**ECE 404 Communication Systems**

***Project 1***

***Vehicle Positioning***

*Introduction*

In nowadays intelligent transportation system, accurate vehicle positioning is a hot topic which all big tech companies are working on which includes autonomous vehicles, traffic management and navigation systems. Traditional methods of vehicle positioning like GPS may not provide reliable results in complex environments like underground parking lots since satellite signals are weak or unavailable. In this project we try to develop a machine learning-based approach to estimate vehicle position using Channel State Information (CSI) from wireless communication systems.

**CSI** describes the properties of a communication channel and can be used to classify the communication link between a vehicle and a remote radio unit (RRU) as either Line-of-Sight (LoS) or Non-Line-of-Sight (NLoS). Using this information, we can develop a system that estimates the vehicle's position with high accuracy. In this project, we will design a neural network (NN) model that takes CSI inputs and classifies the communication link as LoS or NLoS and we will try to estimate the vehicle's position using CSI.

We aim to show the design and implementation of our LoS/NLoS classification program and the vehicle positioning program. We will discuss the results of our experiments and evaluate the performance of our approach.

*Methodology*

We will start this part by analyzing the dataset we will be using in this project. The dataset used in this project consists of CSI measurements collected in an underground parking lot with 476 different locations and 15000 training samples and 5000 validation samples. Each sample includes a complex-valued CSI matrix and a label vector containing the vehicle's position and LoS/NLoS indicator. We will use Python and the PyTorch library to implement and train our NN model.

Note that the Channel State Information (CSI) is a complex-valued matrix that represents the state of the wireless communication channel between the vehicle and the remote radio unit (RRU). The CSI matrix has a dimensionality of 4 × 1632, where 4 is the number of antennas of the RRU, and 1632 is the number of sub-carriers. Each element in the CSI matrix represents the signal strength at a particular antenna and sub-carrier. This CSI matrix is a complex-valued matrix, containing both real and imaginary parts where real part of the CSI represents the in-phase component of the signal and the imaginary part represents the quadrature component.

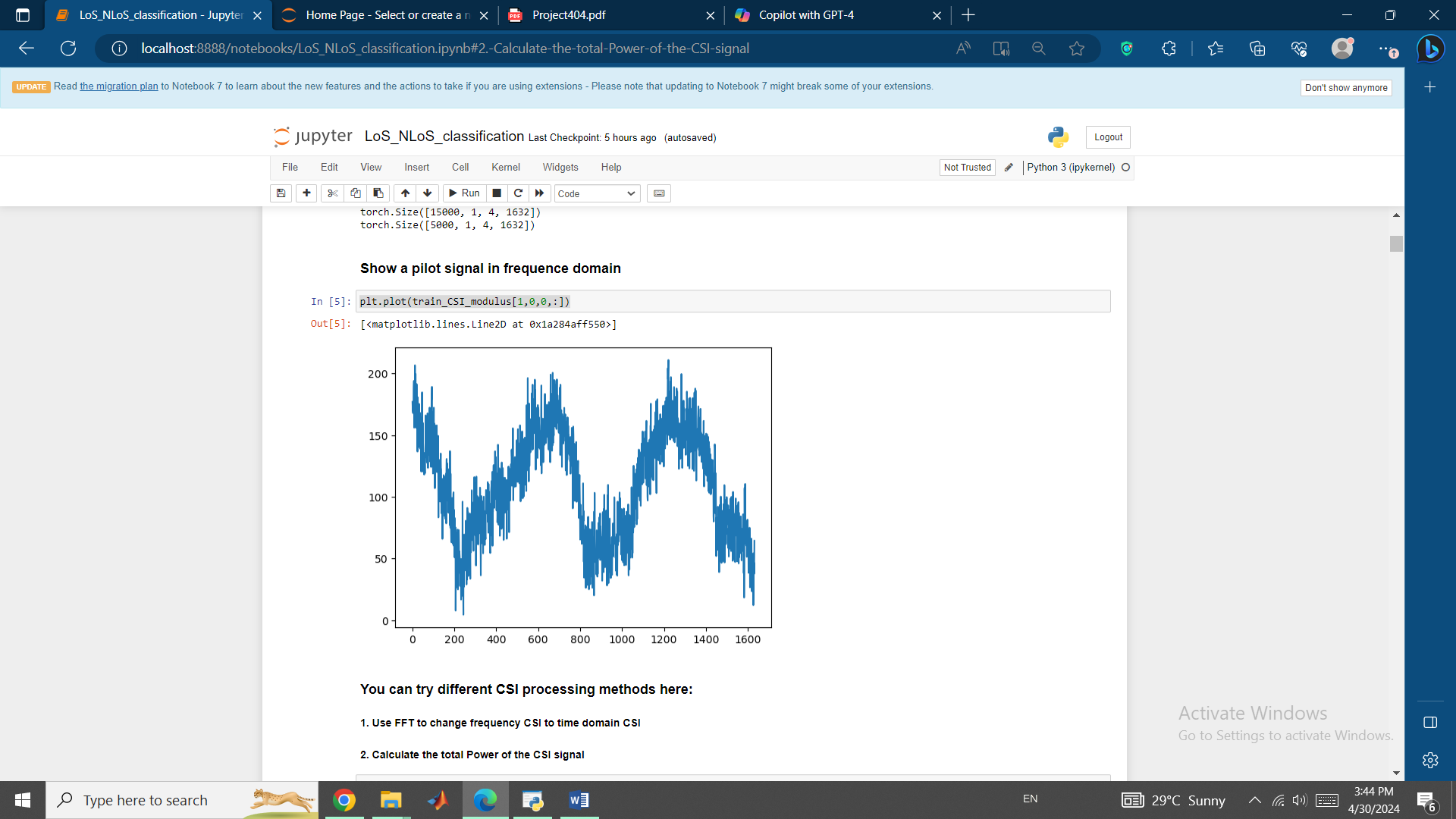
The modulus or absolute value of a complex number is the distance from the origin of the complex plane which can be interpreted as the signal strength or power.

The shapes of modulus of our train and validation dataset are [15000, 1, 4, 1632] and [5000, 1, 4, 1632] respectively. This means we have:

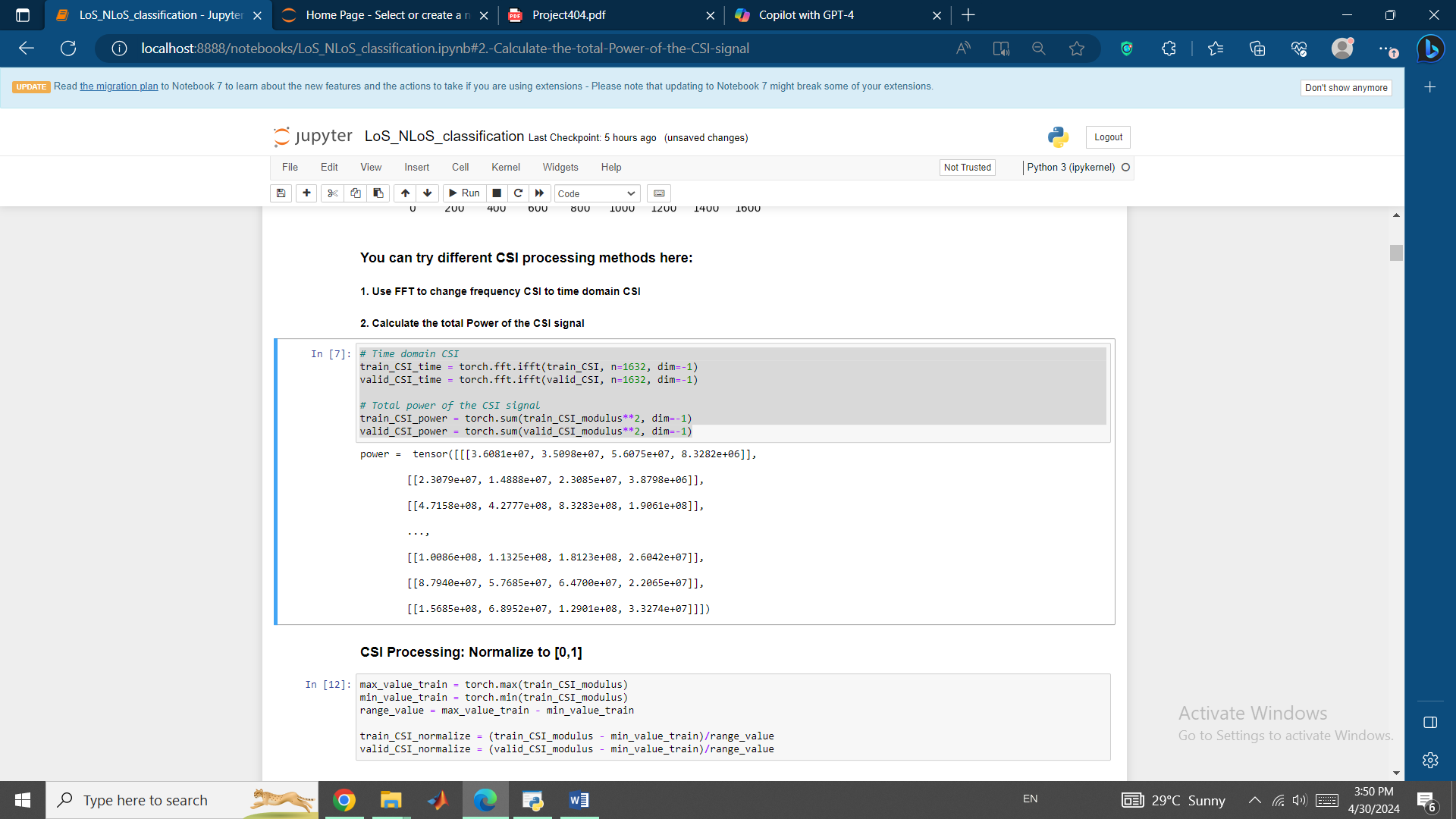
* 15000 training samples and 5000 validation samples
* Each sample is a 1-channel 4 x 1632 matrix where 4 is the number of antennas of the RRU and 1632 is the number of sub-carriers

Also, the labels of our dataset are binary indicators where 0 represents a Line-of-Sight (LoS) communication link and 1 represents a Non-Line-of-Sight (NLoS) communication link and these labales are used to train the machine learning model to classify the communication link as LoS or NLoS.

In order to get a better idea of how this might look, we can plot the signal strength of the first sub-carrier across all 4 antennas for the second training sample which will produce a visualization of the signal in the frequency domain:



We can try different CSI processing methods. For instance, using inverse FFT, we can use Time domain instead of Frequency domain which we used before. The Fast Fourier Transform (FFT) is a method to compute the Discrete Fourier Transform (DFT) and its inverse. DFT converts a sequence of values in the time domain into a sequence of values in the frequency domain. So, we can use FFT to convert the frequency domain CSI to time domain CSI. We can also calculate the power of CSI signal for a particular sample and antenna by squaring the modulus of the CSI matrix and then summing up all the values:



Next step is one of the first stages of preprocessing the data which is normalization. Normalizing the CSI values is an essential step in preparing the data for machine learning models. The main reason for normalizing is to ensure that all features are on the same scale which is important for several reasons:

1. Prevents feature dominance: When features have different scales, some features may dominate the model's predictions leading to biased results. Normalization prevents this by giving equal importance to all features.
2. Improves model convergence: Normalization helps the model converge faster and more accurately by reducing the effect of large values on the optimization process.
3. Enhances model interpretability: Normalization makes it easier to understand the relationships between features and the target variable since make all features be on the same scale.

We normalized the CSI values by subtracting the minimum value and then dividing it by the range value. This maps the original values to the range [0, 1].

In the next step, we want to define a Neural Network model and train this model.

The NN model we implemented is a simple feed-forward neural network model with two hidden layers and has three fully connected (fc) layers:

fc1: 4\*1632 input nodes, 512 output nodes (hidden layer 1)

fc2: 512 input nodes, 128 output nodes (hidden layer 2)

fc3: 128 input nodes, 2 output nodes (output layer)

The **forward method** defines the forward pass through the network, applying the ReLU activation function to the output of each hidden layer.

The forward method refers to the process of propagating input data through the network layer by layer to produce an output. In other words, it's the process of computing the output of the network given an input. It’s like a series of transformations that the input data undergoes as it flows through the network. Each layer applies its own set of weights, biases and activation functions to the input data, transforming it into a new representation. The output of one layer becomes the input to the next layer and so on. The forward method allows the network to make predictions, classify data or generate new data.

As mentioned before we chose ReLU as our activation function. **ReLU (Rectified Linear Unit)** is a type of activation function used in neural networks. An activation function is a mathematical function that is applied to the output of a layer to introduce non-linearity into the model. ReLU is a very widely used activation function used in deep neural networks, defined as:

f(θ) = Max {0, θ}

In other words, if the input θ is positive, the output is x. If the input x is negative, the output is 0.

We chose ReLU since it:

* Introduces non-linearity into the model, allowing the network to learn more complex relationships between inputs and outputs.
* Is computationally efficient since it only requires a simple thresholding operation.
* Helps to reduce the number of active neurons in the network which can improve computational efficiency and reduce overfitting.

And that’s why we chose ReLU over other activation functions such as sigmoid and tanh which can also be used in some other situations.

The optimizer we used for our Neural Network class is **Stochastic Gradient Descent (SGD)**. SGD is a variant of the gradient descent algorithm that is particularly well-suited for large datasets. The SGD optimizer works by iterating through the training dataset one sample at a time and for each sample, it computes the gradient of the loss function with respect to the model's parameters. The gradient is a vector of the same shape as the model's parameters and it indicates the direction in which the parameters should be updated to minimize the loss.

The SGD optimizer then updates the model's parameters using the following formula:

wnew = wold - α \*

where wold is the current value of the model's parameters, α is the learning rate, and  is the gradient of the loss function.

The reason behind choosing SGD over other optimizers for us are:

* SGD is computationally efficient because it only requires computing the gradient of the loss function for a single sample at a time.
* It is a stochastic optimization algorithm which means that it can escape local minima and converge to a better solution.
* SGD can be used with a wide range of learning rates and batch sizes making it a flexible optimization algorithm.

As for our Loss Function for our NN, we chose Cross Entropy.

The **Cross Entropy loss function** is widely used for classification problems. It is defined as:

L(y, yprediction) = -sum(y log(yprediction))

where y is the true label, yprediction is the predicted probability distribution and sum is the sum over all classes.

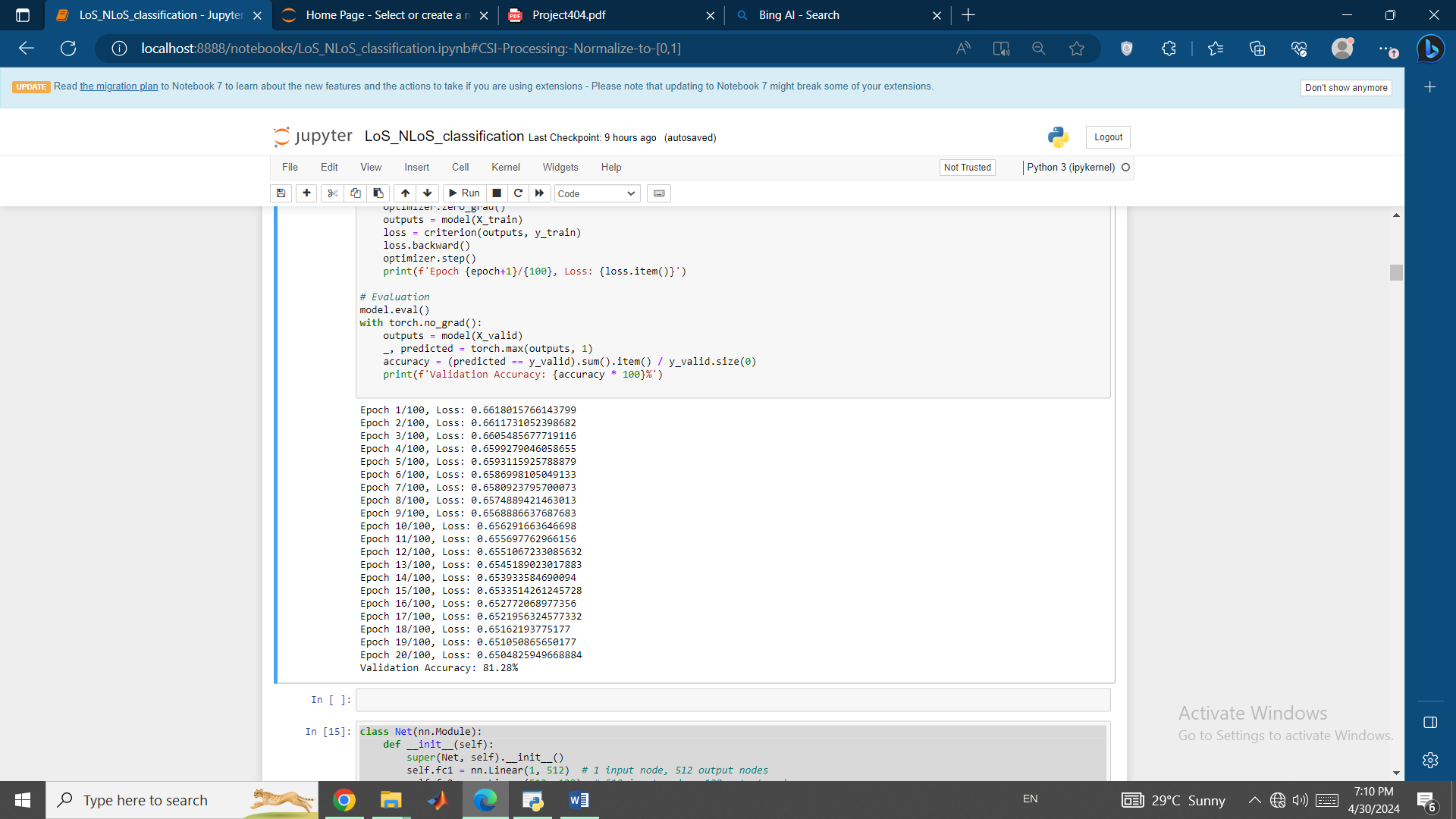
The Cross Entropy measures the difference between the predicted probability distribution and the true label. The goal is to minimize the Cross Entropy loss which means maximizing the log likelihood of the true label.

Cross Entropy is an ideal Loss Function for us since it is a very suitable choice for classification problems like our case of interest in this paper since:

* Cross Entropy is a differentiable loss function which makes it easy to optimize using gradient-based optimization algorithms.
* Provides an interpretable measure of the difference between the predicted probability distribution and the true label.
* It is robust to class imbalance problems where one class has a significantly larger number of samples than the other classes.

SGD and Cross Entropy are especially make a really Good Combination because they complement each other well. SGD is a flexible optimization algorithm that can adapt to a wide range of learning rates and batch sizes while Cross Entropy is a robust loss function that provides an interpretable measure of the difference between the predicted probability distribution and the true label and thus together, SGD and Cross Entropy provide a powerful combination for training neural networks for classification problems.

Result of our defined trained Neural Network model:



As you can see the loss values are decreasing as the epochs progress which shows that the model is learning and improving its performance on the training data.

Last line about Validation accuracy means that the model is able to correctly classify approximately 81.28% of the validation samples and thus this trained Neural Network with all the parameters and different kinds of functions and optimizers and Loss functions that we broadly talked about, was a good choice for classifying Line-of-Sight (LoS)/Non-Light-of-Sight (NLoS) communication link since this output suggests that the model is learning and improving its performance on the training data and it is able to generalize well to new data with an accuracy of 81.28% on the validation dataset.

Now, for each of the CSI inputs (Power, Time domain, Frequency domain), we need to train a separate model using that specific input. Then, we can generate predictions on our validation data.

When we use the power of CSI as input for LoS/NLoS classifications, we get the following output:

Epoch 1/20, Loss: 176030096.0

Epoch 2/20, Loss: 2.225424659008658e+25

Epoch 3/20, Loss: 49524632.0

Epoch 4/20, Loss: 588822912.0

Epoch 5/20, Loss: 461903488.0

Epoch 6/20, Loss: 334984352.0

Epoch 7/20, Loss: 208065280.0

Epoch 8/20, Loss: 81146288.0

Epoch 9/20, Loss: 0.7170279026031494

Epoch 10/20, Loss: 0.7167910933494568

Epoch 11/20, Loss: 0.7165547609329224

Epoch 12/20, Loss: 0.7163187265396118

Epoch 13/20, Loss: 0.7160826921463013

Epoch 14/20, Loss: 0.7158469557762146

Epoch 15/20, Loss: 0.7156115174293518

Epoch 16/20, Loss: 0.7153763175010681

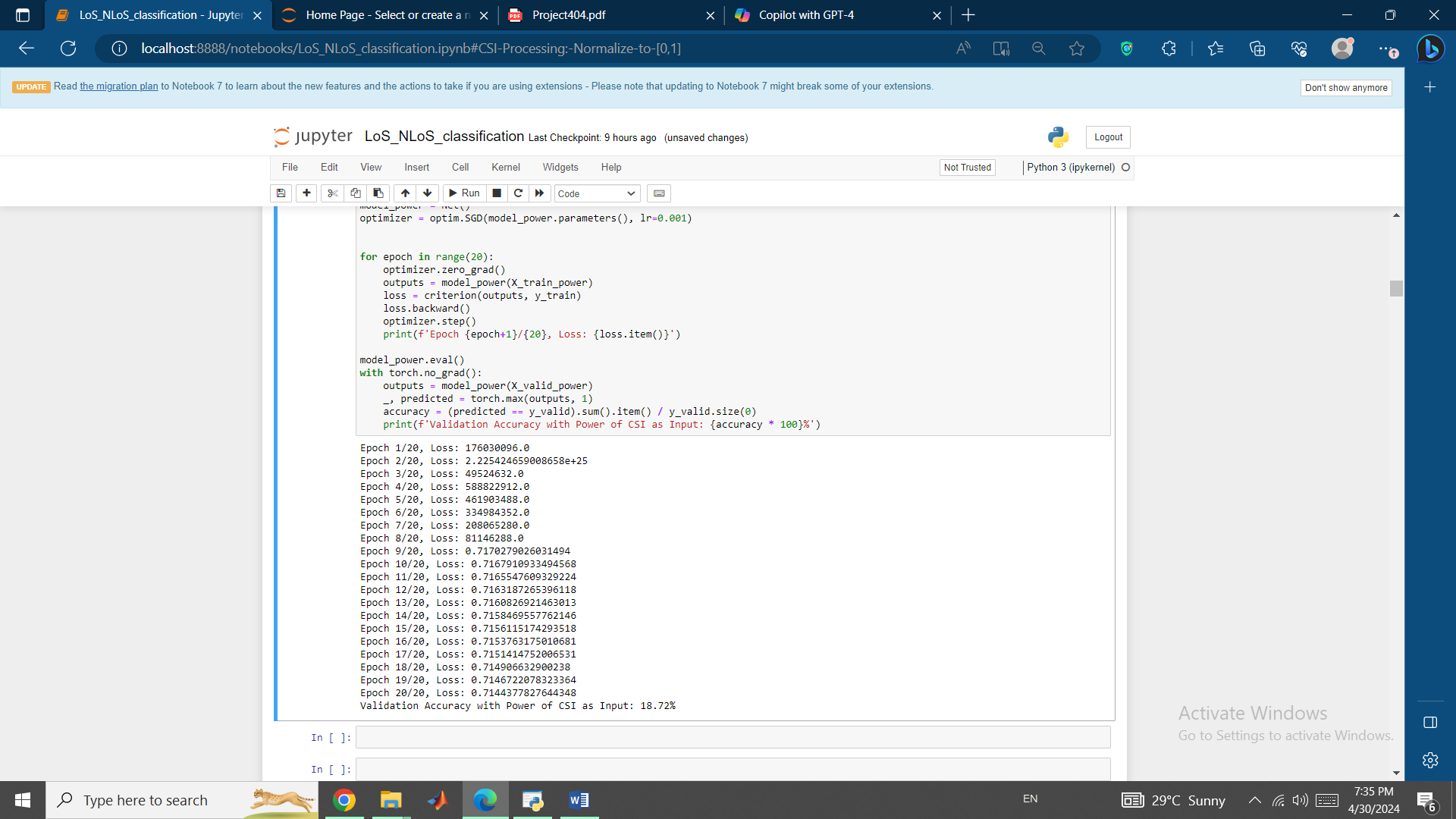
Epoch 17/20, Loss: 0.7151414752006531

Epoch 18/20, Loss: 0.714906632900238

Epoch 19/20, Loss: 0.7146722078323364

Epoch 20/20, Loss: 0.7144377827644348

Validation Accuracy with Power of CSI as Input: 18.72%



From the Validation Accuracy, we can see that the performance of the model with the power of CSI as input is quite low (18.72%).

Some reasons behind this low accuracy can be:

* Model Complexity: The model might be too simple to capture the complex relationship between the power of CSI and the LoS/NLoS conditions.
* Overfitting: If the model is too complex, it might overfit to the training data, resulting in poor performance on the validation data.
* Learning Rate: The learning rate might be too high or too low. If it’s too high, the model might overshoot the optimal solution
* Data Preprocessing: The preprocessing of the data might not be suitable. For example, normalizing the data to a range of [0, 1] might not be the best choice for this particular problem.
* Feature Selection: Using the power of CSI as the only feature might not be sufficient to accurately classify the LoS/NLoS conditions.

In the next step, we use the Time domain CSI as input for LoS/NLoS classifications and we get the following output:

Epoch 1/20, Loss: 0.968691885471344

Epoch 2/20, Loss: 0.534308910369873

Epoch 3/20, Loss: 0.5169869661331177

Epoch 4/20, Loss: 0.5051239728927612

Epoch 5/20, Loss: 0.49585098028182983

Epoch 6/20, Loss: 0.4881400167942047

Epoch 7/20, Loss: 0.4815206527709961

Epoch 8/20, Loss: 0.4757310748100281

Epoch 9/20, Loss: 0.47058337926864624

Epoch 10/20, Loss: 0.4659697711467743

Epoch 11/20, Loss: 0.4617997109889984

Epoch 12/20, Loss: 0.45800554752349854

Epoch 13/20, Loss: 0.45454564690589905

Epoch 14/20, Loss: 0.4513736963272095

Epoch 15/20, Loss: 0.44844815135002136

Epoch 16/20, Loss: 0.44573792815208435

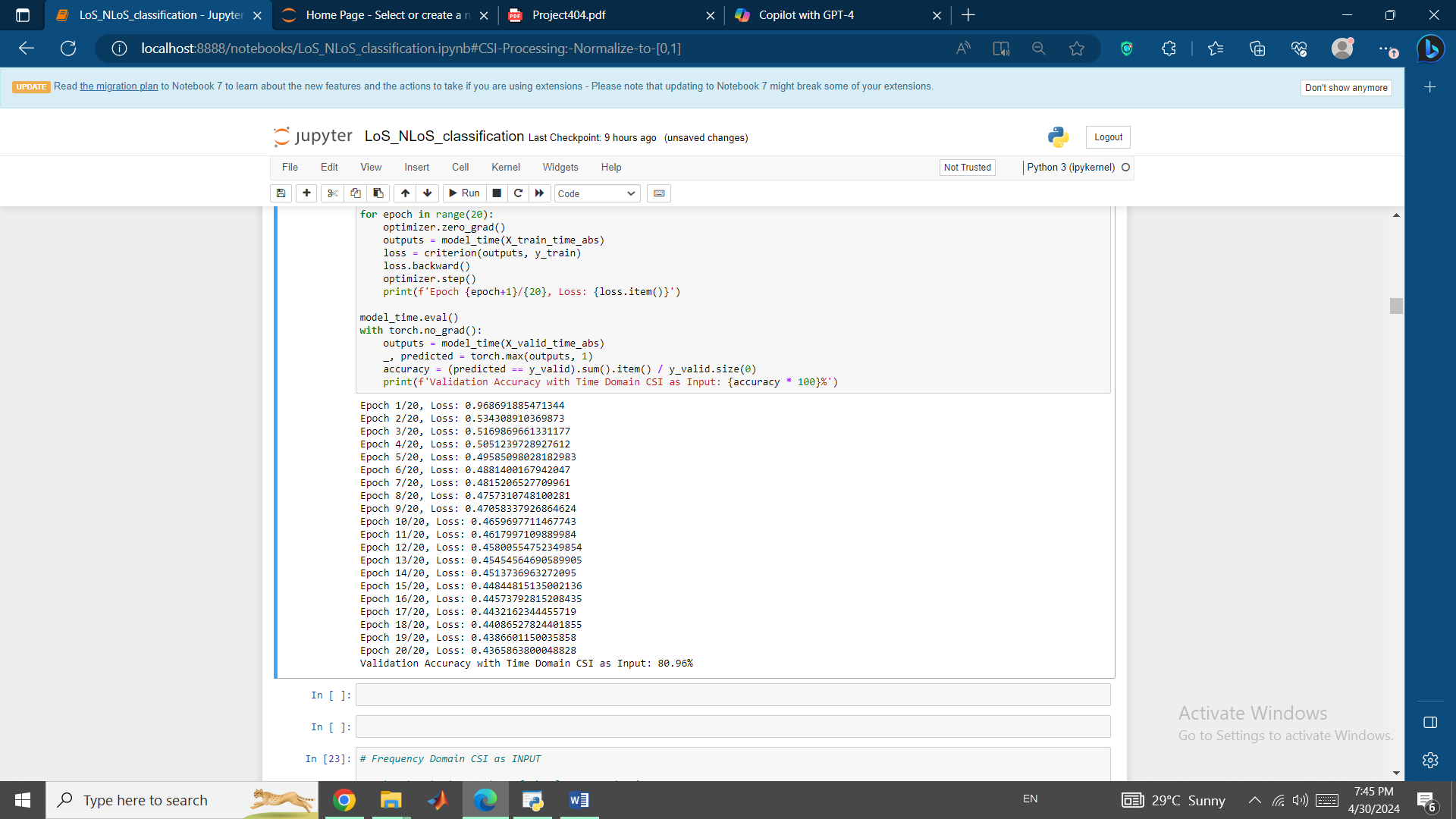
Epoch 17/20, Loss: 0.4432162344455719

Epoch 18/20, Loss: 0.44086527824401855

Epoch 19/20, Loss: 0.4386601150035858

Epoch 20/20, Loss: 0.4365863800048828

Validation Accuracy with Time Domain CSI as Input: 80.96%



The result for the model trained with Time Domain CSI as input responded very well and achieved a validation accuracy of **80.96%.** This is a great result and shows that the time-domain representation of the CSI can be a useful feature for classifying the LoS/NLoS conditions.

In the last phase, we will use Frequency domain CSI as input for LoS/NLoS classifications and we get the following output:

Epoch 1/20, Loss: 7.848504066467285

Epoch 2/20, Loss: 4244.90283203125

Epoch 3/20, Loss: 19587.08203125

Epoch 4/20, Loss: 1145.85400390625

Epoch 5/20, Loss: 589306.1875

Epoch 6/20, Loss: 13194.53515625

Epoch 7/20, Loss: 4226810.5

Epoch 8/20, Loss: 145206832.0

Epoch 9/20, Loss: 1387177.5

Epoch 10/20, Loss: 2542144768.0

Epoch 11/20, Loss: 13851124826112.0

Epoch 12/20, Loss: 872951906304.0

Epoch 13/20, Loss: 4.974206495437619e+16

Epoch 14/20, Loss: 45759197184.0

Epoch 15/20, Loss: 70.79435729980469

Epoch 16/20, Loss: 56.27229690551758

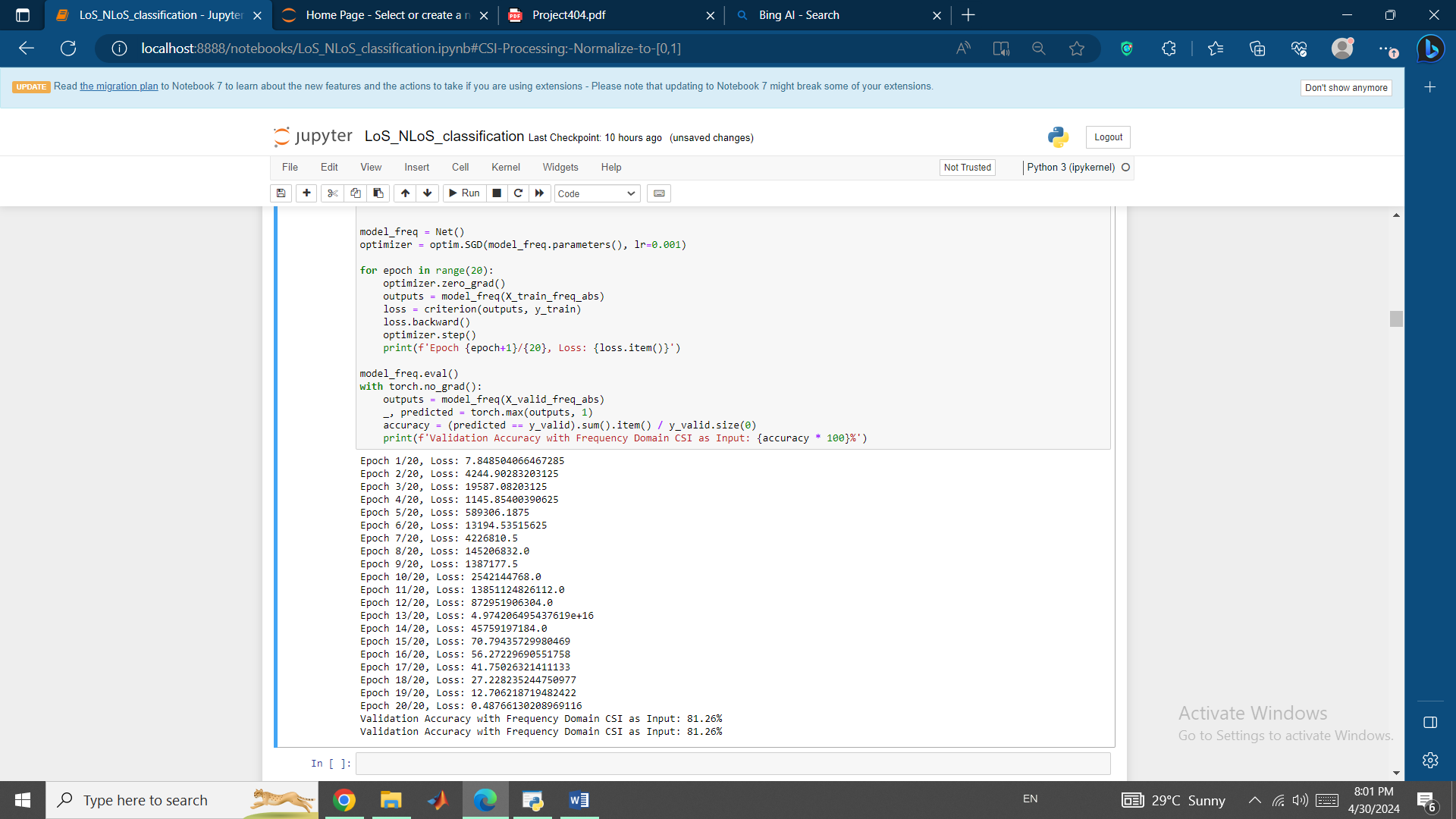
Epoch 17/20, Loss: 41.75026321411133

Epoch 18/20, Loss: 27.228235244750977

Epoch 19/20, Loss: 12.706218719482422

Epoch 20/20, Loss: 0.48766130208969116

Validation Accuracy with Frequency Domain CSI as Input: 81.26%



We can see that the model trained with Frequency Domain CSI as input responded even better than the one trained with Time domain CSI and achieved a validation accuracy of **81.26%** which shows that the Frequency-domain representation of the CSI can be the best feature for classifying the LoS/NLoS conditions.

In the final phase, we want to create a vehicle positioning program that estimates the vehicle position using CSI. Here, we will treat the problem as a regression problem where the goal is to predict the continuous x and y coordinates of the vehicle.

We should start by data preparation where we have to extract the labels of our model where the labels for our regression model are the x and y coordinates of the vehicle. We extract these from the dataset similar to how you extracted the LoS/NLoS labels. Next, we will design our model where in this NN architecture, the output layer should have 2 nodes corresponding to the x and y coordinates. Also, since this is a regression problem, we don’t need an activation function in the output layer. The training process is similar to before, but we use a suitable loss function for regression like Mean Squared Error (MSE) loss and we calculate the MSE loss on our validation data to evaluate our model.

In our first attempts, we encountered the problem where the Epoch loss was infinite and Nan in some of the Epochs and thus the model couldn’t even be evaluated.

Epoch 1/20, Loss: 3019.373291015625

Epoch 2/20, Loss: 3331815112704.0

Epoch 3/20, Loss: inf

Epoch 4/20, Loss: nan

Epoch 5/20, Loss: nan

Epoch 6/20, Loss: nan

Epoch 7/20, Loss: nan

Epoch 8/20, Loss: nan

Epoch 9/20, Loss: nan

Epoch 10/20, Loss: nan

Epoch 11/20, Loss: nan

Epoch 12/20, Loss: nan

Epoch 13/20, Loss: nan

Epoch 14/20, Loss: nan

Epoch 15/20, Loss: nan

Epoch 16/20, Loss: nan

Epoch 17/20, Loss: nan

Epoch 18/20, Loss: nan

Epoch 19/20, Loss: nan

Epoch 20/20, Loss: nan

The error message indicates that the loss became inf or nan during training which is a common issue in training neural networks and can happen because:

* Learning Rate: If the learning rate is too high, the model might overshoot the optimal solution causing the loss to become inf or nan.
* Numerical Stability: The calculations in the model might result in numbers that are too large or too small to be represented accurately causing numerical instability.
* Data Scaling: If the features in the input data have different scales, the model might give more importance to features with larger scales, causing the loss to become inf or nan.
* Model Complexity: If the model is too complex, it might fit the training data too closely causing the loss to become inf or nan.

In order to tackle this problem, we implement three different methods:

1. Gradient Clipping, that is a technique to prevent exploding gradients by artificially limiting them to a range of values.
2. Learning rate scheduling, where we basically decrease the learning rate over epochs
3. Different Optimizer (Adam)

The Adam optimizer is a stochastic gradient descent algorithm that adapts the learning rate for each parameter based on the magnitude of the gradient. The Adam optimizer is designed to adapt to the geometry of the loss function and to be more robust to the choice of learning rate. The Adam optimizer works by maintaining a separate learning rate for each parameter which is adapted based on the magnitude of the gradient. The learning rate for each parameter is computed as follows:

where and are the first and second moment estimates,  and are hyperparameters, is the gradient at time t,  is the learning rate, and  is a small value added for numerical stability.

The Adam optimizer then updates the parameters using the following rule:

Result of Gradient Clipping method:

Epoch 1/20, Loss: 4588.72705078125

Epoch 2/20, Loss: 4494.57958984375

Epoch 3/20, Loss: 4473.93408203125

Epoch 4/20, Loss: 4503.8330078125

Epoch 5/20, Loss: 4452.806640625

Epoch 6/20, Loss: 4441.05517578125

Epoch 7/20, Loss: 4522.12890625

Epoch 8/20, Loss: 4500.447265625

Epoch 9/20, Loss: 4508.9658203125

Epoch 10/20, Loss: 4556.70654296875

Epoch 11/20, Loss: 4501.9677734375

Epoch 12/20, Loss: 4471.20556640625

Epoch 13/20, Loss: 4469.345703125

Epoch 14/20, Loss: 4517.2607421875

Epoch 15/20, Loss: 4538.8798828125

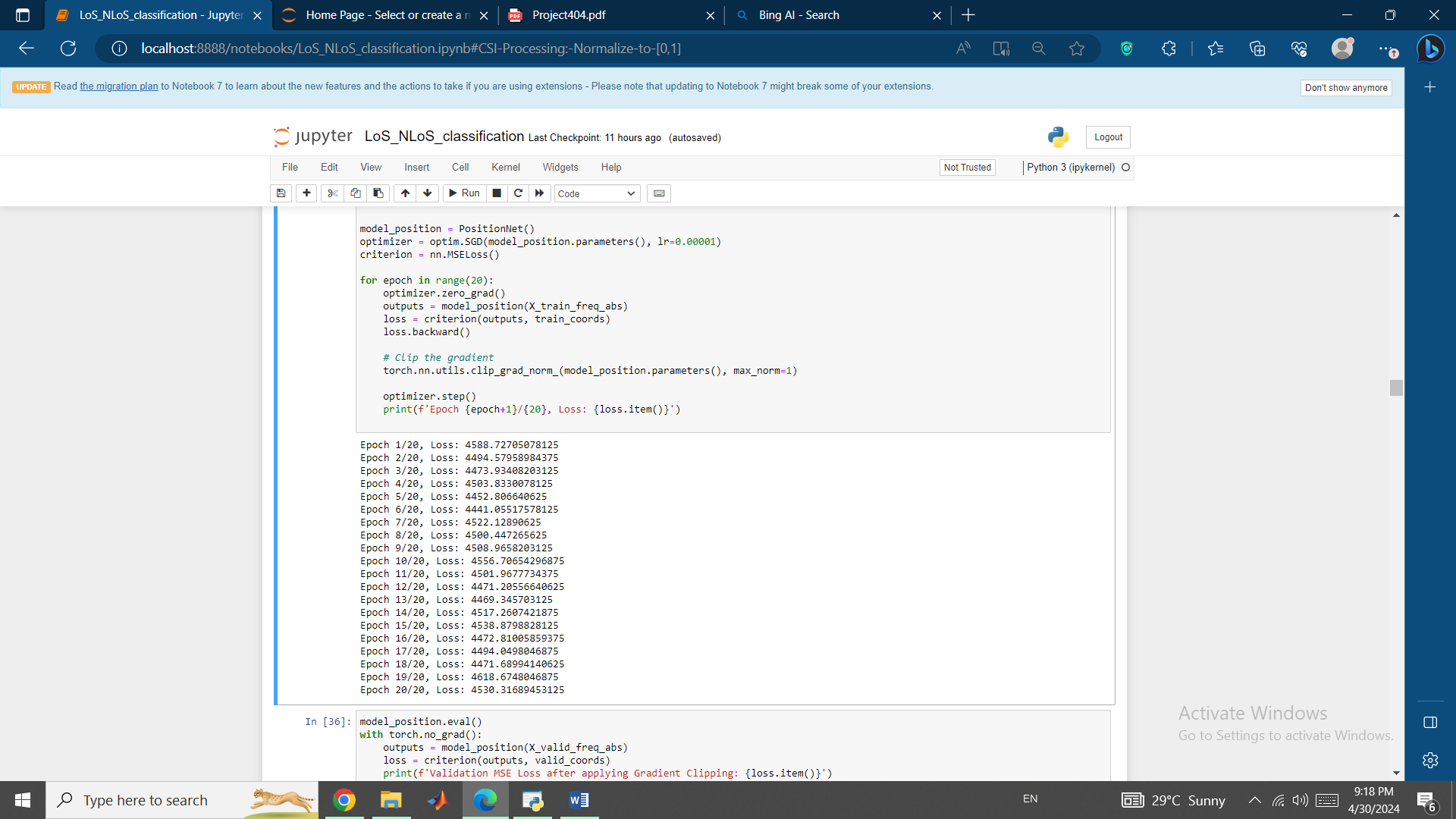
Epoch 16/20, Loss: 4472.81005859375

Epoch 17/20, Loss: 4494.0498046875

Epoch 18/20, Loss: 4471.68994140625

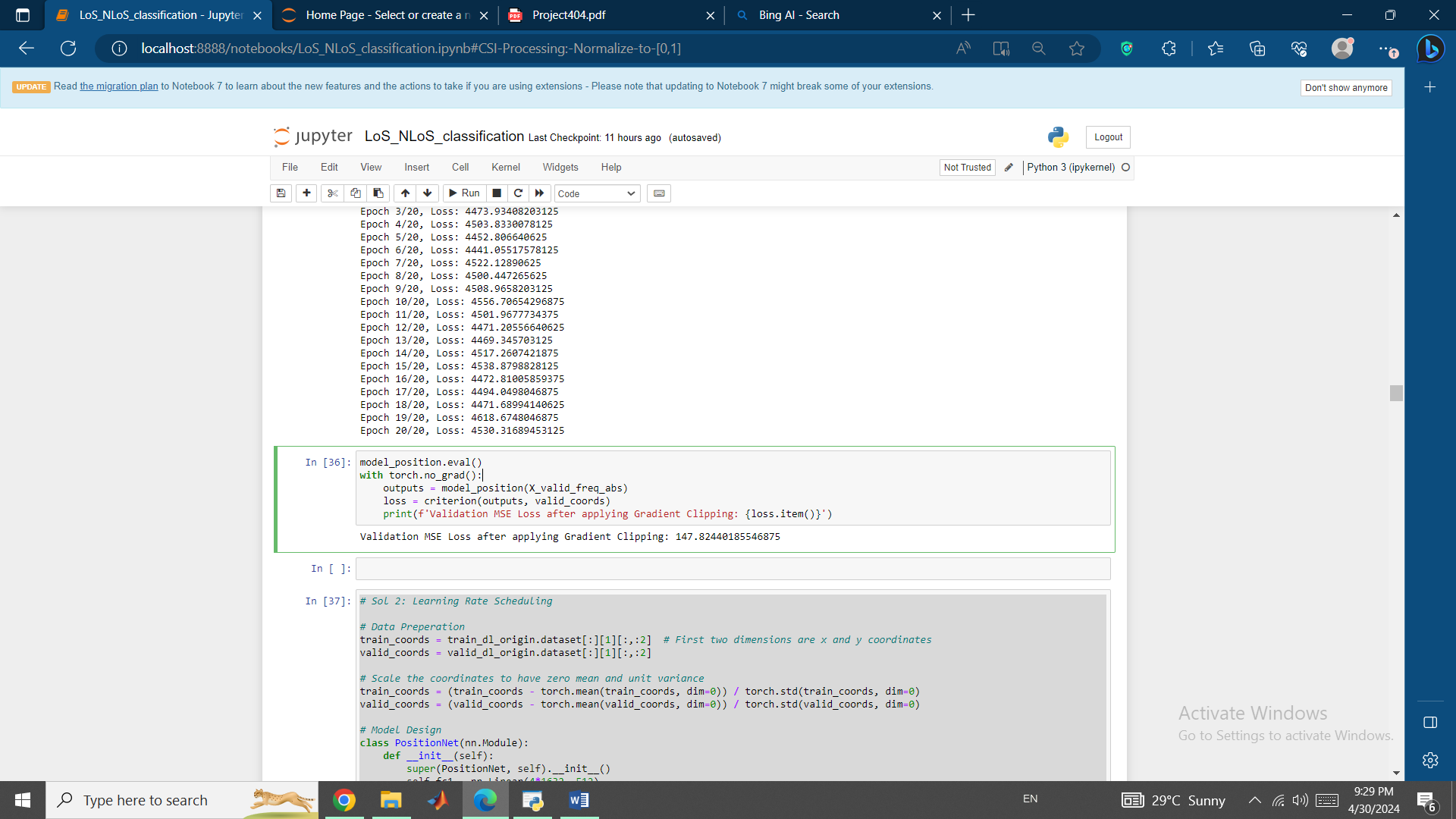
Epoch 19/20, Loss: 4618.6748046875

Epoch 20/20, Loss: 4530.31689453125



We can see that Gradient Clipping has solved our problem. Now, let’s see how well can this trained model predict and fit the data for the estimation of the vehicle position.

Validation MSE Loss after applying Gradient Clipping: 147.82440185546875



Learning Rate Scheduling on the other hand, couldn’t help us solve the issue:

Epoch 1/20, Loss: 3636.274658203125

Epoch 2/20, Loss: 761698.3125

Epoch 3/20, Loss: 47389863837696.0

Epoch 4/20, Loss: inf

Epoch 5/20, Loss: nan

Epoch 6/20, Loss: nan

Epoch 7/20, Loss: nan

Epoch 8/20, Loss: nan

Epoch 9/20, Loss: nan

Epoch 10/20, Loss: nan

Epoch 11/20, Loss: nan

Epoch 12/20, Loss: nan

Epoch 13/20, Loss: nan

Epoch 14/20, Loss: nan

Epoch 15/20, Loss: nan

Epoch 16/20, Loss: nan

Epoch 17/20, Loss: nan

Epoch 18/20, Loss: nan

Epoch 19/20, Loss: nan

Epoch 20/20, Loss: nan

And our 3rd solution, using Adam’s Optimizer, produced the following Loss for its Epochs:

Epoch 1/25, Loss: 4412.85693359375

Epoch 2/25, Loss: 15526.03125

Epoch 3/25, Loss: 1923.53955078125

Epoch 4/25, Loss: 2239.397705078125

Epoch 5/25, Loss: 2388.405029296875

Epoch 6/25, Loss: 2347.241943359375

Epoch 7/25, Loss: 2208.16357421875

Epoch 8/25, Loss: 2222.3193359375

Epoch 9/25, Loss: 2063.809326171875

Epoch 10/25, Loss: 1868.38232421875

Epoch 11/25, Loss: 1714.52783203125

Epoch 12/25, Loss: 1553.96044921875

Epoch 13/25, Loss: 1370.7938232421875

Epoch 14/25, Loss: 1294.9697265625

Epoch 15/25, Loss: 1128.17431640625

Epoch 16/25, Loss: 1001.71337890625

Epoch 17/25, Loss: 894.4698486328125

Epoch 18/25, Loss: 756.4754638671875

Epoch 19/25, Loss: 656.5401611328125

Epoch 20/25, Loss: 559.2646484375

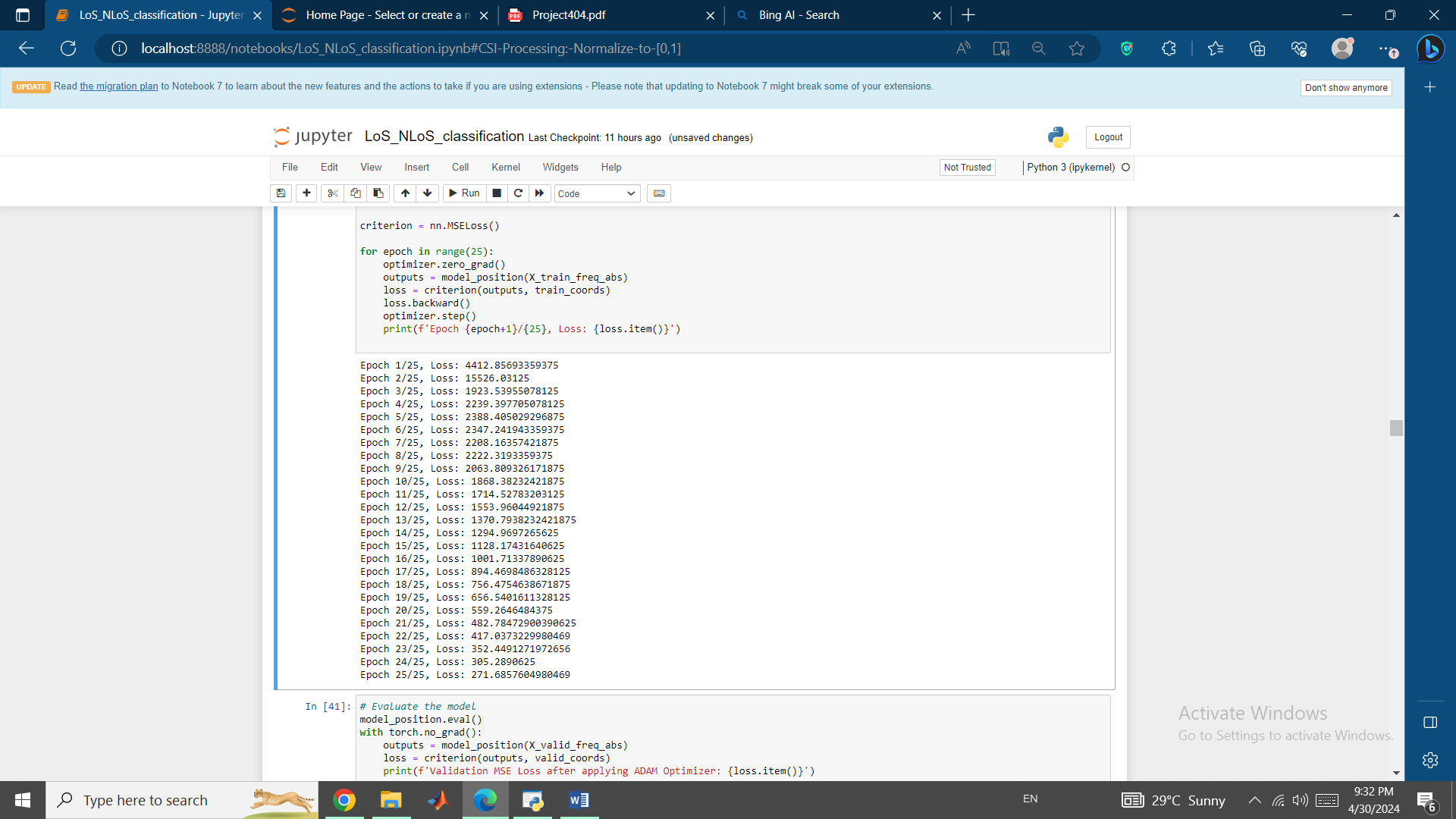
Epoch 21/25, Loss: 482.78472900390625

Epoch 22/25, Loss: 417.0373229980469

Epoch 23/25, Loss: 352.4491271972656

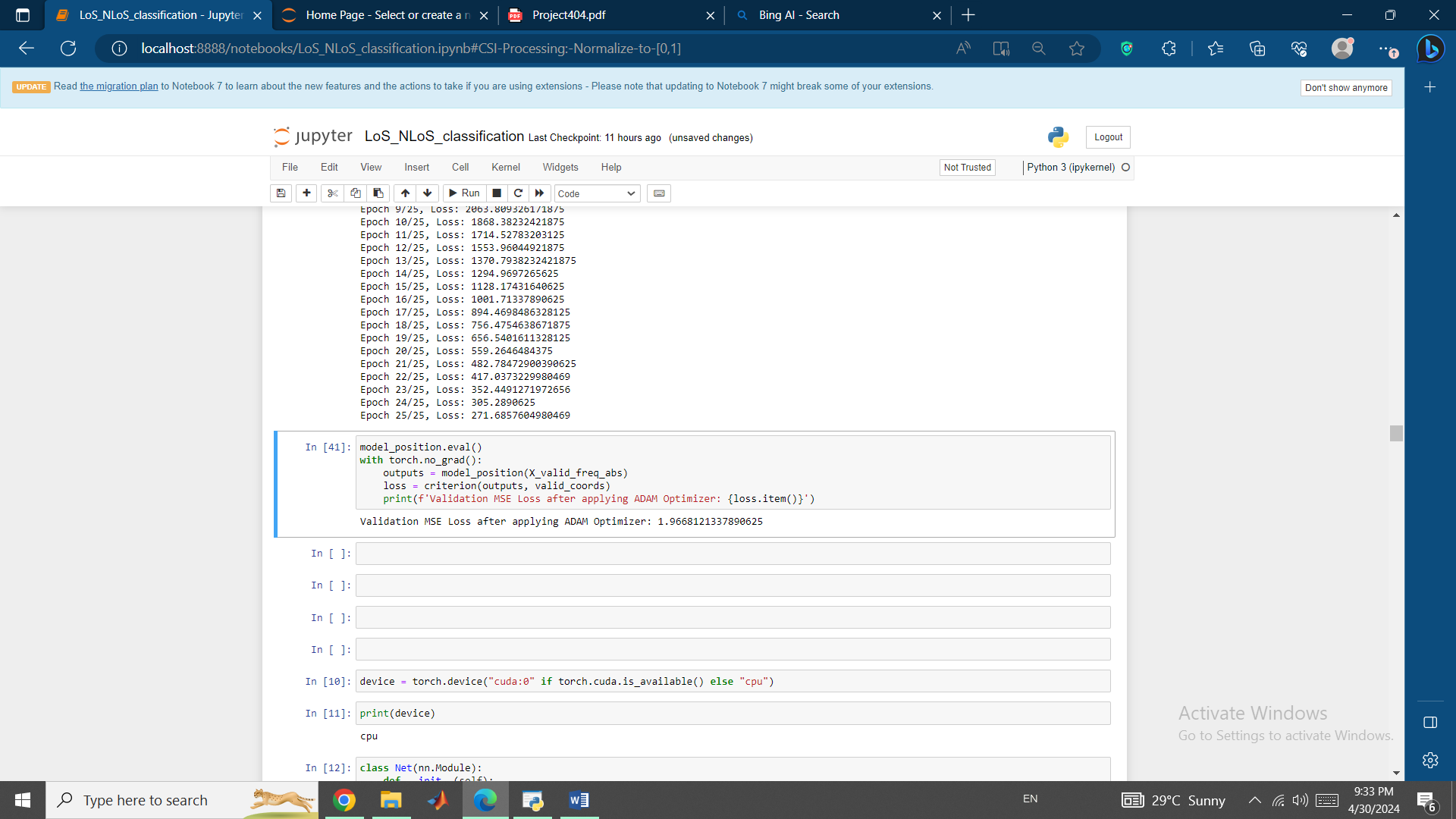
Epoch 24/25, Loss: 305.2890625

Epoch 25/25, Loss: 271.6857604980469



And the model trained by it produced the least Loss function value:

Validation MSE Loss after applying ADAM Optimizer: 1.9668121337890625

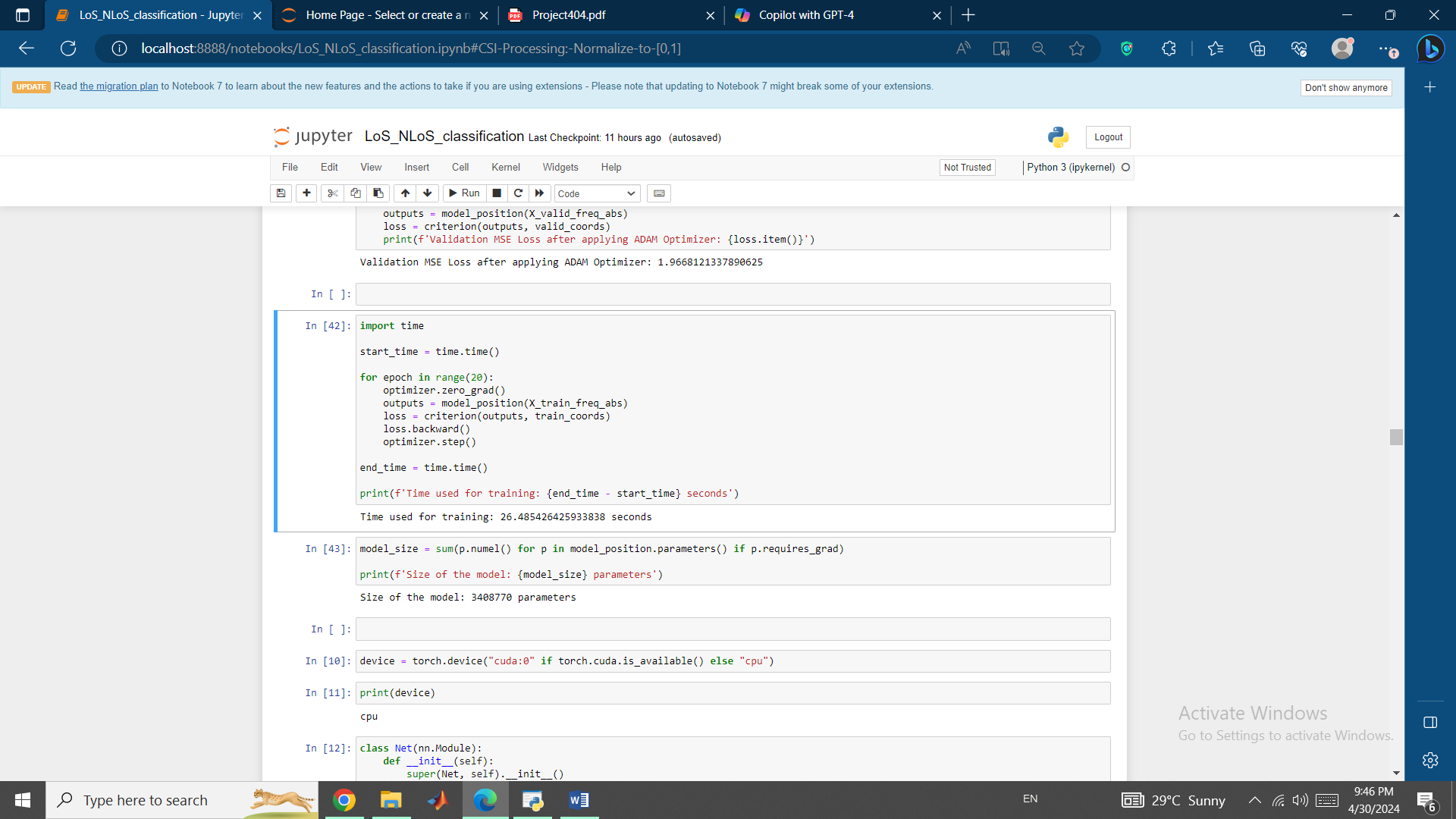


So, we've successfully trained a neural network to predict the x and y coordinates of a vehicle using Channel State Information (CSI) data. After applying the Adam optimizer, the model produced a validation MSE loss of 1.9668121337890625, which is a significantly good result for the model.

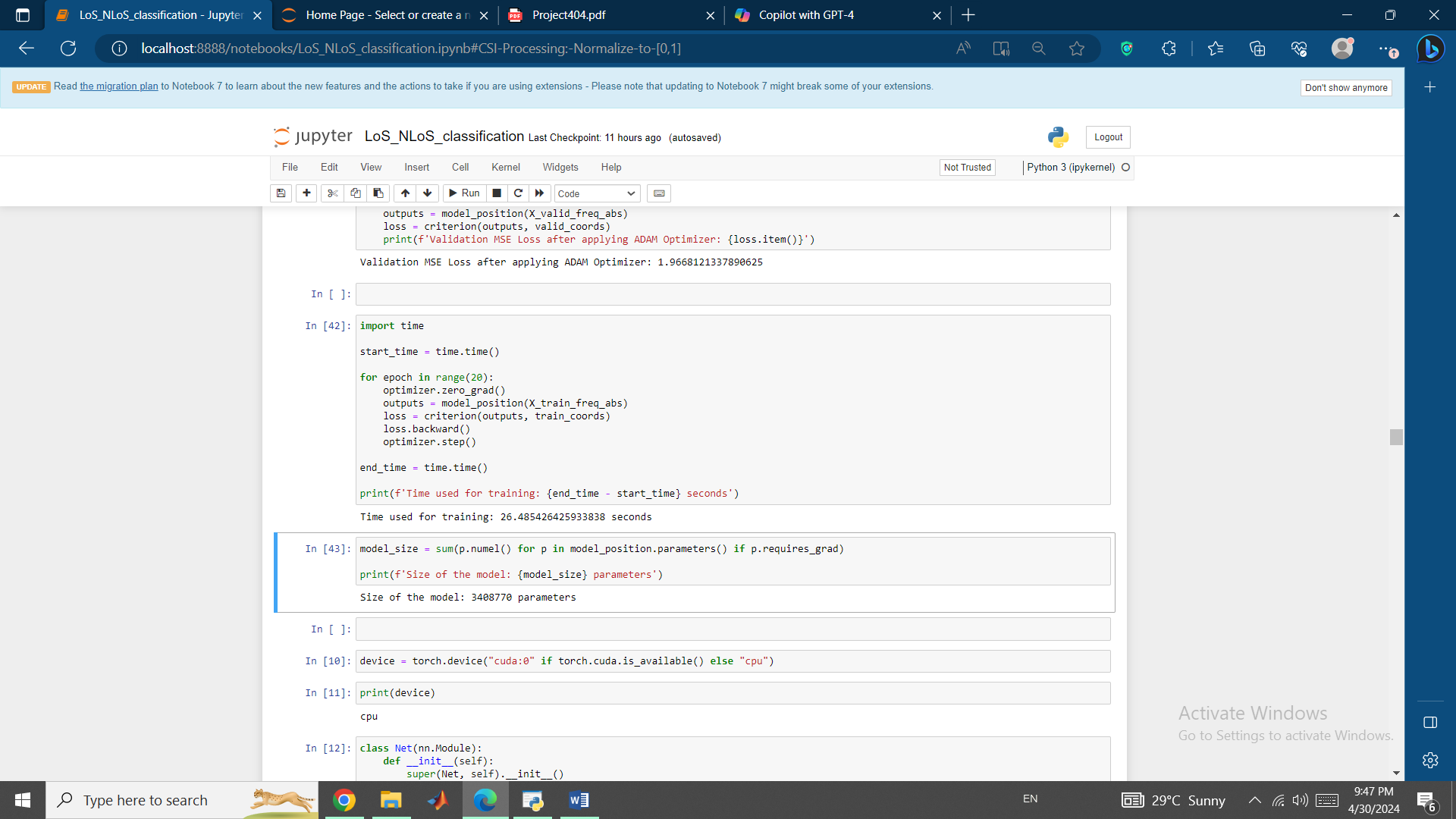
This result suggests that the model is able to predict the x and y coordinates of the vehicle with a reasonable degree of accuracy. The MSE loss value indicates that the model is able to estimate the coordinates with an average error of approximately 1.97 units.

Now, we want to measure the efficiency of our algorithm for our used CSI input. We use time used for training and size of the model to perform this efficiency evaluation.

Time used for training: 26.485426425933838 seconds



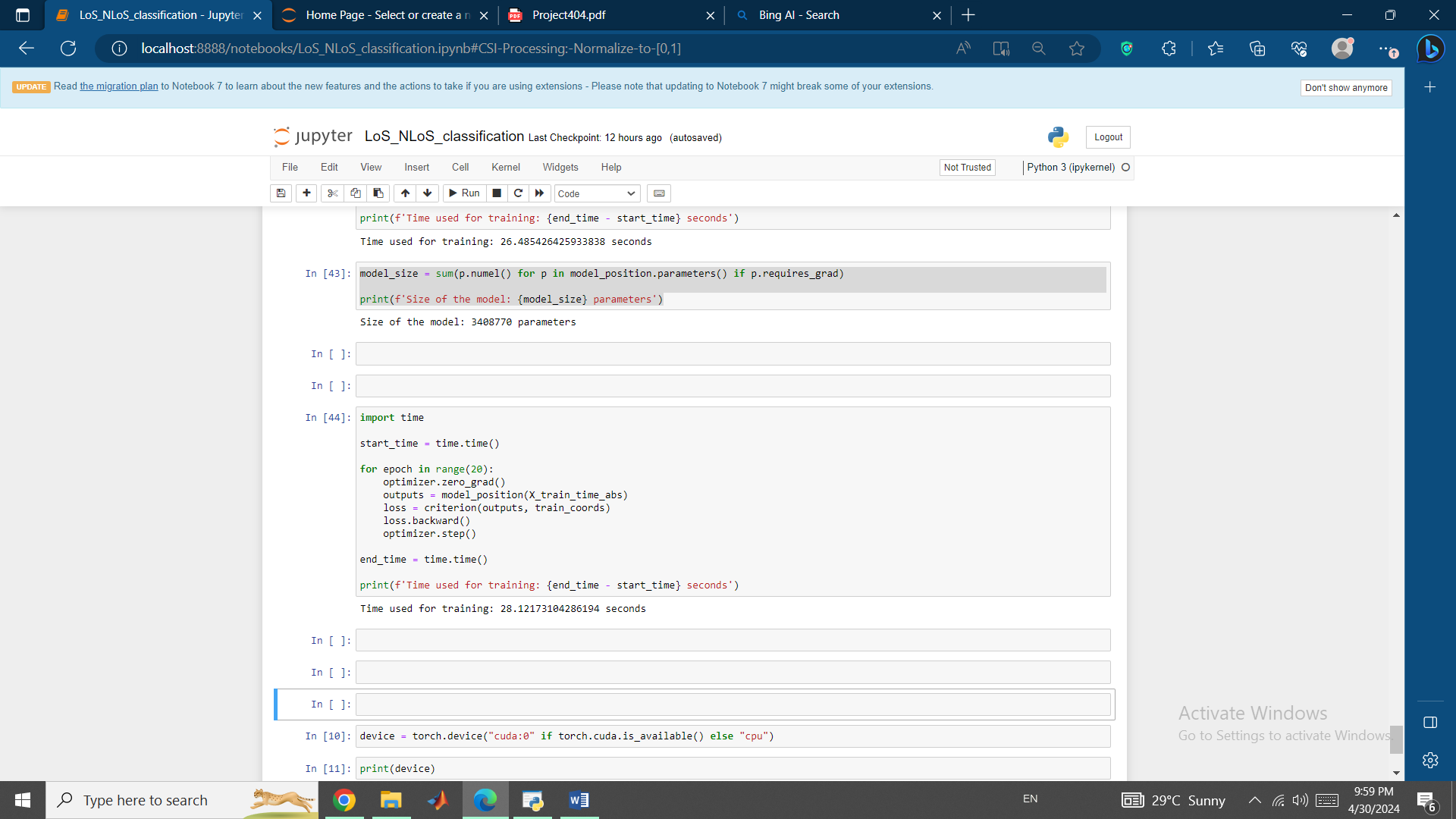
Size of the model: 3408770 parameters



We can see that for training of the model using Frequency domain CSI as input, we have used 26.485426425933838 seconds time and have used 3408770 parameters.

Note that time used for training of the model using Time domain CSI as input is

Time used for training: 28.12173104286194 seconds



Which shows that Frequency domain CSI input is more time efficient than Time domain CSI input.

Also, Power domain CSI input gave us very low Validation Accuracy in compared to Frequency domain and Time domain CSIs (18% against over 80%) which puts it out of the competition.

*Conclusion*

We demonstrated the development of a neural network-based vehicle positioning program using Channel State Information (CSI) data. The program aims to estimate the x and y coordinates of a vehicle using CSI data which is a challenging task due to the complexity of the CSI signals.

The report started by discussing the importance of normalization in preprocessing the CSI data which is essential to prevent feature dominance and improve model convergence. A simple feed-forward neural network model with two hidden layers is designed and the ReLU activation function is chosen for its ability to introduce non-linearity and reduce overfitting.

The report then explored the use of different CSI inputs including Power, Time domain and Frequency domain CSI to train separate models for LoS/NLoS classification. The results show that the Frequency domain CSI input achieves the highest validation accuracy of 81.26%, making it the best feature for classifying LoS/NLoS conditions.

In the next step, the report focused on developing a regression model to predict the continuous x and y coordinates of the vehicle using CSI data. The model is trained using the Mean Squared Error (MSE) loss function and three different methods are implemented to address the issue of infinite and NaN epoch losses:

* Gradient Clipping,
* Learning Rate Scheduling
* Adam optimizer

The Adam optimizer produces the best result with a validation MSE loss of 1.9668121337890625 indicating that the model is able to estimate the coordinates with an average error of approximately 1.97 units.

Finally, the report evaluated the efficiency of the algorithm using the time used for training and the size of the model. The results show that the Frequency domain CSI input is more time-efficient than the Time domain CSI input and chose Frequency domain CSI input as the best input for ANN model overall.