In [1]:
```
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/Dataset3/'
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 = 6

# 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 : (8994, 784, 1)
    Test sample shape : (2252, 784, 1)
  Normalizing data...
  Creating map for the labels...
    0 -  facebook-cloud
    1 -  snapchat
    2 -  crashlytics
    3 -  google-play
    4 -  roblox
    5 -  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, 6)                 6150
=================================================================
Total params: 5,826,438
Trainable params: 5,826,438
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)
```

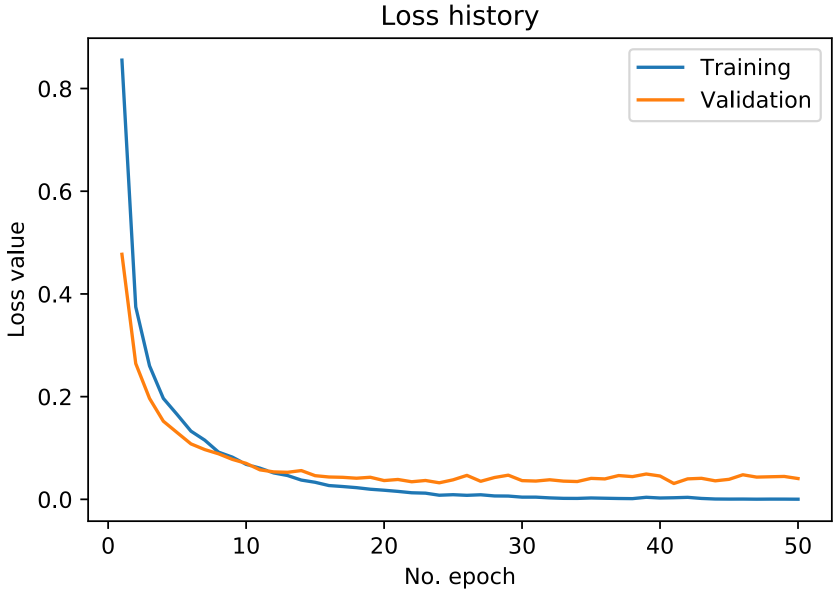
```
Training
Train on 8094 samples, validate on 900 samples
Epoch 1/50
8094/8094 [==============================] - 3s 340us/sample - loss: 0.8553 -
categorical_accuracy: 0.6993 - val_loss: 0.4772 - val_categorical_accuracy:
0.8844
Epoch 2/50
8094/8094 [==============================] - 1s 158us/sample - loss: 0.3748 -
categorical_accuracy: 0.8925 - val_loss: 0.2645 - val_categorical_accuracy:
0.9256
Epoch 3/50
8094/8094 [==============================] - 1s 157us/sample - loss: 0.2599 -
categorical_accuracy: 0.9202 - val_loss: 0.1965 - val_categorical_accuracy:
0.9489
Epoch 4/50
8094/8094 [==============================] - 1s 157us/sample - loss: 0.1966 -
categorical_accuracy: 0.9398 - val_loss: 0.1522 - val_categorical_accuracy:
0.9489
Epoch 5/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.1652 -
categorical_accuracy: 0.9474 - val_loss: 0.1300 - val_categorical_accuracy:
0.9622
Epoch 6/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.1329 -
categorical_accuracy: 0.9582 - val_loss: 0.1080 - val_categorical_accuracy:
0.9656
Epoch 7/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.1153 -
categorical_accuracy: 0.9629 - val_loss: 0.0969 - val_categorical_accuracy:
0.9733
Epoch 8/50
8094/8094 [==============================] - 1s 158us/sample - loss: 0.0919 -
categorical_accuracy: 0.9702 - val_loss: 0.0887 - val_categorical_accuracy:
0.9733
Epoch 9/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.0822 -
categorical_accuracy: 0.9733 - val_loss: 0.0779 - val_categorical_accuracy:
0.9767
Epoch 10/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.0682 -
categorical_accuracy: 0.9794 - val_loss: 0.0701 - val_categorical_accuracy:
0.9811
Epoch 11/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.0608 -
categorical_accuracy: 0.9823 - val_loss: 0.0575 - val_categorical_accuracy:
0.9822
Epoch 12/50
8094/8094 [==============================] - 1s 158us/sample - loss: 0.0514 -
categorical_accuracy: 0.9836 - val_loss: 0.0535 - val_categorical_accuracy:
0.9844
Epoch 13/50
8094/8094 [==============================] - 1s 160us/sample - loss: 0.0464 -
categorical_accuracy: 0.9865 - val_loss: 0.0527 - val_categorical_accuracy:
0.9878
Epoch 14/50
8094/8094 [==============================] - 1s 159us/sample - loss: 0.0375 -
categorical_accuracy: 0.9893 - val_loss: 0.0560 - val_categorical_accuracy:
0.9800
Epoch 15/50
8094/8094 [==============================] - 1s 158us/sample - loss: 0.0332 -
categorical_accuracy: 0.9897 - val_loss: 0.0461 - val_categorical_accuracy:
0.9889
Epoch 16/50
8094/8094 [==============================] - 1s 160us/sample - loss: 0.0269 -
categorical_accuracy: 0.9931 - val_loss: 0.0435 - val_categorical_accuracy:
0.9844
Epoch 17/50
8094/8094 [==============================] - 1s 158us/sample - loss: 0.0250 -
categorical_accuracy: 0.9925 - val_loss: 0.0429 - val_categorical_accuracy:
```

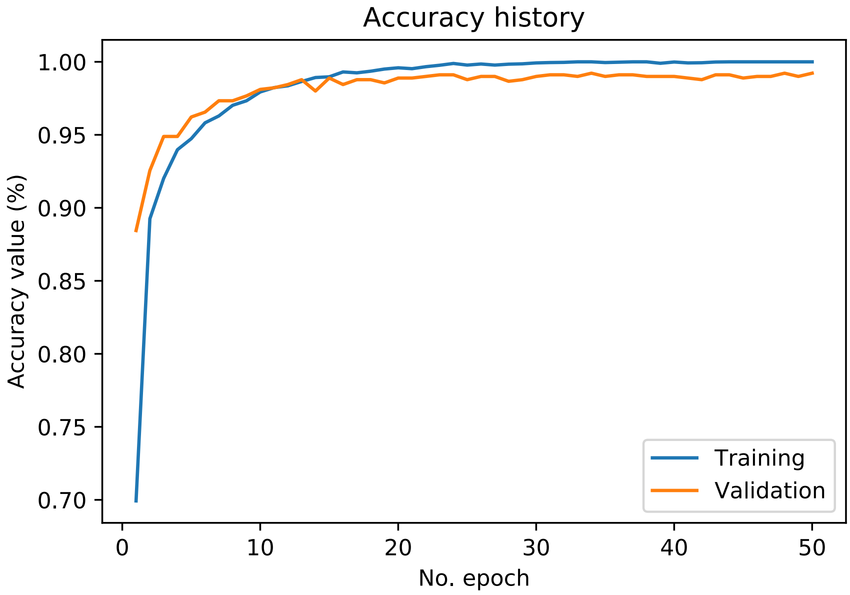
```
0.9878
Epoch 18/50
8094/8094 [==============================] - 1s 160us/sample - loss: 0.0228 -
categorical_accuracy: 0.9936 - val_loss: 0.0411 - val_categorical_accuracy:
0.9878
Epoch 19/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0197 -
categorical_accuracy: 0.9951 - val_loss: 0.0430 - val_categorical_accuracy:
0.9856
Epoch 20/50
8094/8094 [==============================] - 1s 153us/sample - loss: 0.0178 -
categorical_accuracy: 0.9959 - val_loss: 0.0366 - val_categorical_accuracy:
0.9889
Epoch 21/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0155 -
categorical_accuracy: 0.9953 - val_loss: 0.0387 - val_categorical_accuracy:
0.9889
Epoch 22/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0128 -
categorical_accuracy: 0.9967 - val_loss: 0.0343 - val_categorical_accuracy:
0.9900
Epoch 23/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0121 -
categorical_accuracy: 0.9977 - val_loss: 0.0367 - val_categorical_accuracy:
0.9911
Epoch 24/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0080 -
categorical_accuracy: 0.9989 - val_loss: 0.0322 - val_categorical_accuracy:
0.9911
Epoch 25/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0091 -
categorical_accuracy: 0.9978 - val_loss: 0.0380 - val_categorical_accuracy:
0.9878
Epoch 26/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0078 -
categorical_accuracy: 0.9985 - val_loss: 0.0467 - val_categorical_accuracy:
0.9900
Epoch 27/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0089 -
categorical_accuracy: 0.9978 - val_loss: 0.0352 - val_categorical_accuracy:
0.9900
Epoch 28/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0066 -
categorical_accuracy: 0.9984 - val_loss: 0.0425 - val_categorical_accuracy:
0.9867
Epoch 29/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0064 -
categorical_accuracy: 0.9986 - val_loss: 0.0471 - val_categorical_accuracy:
0.9878
Epoch 30/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0043 -
categorical_accuracy: 0.9993 - val_loss: 0.0365 - val_categorical_accuracy:
0.9900
Epoch 31/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0043 -
categorical_accuracy: 0.9995 - val_loss: 0.0356 - val_categorical_accuracy:
0.9911
Epoch 32/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0029 -
categorical_accuracy: 0.9996 - val_loss: 0.0381 - val_categorical_accuracy:
0.9911
Epoch 33/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0019 -
categorical_accuracy: 1.0000 - val_loss: 0.0355 - val_categorical_accuracy:
0.9900
Epoch 34/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0018 -
categorical_accuracy: 1.0000 - val_loss: 0.0347 - val_categorical_accuracy:
0.9922
Epoch 35/50
```

```
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0027 -
categorical_accuracy: 0.9995 - val_loss: 0.0408 - val_categorical_accuracy:
0.9900
Epoch 36/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0021 -
categorical_accuracy: 0.9998 - val_loss: 0.0398 - val_categorical_accuracy:
0.9911
Epoch 37/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0017 -
categorical_accuracy: 1.0000 - val_loss: 0.0464 - val_categorical_accuracy:
0.9911
Epoch 38/50
8094/8094 [==============================] - 1s 156us/sample - loss: 0.0014 -
categorical_accuracy: 1.0000 - val_loss: 0.0443 - val_categorical_accuracy:
0.9900
Epoch 39/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0040 -
categorical_accuracy: 0.9990 - val_loss: 0.0493 - val_categorical_accuracy:
0.9900
Epoch 40/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0026 -
categorical_accuracy: 0.9999 - val_loss: 0.0454 - val_categorical_accuracy:
0.9900
Epoch 41/50
8094/8094 [==============================] - 1s 154us/sample - loss: 0.0031 -
categorical_accuracy: 0.9993 - val_loss: 0.0309 - val_categorical_accuracy:
0.9889
Epoch 42/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0039 -
categorical_accuracy: 0.9994 - val_loss: 0.0399 - val_categorical_accuracy:
0.9878
Epoch 43/50
8094/8094 [==============================] - 1s 155us/sample - loss: 0.0018 -
categorical_accuracy: 0.9999 - val_loss: 0.0410 - val_categorical_accuracy:
0.9911
Epoch 44/50
8094/8094 [==============================] - 1s 155us/sample - loss: 6.6106e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0361 - val_categorical_accura
cy: 0.9911
Epoch 45/50
8094/8094 [==============================] - 1s 154us/sample - loss: 4.8223e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0390 - val_categorical_accura
cy: 0.9889
Epoch 46/50
8094/8094 [==============================] - 1s 156us/sample - loss: 5.6341e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0479 - val_categorical_accura
cy: 0.9900
Epoch 47/50
8094/8094 [==============================] - 1s 154us/sample - loss: 3.1683e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0433 - val_categorical_accura
cy: 0.9900
Epoch 48/50
8094/8094 [==============================] - 1s 154us/sample - loss: 4.7268e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0440 - val_categorical_accura
cy: 0.9922
Epoch 49/50
8094/8094 [==============================] - 1s 155us/sample - loss: 4.4508e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0446 - val_categorical_accura
cy: 0.9900
Epoch 50/50
8094/8094 [==============================] - 1s 153us/sample - loss: 2.7824e-
04 - categorical_accuracy: 1.0000 - val_loss: 0.0405 - val_categorical_accura
cy: 0.9922
```

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-6classes.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-6classes.svg")
plt.show()
```

Visualize result training





In [6]:

```
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 -  google-play
   4 -  roblox
   5 -  tiktok

2252/2252 [==============================] - 0s 118us/sample - loss: 0.0390 -
categorical_accuracy: 0.9911

Display model metrics:

Test accuracy: 0.991119

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       396
           1       0.98      0.98      0.98       329
           2       0.98      1.00      0.99       327
           3       0.99      0.97      0.98       400
           4       1.00      1.00      1.00       419
           5       0.99      1.00      0.99       381

    accuracy                           0.99      2252
   macro avg       0.99      0.99      0.99      2252
weighted avg       0.99      0.99      0.99      2252


Confusion matrix:
```

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