In [1]:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import seaborn as sns
from keras.preprocessing.image import ImageDataGenerator
```

In [2]:

```
#importing the dataset
train=pd.read_csv('sign_mnist_train.csv')
test=pd.read_csv('sign_mnist_test.csv')
```

In [3]:

```
print(train.shape)
print(test.shape)
```

```
(27455, 785)
(7172, 785)
```

In [4]:

```
train.head()
```

Out[4]:

| | label | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 | pixel9 | ... | pixel775 | pixel776 | pixel777 | pixel778 | pixel779 | pixel78 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | 107 | 118 | 127 | 134 | 139 | 143 | 146 | 150 | 153 | ... | 207 | 207 | 207 | 207 | 206 | 20 |
| **1** | 6 | 155 | 157 | 156 | 156 | 156 | 157 | 156 | 158 | 158 | ... | 69 | 149 | 128 | 87 | 94 | 16 |
| **2** | 2 | 187 | 188 | 188 | 187 | 187 | 186 | 187 | 188 | 187 | ... | 202 | 201 | 200 | 199 | 198 | 19 |
| **3** | 2 | 211 | 211 | 212 | 212 | 211 | 210 | 211 | 210 | 210 | ... | 235 | 234 | 233 | 231 | 230 | 22 |
| **4** | 13 | 164 | 167 | 170 | 172 | 176 | 179 | 180 | 184 | 185 | ... | 92 | 105 | 105 | 108 | 133 | 16 |

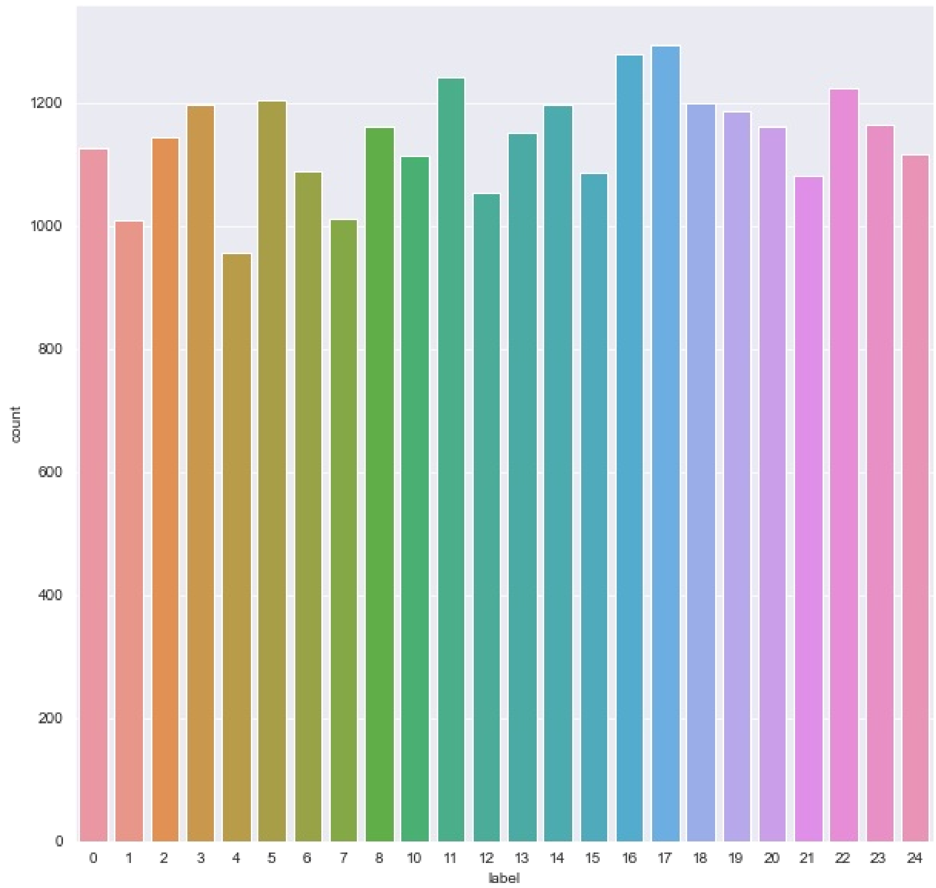
5 rows × 785 columns

In [5]:

```
plt.figure(figsize = (10,10)) # Label Count
sns.set_style("darkgrid")
sns.countplot(train['label'])
```

Out[5]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x25e377c9ee0>
```




In [7]:

```
# Create training and testing arrays
train_set = np.array(train, dtype = 'float32')
test_set = np.array(test, dtype='float32')
```

In [8]:

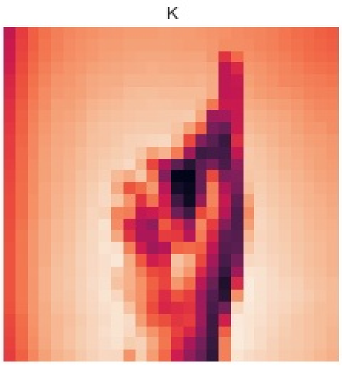
```
#Specifying class labels
class_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K','L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
'U', 'V', 'W', 'X', 'Y' ]
```

In [9]:

```
#See a random image for class label verification
i = random.randint(1,27455)
plt.imshow(train_set[i,1:].reshape((28,28)))
plt.imshow(train_set[i,1:].reshape((28,28)))
label_index = train["label"][i]
plt.title(f"{class_names[label_index]}")
plt.axis('off')
```

Out[9]:

```
(-0.5, 27.5, 27.5, -0.5)
```




In [10]:

```
# Define the dimensions of the plot grid
W_grid = 5
L_grid = 5
fig, axes = plt.subplots(L_grid, W_grid, figsize = (10,10))
axes = axes.ravel() # flatten the 15 x 15 matrix into 225 array
n_train = len(train_set) # get the length of the train dataset
# Select a random number from 0 to n_train
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
 # Select a random number
 index = np.random.randint(0, n_train)
 # read and display an image with the selected index
 axes[i].imshow( train_set[index,1:].reshape((28,28)) )
 label_index = int(train_set[index,0])
 axes[i].set_title(class_names[label_index], fontsize = 8)
 axes[i].axis('off')
plt.subplots_adjust(hspace=0.4)
```

In [11]:

```
# Prepare the training and testing dataset
X_train = train_set[:, 1:] / 255
y_train = train_set[:, 0]
X_test = test_set[:, 1:] / 255
y_test = test_set[:,0]
```

In [12]:

```
#Visualize train images
plt.figure(figsize=(10, 10))
for i in range(25):
 plt.subplot(5, 5, i + 1)
 plt.xticks([])
 plt.yticks([])
 plt.grid(False)
 plt.imshow(X_train[i].reshape((28,28)), cmap=plt.cm.binary)
 label_index = int(y_train[i])
 plt.title(class_names[label_index])
plt.show()
```


In [13]:

```
#Split the training and test sets
from sklearn.model_selection import train_test_split
X_train, X_validate, y_train, y_validate = train_test_split(X_train, y_train, test_size = 0.2, random_state = 123
45)
print(X_train.shape)
print(y_train.shape)
```

```
(21964, 784)
(21964,)
```

In [14]:

```
# Unpack the training and test tuple
X_train = X_train.reshape(X_train.shape[0], *(28, 28, 1))
X_test = X_test.reshape(X_test.shape[0], *(28, 28, 1))
X_validate = X_validate.reshape(X_validate.shape[0], *(28, 28, 1))
print(X_train.shape)
print(y_train.shape)
print(X_validate.shape)
```

```
(21964, 28, 28, 1)
(21964,)
(5491, 28, 28, 1)
```

In [15]:

```
# With data augmentation to prevent overfitting
datagen = ImageDataGenerator(
 featurewise_center=False, # set input mean to 0 over the dataset
 samplewise_center=False, # set each sample mean to 0
 featurewise_std_normalization=False, # divide inputs by std ofthe dataset
 samplewise_std_normalization=False, # divide each input by its std
 zca_whitening=False, # apply ZCA whitening
 rotation_range=10, # randomly rotate images in the range (degrees, 0 to 180)
 zoom_range = 0.1, # Randomly zoom image
 width_shift_range=0.1, # randomly shift images horizontally (fraction of total width)
 height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
 horizontal_flip=False, # randomly flip images
 vertical_flip=False) # randomly flip images
datagen.fit(X_train)
```

In [16]:

```
#Library for CNN Model
import keras
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout,BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau
```

In [17]:

```
#Defining the Convolutional Neural Network
cnn_model = Sequential()
cnn_model.add(Conv2D(32, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Conv2D(64, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(Dropout(0.2))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Conv2D(128, (3, 3), input_shape = (28,28,1), activation='relu'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPooling2D(pool_size = (2, 2)))
cnn_model.add(Flatten())
cnn_model.add(Dense(units = 512, activation = 'relu'))
cnn_model.add(Dropout(0.25))
cnn_model.add(Dense(units = 25, activation = 'softmax'))
cnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
dropout (Dropout)            (None, 11, 11, 64)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 11, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 25)                12825
=================================================================
Total params: 172,441
Trainable params: 171,993
Non-trainable params: 448
_________________________________________________________________
```

In [18]:

```
#Compiling
cnn_model.compile(loss ='sparse_categorical_crossentropy', optimizer='adam' ,metrics =['accuracy'])
```

In [19]:

```
learning_rate_reduction=ReduceLROnPlateau(monitor='val_accuracy',patience=2,verbose=1,factor=0.5,min_lr=0.00001)
```

In [20]:

```
#Training the CNN model
history = cnn_model.fit(datagen.flow(X_train, y_train, batch_size = 512), epochs = 20, verbose=1, validation_data
= (X_validate, y_validate),callbacks=[learning_rate_reduction])
```

```
Epoch 1/20
43/43 [==============================] - 37s 850ms/step - loss: 1.6502 - accuracy: 0.4982 - val_loss
: 3.2344 - val_accuracy: 0.0459
Epoch 2/20
43/43 [==============================] - 36s 836ms/step - loss: 0.5078 - accuracy: 0.8281 - val_loss
: 3.2623 - val_accuracy: 0.0459
Epoch 3/20
43/43 [==============================] - 35s 817ms/step - loss: 0.2523 - accuracy: 0.9173 - val_loss
: 3.2628 - val_accuracy: 0.0563
Epoch 4/20
43/43 [==============================] - 35s 815ms/step - loss: 0.1578 - accuracy: 0.9491 - val_loss
: 3.2407 - val_accuracy: 0.0829
Epoch 5/20
43/43 [==============================] - 35s 817ms/step - loss: 0.1102 - accuracy: 0.9651 - val_loss
: 3.0212 - val_accuracy: 0.1776
Epoch 6/20
43/43 [==============================] - 35s 816ms/step - loss: 0.0844 - accuracy: 0.9734 - val_loss
: 2.9480 - val_accuracy: 0.1945
Epoch 7/20
43/43 [==============================] - 35s 809ms/step - loss: 0.0660 - accuracy: 0.9788 - val_loss
: 2.9165 - val_accuracy: 0.1601
Epoch 8/20
43/43 [==============================] - 35s 810ms/step - loss: 0.0578 - accuracy: 0.9810 - val_loss
: 2.1802 - val_accuracy: 0.3779
Epoch 9/20
43/43 [==============================] - 35s 810ms/step - loss: 0.0518 - accuracy: 0.9838 - val_loss
: 1.4570 - val_accuracy: 0.5305
Epoch 10/20
43/43 [==============================] - 36s 828ms/step - loss: 0.0448 - accuracy: 0.9861 - val_loss
: 0.9138 - val_accuracy: 0.7130
Epoch 11/20
43/43 [==============================] - 35s 823ms/step - loss: 0.0388 - accuracy: 0.9870 - val_loss
: 0.3053 - val_accuracy: 0.9051
Epoch 12/20
43/43 [==============================] - 35s 824ms/step - loss: 0.0342 - accuracy: 0.9889 - val_loss
: 0.2977 - val_accuracy: 0.8986
Epoch 13/20
43/43 [==============================] - 35s 822ms/step - loss: 0.0306 - accuracy: 0.9910 - val_loss
: 0.0701 - val_accuracy: 0.9783
Epoch 14/20
43/43 [==============================] - 34s 790ms/step - loss: 0.0312 - accuracy: 0.9909 - val_loss
: 0.0479 - val_accuracy: 0.9869
Epoch 15/20
43/43 [==============================] - 35s 810ms/step - loss: 0.0259 - accuracy: 0.9914 - val_loss
: 0.0188 - val_accuracy: 0.9931
Epoch 16/20
43/43 [==============================] - 34s 781ms/step - loss: 0.0260 - accuracy: 0.9919 - val_loss
: 0.0184 - val_accuracy: 0.9953
Epoch 17/20
43/43 [==============================] - 34s 795ms/step - loss: 0.0240 - accuracy: 0.9927 - val_loss
: 0.1633 - val_accuracy: 0.9443
Epoch 18/20
43/43 [==============================] - ETA: 0s - loss: 0.0218 - accuracy: 0.9920
Epoch 00018: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
43/43 [==============================] - 34s 797ms/step - loss: 0.0218 - accuracy: 0.9920 - val_loss
: 0.0313 - val_accuracy: 0.9867
Epoch 19/20
43/43 [==============================] - 34s 798ms/step - loss: 0.0147 - accuracy: 0.9953 - val_loss
: 0.0022 - val_accuracy: 0.9995
Epoch 20/20
43/43 [==============================] - 34s 801ms/step - loss: 0.0128 - accuracy: 0.9960 - val_loss
: 0.0018 - val_accuracy: 0.9995
```

In [21]:

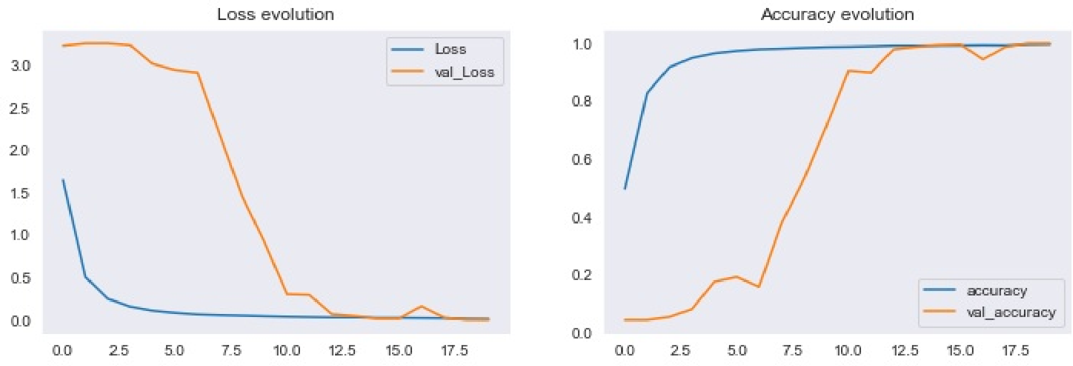
```
#Visualizing the training performance
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['val_loss'], label='val_Loss')
plt.legend()
plt.grid()
plt.title('Loss evolution')

plt.subplot(2, 2, 2)
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.legend()
plt.grid()
plt.title('Accuracy evolution')
```

Out[21]:

```
Text(0.5, 1.0, 'Accuracy evolution')
```




In [22]:

```
#Predictions for the test data
predicted_classes = cnn_model.predict_classes(X_test)
```

```
WARNING:tensorflow:From <ipython-input-22-2f08a57fd4c0>:2: Sequential.predict_classes (from tensorfl
ow.python.keras.engine.sequential) is deprecated and will be removed after 2021-01-01.
Instructions for updating:
Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classi
fication   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("i
nt32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activa
tion).
```

In [23]:

```
L = 5
W = 5
fig, axes = plt.subplots(L, W, figsize = (12,12))
axes = axes.ravel()

for i in np.arange(0, L * W):
    axes[i].imshow(X_test[i].reshape(28,28))
    axes[i].set_title(f"Prediction Class = {predicted_classes[i]:0.1f}\n True Class = {y_test[i]:0.1f}")
    axes[i].axis('off')
plt.subplots_adjust(wspace=0.5)
```


In [24]:

```
from sklearn.metrics import confusion_matrix
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, predicted_classes)
```

In [25]:

```
#Defining function for confusion matrix plot
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=False,
                          title=None,
                          cmap=plt.cm.Blues):
    if not title:
        if normalize:
            title = 'Normalized confusion matrix'
        else:
            title = 'Confusion matrix, without normalization'

    # Computing confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

# Visualizing
    fig, ax = plt.subplots(figsize=(10, 10))
    im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
    ax.figure.colorbar(im, ax=ax)
    # We want to show all ticks...
    ax.set(xticks=np.arange(cm.shape[1]),
           yticks=np.arange(cm.shape[0]),
           xticklabels=classes, yticklabels=classes,
           title=title,
           ylabel='True label',
           xlabel='Predicted label')

   # Rotating the tick labels and setting their alignment.
    plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
             rotation_mode="anchor")
    # Looping over data dimensions and create text annotations.
    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax
np.set_printoptions(precision=2)
```

In [26]:

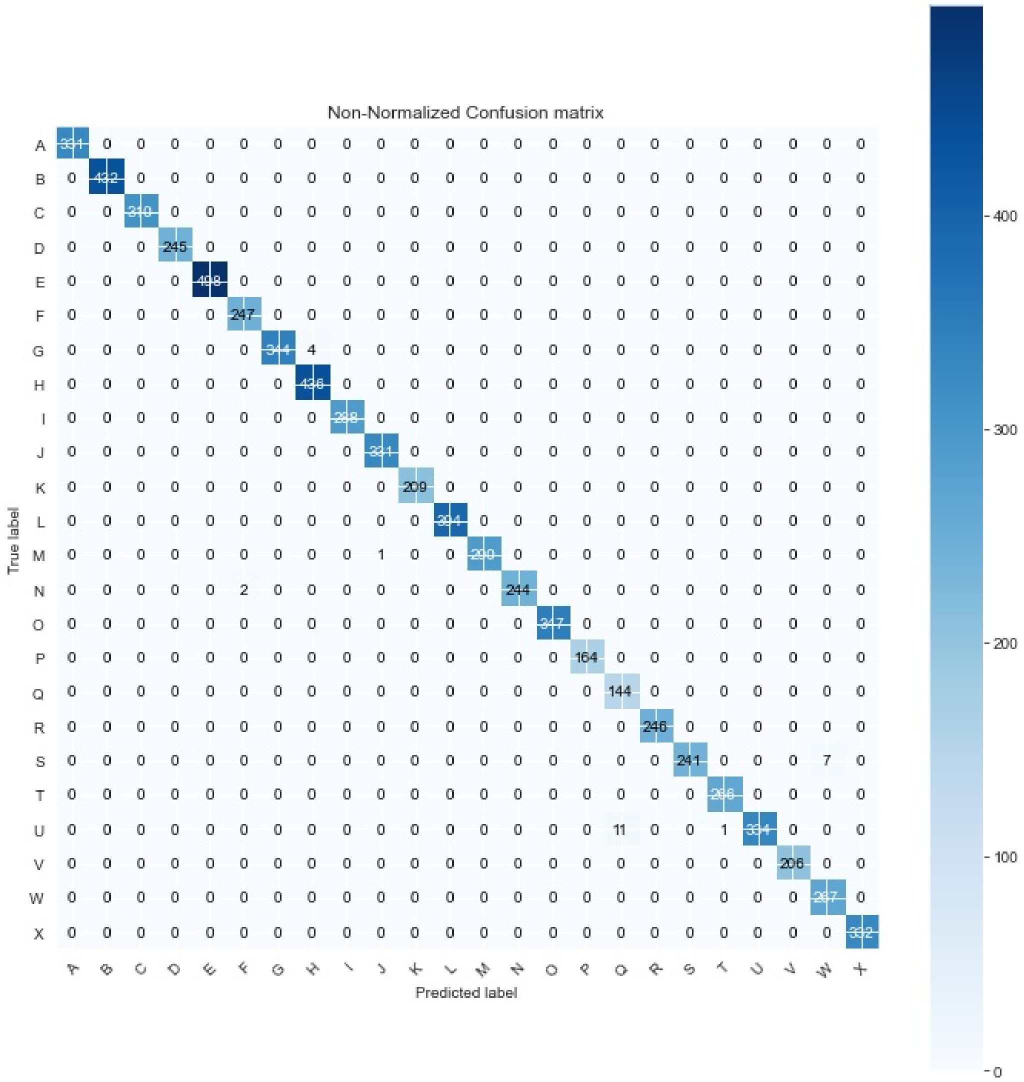
```
#Specifying class labels
class_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T'
, 'U', 'V', 'W', 'X', 'Y' ]
```

In [27]:

```
plt.figure(figsize=(20,20))
plot_confusion_matrix(y_test, predicted_classes, classes = class_names, title='Non-Normalized Confusion matrix')
plt.show()
```

```
Confusion matrix, without normalization
```

```
<Figure size 1440x1440 with 0 Axes>
```

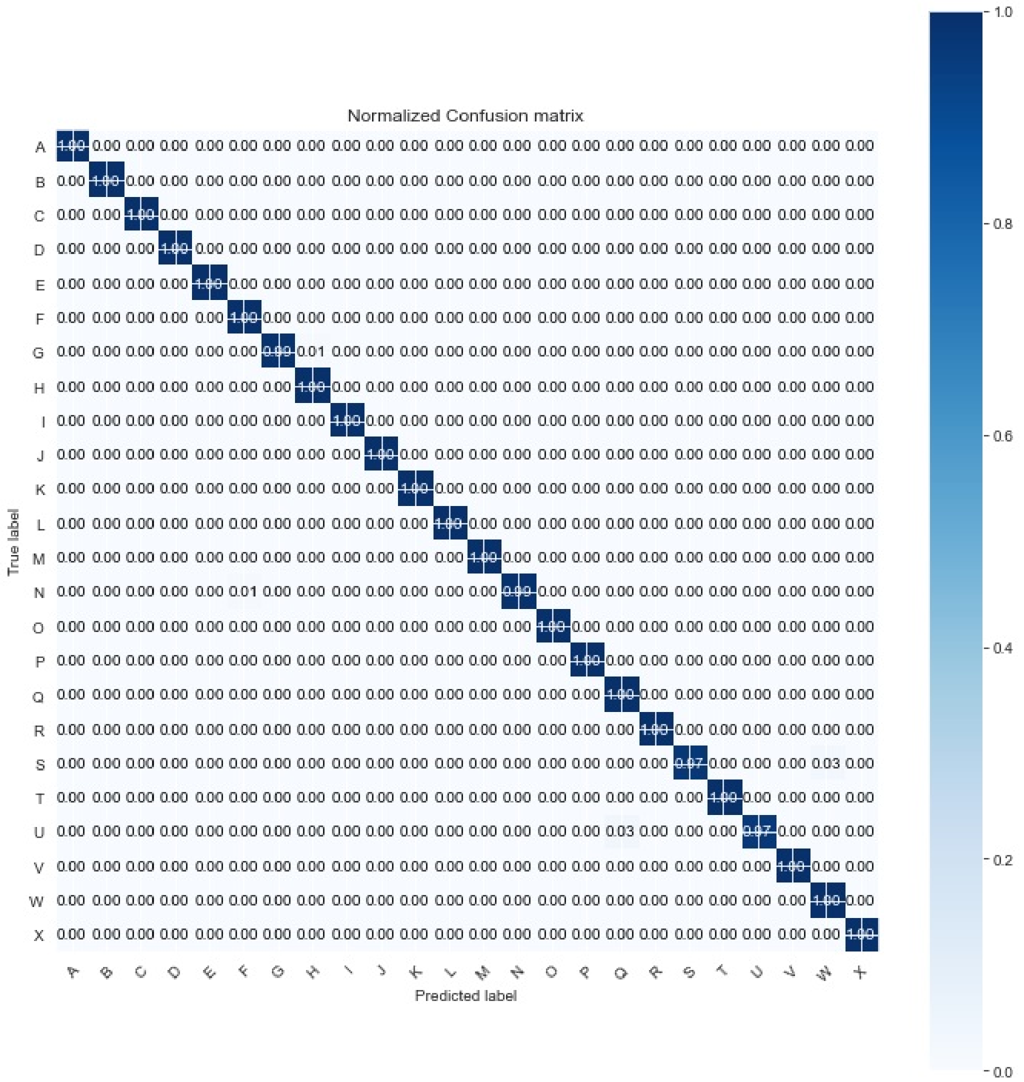


In [28]:

```
plt.figure(figsize=(35,35))
plot_confusion_matrix(y_test, predicted_classes, classes = class_names, normalize=True, title='Normalized Confusi
on matrix')
plt.show()
```

Normalized confusion matrix

<Figure size 2520x2520 with 0 Axes>



In [40]:

```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score

print(classification_report(y_test, predicted_classes))
print("Accuracy:",accuracy_score(y_test, predicted_classes)*100)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       331
         1.0       1.00      1.00      1.00       432
         2.0       1.00      1.00      1.00       310
         3.0       1.00      1.00      1.00       245
         4.0       1.00      1.00      1.00       498
         5.0       0.99      1.00      1.00       247
         6.0       1.00      0.99      0.99       348
         7.0       0.99      1.00      1.00       436
         8.0       1.00      1.00      1.00       288
        10.0       1.00      1.00      1.00       331
        11.0       1.00      1.00      1.00       209
        12.0       1.00      1.00      1.00       394
        13.0       1.00      1.00      1.00       291
        14.0       1.00      0.99      1.00       246
        15.0       1.00      1.00      1.00       347
        16.0       1.00      1.00      1.00       164
        17.0       0.93      1.00      0.96       144
        18.0       1.00      1.00      1.00       246
        19.0       1.00      0.97      0.99       248
        20.0       1.00      1.00      1.00       266
        21.0       1.00      0.97      0.98       346
        22.0       1.00      1.00      1.00       206
        23.0       0.97      1.00      0.99       267
        24.0       1.00      1.00      1.00       332

    accuracy                           1.00      7172
   macro avg       0.99      1.00      1.00      7172
weighted avg       1.00      1.00      1.00      7172

Accuracy: 99.63747908533185
```

In [30]:

```
predicted_classes.shape
```

Out[30]:

```
(7172,)
```

In [31]:

```
y_test.shape
```

Out[31]:

```
(7172,)
```

In [42]:

```
#Convert To One Hot encoded Vector
labels = np.zeros((y_test.shape[0],25))

temp = pd.DataFrame(labels)
temp.head()
```

Out[42]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 25 columns

In [43]:

```
for i in range(y_test.shape[0]):
    index = int(y_test[i])
    labels[i][index] = 1
```

In [44]:

```
print(labels.shape)
```

(7172, 25)

In [45]:

```
temp = pd.DataFrame(y_test)
temp.head()
```

Out[45]:

| | 0 |
|---|---|
| **0** | 6.0 |
| **1** | 5.0 |
| **2** | 10.0 |
| **3** | 0.0 |
| **4** | 3.0 |

In [46]:

```
temp = pd.DataFrame(labels)
temp.head()
```

Out[46]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 25 columns

In [47]:

```
def convert_to_one_hot_encoded(y):
    output_classes = 25
    labels = np.zeros((y.shape[0],output_classes))
    for i in range(y.shape[0]):
        index = int(y[i])
        labels[i][index] = 1
    return labels
```

In [48]:

```
from sklearn.metrics import roc_auc_score

predictions = convert_to_one_hot_encoded(predicted_classes)
y_actual = convert_to_one_hot_encoded(y_test)

scores =  []
for i in range(y_actual.shape[0]):
    scores.append(roc_auc_score(y_actual[i],predictions[i]))

scores = np.array(scores)
print(np.mean(scores)*100)
```

99.81118702361034

In [ ]:

```

```