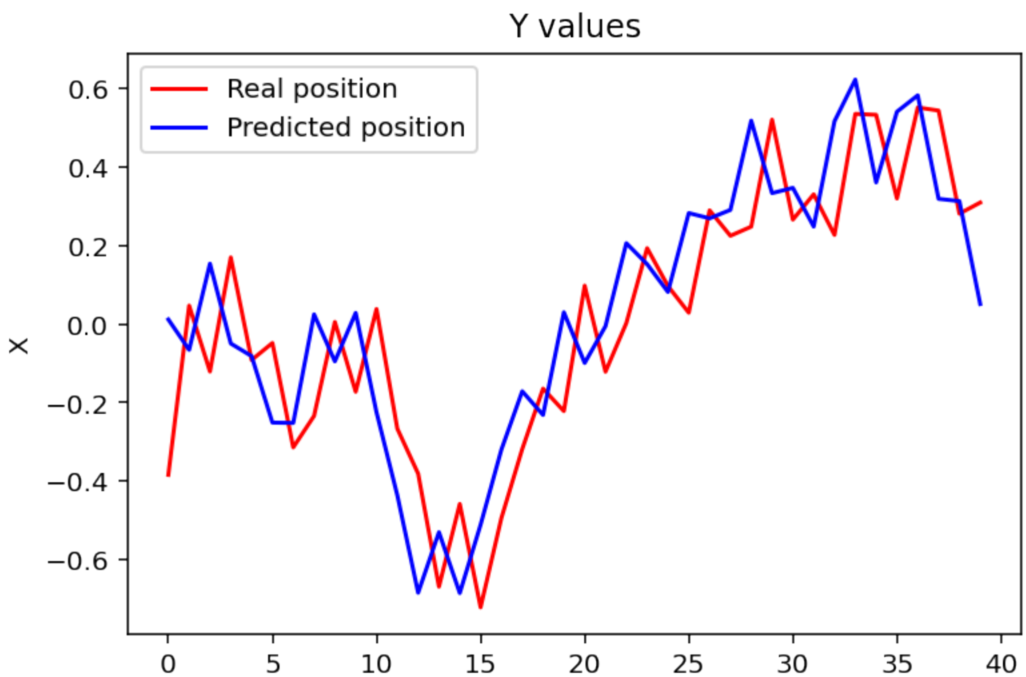
```
0.0574
Epoch 5/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0452 - val_loss:
0.0537
Epoch 6/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0460 - val_loss:
0.0717
Epoch 7/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0409 - val_loss:
0.0620
Epoch 8/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0392 - val_loss:
0.0819
Epoch 9/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0424 - val_loss:
0.0612
Epoch 10/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0402 - val_loss:
0.0589
Epoch 11/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0414 - val_loss:
0.0709
Epoch 12/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0466 - val_loss:
0.0882
Epoch 13/50
15/15 [==============================] - 0s 16ms/step - loss: 0.0358 - val_loss:
0.0707
Epoch 14/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0393 - val_loss:
0.1082
Epoch 15/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0424 - val_loss:
0.0858
Epoch 16/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0446 - val_loss:
0.0722
Epoch 17/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0465 - val_loss:
0.1270
Epoch 18/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0450 - val_loss:
0.0623
Epoch 19/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0453 - val_loss:
0.0855
Epoch 20/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0437 - val_loss:
```

```
0.0955
Epoch 21/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0375 - val_loss:
0.0755
Epoch 22/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0426 - val_loss:
0.0796
Epoch 23/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0400 - val_loss:
0.0636
Epoch 24/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0423 - val_loss:
0.0743
Epoch 25/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0378 - val_loss:
0.0720
Epoch 26/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0398 - val_loss:
0.0781
Epoch 27/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0428 - val_loss:
0.0781
Epoch 28/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0373 - val_loss:
0.0836
Epoch 29/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0444 - val_loss:
0.0846
Epoch 30/50
15/15 [==============================] - 0s 16ms/step - loss: 0.0395 - val_loss:
0.1001
Epoch 31/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0420 - val_loss:
0.0917
Epoch 32/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0385 - val_loss:
0.0747
Epoch 33/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0390 - val_loss:
0.0739
Epoch 34/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0404 - val_loss:
0.0949
Epoch 35/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0396 - val_loss:
0.0648
Epoch 36/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0358 - val_loss:
```

```
0.0788
Epoch 37/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0376 - val_loss:
0.0836
Epoch 38/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0494 - val_loss:
0.0925
Epoch 39/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0369 - val_loss:
0.0889
Epoch 40/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0383 - val_loss:
0.0903
Epoch 41/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0369 - val_loss:
0.0861
Epoch 42/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0359 - val_loss:
0.0741
Epoch 43/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0395 - val_loss:
0.0815
Epoch 44/50
15/15 [==============================] - 0s 16ms/step - loss: 0.0371 - val_loss:
0.0868
Epoch 45/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0413 - val_loss:
0.0813
Epoch 46/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0415 - val_loss:
0.0933
Epoch 47/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0372 - val_loss:
0.0847
Epoch 48/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0385 - val_loss:
0.0904
Epoch 49/50
15/15 [==============================] - 0s 14ms/step - loss: 0.0394 - val_loss:
0.1009
Epoch 50/50
15/15 [==============================] - 0s 15ms/step - loss: 0.0371 - val_loss:
0.0846
```

[175]:
```
plt.plot(x_pred, color='red',label='Real position')
plt.plot(y_test, color='blue',label='Predicted position');
plt.title('Y values')
```

```
#plt.xlabel('Date')
plt.ylabel('X')
plt.legend()
plt.show()

MSE1=mean_squared_error(x_pred, y_test, squared=False)
print("MSE = ", MSE1)
```



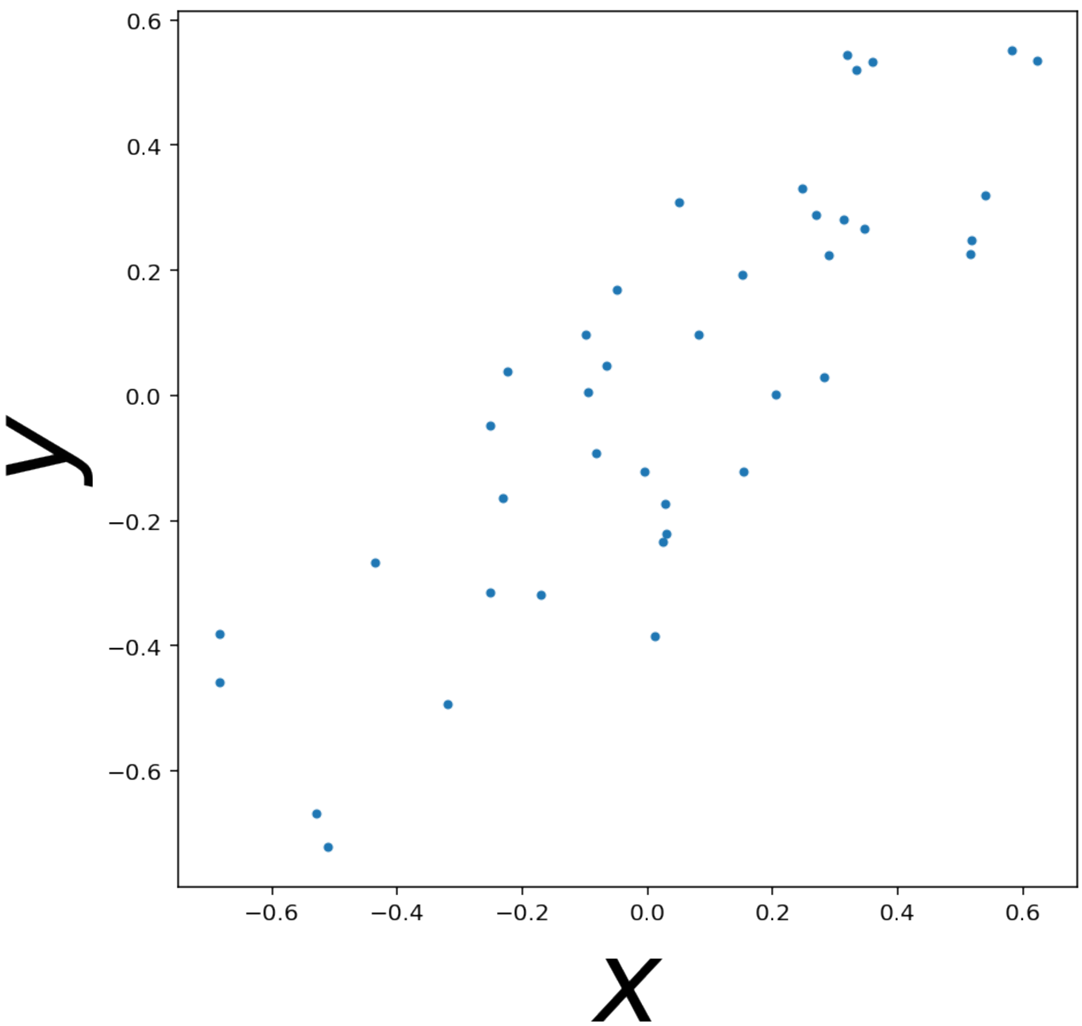

```
MSE =  0.190686
```

[139]:
```
x_pred
```

[139]:
```
array([[ 0.02891346],
       [-0.25876385],
       [-0.09110858],
       [ 0.08409639],
       [-0.03282701],
       [-0.1471768 ],
       [ 0.1513299 ],
       [ 0.12525119],
       [ 0.06137033],
       [ 0.47542968],
       [ 0.11650269],
       [ 0.24012011],
       [ 0.12309794],
```

```
[ 0.4640012 ],
[ 0.4176118 ],
[ 0.29948637],
[ 0.4742349 ],
[ 0.4235113 ],
[ 0.16309182],
[ 0.21584761]], dtype=float32)
```

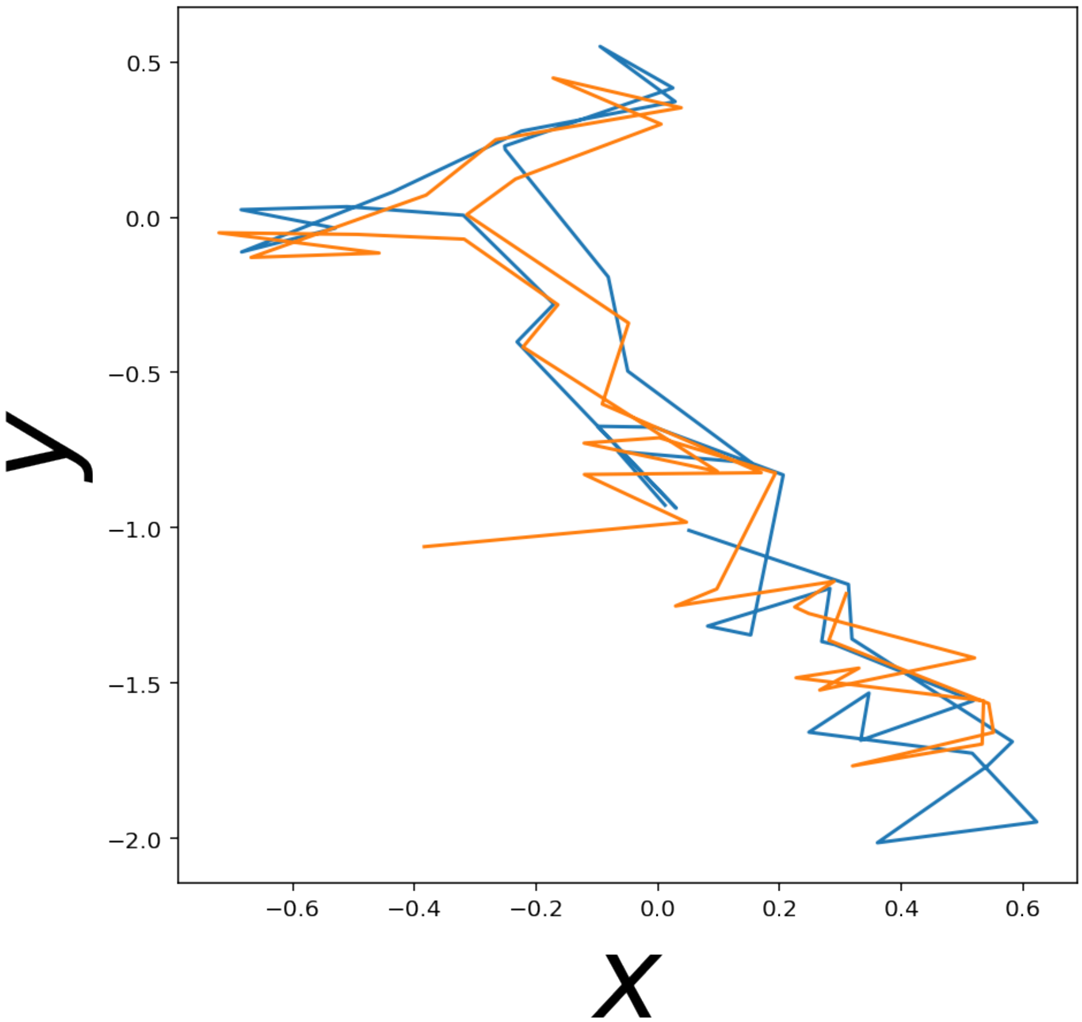
[176]:
```
figure(1, [7, 7])
plot(y_test,x_pred,'.')
#xlim(50, 200)
#ylim(50, 200)


xlabel(r'$x$', fontsize=50)
ylabel(r'$y$', fontsize=50);
```



[ ]:

[178]:

```
figure(1, [7, 7])
plot(y_test,yy_test)
#xlim(50, 200)
#ylim(50, 200)
plot(x_pred, y_pred)

xlabel(r'$x$', fontsize=50)
ylabel(r'$y$', fontsize=50);
```



[227]:

```
tracks[250:,1,0]
```

[227]: <tf.Tensor: shape=(50,), dtype=float32, numpy=

```
array([140.36302, 140.77554, 142.66766, 143.36247, 138.62723, 139.49977,
       137.80324, 138.58437, 140.95016, 139.20639, 142.53313, 142.88295,
       143.422  , 144.35042, 144.1716 , 147.77264, 148.27075, 149.94284,
       149.1331 , 151.54823, 152.08319, 147.50658, 150.85767, 148.4269 ,
       146.19507, 144.34917, 144.93454, 145.3479 , 146.17107, 144.64603,
       142.63339, 144.39685, 147.15265, 147.89957, 146.23137, 146.45193,
       145.42044, 145.14705, 143.46056, 145.62779, 147.03528, 146.47115,
       146.88373, 144.29402, 147.31657, 150.4178 , 150.03902, 149.02408,
       146.62305, 147.32089], dtype=float32)>
```

[242]:

```
values[0][0]
```

[242]:

```
array([[150.30185],
       [151.6066 ],
       [150.32213],
       [149.27074],
       [151.90056],
       [153.61559],
       [153.69879],
       [153.30333],
       [152.88086],
       [150.43613],
       [149.56084],
       [149.60457],
       [148.94038],
       [148.6731 ],
       [151.73859],
       [150.84282],
       [148.69049],
       [150.13455],
       [152.71347],
       [152.8232 ]], dtype=float32)
```

[243]:

```
y_test
```

[243]:

```
array([142.63339, 144.39685, 147.15265, 147.89957, 146.23137, 146.45193,
       145.42044, 145.14705, 143.46056, 145.62779, 147.03528, 146.47115,
       146.88373, 144.29402, 147.31657, 150.4178 , 150.03902, 149.02408,
       146.62305, 147.32089], dtype=float32)
```

[ ]:

```

```