Create Model

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

dropout1 = 0.5
dropout2 = 0.2

# learning_rate = 0.001

#conv1D and LSTM model
model_lstm = Sequential()
model_lstm.add(Conv1D(filters=64, kernel_size=1, activation='relu', input_
model_lstm.add(LSTM(128, return_sequences=True))
model_lstm.add(Dropout(dropout1))
model_lstm.add(LSTM(64, return_sequences=False))
model_lstm.add(Dropout(dropout2))
model_lstm.add(Dense(12, activation='sigmoid'))

print(train.shape[1], train.shape[2])

model_lstm.compile(loss='binary_crossentropy',optimizer='adam',metrics=['
```

```
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_ConvLSTM.keras



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

```
Epoch 1/30
2024-06-18 07:14:16.506213: I external/local_xla/xla/stream_executor/cuda/
cuda_dnn.cc:465] Loaded cuDNN version 8907
```

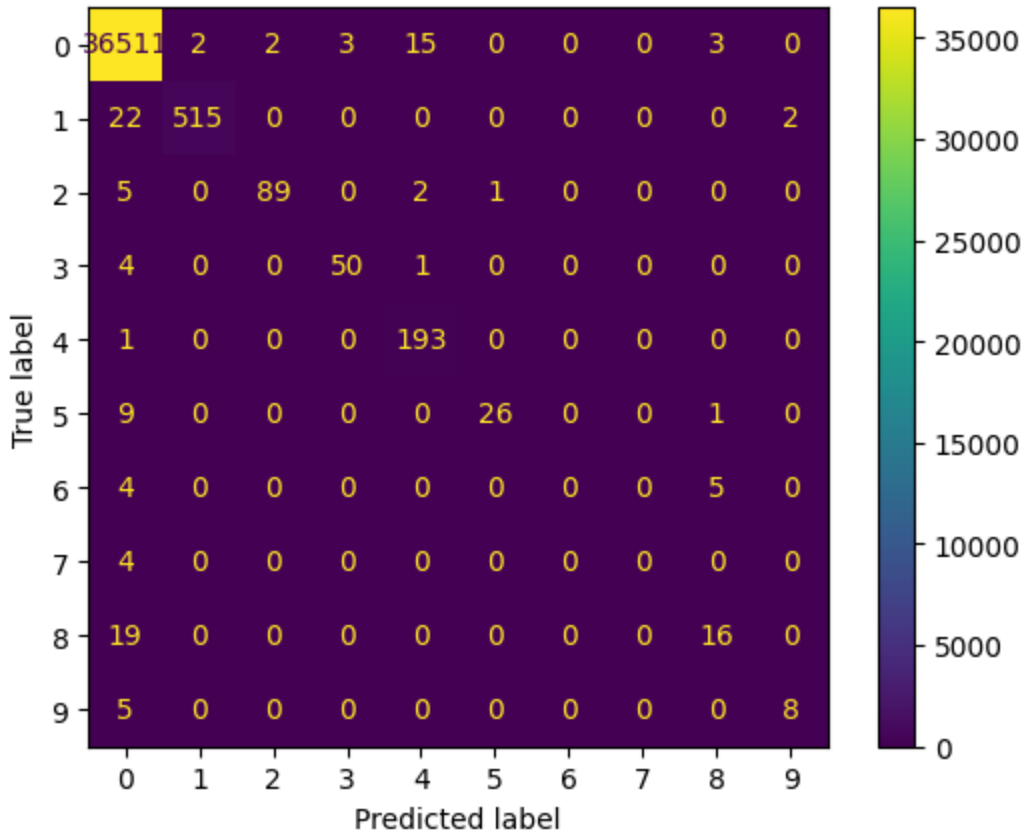
```
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 19s 8ms/step - accuracy: 0.0476 - loss: 0.0
982 - val_accuracy: 0.0144 - val_loss: 0.0103 - learning_rate: 0.0010
Epoch 2/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 17s 8ms/step - accuracy: 0.1907 - loss: 0.0
093 - val_accuracy: 0.0244 - val_loss: 0.0063 - learning_rate: 0.0010
Epoch 3/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.2406 - loss: 0.0
058 - val_accuracy: 0.0769 - val_loss: 0.0042 - learning_rate: 0.0010
Epoch 4/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.0889 - loss: 0.0
041 - val_accuracy: 0.0444 - val_loss: 0.0032 - learning_rate: 0.0010
Epoch 5/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.0478 - loss: 0.0
033 - val_accuracy: 0.0406 - val_loss: 0.0030 - learning_rate: 0.0010
Epoch 6/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.0542 - loss: 0.0
031 - val_accuracy: 0.0365 - val_loss: 0.0026 - learning_rate: 0.0010
Epoch 7/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.0555 - loss: 0.0
027 - val_accuracy: 0.0363 - val_loss: 0.0024 - learning_rate: 0.0010
Epoch 8/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.0634 - loss: 0.0
026 - val_accuracy: 0.0371 - val_loss: 0.0022 - learning_rate: 0.0010
Epoch 9/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.0542 - loss: 0.0
025 - val_accuracy: 0.0408 - val_loss: 0.0018 - learning_rate: 0.0010
Epoch 10/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.0691 - loss: 0.00
21 - val_accuracy: 0.0388 - val_loss: 0.0018 - learning_rate: 0.0010
Epoch 11/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.0965 - loss: 0.0
020 - val_accuracy: 0.0462 - val_loss: 0.0015 - learning_rate: 0.0010
Epoch 12/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.1136 - loss: 0.0
018 - val_accuracy: 0.0442 - val_loss: 0.0019 - learning_rate: 0.0010
Epoch 13/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.1219 - loss: 0.0
017 - val_accuracy: 0.0523 - val_loss: 0.0017 - learning_rate: 0.0010
Epoch 14/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.1280 - loss: 0.0
016 - val_accuracy: 0.0429 - val_loss: 0.0013 - learning_rate: 0.0010
Epoch 15/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 13s 7ms/step - accuracy: 0.1197 - loss: 0.0
014 - val_accuracy: 0.0505 - val_loss: 0.0018 - learning_rate: 0.0010
Epoch 16/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.1195 - loss: 0.0
015 - val_accuracy: 0.0489 - val_loss: 0.0014 - learning_rate: 0.0010
Epoch 17/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 21s 7ms/step - accuracy: 0.1189 - loss: 0.0
013 - val_accuracy: 0.0413 - val_loss: 0.0013 - learning_rate: 0.0010
Epoch 18/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 15s 9ms/step - accuracy: 0.0846 - loss: 0.0
014 - val_accuracy: 0.0487 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 19/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.1441 - loss: 0.0
013 - val_accuracy: 0.0421 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 20/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 22s 8ms/step - accuracy: 0.0988 - loss: 0.0
013 - val_accuracy: 0.0431 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 21/30
```

```
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.1515 - loss: 0.0
012 - val_accuracy: 0.0480 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 22/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 20s 8ms/step - accuracy: 0.1173 - loss: 0.0
012 - val_accuracy: 0.0456 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 23/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.1313 - loss: 0.0
013 - val_accuracy: 0.0444 - val_loss: 0.0010 - learning_rate: 0.0010
Epoch 24/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.1403 - loss: 0.0
011 - val_accuracy: 0.0507 - val_loss: 0.0014 - learning_rate: 0.0010
Epoch 25/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.1283 - loss: 0.0
012 - val_accuracy: 0.0440 - val_loss: 0.0010 - learning_rate: 0.0010
Epoch 26/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.1094 - loss: 0.00
11 - val_accuracy: 0.0490 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 27/30
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.1559 - loss: 0.00
12 - val_accuracy: 0.0473 - val_loss: 0.0012 - learning_rate: 0.0010
Epoch 28/30
1753/1759 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.1354 - loss: 0.00
11
Epoch 28: ReduceLROnPlateau reducing learning rate to 0.000200000009499490
26.
1759/1759 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.1354 - loss: 0.00
11 - val_accuracy: 0.0560 - 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)}%')
```

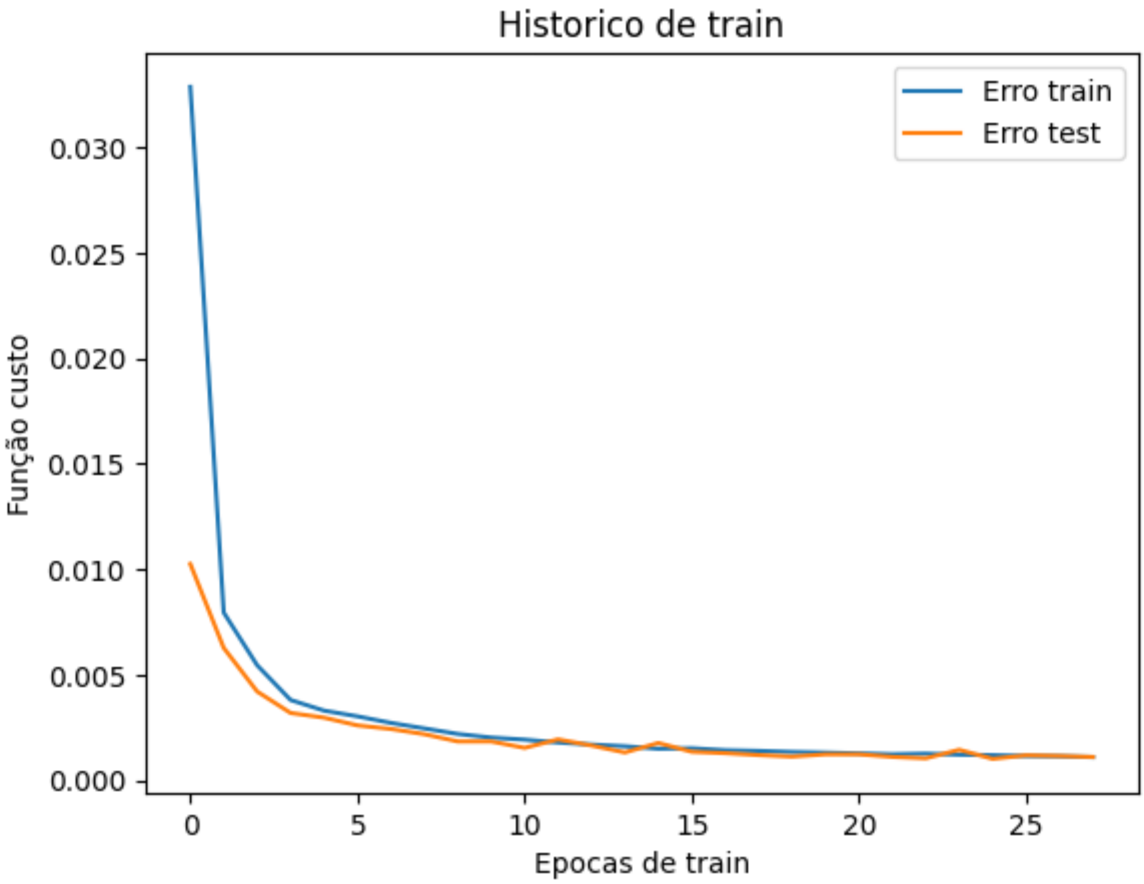
```
precision = precision_score(y_true =  y_test, y_pred = pred, average='mac
print(f'Precision - Macro: {np.round(precision*100,2)}%')

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 : 99.71%
Precision - Macro: 72.33%
Recall - Macro: 65.71%
F1-score - Macro: 68.54%
Precision - Weighted: 99.66%
Recall - Weighted: 99.71%
F1-score - Weighted: 99.68%
```

TEST THE NETWORK

In [ ]:

```
plt.plot(model_lstm_output.history['loss'])
plt.plot(model_lstm_output.history['val_loss'])
plt.title('Historico de train')
plt.xlabel('Epocas de train')
plt.ylabel('Função custo')
plt.legend(['Erro train', 'Erro test'])
plt.show()
```



In [ ]:
```
test[0]
test.shape
```

Out[ ]: