TORQUE PREDICTION FROM EV GEARBOX

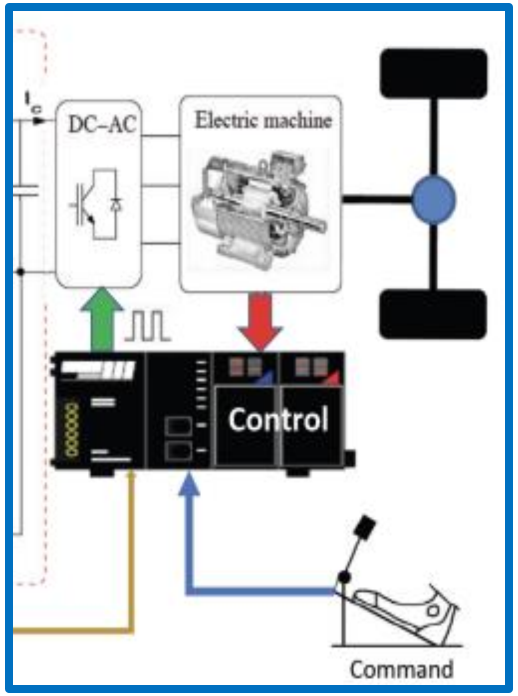
Rodrigo Pérez Salinas

Desmond Nii Ashitey Hammond

Niklas Handke

INTRODUCTION

- **Electrical Torque estimation** is required to control electrical vehicles
- **Precise control** of the motor requires input of costly sensors (current, voltage, rpm, **torque**…)



INTRODUCTION

- **Model of the electrical motor** uses voltage, current, flux, inductances, speed and resistance as an input
- **Electrical torque** given by equation with current and flux
- **Flux** is directly correlated to voltage, but expensive to measure and very difficult to estimate.
- **Task** is to approximate the solution of a fifth order differential equation

The synchronous motor modeling follows:

$$v_{ds} = r_s i_{ds} + \rho\lambda_{ds} - \omega_r \lambda_{qs} \quad 1$$

$$v_{qs} = r_s i_{qs} + \rho\lambda_{qs} + \omega_r \lambda_{ds} \quad 2$$

$$\lambda_{ds} = L_d i_{ds} + \lambda_m \quad 3$$

$$\lambda_{qs} = L_q i_{qs} \quad 4$$

The electrical torque equation is:

$$T_e = \left(\frac{3}{2}\right)\left(\frac{P}{2}\right)(\lambda_{ds} i_{qs} - \lambda_{qs} i_{ds}) \quad 5$$

And, the mechanical model of torque is

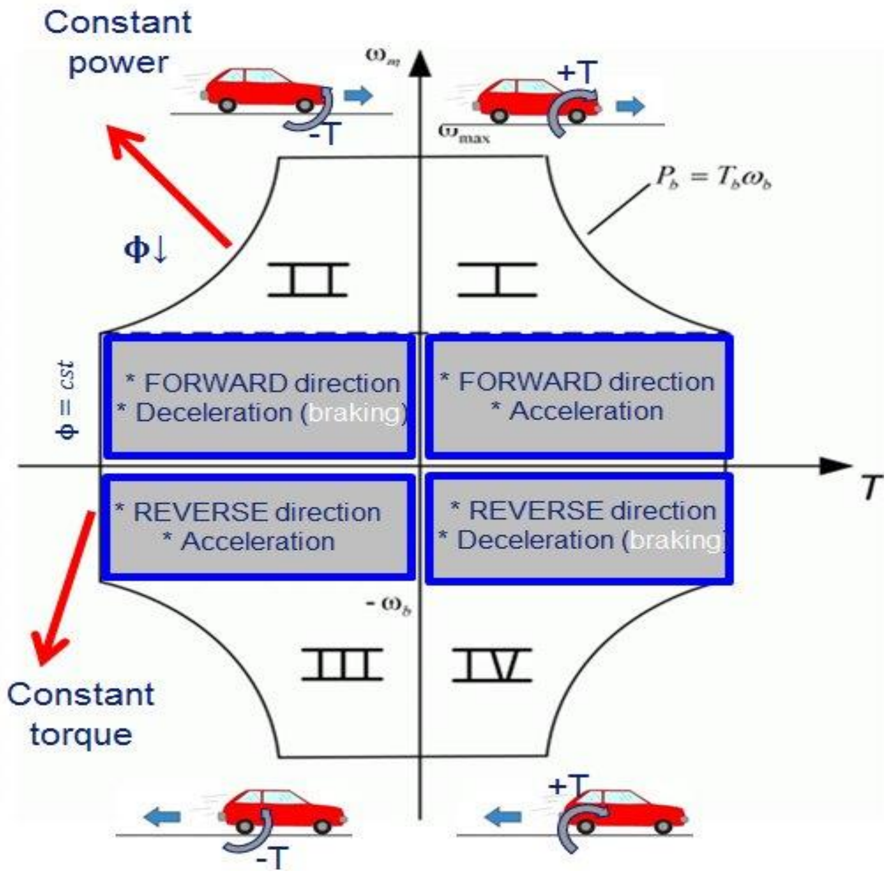
$$T_e = J\frac{d}{dt}\omega_r + B_m\omega_r + T_L \quad 6$$

DATASET

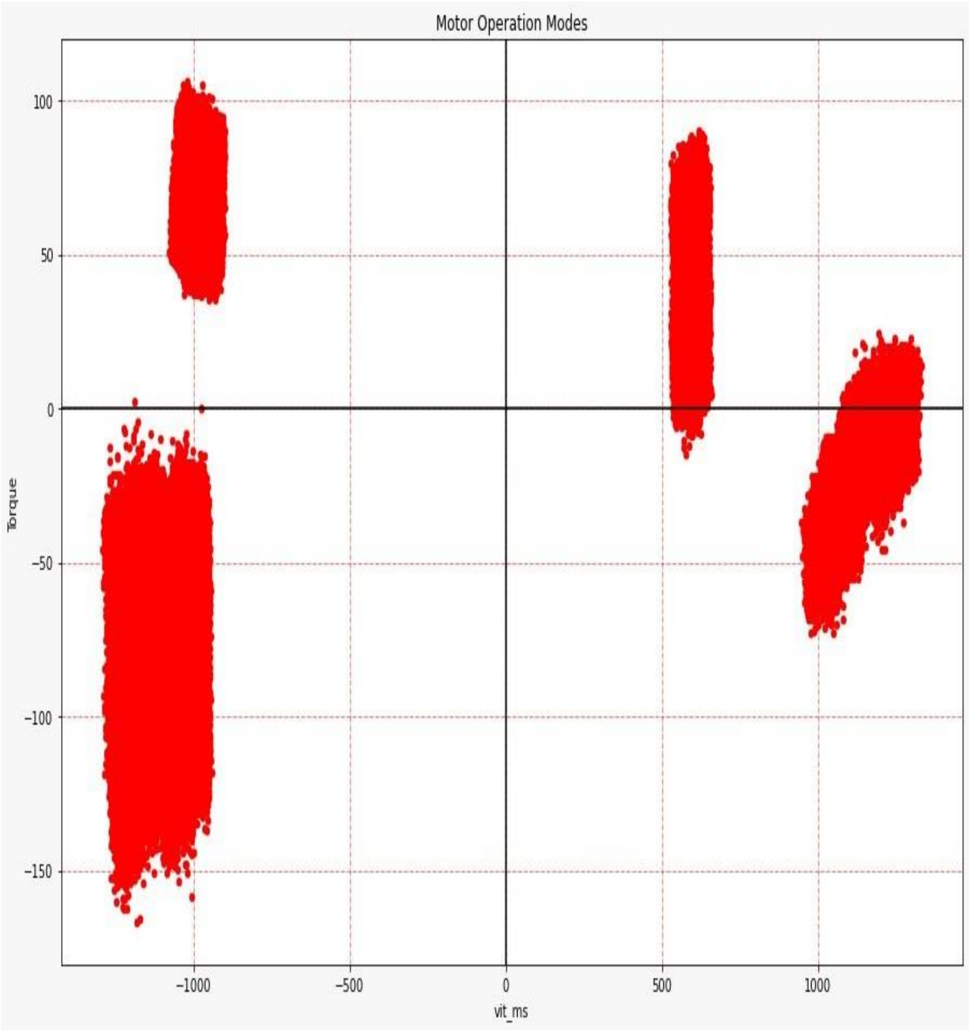
- **10** motor testbench results
- **390k** datapoints with 46 parameters each, given in Excel-sheets
- measured and **estimated** inputs and one output parameter for torque

- **Problems:**
  - Some of the data is estimated
  - Not very balanced dataset

DATA ISSUES



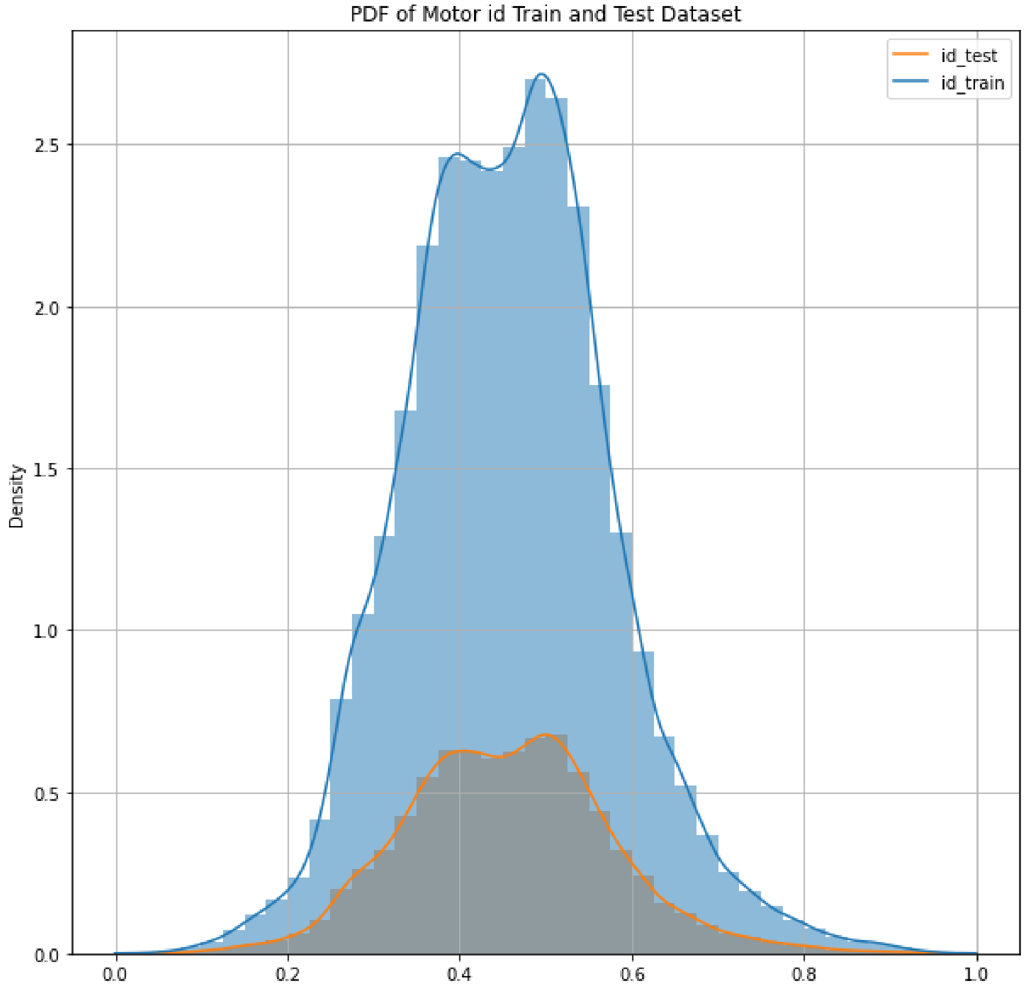
Sarrazin, Mathieu & Gillijns, Steven & Janssens, Karl & Van der Auweraer, Herman & Verhaege, Kevin. (2014). Vibro-acoustic measurements and techniques for electric automotive applications. INTERNOISE 2014 - 43rd International Congress on Noise Control Engineering: Improving the World Through Noise Control.

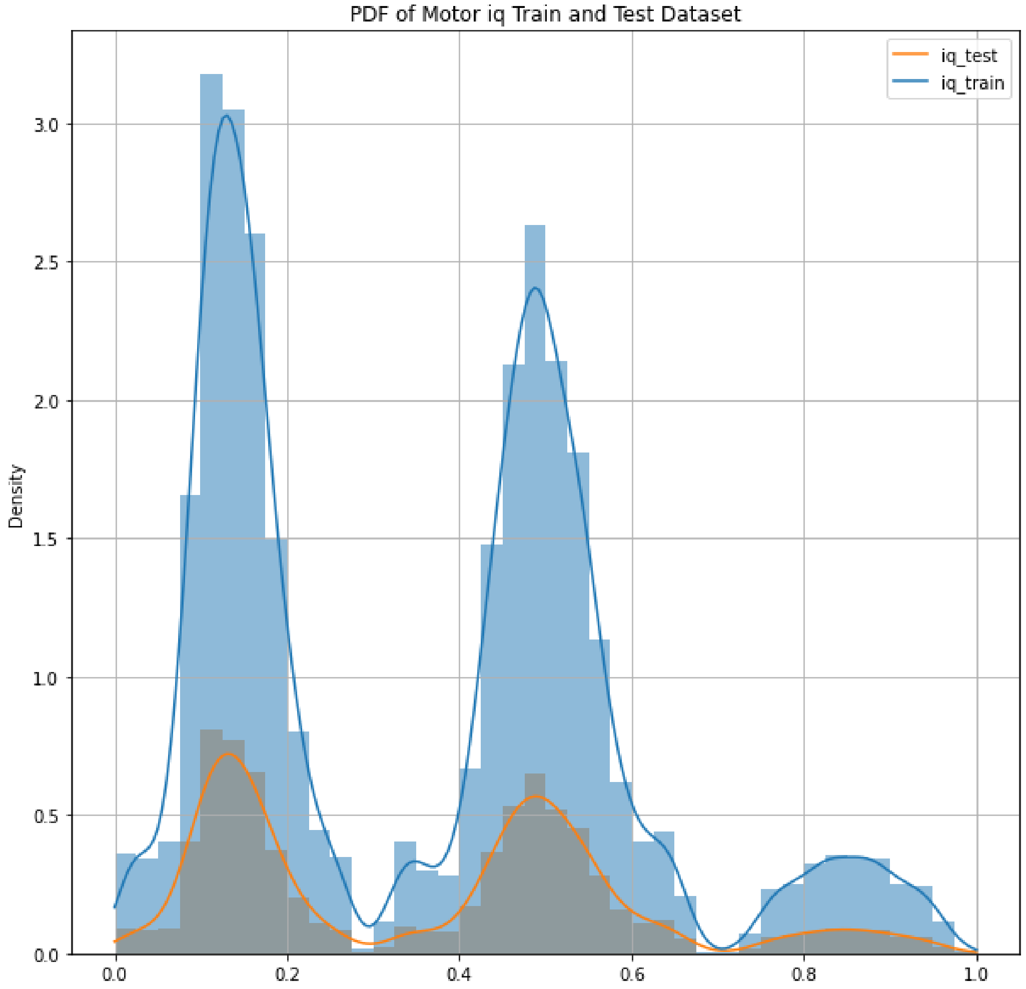


DATA INPUT

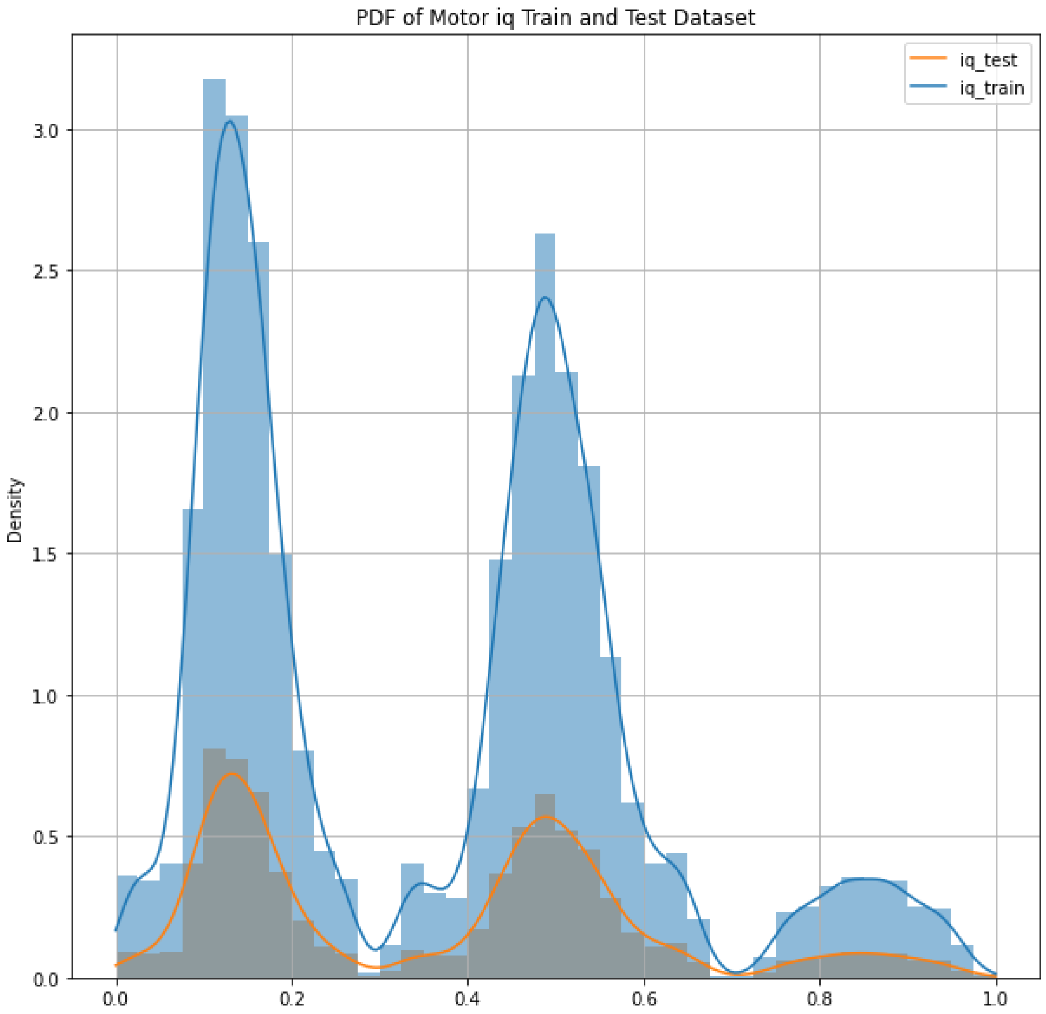
- **4 Inputs**: 2 Currents (i_ds and i_dq) in A and 2 voltages (v_d and v_q) in V
- **1 output**: torque in Nm
- **Normalize** the data to the unit interval by MinMaxScaler
- **Training split** of 80%

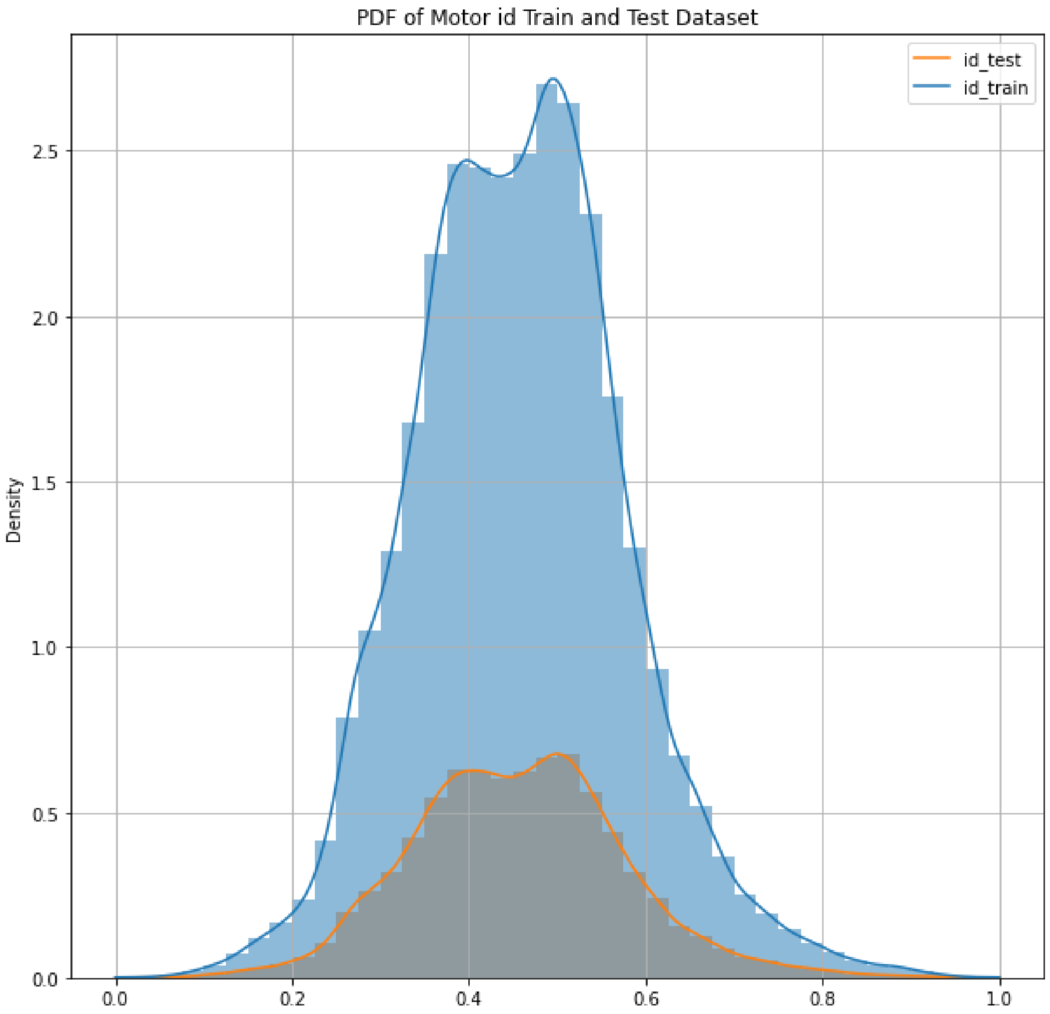
DATA DISTRIBUTION



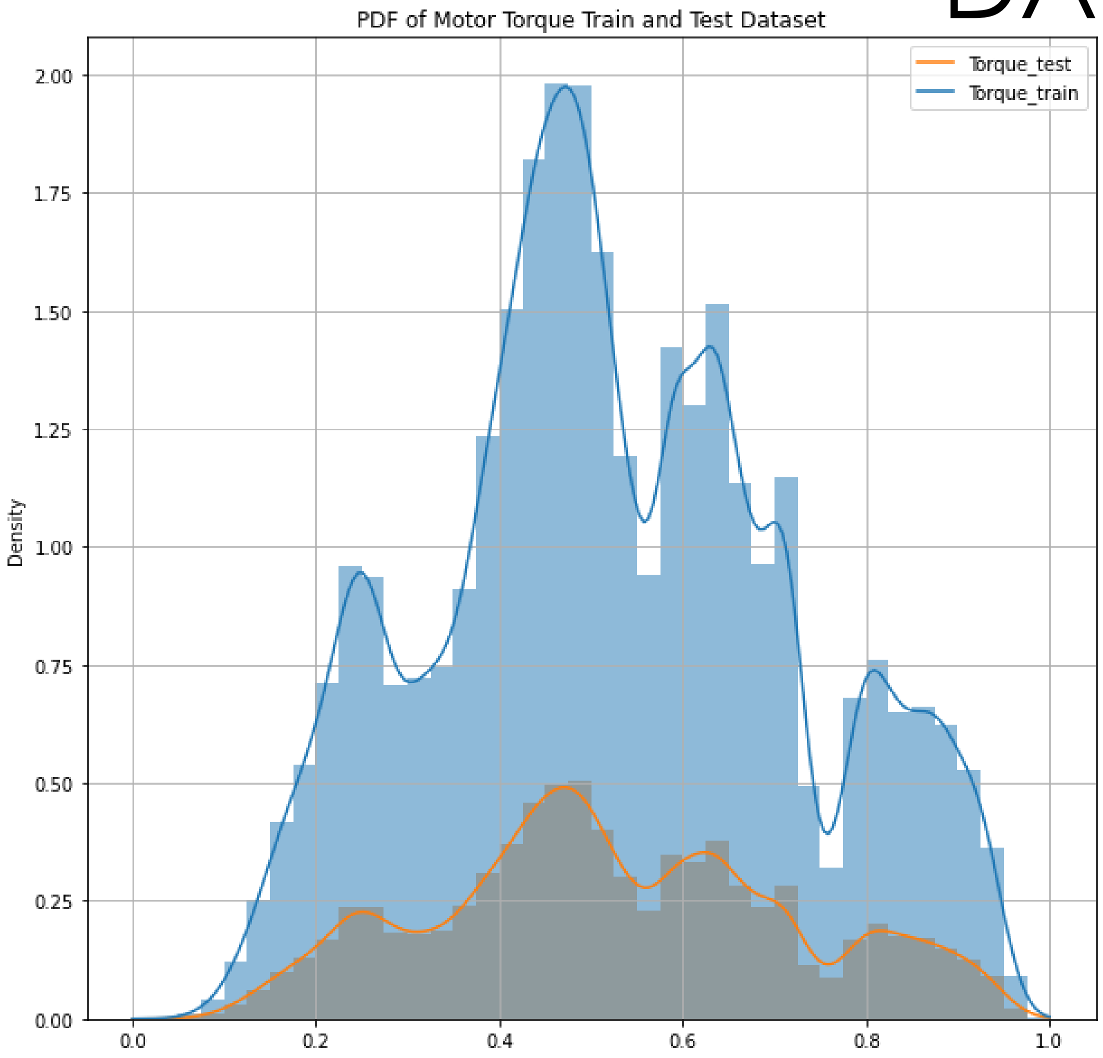


DATA DISTRIBUTION





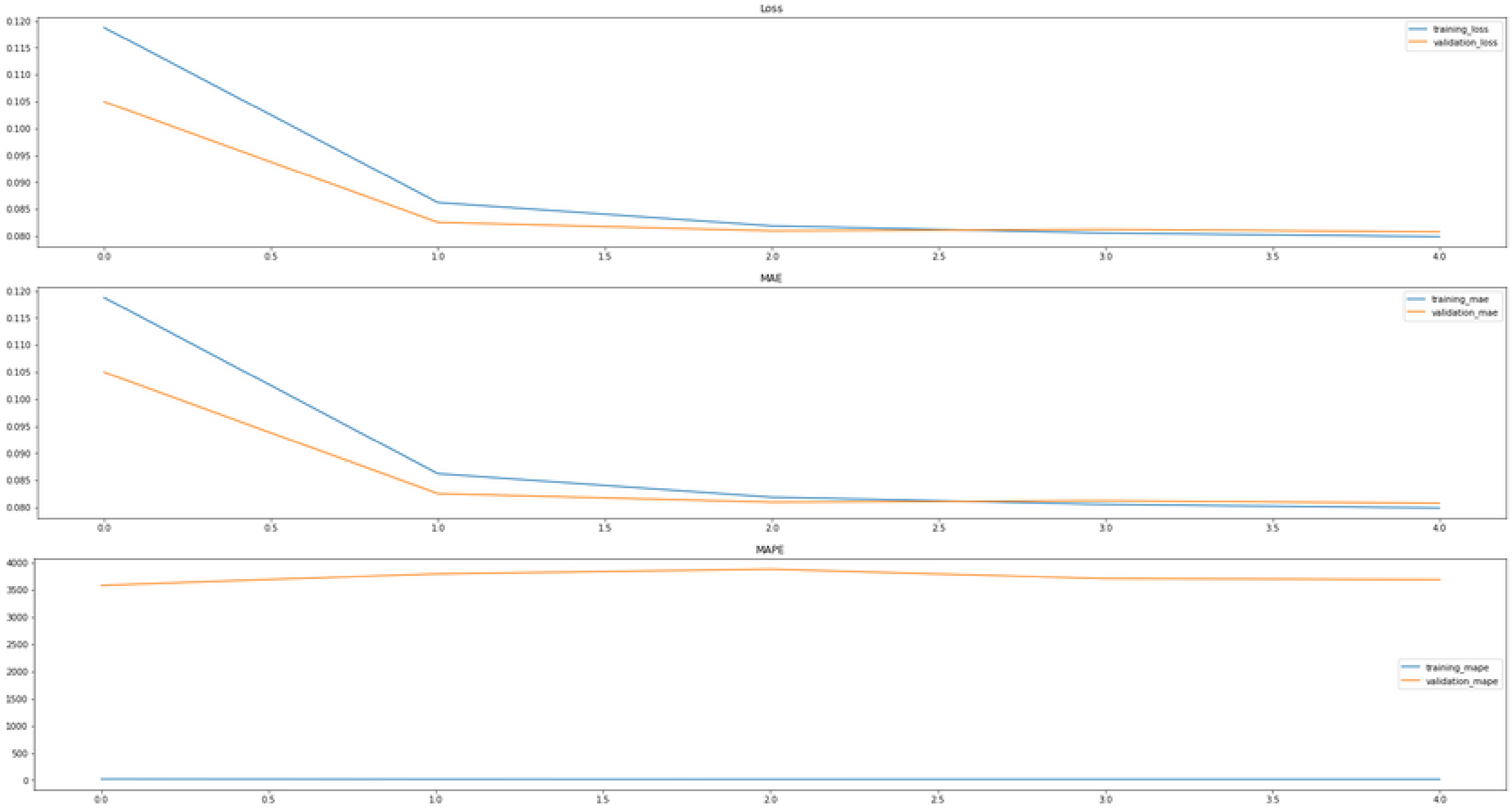
DATA DISTRIBUTION



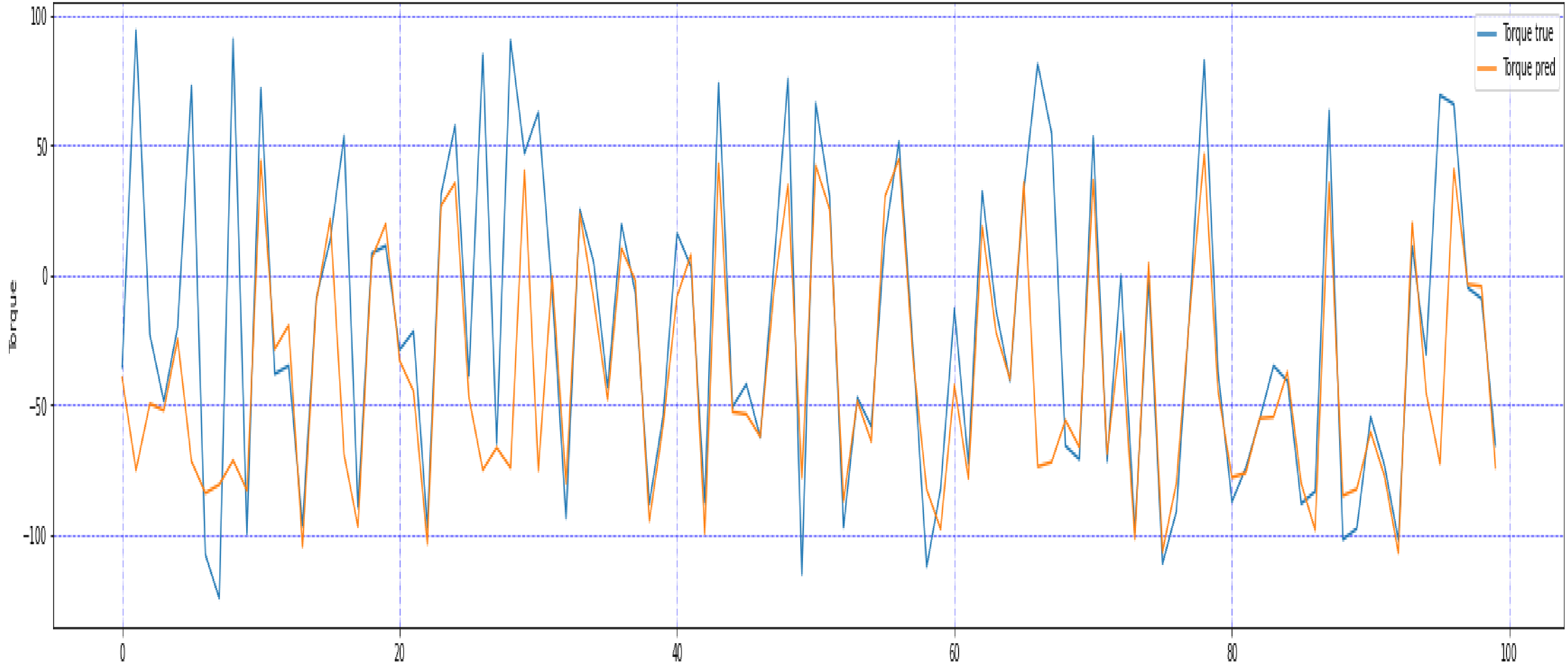
FIRST APPROACH

```
EPOCHS=5

model =  Sequential(name='Torque_Prediciton_Model')
model.add(Input(name='Input_Layer', shape=( x_train.shape[-1])))
model.add(Dense(units=50, activation='gelu', name='Hidden_Layer'))
model.add(Dense(units=y_train.shape[-1], activation='linear', name='Output_Layer'))

optimizer = Adam(learning_rate=1e-3, amsgrad=False)
model.compile(loss=['mae','mse','mape'], optimizer=optimizer, metrics=['mse', 'mape', 'mae'])
model.summary()
```

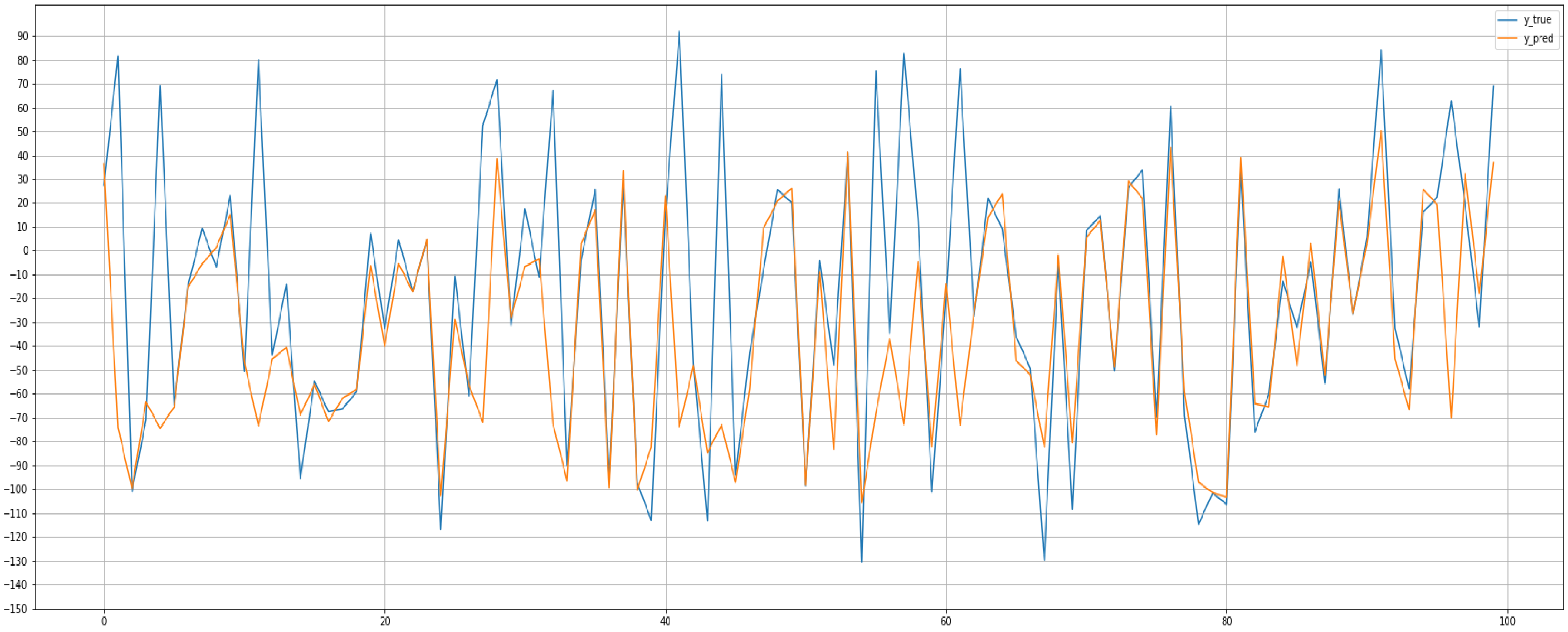
FIRST APPROACH



FIRST APPROACH



FIRST APPROACH



SECOND APPROACH

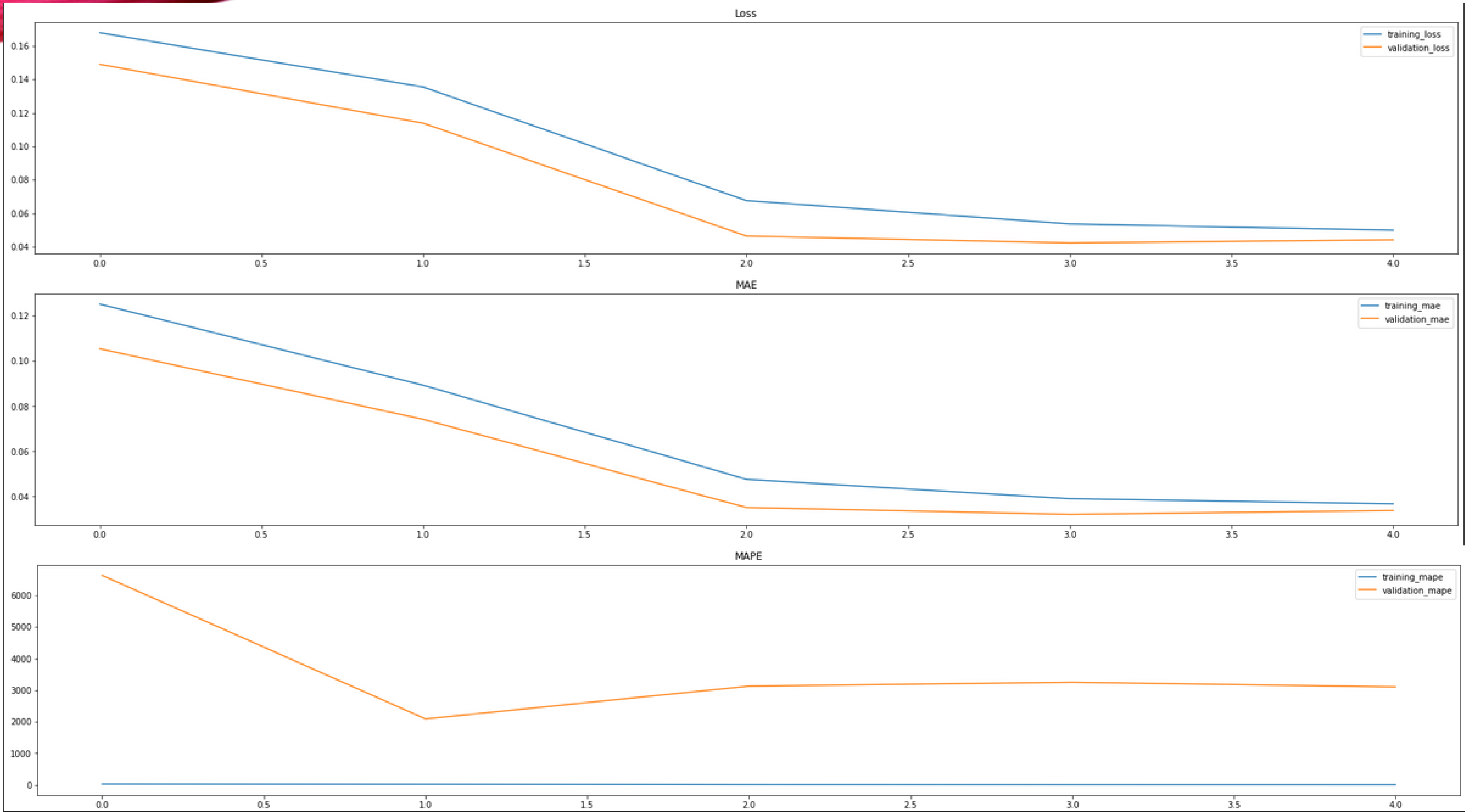
```
def create_model():
    inputs = keras.Input(name='InputLayer', shape=(num_x_signals))
    dropout=0.01
    num_layers=2
    num_units = 50
    steps = 5

    x_encode = Dense(units=num_units,  activation='gelu', use_bias=True)(inputs)
    x_encode = tf.expand_dims((x_encode), axis=-2)
    x_encode = Conv1D(filters=num_units, kernel_size=4, strides=1, activation='gelu', data_format='channels_first')(x_encode)
    x_encode = Dropout(dropout)(x_encode)

    for i in range(num_layers):
        #num_units = num_units - steps
        # x_encode = LayerNormalization()(x_encode)
        x_encode = Dense(units=num_units,  activation='gelu', use_bias=True)(x_encode)
        # x_encode = tf.expand_dims((x_encode), axis=1)
        #x_encode = Conv1D(filters=num_units, kernel_size=4, strides=1, activation='gelu', data_format='channels_first')(x_encode)
        x_encode = Dropout(dropout)(x_encode)

    x_encode = Flatten()(x_encode)
    x_encode = Dropout(dropout)(x_encode)
    outputs = Dense(units=num_y_signals,  activation='gelu', use_bias=True, name='Output_Layer')(x_encode)

    #optimizer = Adam(learning_rate=1e-3, amsgrad=True)
    model = Model(inputs, outputs, name='Torque_Prediciton_model')
    optimizer = Adam(learning_rate=1e-3, amsgrad=False)
    model.compile(loss=CustomLoss, optimizer=optimizer, metrics=['mse', 'mape', 'mae'])
    model.build((None, 1, num_x_signals))

    return model

model = create_model()
model.summary()
```

SECOND APPROACH

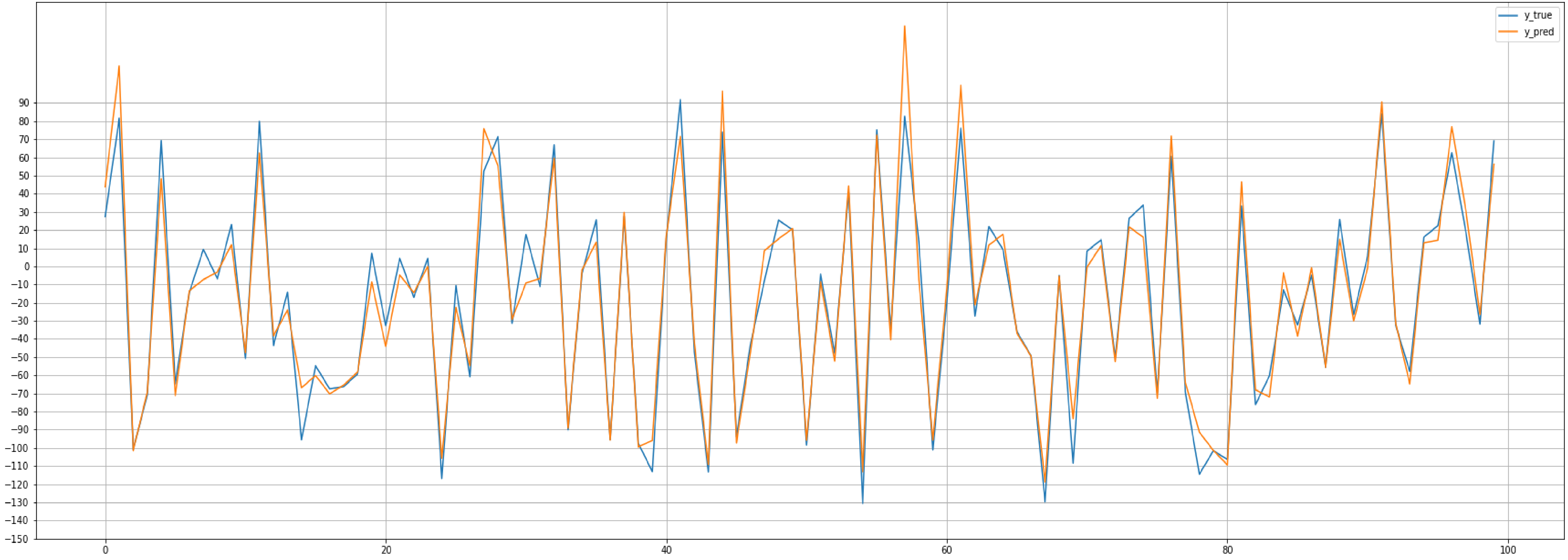
- 3 dense layers
- 1 Convolution1D layer
- 4 dropout layers

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 4)]               0

 dense (Dense)               (None, 50)                250

 tf.expand_dims (TFOpLambda)  (None, 1, 50)            0

 conv1d (Conv1D)             (None, 50, 47)            250

 dropout (Dropout)           (None, 50, 47)            0

 dense_1 (Dense)             (None, 50, 50)            2400

 dropout_1 (Dropout)         (None, 50, 50)            0

 dense_2 (Dense)             (None, 50, 50)            2550

 dropout_2 (Dropout)         (None, 50, 50)            0

 flatten (Flatten)           (None, 2500)              0

 dropout_3 (Dropout)         (None, 2500)              0

 Output_Layer (Dense)        (None, 1)                 2501

=================================================================
Total params: 7,951
Trainable params: 7,951
Non-trainable params: 0
_________________________________________________________________
```

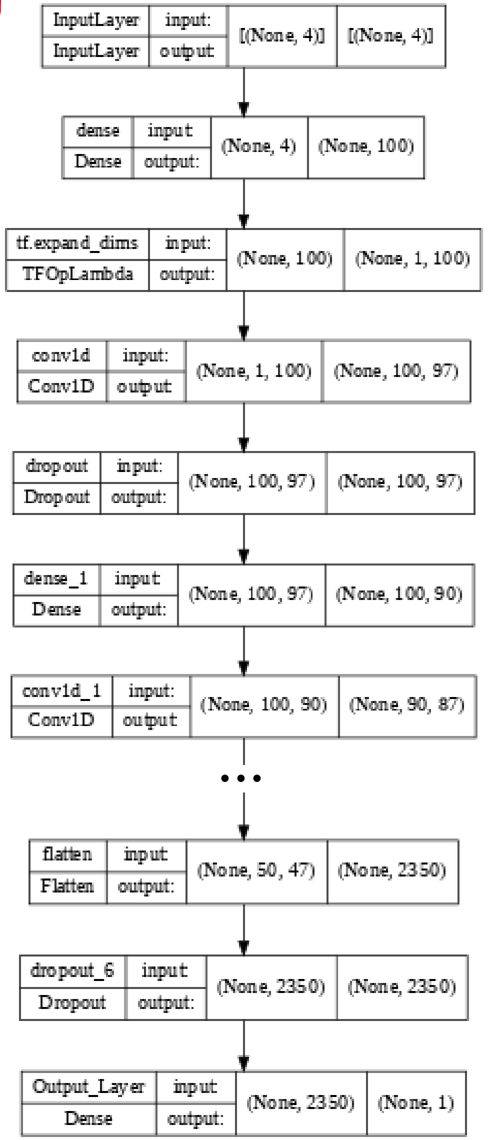
SECOND APPROACH



SECOND APPROACH



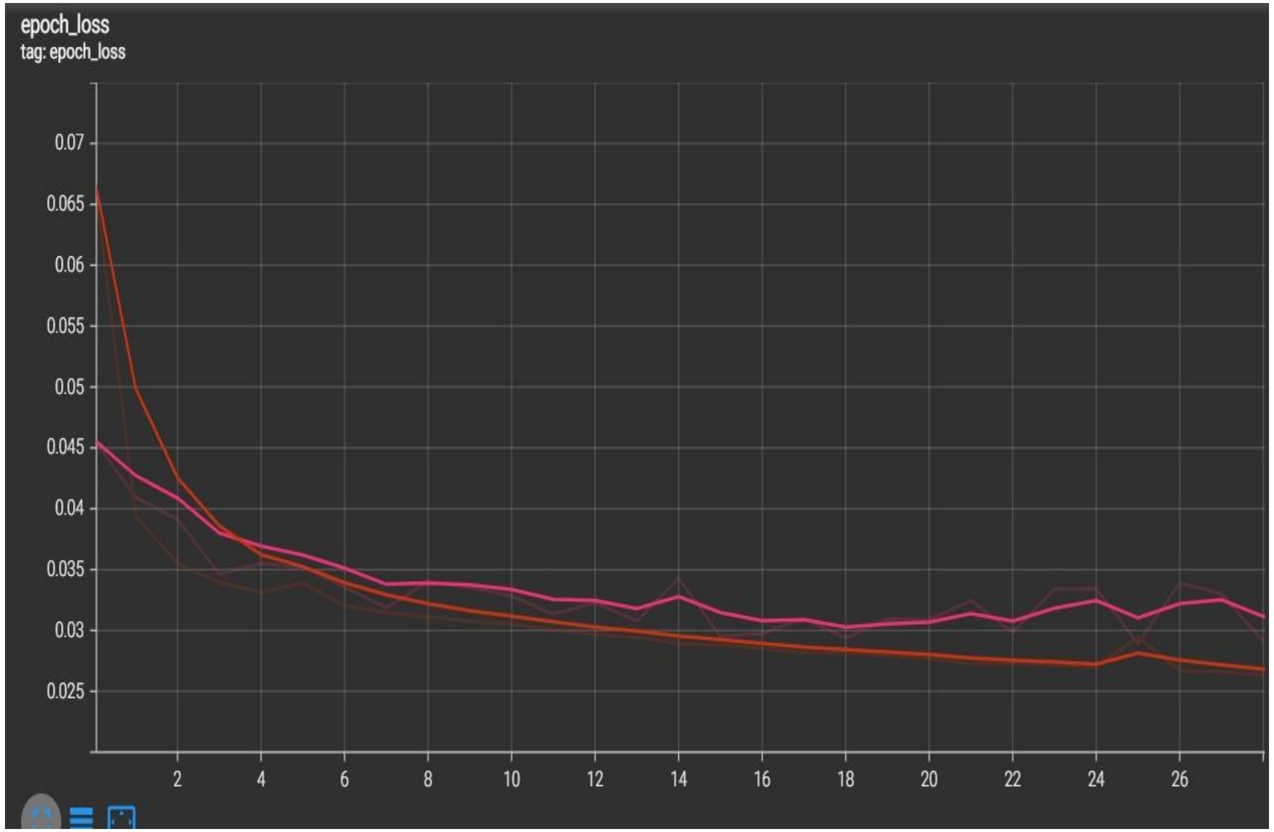
FINAL MODEL

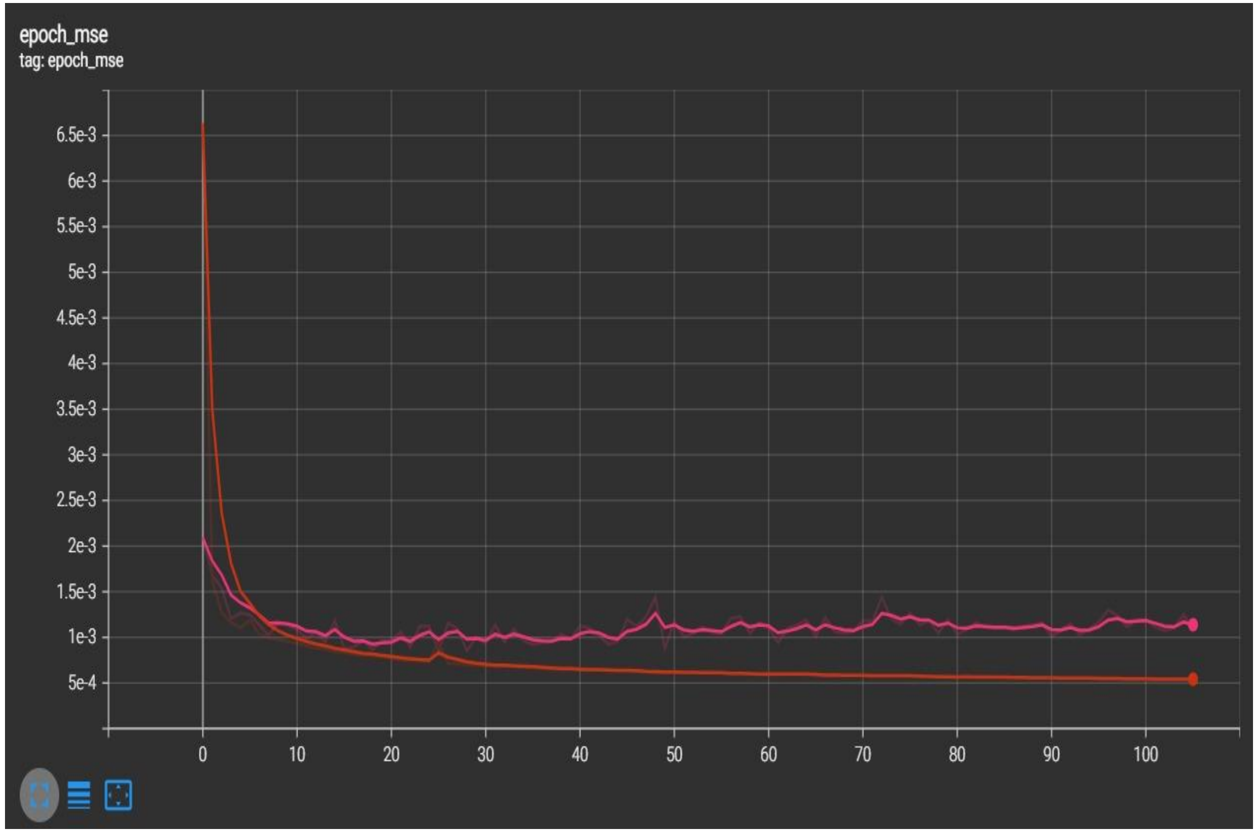



```
Model: "Torque_Prediciton_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 4)]               0
 dense_1 (Dense)             (None, 100)               500
 tf.expand_dims (TFOpLambda)  (None, 1, 100)           0
 conv1d (Conv1D)             (None, 100, 97)           500
 dropout (Dropout)           (None, 100, 97)           0
 dense_2 (Dense)             (None, 100, 90)           8820
 conv1d_1 (Conv1D)           (None, 90, 87)            36090
 dropout_1 (Dropout)         (None, 90, 87)            0
 dense_3 (Dense)             (None, 90, 80)            7040
 conv1d_2 (Conv1D)           (None, 80, 77)            28880
 dropout_2 (Dropout)         (None, 80, 77)            0
 dense_4 (Dense)             (None, 80, 70)            5460
 conv1d_3 (Conv1D)           (None, 70, 67)            22470
 dropout_3 (Dropout)         (None, 70, 67)            0
 dense_5 (Dense)             (None, 70, 60)            4080
 conv1d_4 (Conv1D)           (None, 60, 57)            16860
 dropout_4 (Dropout)         (None, 60, 57)            0
 dense_6 (Dense)             (None, 60, 50)            2900
 conv1d_5 (Conv1D)           (None, 50, 47)            12050
 dropout_5 (Dropout)         (None, 50, 47)            0
 flatten (Flatten)           (None, 2350)              0
 dropout_6 (Dropout)         (None, 2350)              0
 Output_Layer (Dense)        (None, 1)                 2351
=================================================================
Total params: 148,001
Trainable params: 148,001
Non-trainable params: 0
```

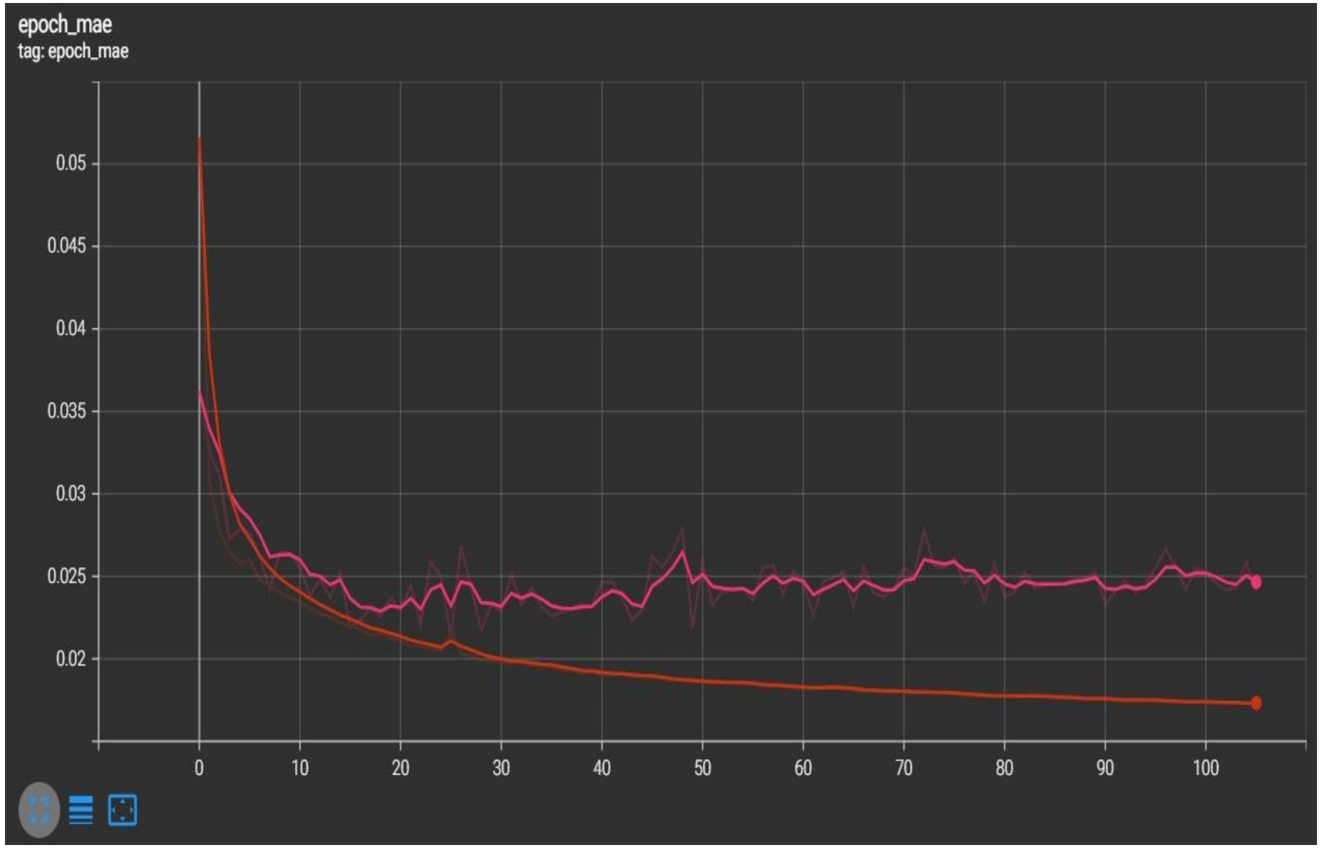
- Metrics: MAE, MSE and MAPE
- 5CNN +100 units DNN -10/step
- GeLu for CNNs and ReLu for DNNs
- Dropout layers 0.2
- Optimizer=Adam + amsgrad
- Batch size: 256
- Early stopping patience=40

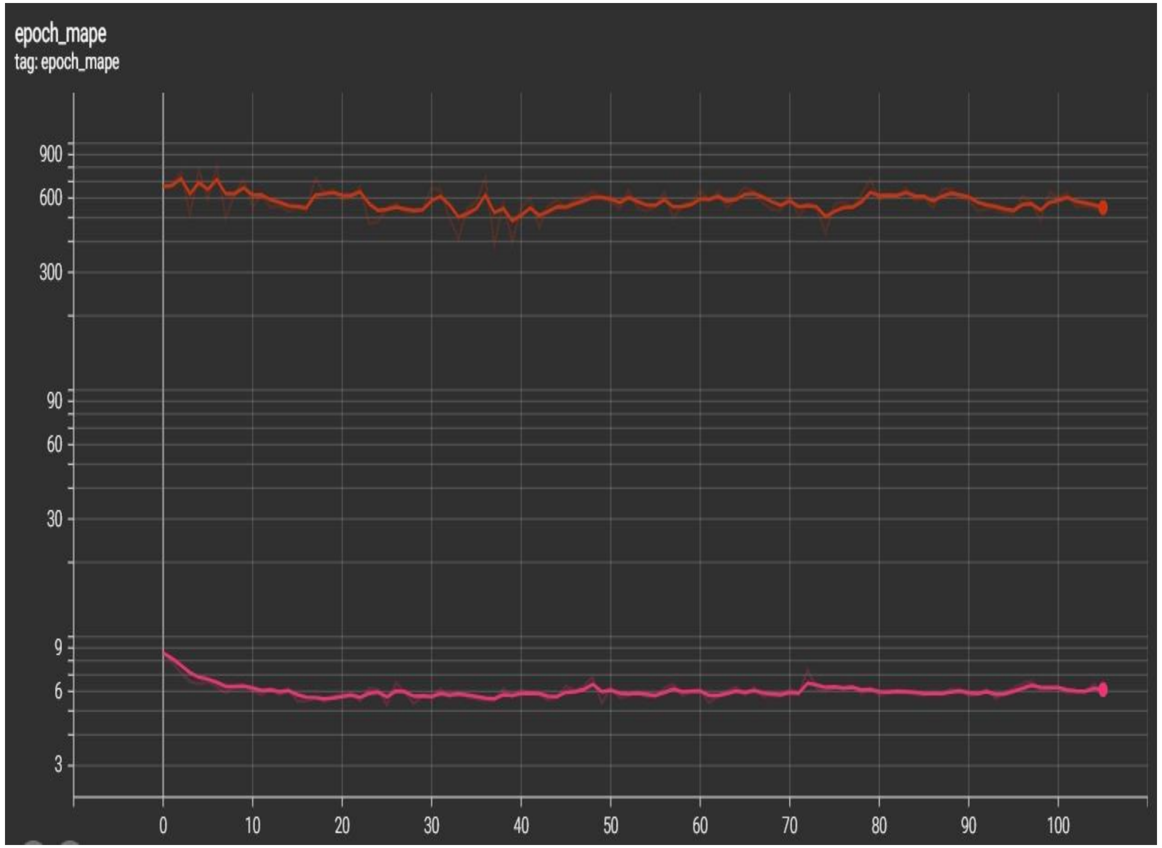
FINAL MODEL


epoch_loss
tag: epoch_loss
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0.025
2
4
6
8
10
12
14
16
18
20
22
24
26

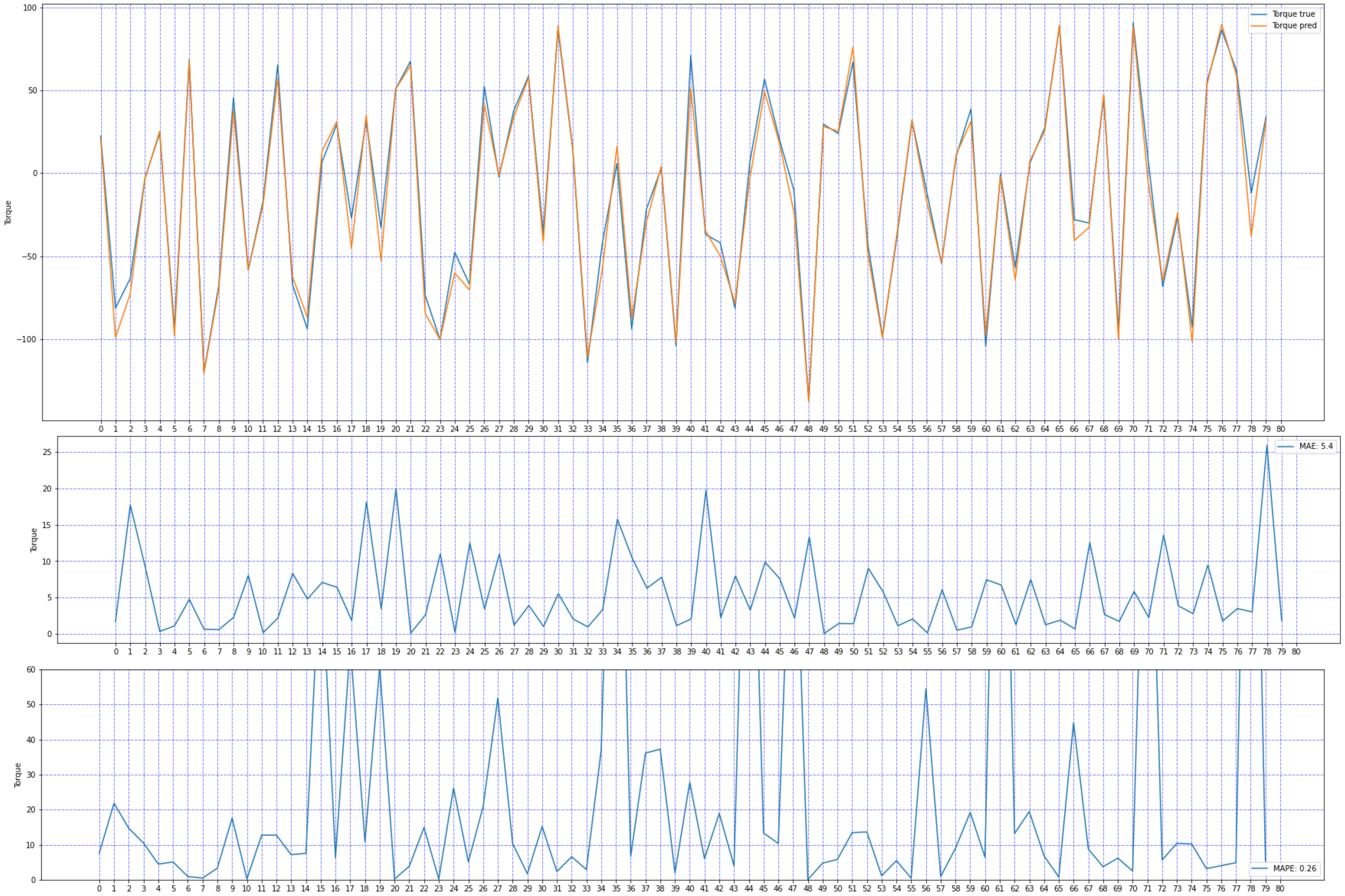

epoch_mse
tag: epoch_mse
6.5e-3
6e-3
5.5e-3
5e-3
4.5e-3
4e-3
3.5e-3
3e-3
2.5e-3
2e-3
1.5e-3
1e-3
5e-4
0
10
20
30
40
50
60
70
80
90
100

FINAL MODEL


epoch_mae
tag: epoch_mae
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0
10
20
30
40
50
60
70
80
90
100


epoch_mape
tag: epoch_mape
900
600
300
90
60
30
9
6
3
0
10
20
30
40
50
60
70
80
90
100

FINAL MODEL



- **RMSE = 7.66**
- **MAE = 5.4**

Benchmark, to be proved and beaten, **Model Reference Adaptive System**:

- RMSE = 0.3871

NEXT STEPS

- Improve or add meaningful metrics
  - SMAPE, MASE
  - Electrical Engineering performance metrics

$$\text{MASE} = \text{mean}\left(\frac{|e_j|}{\frac{1}{T-1}\sum_{t=2}^{T}|Y_t - Y_{t-1}|}\right)$$

- Check against a simulation of a motor
- Test with a different motor / dataset
- Adding speed of the motor as input for a more precise model

- Real implementation requirements
  - Sampling time
  - Embedded systems

THANKS FOR LISTENING!