Create Model

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

# Model configuration
dropout1 = 0.5
dropout2 = 0.2
dropout3 = 0.1
initial_learning_rate = 0.001

# Define the model
model_lstm = Sequential()
model_lstm.add(Bidirectional(LSTM(64, return_sequences=True), input_shape
model_lstm.add(BatchNormalization())
model_lstm.add(Dropout(dropout1))
model_lstm.add(Bidirectional(LSTM(64, return_sequences=False)))
model_lstm.add(BatchNormalization())
model_lstm.add(Dropout(dropout2))
model_lstm.add(Dense(64, activation='relu'))
model_lstm.add(BatchNormalization())
model_lstm.add(Dropout(dropout3))
model_lstm.add(Dense(12, activation='sigmoid'))



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

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_BILSTM.keras',



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

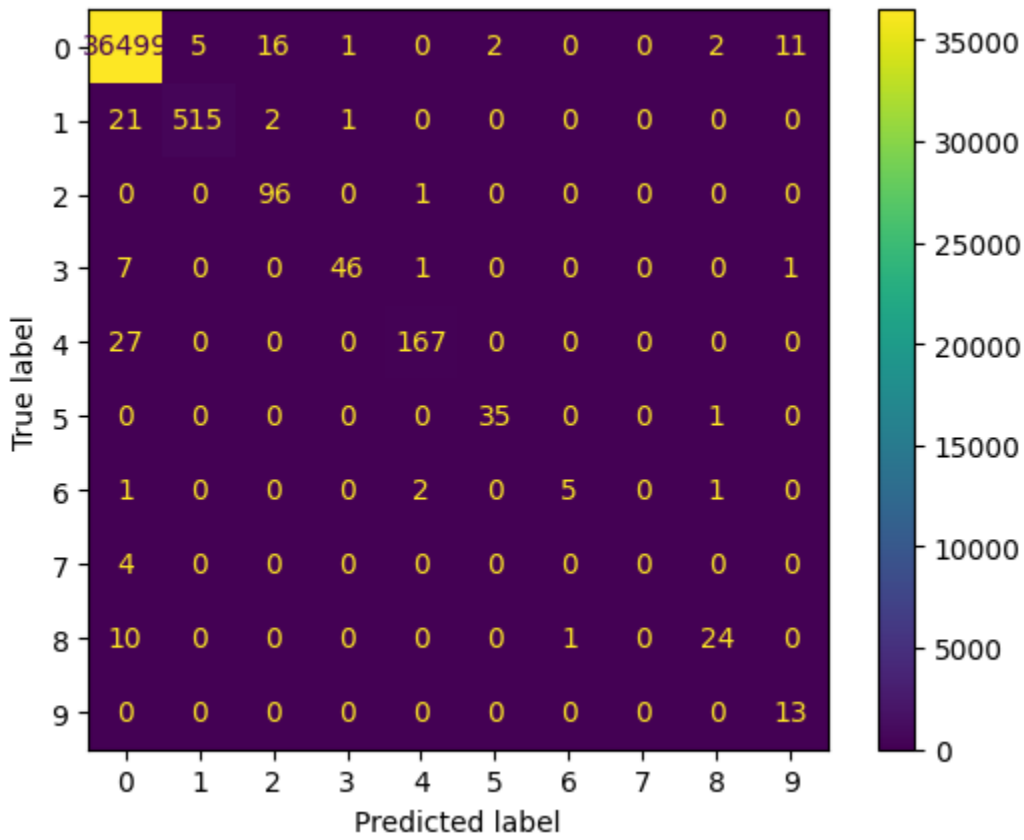
```
Epoch 1/30
2024-06-18 07:21:56.023662: I external/local_xla/xla/stream_executor/cuda/
cuda_dnn.cc:465] Loaded cuDNN version 8907
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 25s 11ms/step - accuracy: 0.0993 - loss: 0.
2120 - val_accuracy: 0.2156 - val_loss: 0.0041 - learning_rate: 0.0010
Epoch 2/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 21s 12ms/step - accuracy: 0.1781 - loss: 0.
0058 - val_accuracy: 0.2757 - val_loss: 0.0027 - learning_rate: 0.0010
Epoch 3/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.1962 - loss: 0.0
044 - val_accuracy: 0.2210 - val_loss: 0.0025 - learning_rate: 0.0010
Epoch 4/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.2180 - loss: 0.0
034 - val_accuracy: 0.2295 - val_loss: 0.0026 - learning_rate: 0.0010
Epoch 5/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.2266 - loss: 0.0
030 - val_accuracy: 0.1664 - val_loss: 0.0019 - learning_rate: 0.0010
Epoch 6/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 16s 9ms/step - accuracy: 0.2518 - loss: 0.0
028 - val_accuracy: 0.2284 - val_loss: 0.0020 - learning_rate: 0.0010
Epoch 7/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 23s 10ms/step - accuracy: 0.2339 - loss: 0.
0024 - val_accuracy: 0.2698 - val_loss: 0.0025 - learning_rate: 0.0010
Epoch 8/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 22s 11ms/step - accuracy: 0.2113 - loss: 0.
0023 - val_accuracy: 0.1696 - val_loss: 0.0016 - learning_rate: 0.0010
Epoch 9/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.2194 - loss: 0.
0021 - val_accuracy: 0.2974 - val_loss: 0.0015 - learning_rate: 0.0010
Epoch 10/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.2418 - loss: 0.
0019 - val_accuracy: 0.2889 - val_loss: 0.0018 - learning_rate: 0.0010
Epoch 11/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.2206 - loss: 0.
0019 - val_accuracy: 0.3382 - val_loss: 0.0021 - learning_rate: 0.0010
Epoch 12/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 9ms/step - accuracy: 0.2089 - loss: 0.0
018 - val_accuracy: 0.1427 - val_loss: 0.0014 - learning_rate: 0.0010
Epoch 13/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.1940 - loss: 0.
0018 - val_accuracy: 0.4554 - val_loss: 0.0015 - learning_rate: 0.0010
Epoch 14/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 18s 10ms/step - accuracy: 0.1848 - loss: 0.
0017 - val_accuracy: 0.3518 - val_loss: 0.0014 - learning_rate: 0.0010
Epoch 15/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 10ms/step - accuracy: 0.1458 - loss: 0.
0018 - val_accuracy: 0.3367 - val_loss: 0.0015 - 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:

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

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 : 99.69%
Precision - Macro: 79.22%
Recall - Macro: 78.55%
F1-score - Macro: 77.61%
Precision - Weighted: 99.69%
Recall - Weighted: 99.69%
F1-score - Weighted: 99.68%
```

TEST THE NETWORK

In [ ]: