```
import pickle
import pandas as pd
import numpy as np

def unpickle(file):
        with open(file, 'rb') as fo:
                dict = pickle.load(fo, encoding='bytes')
        return dict

def load_labels_name(filename):
        with open(filename, 'rb') as f:
                obj = pickle.load(f)
        return obj


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# load data
train = unpickle('./drive/MyDrive/cifar-100-python/train')
X_train = train.get(b'data')
y_train = train.get(b'fine_labels')

name = load_labels_name('./drive/MyDrive/cifar-100-python/meta')

test = unpickle('./drive/MyDrive/cifar-100-python/test')
X_test = test.get(b'data')
y_test = test.get(b'fine_labels')
print(X_test.shape)
```

```
(10000, 3072)
```

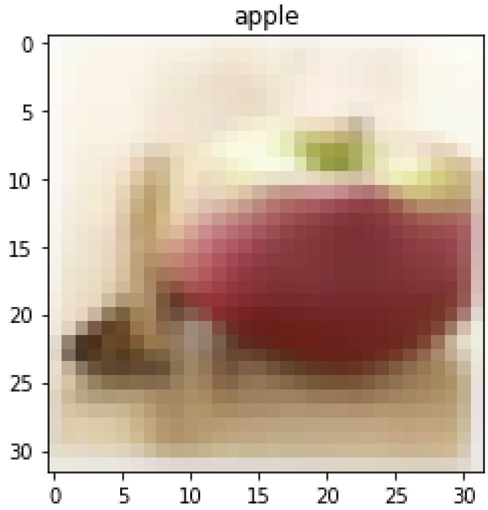
```
# vistualize sample image
# from PIL import Image
# %matplotlib inline
from matplotlib import pyplot as plt

X_train=X_train.reshape(-1,3,32,32)
X_train=np.rollaxis(X_train, 1, 4)
X_test=X_test.reshape(-1,3,32,32)
X_test=np.rollaxis(X_test, 1, 4)
X_train.shape
```

```
(50000, 32, 32, 3)
```

```
plt.figure(figsize=(4,4))
```

```
index = 2
plt.rcParams["axes.grid"] = False
plt.imshow(X_train[index])
plt.title(name['fine_label_names'][y_train[index]])
```

```
Text(0.5, 1.0, 'apple')
```



```
# from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# print(X_train)

X_train = X_train / 255.0
X_test = X_test / 255.0
```

CNN

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.losses import sparse_categorical_crossentropy
from tensorflow.keras.optimizers import Adam
# Create the model
model = Sequential()
model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Conv2D(128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(100, activation='softmax'))


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        896       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 13, 13, 64)        18496     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 128)         73856     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 2, 2, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
dense (Dense)                (None, 256)               131328    
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896     
_________________________________________________________________
dense_2 (Dense)              (None, 100)               12900     
=================================================================
Total params: 270,372
Trainable params: 270,372
Non-trainable params: 0
_________________________________________________________________
```

```
# Model configuration
batch_size = 256
verbosity = 1
epochs = 50

X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer=Adam(), metrics=['acc

# Fit data to model
history = model.fit(X_train, y_train, batch_size=batch_size,
```

```
epochs=epochs,
verbose=verbosity,validation_split=0.2)
```

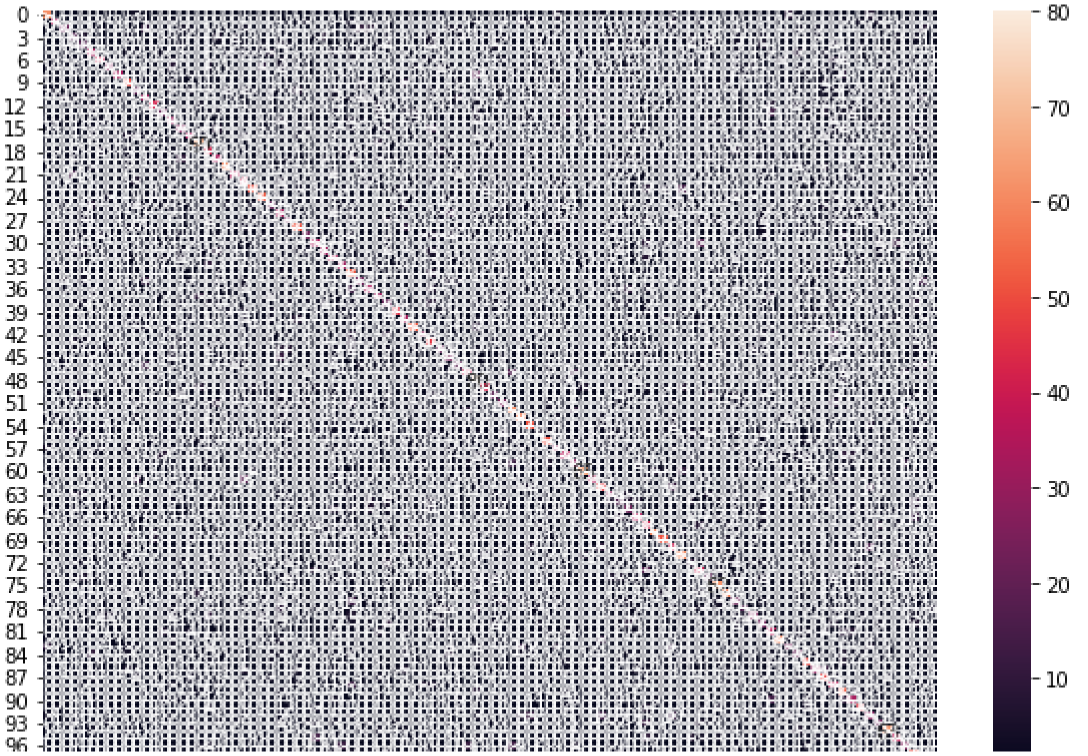
```
Epoch 1/50
157/157 [==============================] - 48s 300ms/step - loss: 4.2189 - accuracy: 0.05
Epoch 2/50
157/157 [==============================] - 47s 298ms/step - loss: 3.6949 - accuracy: 0.13
Epoch 3/50
157/157 [==============================] - 47s 299ms/step - loss: 3.3707 - accuracy: 0.18
Epoch 4/50
157/157 [==============================] - 47s 298ms/step - loss: 3.1455 - accuracy: 0.22
Epoch 5/50
157/157 [==============================] - 47s 299ms/step - loss: 2.9889 - accuracy: 0.25
Epoch 6/50
157/157 [==============================] - 47s 300ms/step - loss: 2.8632 - accuracy: 0.28
Epoch 7/50
157/157 [==============================] - 47s 300ms/step - loss: 2.7629 - accuracy: 0.30
Epoch 8/50
157/157 [==============================] - 47s 300ms/step - loss: 2.6541 - accuracy: 0.32
Epoch 9/50
157/157 [==============================] - 47s 301ms/step - loss: 2.5611 - accuracy: 0.34
Epoch 10/50
157/157 [==============================] - 47s 300ms/step - loss: 2.4853 - accuracy: 0.35
Epoch 11/50
157/157 [==============================] - 47s 300ms/step - loss: 2.4058 - accuracy: 0.37
Epoch 12/50
157/157 [==============================] - 47s 299ms/step - loss: 2.3515 - accuracy: 0.38
Epoch 13/50
157/157 [==============================] - 47s 299ms/step - loss: 2.2799 - accuracy: 0.40
Epoch 14/50
157/157 [==============================] - 47s 299ms/step - loss: 2.2076 - accuracy: 0.42
Epoch 15/50
157/157 [==============================] - 47s 300ms/step - loss: 2.1496 - accuracy: 0.43
Epoch 16/50
157/157 [==============================] - 47s 301ms/step - loss: 2.0859 - accuracy: 0.44
Epoch 17/50
157/157 [==============================] - 47s 300ms/step - loss: 2.0530 - accuracy: 0.45
Epoch 18/50
157/157 [==============================] - 47s 300ms/step - loss: 1.9732 - accuracy: 0.47
Epoch 19/50
157/157 [==============================] - 47s 300ms/step - loss: 1.9300 - accuracy: 0.48
Epoch 20/50
157/157 [==============================] - 47s 300ms/step - loss: 1.8768 - accuracy: 0.49
Epoch 21/50
157/157 [==============================] - 47s 300ms/step - loss: 1.8344 - accuracy: 0.50
Epoch 22/50
157/157 [==============================] - 47s 300ms/step - loss: 1.7856 - accuracy: 0.51
Epoch 23/50
157/157 [==============================] - 47s 300ms/step - loss: 1.7436 - accuracy: 0.52
Epoch 24/50
157/157 [==============================] - 47s 300ms/step - loss: 1.6926 - accuracy: 0.53
Epoch 25/50
157/157 [==============================] - 47s 301ms/step - loss: 1.6372 - accuracy: 0.55
Epoch 26/50
157/157 [==============================] - 47s 301ms/step - loss: 1.6031 - accuracy: 0.55
Epoch 27/50
157/157 [==============================] - 47s 300ms/step - loss: 1.5539 - accuracy: 0.57
Epoch 28/50
157/157 [==============================] - 47s 300ms/step - loss: 1.5101 - accuracy: 0.57
Epoch 29/50
157/157 [==============================] - 47s 299ms/step - loss: 1.4727 - accuracy: 0.58
```

```
score = model.evaluate(X_test, y_test)
print(f'Test loss: {score[0]} / Test accuracy: {score[1]}')
```

```
313/313 [==============================] - 4s 12ms/step - loss: 3.8395 - accuracy: 0.3620
Test loss: 3.8395190238952637 / Test accuracy: 0.3619999885559082
```

```
from sklearn import metrics
import seaborn as sn
actual = y_test
y_prob = model.predict(X_test)
predicted = np.argmax(y_prob,axis=1)
print(metrics.classification_report(actual, predicted))

# show the confusion matrix
print("confusion matrix:")
m = metrics.confusion_matrix(actual, predicted)
print(m)
mplot = pd.DataFrame(m)
plt.figure(figsize=(10,7))
sn.heatmap(mplot,annot = True)
plt.show()
```

```
          78       0.30      0.15      0.21       100
          79       0.35      0.26      0.30       100
          80       0.16      0.22      0.19       100
          81       0.34      0.37      0.35       100
          82       0.71      0.60      0.65       100
          83       0.35      0.23      0.28       100
          84       0.18      0.20      0.19       100
          85       0.40      0.49      0.44       100
          86       0.39      0.29      0.33       100
          87       0.39      0.48      0.43       100
          88       0.22      0.38      0.28       100
          89       0.29      0.52      0.37       100
          90       0.30      0.33      0.32       100
          91       0.42      0.46      0.44       100
          92       0.28      0.27      0.28       100
          93       0.21      0.21      0.21       100
          94       0.62      0.63      0.62       100
          95       0.46      0.39      0.42       100
          96       0.38      0.25      0.30       100
          97       0.27      0.50      0.35       100
          98       0.19      0.15      0.17       100
          99       0.50      0.32      0.39       100

    accuracy                           0.36     10000
   macro avg       0.38      0.36      0.36     10000
weighted avg       0.38      0.36      0.36     10000

confusion matrix:
[[61  0  0 ...  0  0  0]
 [ 0 36  0 ...  0  0  0]
 [ 1  0 21 ...  0  7  0]
 ...
 [ 0  0  0 ... 50  0  0]
 [ 0  0  2 ...  1 15  1]
 [ 0  0  1 ...  1  0 32]]
```



```
# Plot history: Accuracy
plt.plot(range(epochs), history.history['accuracy'], label='accuracy')
plt.plot(range(epochs), history.history['val_accuracy'], label='val_accuracy')
plt.title('accuracy history')
plt.ylabel('Accuracy value')
plt.xlabel('No. epoch')
plt.legend()
plt.grid(True)
```