📷 Cifar-10 Image Classifiction

The `CIFAR-10` dataset consists of `60000` `32x32` color images in `10` classes, with `6000` images per class. There are `50000` training images and `10000` test images.

🔬 Problem Definition:

Given an image, can we predict the correct class of this image?

The images are very small (`32x32`) and by visualizing them you will notice how difficult it is to distinguish them even for a human.

In this notebook we are going to build a CNN model that can classify images of various objects. We have `10` class of images:

1. Airplane
2. Automobile
3. Bird
4. Cat
5. Deer
6. Dog
7. Frog
8. Horse
9. Ship
10. Truck

🎯 Evaluation:

We have `10` classes, so if we pick a image and we randomly gues it class, we have `1/10` probability to be true.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.image import ImageDataGenerator

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

📥 Load the data

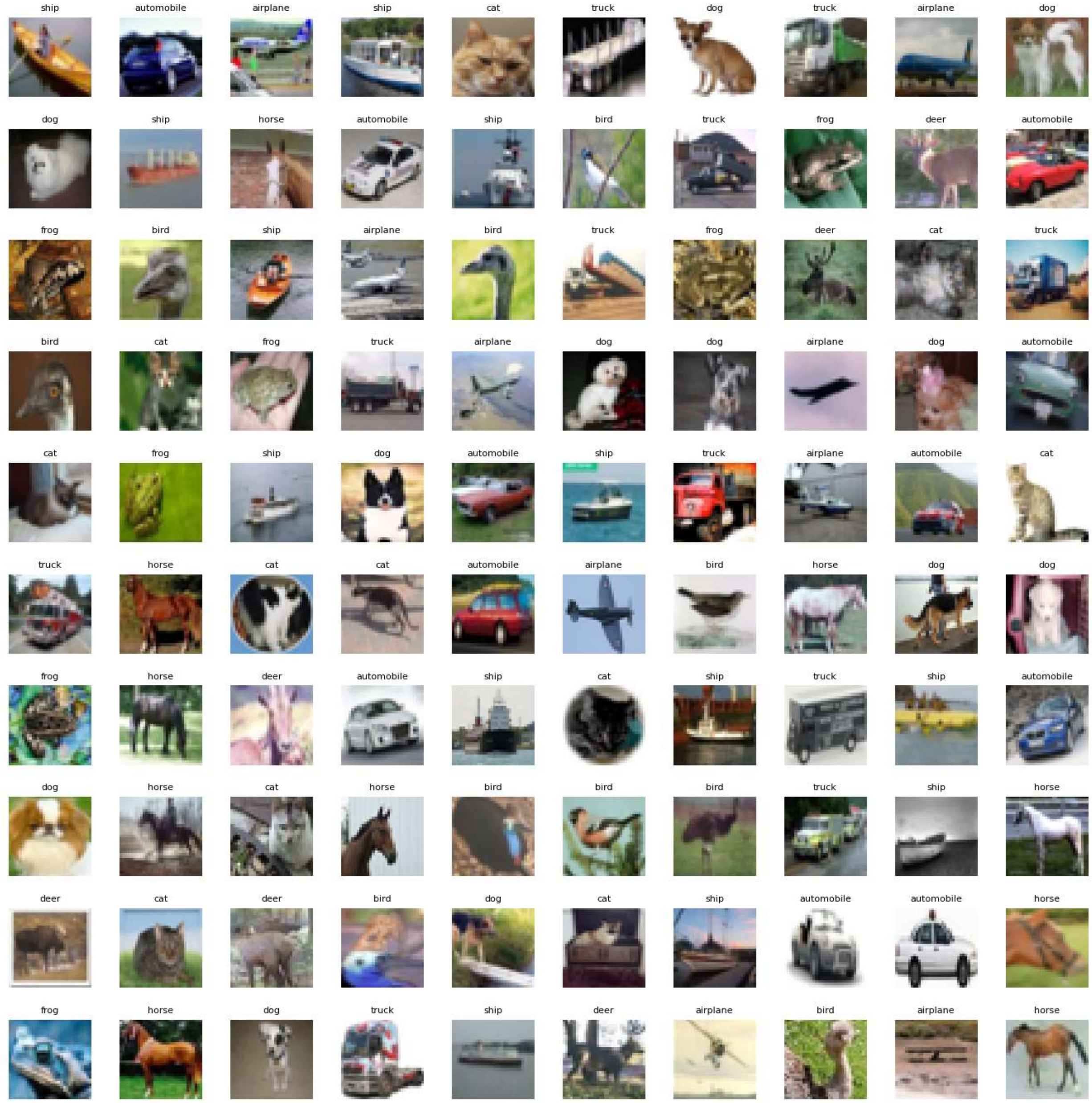
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

print(f"X_train shape: {X_train.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 6s 0us/step
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

🖼 Data Visualization

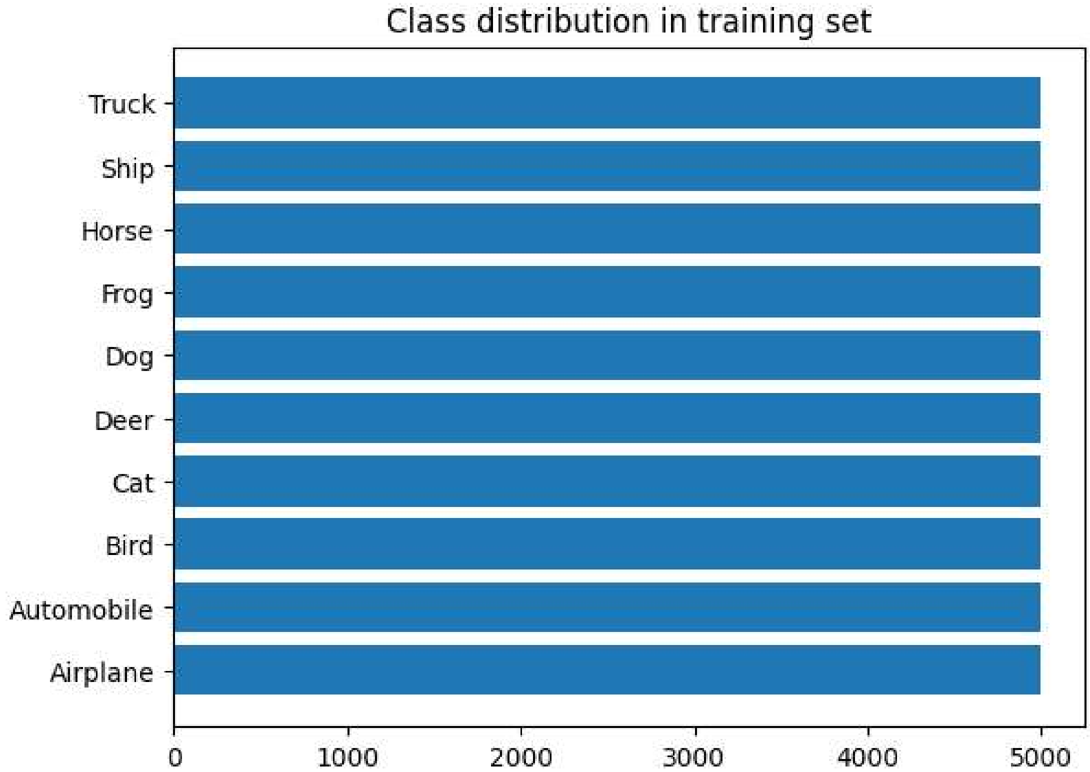
```
# Define the labels of the dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']

# Let's view more images in a grid format
# Define the dimensions of the plot grid
W_grid = 10
L_grid = 10

# fig, axes = plt.subplots(L_grid, W_grid)
# subplot return the figure object and axes object
# we can use the axes object to plot specific figures at various locations

fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array
```

```
n_train = len(X_train) # 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(X_train[index,1:])
    label_index = int(y_train[index])
    axes[i].set_title(labels[label_index], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```



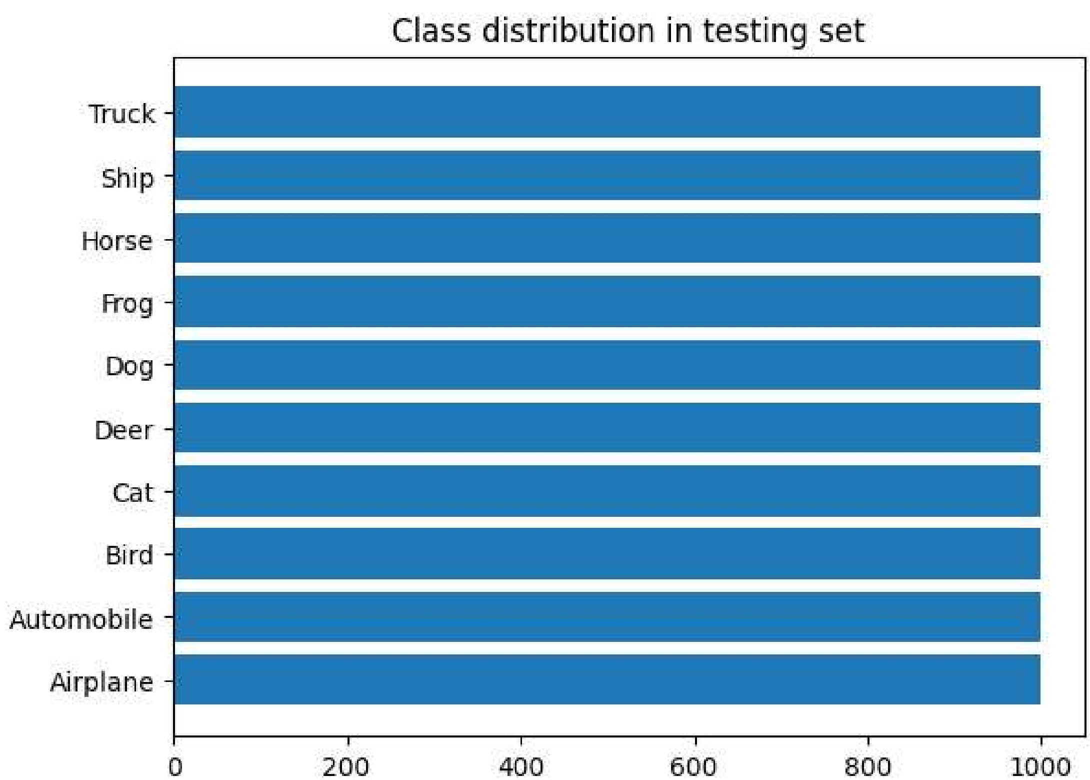

```
classes_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Tr

classes, counts = np.unique(y_train, return_counts=True)
plt.barh(classes_name, counts)
plt.title('Class distribution in training set')
```

```
Text(0.5, 1.0, 'Class distribution in training set')
```




```
classes, counts = np.unique(y_test, return_counts=True)
plt.barh(classes_name, counts)
plt.title('Class distribution in testing set')
```

```
Text(0.5, 1.0, 'Class distribution in testing set')
```



The class are equally distributed

🔄 Data Preprocessing

```
# Scale the data
X_train = X_train / 255.0
X_test = X_test / 255.0

# Transform target variable into one-hotencoding
y_cat_train = to_categorical(y_train, 10)
y_cat_test = to_categorical(y_test, 10)
```

```
y_cat_train
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.]], dtype=float32)
```

🤖 Model Building

```
INPUT_SHAPE = (32, 32, 3)
KERNEL_SIZE = (3, 3)
model = Sequential()

# Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', pa
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', pa
model.add(BatchNormalization())
# Pooling layer
model.add(MaxPool2D(pool_size=(2, 2)))
# Dropout layers
model.add(Dropout(0.25))

model.add(Conv2D(filters=64, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', pa
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', pa
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', p
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, activation='relu', p
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
# model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(10, activation='softmax'))

METRICS = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=METRICS)


model.summary()
```

```
      conv2d (Conv2D)             (None, 32, 32, 32)        896

      batch_normalization (Batch  (None, 32, 32, 32)        128
      Normalization)

      conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

      batch_normalization_1 (Bat  (None, 32, 32, 32)        128
      chNormalization)
```

```
 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Bat  (None, 16, 16, 64)        256
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Bat  (None, 8, 8, 128)         512
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 552362 (2.11 MB)
Trainable params: 551466 (2.10 MB)
Non-trainable params: 896 (3.50 KB)
```

Early Stopping

```
early_stop = EarlyStopping(monitor='val_loss', patience=2)
```

Data Augmentations

```
batch_size = 32
data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=Tru
train_generator = data_generator.flow(X_train, y_cat_train, batch_size)
steps_per_epoch = X_train.shape[0] // batch_size
```

```
r = model.fit(train_generator,
              epochs=50,
              steps_per_epoch=steps_per_epoch,
              validation_data=(X_test, y_cat_test),
#               callbacks=[early_stop],
#               batch_size=batch_size,
             )
```

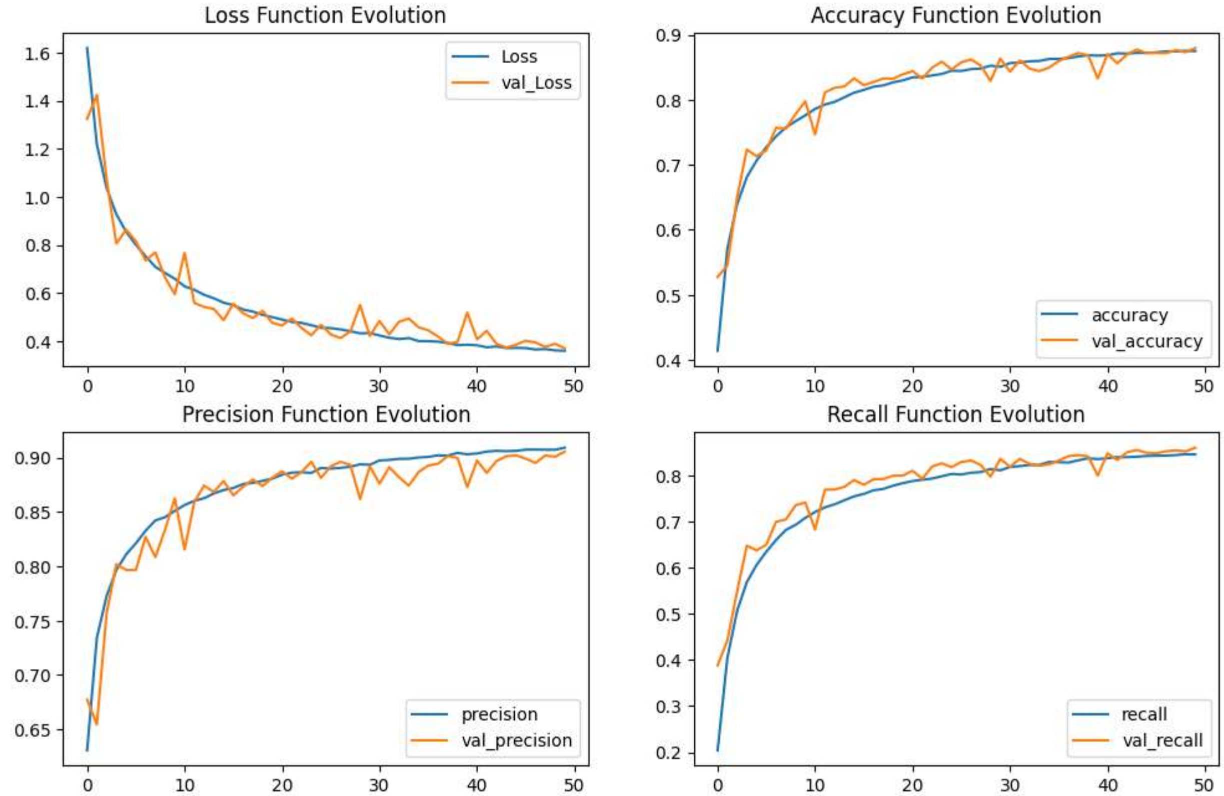
```
1562/1562 [==============================] - 45s 29ms/step - loss: 0.4800 - accuracy: 0.8352 -
Epoch 23/50
1562/1562 [==============================] - 44s 28ms/step - loss: 0.4758 - accuracy: 0.8375 -
Epoch 24/50
1562/1562 [==============================] - 44s 28ms/step - loss: 0.4662 - accuracy: 0.8398 -
Epoch 25/50
1562/1562 [==============================] - 47s 30ms/step - loss: 0.4555 - accuracy: 0.8448 -
Epoch 26/50
```

```
Epoch 48/50
1562/1562 [==============================] - 43s 28ms/step - loss: 0.3669 - accuracy: 0.8744 -
Epoch 49/50
1562/1562 [==============================] - 43s 27ms/step - loss: 0.3615 - accuracy: 0.8753 -
Epoch 50/50
1562/1562 [==============================] - 44s 28ms/step - loss: 0.3599 - accuracy: 0.8750 -
```

📊 Model Evaluation

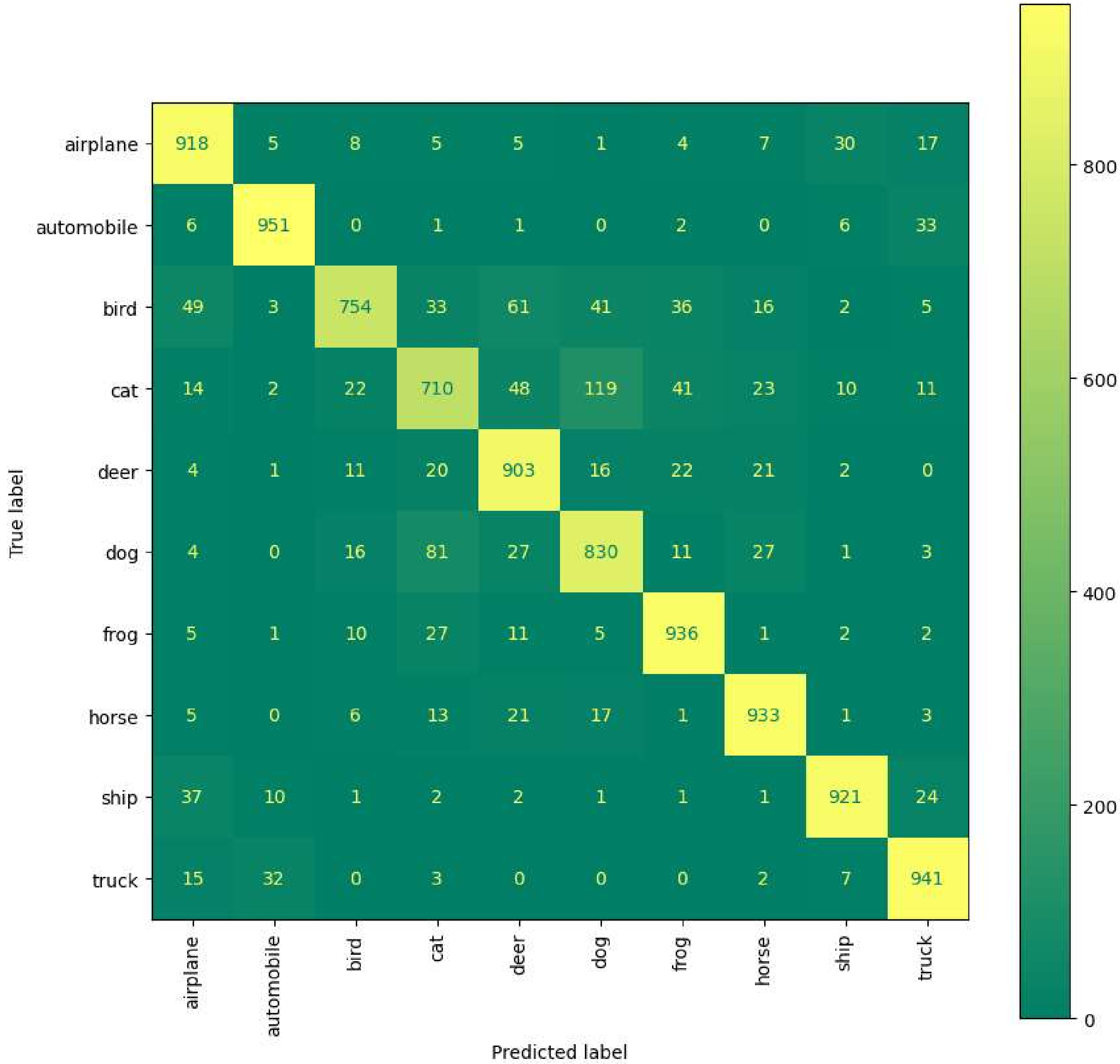
```
plt.figure(figsize=(12, 16))

plt.subplot(4, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='val_Loss')
plt.title('Loss Function Evolution')
plt.legend()

plt.subplot(4, 2, 2)
plt.plot(r.history['accuracy'], label='accuracy')
plt.plot(r.history['val_accuracy'], label='val_accuracy')
plt.title('Accuracy Function Evolution')
plt.legend()

plt.subplot(4, 2, 3)
plt.plot(r.history['precision'], label='precision')
plt.plot(r.history['val_precision'], label='val_precision')
plt.title('Precision Function Evolution')
plt.legend()

plt.subplot(4, 2, 4)
plt.plot(r.history['recall'], label='recall')
plt.plot(r.history['val_recall'], label='val_recall')
plt.title('Recall Function Evolution')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x783d9038f280>
```




```
evaluation = model.evaluate(X_test, y_cat_test)
print(f'Test Accuracy : {evaluation[1] * 100:.2f}%')

y_pred = model.predict(X_test)
y_pred = np.argmax(y_pred, axis=1)
cm = confusion_matrix(y_test, y_pred)


disp = ConfusionMatrixDisplay(confusion_matrix=cm,
                              display_labels=labels)


# NOTE: Fill all variables here with default values of the plot_confusion_matrix
fig, ax = plt.subplots(figsize=(10, 10))
disp = disp.plot(xticks_rotation='vertical', ax=ax,cmap='summer')

plt.show()
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.3702 - accuracy: 0.879
Test Accuracy : 87.97%
313/313 [==============================] - 1s 3ms/step
```




```
print(classification_report(y_test, y_pred))
```

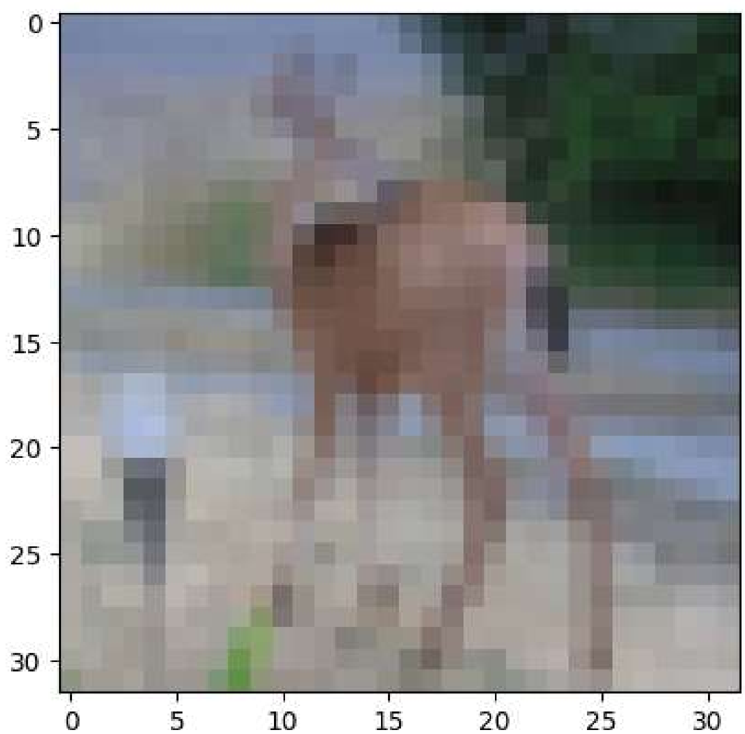
```
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      1000
           1       0.95      0.95      0.95      1000
           2       0.91      0.75      0.82      1000
           3       0.79      0.71      0.75      1000
           4       0.84      0.90      0.87      1000
           5       0.81      0.83      0.82      1000
```

```
           6       0.89      0.94      0.91      1000
           7       0.90      0.93      0.92      1000
           8       0.94      0.92      0.93      1000
           9       0.91      0.94      0.92      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

Test on one image

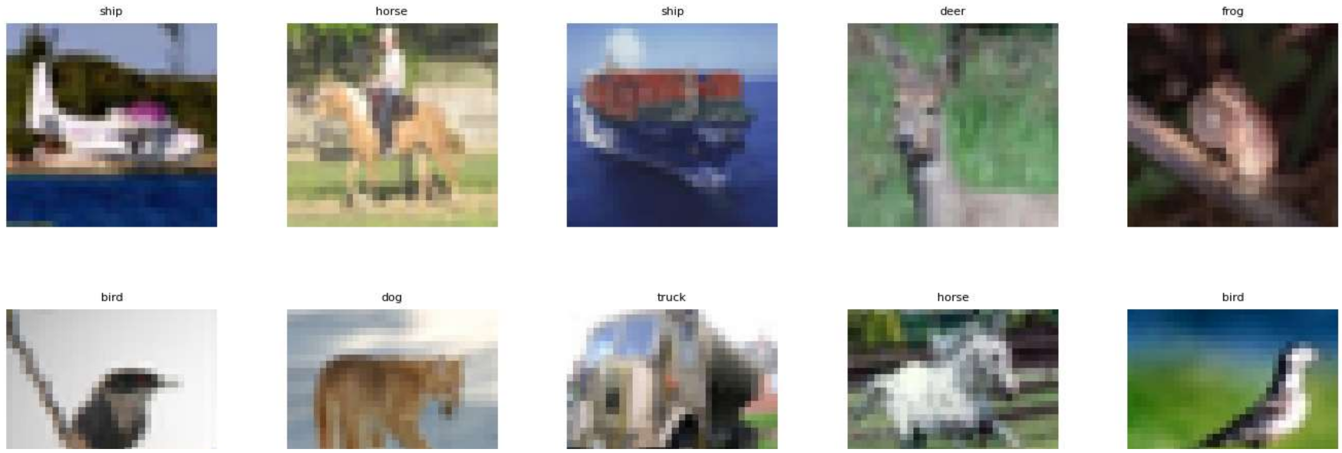
```
my_image = X_test[100]
plt.imshow(my_image)

# that's a Deer
print(f" Image 100 is {y_test[100]}")

# correctly predicted as a Deer
pred_100 = np.argmax(model.predict(my_image.reshape(1, 32, 32, 3)))
print(f"The model predict that image 100 is {pred_100}")
```

```
 Image 100 is [4]
1/1 [==============================] - 0s 369ms/step
The model predict that image 100 is 4
```




```
# Define the labels of the dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']

# Let's view more images in a grid format
# Define the dimensions of the plot grid
```

```
W_grid = 5
L_grid = 5

# fig, axes = plt.subplots(L_grid, W_grid)
# subplot return the figure object and axes object
# we can use the axes object to plot specific figures at various locations

fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))

axes = axes.ravel() # flaten the 15 x 15 matrix into 225 array

n_test = len(X_test) # 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_test)
    # read and display an image with the selected index
    axes[i].imshow(X_test[index,1:])
    label_index = int(y_pred[index])
    axes[i].set_title(labels[label_index], fontsize = 8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.4)
```


ship
horse
ship
deer
frog
bird
dog
truck
horse
bird