Create Model

In [ ]:
```
# K.clear_session()

model_lstm = Sequential()
model_lstm.add(LSTM(64, input_shape=(1, 12), activation='relu'))
model_lstm.add(Dropout(0.2))
model_lstm.add(Dense(32,activation='relu'))
model_lstm.add(Dense(12,activation='sigmoid'))

# Compile the model
model_lstm.compile(optimizer='adam', loss='binary_crossentropy', metrics=

print(train.shape)
```

```
(112552, 1, 12)
```

Train Model

Train the model with the training data and validate it with the test data.

Training configuration:

- 30 epochs
- Batch size of 64
- Early stopping to prevent overfitting
- Learning rate reduction on plateau
- Model checkpoint to save the best model based on validation loss

In [ ]:
```
# Define callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_be
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,
model_checkpoint = ModelCheckpoint('best_models/best_model_LSTM.keras', m



# Train the model
model_lstm_output = model_lstm.fit(train, y_train, epochs=30,  batch_size
```

```
Epoch 1/30
WARNING: All log messages before absl::InitializeLog() is called are writt
en to STDERR
I0000 00:00:1718691056.393010   45583 service.cc:145] XLA service 0x7d05d4
004120 initialized for platform CUDA (this does not guarantee that XLA wil
l be used). Devices:
I0000 00:00:1718691056.393147   45583 service.cc:153]   StreamExecutor dev
ice (0): NVIDIA GeForce GTX 1060 6GB, Compute Capability 6.1
2024-06-18 07:10:56.429617: I tensorflow/compiler/mlir/tensorflow/utils/du
mp_mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CR
ASH_REPRODUCER_DIRECTORY` to enable.
2024-06-18 07:10:56.741958: I external/local_xla/xla/stream_executor/cuda/
cuda_dnn.cc:465] Loaded cuDNN version 8907
```

```
  82/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.8168 - loss: 0.66
92
I0000 00:00:1718691058.689651   45583 device_compiler.h:188] Compiled clus
ter using XLA!  This line is logged at most once for the lifetime of the p
rocess.
```

```
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 12s 4ms/step - accuracy: 0.4099 - loss: 0.1
667 - val_accuracy: 0.8630 - val_loss: 0.0086 - learning_rate: 0.0010
Epoch 2/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.6943 - loss: 0.00
78 - val_accuracy: 0.1798 - val_loss: 0.0062 - learning_rate: 0.0010
Epoch 3/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.2251 - loss: 0.00
59 - val_accuracy: 0.0689 - val_loss: 0.0048 - learning_rate: 0.0010
Epoch 4/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1380 - loss: 0.00
46 - val_accuracy: 0.0849 - val_loss: 0.0039 - learning_rate: 0.0010
Epoch 5/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.0974 - loss: 0.00
40 - val_accuracy: 0.0698 - val_loss: 0.0033 - learning_rate: 0.0010
Epoch 6/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.0752 - loss: 0.00
31 - val_accuracy: 0.0613 - val_loss: 0.0027 - learning_rate: 0.0010
Epoch 7/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.0595 - loss: 0.00
27 - val_accuracy: 0.0563 - val_loss: 0.0024 - learning_rate: 0.0010
Epoch 8/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.0535 - loss: 0.00
23 - val_accuracy: 0.0562 - val_loss: 0.0021 - learning_rate: 0.0010
Epoch 9/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.0527 - loss: 0.00
22 - val_accuracy: 0.0513 - val_loss: 0.0020 - learning_rate: 0.0010
Epoch 10/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 1ms/step - accuracy: 0.0521 - loss: 0.00
20 - val_accuracy: 0.0516 - val_loss: 0.0017 - learning_rate: 0.0010
Epoch 11/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.0536 - loss: 0.00
20 - val_accuracy: 0.0589 - val_loss: 0.0017 - learning_rate: 0.0010
Epoch 12/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0553 - loss: 0.00
16 - val_accuracy: 0.0491 - val_loss: 0.0017 - learning_rate: 0.0010
Epoch 13/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0538 - loss: 0.00
15 - val_accuracy: 0.0517 - val_loss: 0.0015 - learning_rate: 0.0010
Epoch 14/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 991us/step - accuracy: 0.0507 - loss: 0.
0015 - val_accuracy: 0.0535 - val_loss: 0.0015 - learning_rate: 0.0010
Epoch 15/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.0501 - loss: 0.00
15 - val_accuracy: 0.0514 - val_loss: 0.0013 - learning_rate: 0.0010
Epoch 16/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0510 - loss: 0.00
14 - val_accuracy: 0.0487 - val_loss: 0.0013 - learning_rate: 0.0010
Epoch 17/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0487 - loss: 0.00
14 - val_accuracy: 0.0483 - val_loss: 0.0013 - learning_rate: 0.0010
Epoch 18/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0487 - loss: 0.00
13 - val_accuracy: 0.0470 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 19/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0469 - loss: 0.00
13 - val_accuracy: 0.0485 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 20/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1000us/step - accuracy: 0.0477 - loss: 0
.0012 - val_accuracy: 0.0485 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 21/30
```

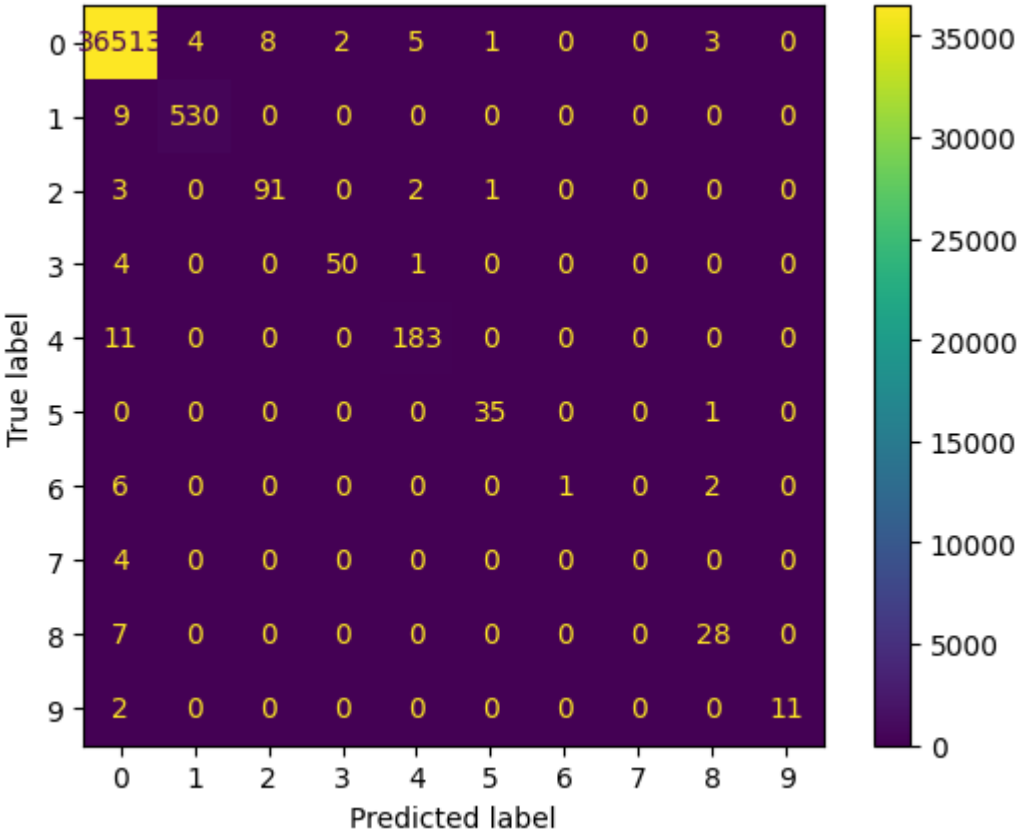
```
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0474 - loss: 0.00
13 - val_accuracy: 0.0467 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 22/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0470 - loss: 0.00
13 - val_accuracy: 0.0472 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 23/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 987us/step - accuracy: 0.0449 - loss: 0.
0011 - val_accuracy: 0.0455 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 24/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 998us/step - accuracy: 0.0461 - loss: 0.
0011 - val_accuracy: 0.0462 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 25/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0435 - loss: 0.00
12 - val_accuracy: 0.0482 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 26/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0448 - loss: 0.00
11 - val_accuracy: 0.0446 - val_loss: 0.0010 - learning_rate: 0.0010
Epoch 27/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0450 - loss: 0.00
11 - val_accuracy: 0.0464 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 28/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0459 - loss: 0.00
10 - val_accuracy: 0.0428 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 29/30
1755/1759 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.0429 - loss: 0.00
10
Epoch 29: ReduceLROnPlateau reducing learning rate to 0.000200000009499490
26.
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 2s 1ms/step - accuracy: 0.0429 - loss: 0.00
10 - val_accuracy: 0.0426 - val_loss: 0.0011 - learning_rate: 0.0010
```

RESULTS

In [ ]:
```
# clabels = ['AccelY', 'BreakY', 'TurnRightX', 'Turn LeftX', 'PositiveZ',

# cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_mat
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = conf_mat)

# display matrix
cm_display.plot()
plt.show()
```



Performance Metrics

- Accuracy = $= \frac{Correct\ Predictions}{All\ Predictions}$
- Precision for a given class = $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Predictions\ for\ the\ Class}$
- Recall for a given class = $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Instances\ of\ the\ Class}$
- Averaging is a way to get a single number for multiclass. Depending on the importance one wants to give to minority classes:
  - Macro average: Compute the metric for each class, and returns the average without considering the proportion for each class in the dataset. For instance:

    Precision = $= \frac{P_{class1} + P_{class2} + ... + P_{classn}}{N}$
  - Weighted average: Compute the metric for each class, and returns the average considering the proportion (weighted) for each class in the dataset. For instance:

    Precision = $= \frac{N_1 * P_{class1} + N_2 * P_{class2} + ... + N_n * P_{classn}}{N}$

In [ ]:
```
# Calculates performance metrics
acc = accuracy_score(y_true =  y_test, y_pred = pred)
print(f'Accuracy : {np.round(acc*100,2)}%')
```

```
precision = precision_score(y_true =  y_test, y_pred = pred, average='mac
print(f'Precision - Macro: {np.round(precision*100,2)}%')

recall = recall_score(y_true =  y_test, y_pred = pred, average='macro')
print(f'Recall - Macro: {np.round(recall*100,2)}%')

f1 = f1_score(y_true =  y_test, y_pred = pred, average='macro')
print(f'F1-score - Macro: {np.round(f1*100,2)}%')

precision = precision_score(y_true =  y_test, y_pred = pred, average='wei
print(f'Precision - Weighted: {np.round(precision*100,2)}%')

recall = recall_score(y_true =  y_test, y_pred = pred, average='weighted'
print(f'Recall - Weighted: {np.round(recall*100,2)}%')

f1 = f1_score(y_true =  y_test, y_pred = pred, average='weighted')
print(f'F1-score - Weighted: {np.round(f1*100,2)}%')
```

```
Accuracy : 99.8%
Precision - Macro: 86.0%
Recall - Macro: 75.03%
F1-score - Macro: 76.88%
Precision - Weighted: 99.79%
Recall - Weighted: 99.8%
F1-score - Weighted: 99.78%
```