In [1]:
```python
print("Import needed packages")

from mlxtend.data import loadlocal_mnist
import os
import tensorflow as tf
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, precis
ion_score, recall_score, f1_score, classification_report
import matplotlib.pyplot as plt
import matplotlib as mpl
from tensorflow.keras.utils import to_categorical
```

```
Import needed packages
```

In [2]:
```
print("Import dataset...")

# Initialization
dirDataset = '/workspaces/flow-based-1dcnn/dataset/Preprocessed/Dataset4/'
fileTrainData = 'train-images-idx3-ubyte'
fileTrainLabels = 'train-labels-idx1-ubyte'
fileTestData = 'test-images-idx3-ubyte'
fileTestLabels = 'test-labels-idx1-ubyte'
pathMapLabels = 'PNGs/Split/mapLabels.txt'
numClasses = 12

# Unzip
#   If the data is already unzip, a warning message is display, but it does
not stop the script
print("  Uncompressing data...")
if os.path.isfile(dirDataset + fileTrainData + '.gz'):
    print("    Uncompress file : %s" % (dirDataset + fileTrainData + '.gz'))
    os.system('gunzip '+dirDataset+fileTrainData+'.gz')
if os.path.isfile(dirDataset + fileTrainLabels + '.gz'):
    print("    Uncompress file : %s" % (dirDataset + fileTrainLabels + '.gz
'))
    os.system('gunzip '+dirDataset+fileTrainLabels+'.gz')
if os.path.isfile(dirDataset + fileTestData + '.gz'):
    print("    Uncompress file : %s" % (dirDataset + fileTestData + '.gz'))
    os.system('gunzip '+dirDataset+fileTestData+'.gz')
if os.path.isfile(dirDataset + fileTestLabels + '.gz'):
    print("    Uncompress file : %s" % (dirDataset + fileTestLabels + '.gz
'))
    os.system('gunzip '+dirDataset+fileTestLabels+'.gz')

# Load
print("  Loading data...")
X_train, Y_train = loadlocal_mnist(images_path=os.path.join(dirDataset, file
TrainData), labels_path=os.path.join(dirDataset, fileTrainLabels))
X_test, Y_test = loadlocal_mnist(images_path=os.path.join(dirDataset, fileTe
stData), labels_path=os.path.join(dirDataset, fileTestLabels))

# One-hot encoding of the labels
Y_train_OH = to_categorical(Y_train)
Y_test_OH = to_categorical(Y_test)

# Modify dimensions input data
print("  Dimensions of the input data:")
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
print("    Train sample shape : %s" % (X_train.shape,))
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)
print("    Test sample shape : %s" % (X_test.shape,))

# Normalize
print("  Normalizing data...")
X_train = X_train / 255.0
X_test = X_test / 255.0

# Create dictionary for the label mapping
print("  Creating map for the labels...")
label_dict = {}
fileMap = open(os.path.join(dirDataset, pathMapLabels), 'r', newline='\n')
for line in fileMap:
    x = line.strip().split(",")
    label_dict[int(x[0])] = x[1]
for element in label_dict:
    print("    %d - %s" % (element, label_dict[element]))
```

```
Import dataset...
  Uncompressing data...
  Loading data...
  Dimensions of the input data:
    Train sample shape : (15732, 784, 1)
    Test sample shape : (3940, 784, 1)
  Normalizing data...
  Creating map for the labels...
    0 -  facebook-cloud
    1 -  snapchat
    2 -  crashlytics
    3 -  netflix
    4 -  google-cloud
    5 -  apple-music
    6 -  google-play
    7 -  moat
    8 -  roblox
    9 -  nbc-services
    10 -  adjust
    11 -  tiktok
```

In [3]:

```
print("Build model")

model_1D_CNN = tf.keras.models.Sequential()
model_1D_CNN.add(tf.keras.layers.Conv1D(32, 25, strides=1, padding='same', a
ctivation='relu', input_shape=(784,1)))
model_1D_CNN.add(tf.keras.layers.MaxPooling1D(pool_size=3, strides=3, paddin
g='same'))
model_1D_CNN.add(tf.keras.layers.Conv1D(64, 25, strides=1, padding='same', a
ctivation='relu'))
model_1D_CNN.add(tf.keras.layers.MaxPooling1D(pool_size=3, strides=3, paddin
g='same'))
model_1D_CNN.add(tf.keras.layers.Flatten())
model_1D_CNN.add(tf.keras.layers.Dense(1024, activation='relu'))
model_1D_CNN.add(tf.keras.layers.Dropout(0.5))
model_1D_CNN.add(tf.keras.layers.Dense(numClasses, activation='softmax'))

# Print model summary
model_1D_CNN.summary()
```

```
Build model
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 784, 32)           832
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 262, 32)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 262, 64)           51264
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 88, 64)            0
_________________________________________________________________
flatten (Flatten)            (None, 5632)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              5768192
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 12)                12300
=================================================================
Total params: 5,832,588
Trainable params: 5,832,588
Non-trainable params: 0
_________________________________________________________________
```

In [4]:
```
print("Training")

# Parameters
learningRate = 1e-4
batchSize = 50
numEpoch = 50

# Initialization
optimizer = tf.keras.optimizers.Adam(learning_rate=learningRate)        # t
f.keras.optimizers.SGD(learning_rate=learningRate)
loss = tf.keras.losses.CategoricalCrossentropy()                        # Sp
arseCategoricalCrossentropy
metrics = 'categorical_accuracy'                                        # 's
parse_categorical_accuracy'

# Configure model
model_1D_CNN.compile(optimizer=optimizer,
              loss=loss,
              metrics=[metrics])

# Train
history = model_1D_CNN.fit(X_train, Y_train_OH, epochs=numEpoch, batch_size=
batchSize, validation_split=0.1, verbose=1)
```

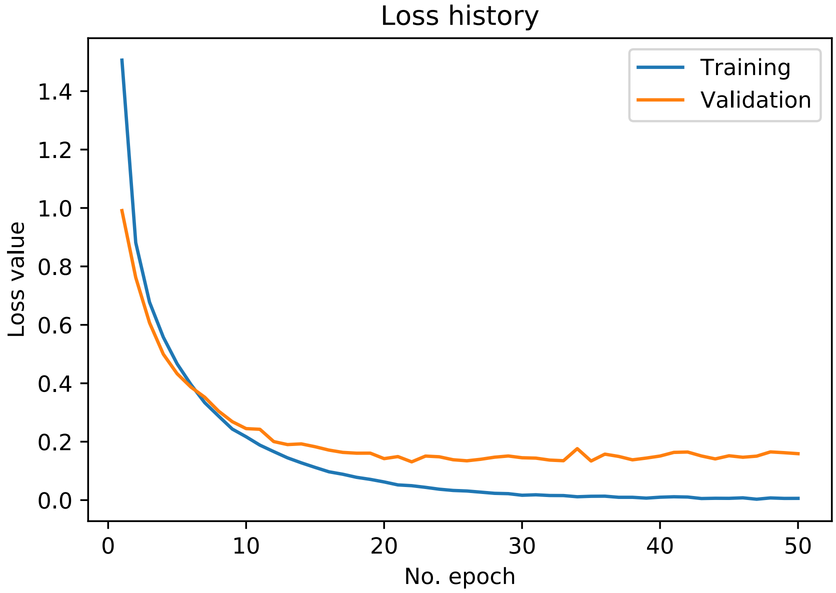
```
Training
Train on 14158 samples, validate on 1574 samples
Epoch 1/50
14158/14158 [==============================] - 4s 259us/sample - loss: 1.5063
- categorical_accuracy: 0.5094 - val_loss: 0.9912 - val_categorical_accuracy:
0.6976
Epoch 2/50
14158/14158 [==============================] - 2s 158us/sample - loss: 0.8815
- categorical_accuracy: 0.7261 - val_loss: 0.7621 - val_categorical_accuracy:
0.7624
Epoch 3/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.6775
- categorical_accuracy: 0.7918 - val_loss: 0.6071 - val_categorical_accuracy:
0.8119
Epoch 4/50
14158/14158 [==============================] - 2s 161us/sample - loss: 0.5576
- categorical_accuracy: 0.8231 - val_loss: 0.4995 - val_categorical_accuracy:
0.8431
Epoch 5/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.4666
- categorical_accuracy: 0.8503 - val_loss: 0.4328 - val_categorical_accuracy:
0.8526
Epoch 6/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.3959
- categorical_accuracy: 0.8734 - val_loss: 0.3869 - val_categorical_accuracy:
0.8679
Epoch 7/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.3336
- categorical_accuracy: 0.8936 - val_loss: 0.3523 - val_categorical_accuracy:
0.8812
Epoch 8/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.2880
- categorical_accuracy: 0.9058 - val_loss: 0.3049 - val_categorical_accuracy:
0.8958
Epoch 9/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.2432
- categorical_accuracy: 0.9222 - val_loss: 0.2686 - val_categorical_accuracy:
0.9098
Epoch 10/50
14158/14158 [==============================] - 2s 161us/sample - loss: 0.2170
- categorical_accuracy: 0.9309 - val_loss: 0.2448 - val_categorical_accuracy:
0.9282
Epoch 11/50
14158/14158 [==============================] - 2s 159us/sample - loss: 0.1882
- categorical_accuracy: 0.9400 - val_loss: 0.2425 - val_categorical_accuracy:
0.9263
Epoch 12/50
14158/14158 [==============================] - 2s 160us/sample - loss: 0.1662
- categorical_accuracy: 0.9462 - val_loss: 0.2005 - val_categorical_accuracy:
0.9358
Epoch 13/50
14158/14158 [==============================] - 2s 162us/sample - loss: 0.1448
- categorical_accuracy: 0.9533 - val_loss: 0.1902 - val_categorical_accuracy:
0.9403
Epoch 14/50
14158/14158 [==============================] - 2s 162us/sample - loss: 0.1277
- categorical_accuracy: 0.9591 - val_loss: 0.1926 - val_categorical_accuracy:
0.9422
Epoch 15/50
14158/14158 [==============================] - 2s 160us/sample - loss: 0.1120
- categorical_accuracy: 0.9640 - val_loss: 0.1829 - val_categorical_accuracy:
0.9485
Epoch 16/50
14158/14158 [==============================] - 2s 161us/sample - loss: 0.0970
- categorical_accuracy: 0.9709 - val_loss: 0.1714 - val_categorical_accuracy:
0.9511
Epoch 17/50
14158/14158 [==============================] - 2s 160us/sample - loss: 0.0887
- categorical_accuracy: 0.9728 - val_loss: 0.1635 - val_categorical_accuracy:
```

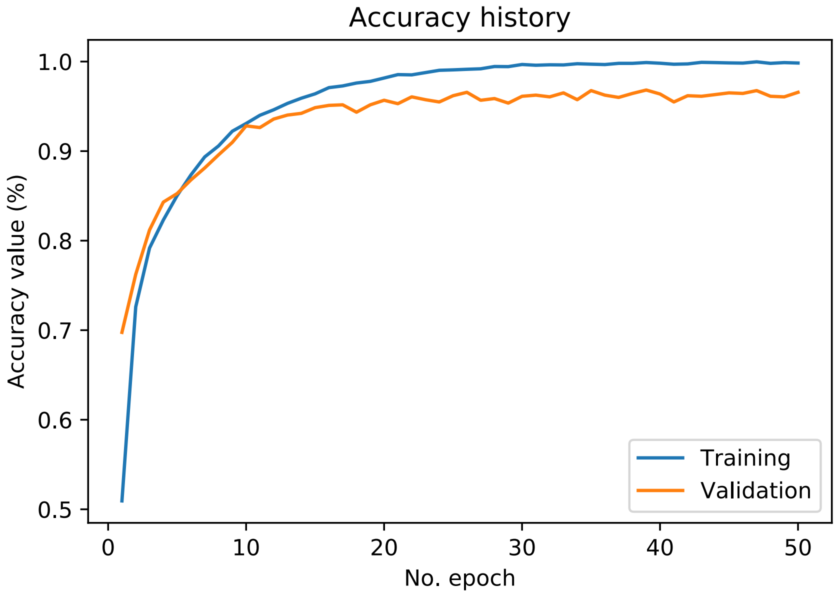
```
0.9517
Epoch 18/50
14158/14158 [==============================] - 2s 160us/sample - loss: 0.0781
- categorical_accuracy: 0.9761 - val_loss: 0.1607 - val_categorical_accuracy:
0.9435
Epoch 19/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0711
- categorical_accuracy: 0.9779 - val_loss: 0.1608 - val_categorical_accuracy:
0.9517
Epoch 20/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0623
- categorical_accuracy: 0.9816 - val_loss: 0.1421 - val_categorical_accuracy:
0.9568
Epoch 21/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0522
- categorical_accuracy: 0.9854 - val_loss: 0.1491 - val_categorical_accuracy:
0.9530
Epoch 22/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0494
- categorical_accuracy: 0.9852 - val_loss: 0.1311 - val_categorical_accuracy:
0.9606
Epoch 23/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0438
- categorical_accuracy: 0.9877 - val_loss: 0.1509 - val_categorical_accuracy:
0.9574
Epoch 24/50
14158/14158 [==============================] - 2s 157us/sample - loss: 0.0374
- categorical_accuracy: 0.9903 - val_loss: 0.1484 - val_categorical_accuracy:
0.9549
Epoch 25/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0331
- categorical_accuracy: 0.9907 - val_loss: 0.1383 - val_categorical_accuracy:
0.9619
Epoch 26/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0313
- categorical_accuracy: 0.9914 - val_loss: 0.1347 - val_categorical_accuracy:
0.9657
Epoch 27/50
14158/14158 [==============================] - 2s 157us/sample - loss: 0.0273
- categorical_accuracy: 0.9919 - val_loss: 0.1399 - val_categorical_accuracy:
0.9568
Epoch 28/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0234
- categorical_accuracy: 0.9945 - val_loss: 0.1470 - val_categorical_accuracy:
0.9587
Epoch 29/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0221
- categorical_accuracy: 0.9943 - val_loss: 0.1512 - val_categorical_accuracy:
0.9536
Epoch 30/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0167
- categorical_accuracy: 0.9968 - val_loss: 0.1451 - val_categorical_accuracy:
0.9612
Epoch 31/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0183
- categorical_accuracy: 0.9959 - val_loss: 0.1438 - val_categorical_accuracy:
0.9625
Epoch 32/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0159
- categorical_accuracy: 0.9964 - val_loss: 0.1371 - val_categorical_accuracy:
0.9606
Epoch 33/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0156
- categorical_accuracy: 0.9963 - val_loss: 0.1349 - val_categorical_accuracy:
0.9651
Epoch 34/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0116
- categorical_accuracy: 0.9976 - val_loss: 0.1763 - val_categorical_accuracy:
0.9574
Epoch 35/50
```

```
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0132
- categorical_accuracy: 0.9971 - val_loss: 0.1339 - val_categorical_accuracy:
0.9676
Epoch 36/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0135
- categorical_accuracy: 0.9967 - val_loss: 0.1575 - val_categorical_accuracy:
0.9625
Epoch 37/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0097
- categorical_accuracy: 0.9980 - val_loss: 0.1497 - val_categorical_accuracy:
0.9600
Epoch 38/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0097
- categorical_accuracy: 0.9980 - val_loss: 0.1380 - val_categorical_accuracy:
0.9644
Epoch 39/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0069
- categorical_accuracy: 0.9989 - val_loss: 0.1440 - val_categorical_accuracy:
0.9682
Epoch 40/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0100
- categorical_accuracy: 0.9982 - val_loss: 0.1510 - val_categorical_accuracy:
0.9638
Epoch 41/50
14158/14158 [==============================] - 2s 157us/sample - loss: 0.0116
- categorical_accuracy: 0.9970 - val_loss: 0.1634 - val_categorical_accuracy:
0.9549
Epoch 42/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0106
- categorical_accuracy: 0.9974 - val_loss: 0.1647 - val_categorical_accuracy:
0.9619
Epoch 43/50
14158/14158 [==============================] - 2s 158us/sample - loss: 0.0055
- categorical_accuracy: 0.9992 - val_loss: 0.1512 - val_categorical_accuracy:
0.9612
Epoch 44/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0064
- categorical_accuracy: 0.9989 - val_loss: 0.1411 - val_categorical_accuracy:
0.9632
Epoch 45/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0061
- categorical_accuracy: 0.9985 - val_loss: 0.1519 - val_categorical_accuracy:
0.9651
Epoch 46/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0079
- categorical_accuracy: 0.9983 - val_loss: 0.1468 - val_categorical_accuracy:
0.9644
Epoch 47/50
14158/14158 [==============================] - 2s 155us/sample - loss: 0.0032
- categorical_accuracy: 0.9997 - val_loss: 0.1506 - val_categorical_accuracy:
0.9676
Epoch 48/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0076
- categorical_accuracy: 0.9980 - val_loss: 0.1651 - val_categorical_accuracy:
0.9612
Epoch 49/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0059
- categorical_accuracy: 0.9989 - val_loss: 0.1624 - val_categorical_accuracy:
0.9606
Epoch 50/50
14158/14158 [==============================] - 2s 156us/sample - loss: 0.0061
- categorical_accuracy: 0.9984 - val_loss: 0.1590 - val_categorical_accuracy:
0.9657
```

In [5]:
```
print("Visualize result training")

# Create x-axis
xaxis = range(1, numEpoch + 1)

# Plot history: Loss
plt.plot(xaxis, history.history['loss'], xaxis, history.history['val_loss'])
plt.title('Loss history')
plt.ylabel('Loss value')
plt.xlabel('No. epoch')
plt.legend(["Training", "Validation"])
plt.savefig("Plots/loss_history-1DCNN-12classes.svg")
plt.show()

# Plot history: Accuracy
plt.plot(xaxis, history.history[metrics], xaxis, history.history['val_' + me
trics])
plt.title('Accuracy history')
plt.ylabel('Accuracy value (%)')
plt.xlabel('No. epoch')
plt.legend(["Training", "Validation"])
plt.savefig("Plots/accuracy_history-1DCNN-12classes.svg")
plt.show()
```

```
Visualize result training
```

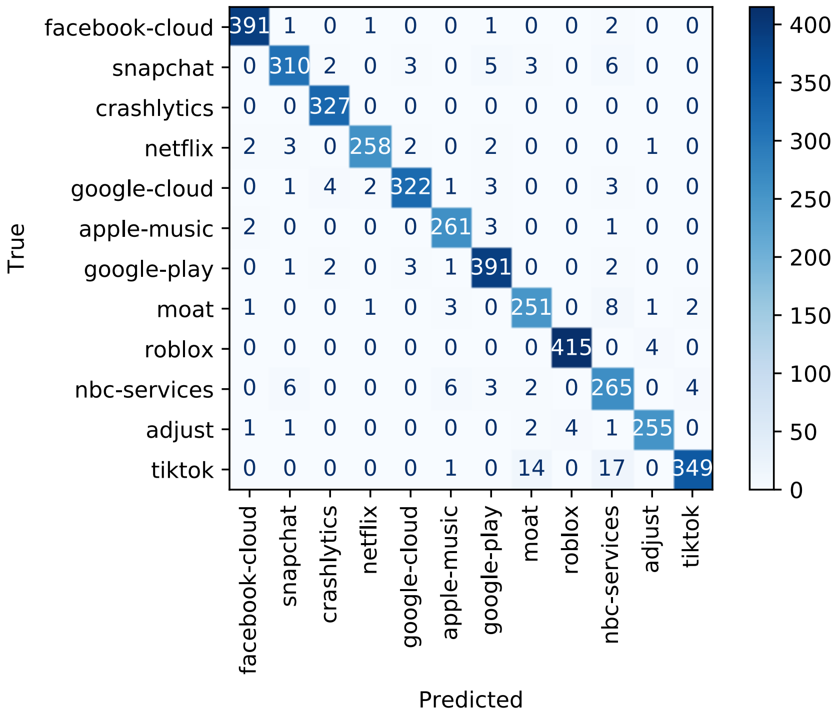




In [6]:
```python
print("Testing...")

# Create list labels for plot
listLabels = []
print("Map labels:")
for element in label_dict:
    print("  %2d - %s" % (element, label_dict[element]))
    listLabels.append(label_dict[element])
print(" ")

# Predict outcome for the test set
y_pred = model_1D_CNN.predict(X_test)

# Print metrics
test_loss, test_acc = model_1D_CNN.evaluate(X_test, Y_test_OH, verbose=1)
print("\nDisplay model metrics:")
print('\nTest accuracy:', test_acc)
print(" ")
print(classification_report(Y_test_OH.argmax(axis=1), y_pred.argmax(axis=
1)))

# Create and plot confusion matrix
print("\nConfusion matrix:")
confMat = confusion_matrix(Y_test_OH.argmax(axis=1), y_pred.argmax(axis=1))
disp = ConfusionMatrixDisplay(confusion_matrix=confMat, display_labels=listL
abels)
disp.plot(xticks_rotation='vertical', cmap='Blues')
disp.ax_.set(xlabel='Predicted', ylabel='True', )
```

```
Testing...
Map labels:
   0 -  facebook-cloud
   1 -  snapchat
   2 -  crashlytics
   3 -  netflix
   4 -  google-cloud
   5 -  apple-music
   6 -  google-play
   7 -  moat
   8 -  roblox
   9 -  nbc-services
  10 -  adjust
  11 -  tiktok

3940/3940 [==============================] - 0s 86us/sample - loss: 0.1238 -
categorical_accuracy: 0.9632

Display model metrics:

Test accuracy: 0.96319795

              precision    recall  f1-score   support

           0       0.98      0.99      0.99       396
           1       0.96      0.94      0.95       329
           2       0.98      1.00      0.99       327
           3       0.98      0.96      0.97       268
           4       0.98      0.96      0.97       336
           5       0.96      0.98      0.97       267
           6       0.96      0.98      0.97       400
           7       0.92      0.94      0.93       267
           8       0.99      0.99      0.99       419
           9       0.87      0.93      0.90       286
          10       0.98      0.97      0.97       264
          11       0.98      0.92      0.95       381

    accuracy                           0.96      3940
   macro avg       0.96      0.96      0.96      3940
weighted avg       0.96      0.96      0.96      3940


Confusion matrix:
```

Out[6]:
```
[Text(0, 0.5, 'True'), Text(0.5, 0, 'Predicted')]
```


True
Predicted
facebook-cloud
snapchat
crashlytics
netflix
google-cloud
apple-music
google-play
moat
roblox
nbc-services
adjust
tiktok