I used:  
X\_jma shape: (85325, 10, 10)

y\_jma shape: (85325, 1)  
  
import tensorflow as tf  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv1D, MaxPooling1D, BatchNormalization, GRU, Dropout, Dense  
from tensorflow.keras.regularizers import l2  
from tensorflow.keras.optimizers import Adam  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
  
# Define the CNN-GRU model  
def build\_cnn\_gru\_model(input\_shape):  
 model = Sequential()  
  
 # CNN layers  
 model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=input\_shape))  
 model.add(BatchNormalization())  
 model.add(MaxPooling1D(pool\_size=2))  
  
 model.add(Conv1D(filters=128, kernel\_size=3, activation='relu'))  
 model.add(BatchNormalization())  
 model.add(MaxPooling1D(pool\_size=2))  
  
 # GRU layers  
 model.add(GRU(units=256, return\_sequences=True, kernel\_regularizer=l2(0.001)))  
 model.add(Dropout(0.3))  
 model.add(GRU(units=128, return\_sequences=False, kernel\_regularizer=l2(0.001)))  
 model.add(Dropout(0.3))  
  
 # Dense layers  
 model.add(Dense(units=64, activation='relu', kernel\_regularizer=l2(0.001)))  
 model.add(Dropout(0.4))  
 model.add(Dense(units=1, activation='linear')) # Regression output  
  
 # Compile the model with a learning rate scheduler  
 model.compile(optimizer=Adam(learning\_rate=0.0005), loss='mse', metrics=['mae'])  
  
 return model

# Build the model  
input\_shape = (10, 10) # Adjust based on your data  
model = build\_cnn\_gru\_model(input\_shape)

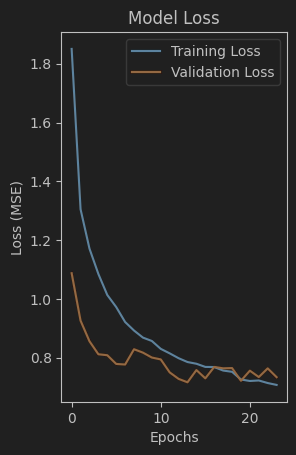
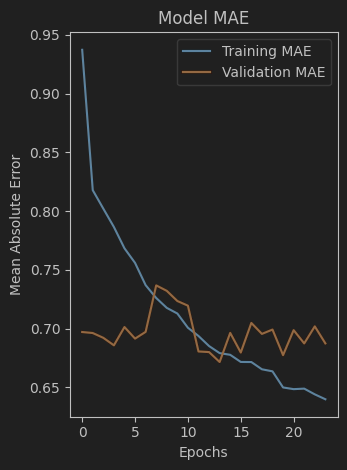
# Define Callbacks  
early\_stopping = EarlyStopping(  
 monitor='val\_loss',  
 patience=10, # Stop if no improvement after 10 epochs  
 restore\_best\_weights=True  
)  
reduce\_lr = ReduceLROnPlateau(  
 monitor='val\_loss',  
 factor=0.5, # Reduce learning rate by half  
 patience=5, # After 5 epochs of no improvement  
 min\_lr=1e-6 # Minimum learning rate  
)

# Adjust the input shape to match the new data  
input\_shape = (X\_jma.shape[1], X\_jma.shape[2]) # (10, 10)

# Train the model  
history = model.fit(  
 X\_jma, y\_jma,  
 validation\_split=0.2,  
 epochs=100, # Increased to 100 to allow for more learning  
 batch\_size=64,  
 callbacks=[early\_stopping, reduce\_lr]  
)

I got:  
Training Loss: 0.7965

Training MAE: 0.6922



Epoch 1/100

1067/1067 [==============================] - 11s 8ms/step - loss: 1.8504 - mae: 0.9374 - val\_loss: 1.0872 - val\_mae: 0.6970 - lr: 5.0000e-04

Epoch 2/100

1067/1067 [==============================] - 8s 8ms/step - loss: 1.3047 - mae: 0.8178 - val\_loss: 0.9257 - val\_mae: 0.6962 - lr: 5.0000e-04

Epoch 3/100

1067/1067 [==============================] - 8s 8ms/step - loss: 1.1708 - mae: 0.8021 - val\_loss: 0.8556 - val\_mae: 0.6921 - lr: 5.0000e-04

Epoch 4/100

1067/1067 [==============================] - 8s 8ms/step - loss: 1.0845 - mae: 0.7865 - val\_loss: 0.8114 - val\_mae: 0.6857 - lr: 5.0000e-04

Epoch 5/100

1067/1067 [==============================] - 8s 8ms/step - loss: 1.0130 - mae: 0.7683 - val\_loss: 0.8080 - val\_mae: 0.7013 - lr: 5.0000e-04

Epoch 6/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.9720 - mae: 0.7559 - val\_loss: 0.7784 - val\_mae: 0.6914 - lr: 5.0000e-04

Epoch 7/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.9211 - mae: 0.7370 - val\_loss: 0.7765 - val\_mae: 0.6971 - lr: 5.0000e-04

Epoch 8/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.8920 - mae: 0.7260 - val\_loss: 0.8284 - val\_mae: 0.7367 - lr: 5.0000e-04

Epoch 9/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.8680 - mae: 0.7176 - val\_loss: 0.8169 - val\_mae: 0.7321 - lr: 5.0000e-04

Epoch 10/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.8565 - mae: 0.7128 - val\_loss: 0.8002 - val\_mae: 0.7234 - lr: 5.0000e-04

Epoch 11/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.8297 - mae: 0.7007 - val\_loss: 0.7938 - val\_mae: 0.7195 - lr: 5.0000e-04

Epoch 12/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.8144 - mae: 0.6937 - val\_loss: 0.7497 - val\_mae: 0.6805 - lr: 5.0000e-04

Epoch 13/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7980 - mae: 0.6851 - val\_loss: 0.7276 - val\_mae: 0.6799 - lr: 5.0000e-04

Epoch 14/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7849 - mae: 0.6790 - val\_loss: 0.7160 - val\_mae: 0.6715 - lr: 5.0000e-04

Epoch 15/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7791 - mae: 0.6776 - val\_loss: 0.7576 - val\_mae: 0.6963 - lr: 5.0000e-04

Epoch 16/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7682 - mae: 0.6715 - val\_loss: 0.7294 - val\_mae: 0.6796 - lr: 5.0000e-04

Epoch 17/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7679 - mae: 0.6715 - val\_loss: 0.7682 - val\_mae: 0.7048 - lr: 5.0000e-04

Epoch 18/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7559 - mae: 0.6653 - val\_loss: 0.7635 - val\_mae: 0.6954 - lr: 5.0000e-04

Epoch 19/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7517 - mae: 0.6635 - val\_loss: 0.7644 - val\_mae: 0.6991 - lr: 5.0000e-04

Epoch 20/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7255 - mae: 0.6498 - val\_loss: 0.7211 - val\_mae: 0.6773 - lr: 2.5000e-04

Epoch 21/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7204 - mae: 0.6484 - val\_loss: 0.7555 - val\_mae: 0.6986 - lr: 2.5000e-04

Epoch 22/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7222 - mae: 0.6488 - val\_loss: 0.7338 - val\_mae: 0.6873 - lr: 2.5000e-04

Epoch 23/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7134 - mae: 0.6439 - val\_loss: 0.7633 - val\_mae: 0.7018 - lr: 2.5000e-04

Epoch 24/100

1067/1067 [==============================] - 8s 8ms/step - loss: 0.7070 - mae: 0.6398 - val\_loss: 0.7334 - val\_mae: 0.6874 - lr: 2.5000e-04