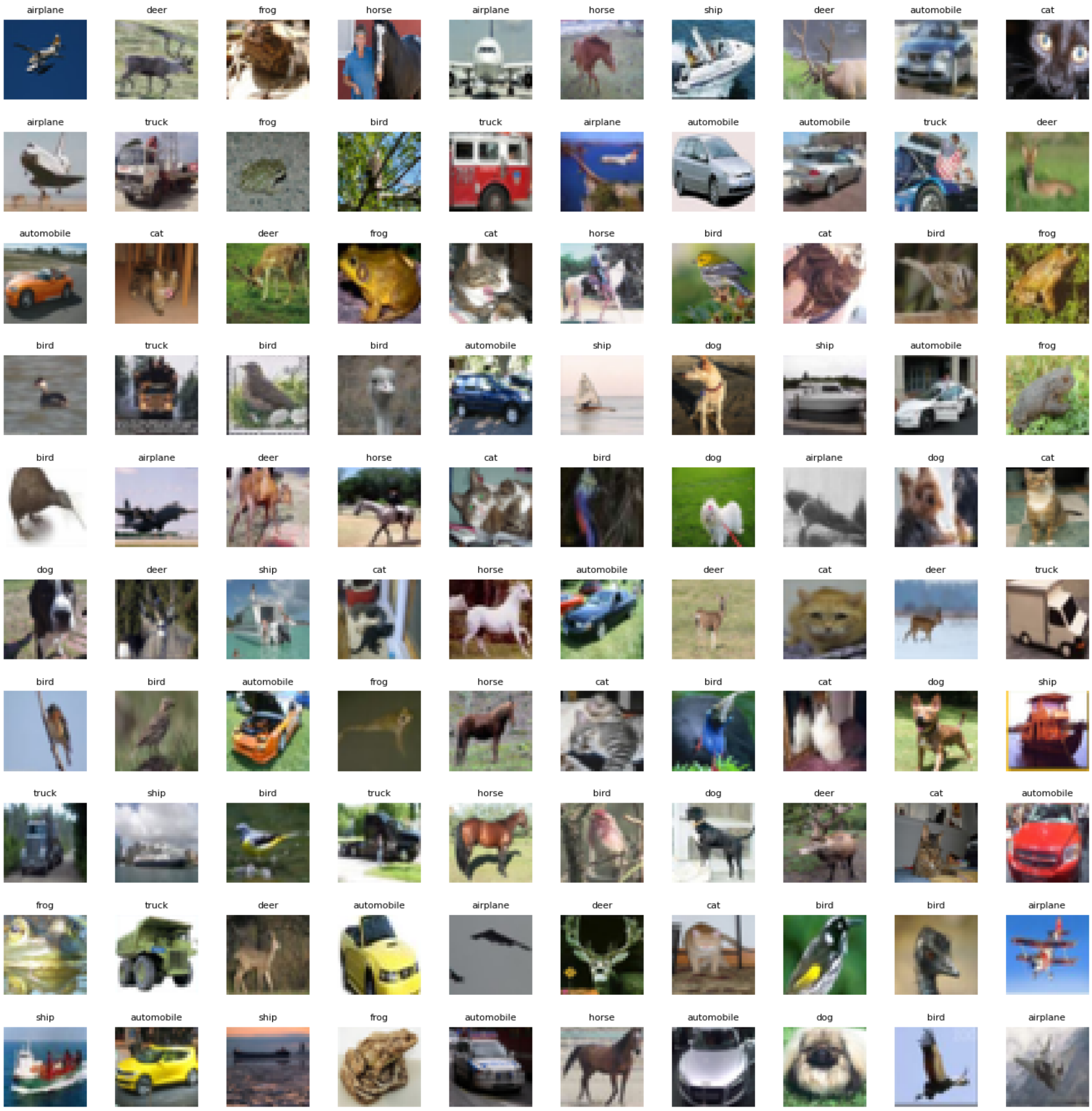
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

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

In [2]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, Bat
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
```

In [3]:
```
(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}")
```

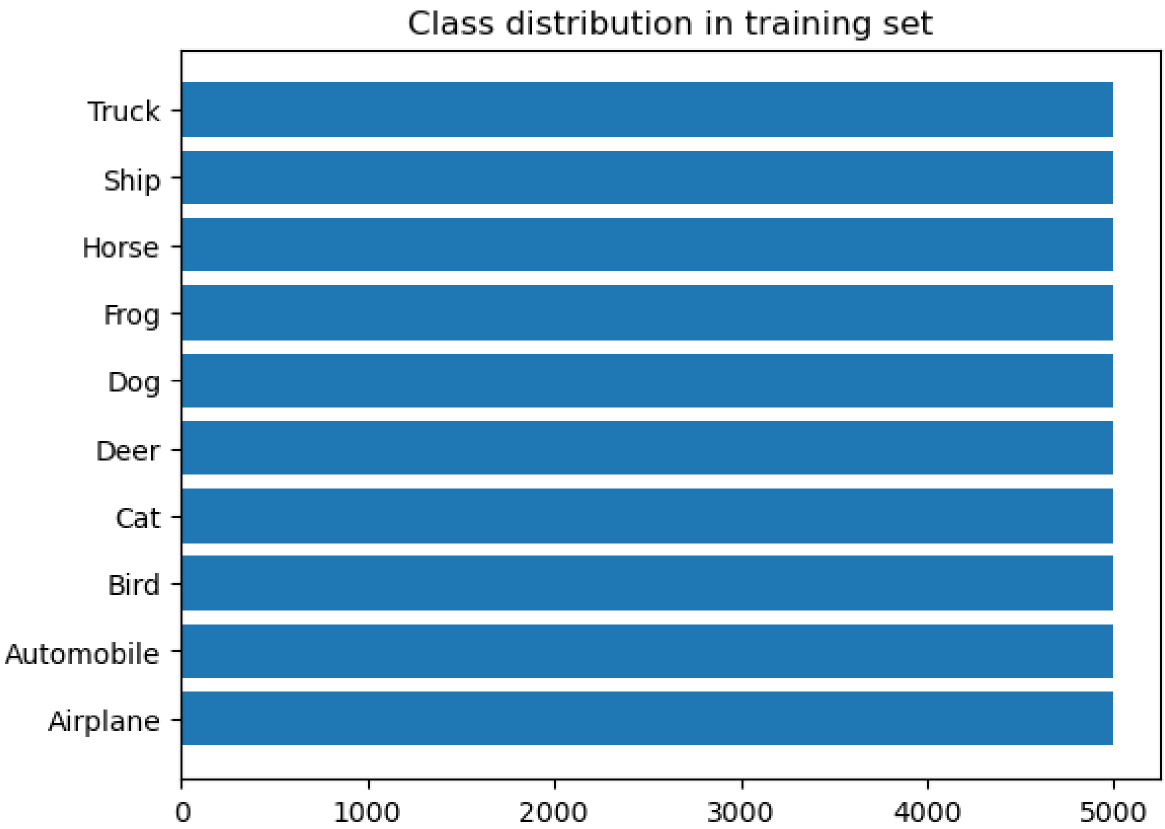
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 1009s 6us/step
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

In [5]:
```
# 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)
```



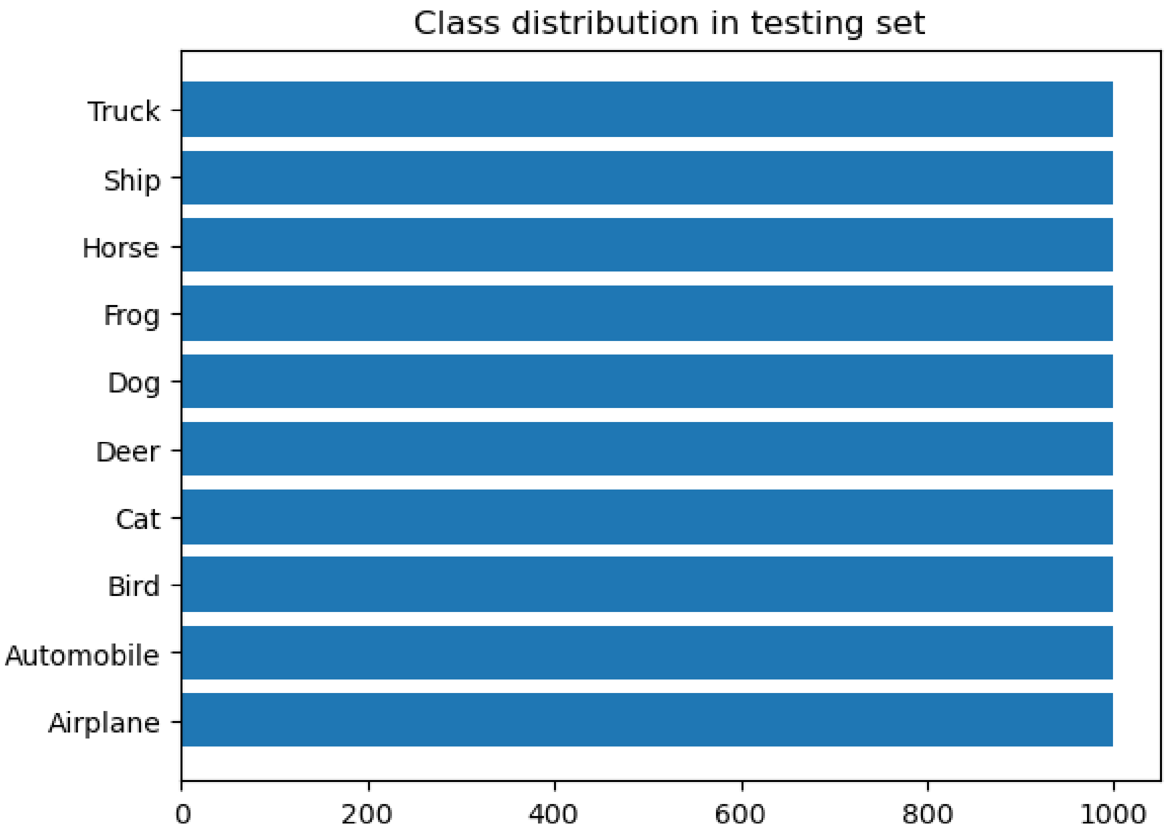
In [6]:
```
classes_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Ho

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

Out[6]:
```
Text(0.5, 1.0, 'Class distribution in training set')
```



In [7]:
```
classes, counts = np.unique(y_test, return_counts=True)
plt.barh(classes_name, counts)
plt.title('Class distribution in testing set')
```

Out[7]:
```
Text(0.5, 1.0, 'Class distribution in testing set')
```



In [8]:
```
# 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)
```

In [9]:

```
y_cat_train
```

Out[9]:

```
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)
```

In [10]:

```
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, acti
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, acti
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, acti
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, acti
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, act
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=KERNEL_SIZE, input_shape=INPUT_SHAPE, act
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)
```

In [11]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 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)                                                
                                                                 
 max_pooling2d (MaxPooling2  (None, 16, 16, 32)        0         
 D)                                                              
                                                                 
 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)
_________________________________________________________________
```

In [12]:

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

In [13]:

```
batch_size = 32
data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1,
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,
             )
```

```
Epoch 1/50
1562/1562 [==============================] - 280s 176ms/step - loss: 1.6348 - accu
racy: 0.4067 - precision: 0.6259 - recall: 0.1954 - val_loss: 1.7578 - val_accurac
y: 0.4262 - val_precision: 0.5140 - val_recall: 0.3071
Epoch 2/50
1562/1562 [==============================] - 282s 181ms/step - loss: 1.2277 - accu
racy: 0.5639 - precision: 0.7242 - recall: 0.3981 - val_loss: 1.0447 - val_accurac
y: 0.6356 - val_precision: 0.7621 - val_recall: 0.5083
Epoch 3/50
1562/1562 [==============================] - 279s 178ms/step - loss: 1.0493 - accu
racy: 0.6376 - precision: 0.7714 - recall: 0.5008 - val_loss: 0.8279 - val_accurac
y: 0.7176 - val_precision: 0.8213 - val_recall: 0.6140
Epoch 4/50
1562/1562 [==============================] - 275s 176ms/step - loss: 0.9370 - accu
racy: 0.6782 - precision: 0.7939 - recall: 0.5642 - val_loss: 0.8358 - val_accurac
y: 0.7178 - val_precision: 0.8077 - val_recall: 0.6274
Epoch 5/50
1562/1562 [==============================] - 274s 175ms/step - loss: 0.8614 - accu
racy: 0.7081 - precision: 0.8128 - recall: 0.6069 - val_loss: 0.8608 - val_accurac
y: 0.7173 - val_precision: 0.7981 - val_recall: 0.6473
Epoch 6/50
1562/1562 [==============================] - 306s 196ms/step - loss: 0.8059 - accu
racy: 0.7269 - precision: 0.8229 - recall: 0.6349 - val_loss: 0.7250 - val_accurac
y: 0.7638 - val_precision: 0.8502 - val_recall: 0.6805
Epoch 7/50
1562/1562 [==============================] - 289s 185ms/step - loss: 0.7634 - accu
racy: 0.7428 - precision: 0.8340 - recall: 0.6557 - val_loss: 0.7032 - val_accurac
y: 0.7642 - val_precision: 0.8218 - val_recall: 0.7073
Epoch 8/50
1562/1562 [==============================] - 288s 184ms/step - loss: 0.7222 - accu
racy: 0.7549 - precision: 0.8396 - recall: 0.6765 - val_loss: 0.6342 - val_accurac
y: 0.7872 - val_precision: 0.8574 - val_recall: 0.7287
Epoch 9/50
1562/1562 [==============================] - 285s 182ms/step - loss: 0.6954 - accu
racy: 0.7663 - precision: 0.8472 - recall: 0.6905 - val_loss: 0.6603 - val_accurac
y: 0.7810 - val_precision: 0.8438 - val_recall: 0.7297
Epoch 10/50
1562/1562 [==============================] - 286s 183ms/step - loss: 0.6684 - accu
racy: 0.7725 - precision: 0.8501 - recall: 0.7035 - val_loss: 0.5929 - val_accurac
y: 0.8031 - val_precision: 0.8621 - val_recall: 0.7430
Epoch 11/50
1562/1562 [==============================] - 283s 181ms/step - loss: 0.6427 - accu
racy: 0.7824 - precision: 0.8548 - recall: 0.7169 - val_loss: 0.6271 - val_accurac
y: 0.7914 - val_precision: 0.8490 - val_recall: 0.7434
Epoch 12/50
1562/1562 [==============================] - 287s 184ms/step - loss: 0.6201 - accu
racy: 0.7922 - precision: 0.8592 - recall: 0.7274 - val_loss: 0.5598 - val_accurac
y: 0.8155 - val_precision: 0.8708 - val_recall: 0.7632
Epoch 13/50
1562/1562 [==============================] - 293s 188ms/step - loss: 0.6018 - accu
racy: 0.7957 - precision: 0.8625 - recall: 0.7357 - val_loss: 0.5439 - val_accurac
y: 0.8169 - val_precision: 0.8679 - val_recall: 0.7767
Epoch 14/50
1562/1562 [==============================] - 278s 178ms/step - loss: 0.5825 - accu
racy: 0.8037 - precision: 0.8659 - recall: 0.7465 - val_loss: 0.5620 - val_accurac
y: 0.8156 - val_precision: 0.8689 - val_recall: 0.7694
Epoch 15/50
1562/1562 [==============================] - 14687s 9s/step - loss: 0.5643 - accur
acy: 0.8082 - precision: 0.8697 - recall: 0.7533 - val_loss: 0.5283 - val_accurac
y: 0.8237 - val_precision: 0.8747 - val_recall: 0.7820
Epoch 16/50
1562/1562 [==============================] - 24473s 16s/step - loss: 0.5558 - accu
racy: 0.8136 - precision: 0.8725 - recall: 0.7580 - val_loss: 0.5060 - val_accurac
y: 0.8317 - val_precision: 0.8875 - val_recall: 0.7901
```

```
Epoch 17/50
1562/1562 [==============================] - 300s 192ms/step - loss: 0.5392 - accu
racy: 0.8167 - precision: 0.8762 - recall: 0.7659 - val_loss: 0.4698 - val_accurac
y: 0.8456 - val_precision: 0.8843 - val_recall: 0.8140
Epoch 18/50
1562/1562 [==============================] - 293s 188ms/step - loss: 0.5318 - accu
racy: 0.8213 - precision: 0.8766 - recall: 0.7706 - val_loss: 0.4832 - val_accurac
y: 0.8373 - val_precision: 0.8822 - val_recall: 0.8033
Epoch 19/50
1562/1562 [==============================] - 297s 190ms/step - loss: 0.5154 - accu
racy: 0.8250 - precision: 0.8790 - recall: 0.7748 - val_loss: 0.5067 - val_accurac
y: 0.8282 - val_precision: 0.8732 - val_recall: 0.7943
Epoch 20/50
1562/1562 [==============================] - 298s 191ms/step - loss: 0.5074 - accu
racy: 0.8281 - precision: 0.8815 - recall: 0.7799 - val_loss: 0.5168 - val_accurac
y: 0.8333 - val_precision: 0.8726 - val_recall: 0.7980
Epoch 21/50
1562/1562 [==============================] - 295s 189ms/step - loss: 0.4952 - accu
racy: 0.8321 - precision: 0.8829 - recall: 0.7852 - val_loss: 0.5882 - val_accurac
y: 0.8134 - val_precision: 0.8631 - val_recall: 0.7751
Epoch 22/50
1562/1562 [==============================] - 290s 186ms/step - loss: 0.4915 - accu
racy: 0.8331 - precision: 0.8841 - recall: 0.7878 - val_loss: 0.4723 - val_accurac
y: 0.8419 - val_precision: 0.8928 - val_recall: 0.7956
Epoch 23/50
1562/1562 [==============================] - 296s 189ms/step - loss: 0.4799 - accu
racy: 0.8359 - precision: 0.8870 - recall: 0.7925 - val_loss: 0.5121 - val_accurac
y: 0.8338 - val_precision: 0.8777 - val_recall: 0.8004
Epoch 24/50
1562/1562 [==============================] - 287s 184ms/step - loss: 0.4675 - accu
racy: 0.8420 - precision: 0.8903 - recall: 0.7991 - val_loss: 0.5167 - val_accurac
y: 0.8310 - val_precision: 0.8740 - val_recall: 0.7962
Epoch 25/50
1562/1562 [==============================] - 294s 188ms/step - loss: 0.4643 - accu
racy: 0.8429 - precision: 0.8901 - recall: 0.8007 - val_loss: 0.4694 - val_accurac
y: 0.8474 - val_precision: 0.8803 - val_recall: 0.8193
Epoch 26/50
1562/1562 [==============================] - 300s 192ms/step - loss: 0.4590 - accu
racy: 0.8432 - precision: 0.8902 - recall: 0.8024 - val_loss: 0.5471 - val_accurac
y: 0.8221 - val_precision: 0.8637 - val_recall: 0.7898
Epoch 27/50
1562/1562 [==============================] - 299s 191ms/step - loss: 0.4491 - accu
racy: 0.8471 - precision: 0.8940 - recall: 0.8060 - val_loss: 0.4521 - val_accurac
y: 0.8528 - val_precision: 0.8884 - val_recall: 0.8227
Epoch 28/50
1562/1562 [==============================] - 320s 205ms/step - loss: 0.4487 - accu
racy: 0.8490 - precision: 0.8927 - recall: 0.8072 - val_loss: 0.4716 - val_accurac
y: 0.8492 - val_precision: 0.8851 - val_recall: 0.8200
Epoch 29/50
1562/1562 [==============================] - 331s 212ms/step - loss: 0.4407 - accu
racy: 0.8502 - precision: 0.8953 - recall: 0.8110 - val_loss: 0.4851 - val_accurac
y: 0.8428 - val_precision: 0.8826 - val_recall: 0.8075
Epoch 30/50
1562/1562 [==============================] - 294s 188ms/step - loss: 0.4362 - accu
racy: 0.8491 - precision: 0.8917 - recall: 0.8104 - val_loss: 0.4794 - val_accurac
y: 0.8427 - val_precision: 0.8806 - val_recall: 0.8157
Epoch 31/50
1562/1562 [==============================] - 287s 184ms/step - loss: 0.4306 - accu
racy: 0.8535 - precision: 0.8952 - recall: 0.8149 - val_loss: 0.4355 - val_accurac
y: 0.8546 - val_precision: 0.8934 - val_recall: 0.8203
Epoch 32/50
1562/1562 [==============================] - 4009s 3s/step - loss: 0.4234 - accura
cy: 0.8549 - precision: 0.8976 - recall: 0.8192 - val_loss: 0.3986 - val_accuracy:
0.8710 - val_precision: 0.9010 - val_recall: 0.8439
```

```
Epoch 33/50
1562/1562 [==============================] - 331s 212ms/step - loss: 0.4186 - accu
racy: 0.8566 - precision: 0.8982 - recall: 0.8203 - val_loss: 0.4435 - val_accurac
y: 0.8569 - val_precision: 0.8902 - val_recall: 0.8298
Epoch 34/50
1562/1562 [==============================] - 319s 204ms/step - loss: 0.4118 - accu
racy: 0.8600 - precision: 0.8991 - recall: 0.8240 - val_loss: 0.4004 - val_accurac
y: 0.8683 - val_precision: 0.9032 - val_recall: 0.8388
Epoch 35/50
1562/1562 [==============================] - 354s 227ms/step - loss: 0.4134 - accu
racy: 0.8589 - precision: 0.8997 - recall: 0.8229 - val_loss: 0.4204 - val_accurac
y: 0.8642 - val_precision: 0.8967 - val_recall: 0.8378
Epoch 36/50
1562/1562 [==============================] - 340s 217ms/step - loss: 0.4057 - accu
racy: 0.8618 - precision: 0.9013 - recall: 0.8253 - val_loss: 0.4233 - val_accurac
y: 0.8631 - val_precision: 0.8948 - val_recall: 0.8391
Epoch 37/50
1562/1562 [==============================] - 348s 223ms/step - loss: 0.4055 - accu
racy: 0.8617 - precision: 0.9008 - recall: 0.8272 - val_loss: 0.4539 - val_accurac
y: 0.8546 - val_precision: 0.8920 - val_recall: 0.8246
Epoch 38/50
1562/1562 [==============================] - 337s 216ms/step - loss: 0.3994 - accu
racy: 0.8628 - precision: 0.9027 - recall: 0.8298 - val_loss: 0.3946 - val_accurac
y: 0.8703 - val_precision: 0.9054 - val_recall: 0.8377
Epoch 39/50
1562/1562 [==============================] - 340s 217ms/step - loss: 0.3964 - accu
racy: 0.8654 - precision: 0.9020 - recall: 0.8307 - val_loss: 0.4203 - val_accurac
y: 0.8615 - val_precision: 0.8965 - val_recall: 0.8334
Epoch 40/50
1562/1562 [==============================] - 336s 215ms/step - loss: 0.3972 - accu
racy: 0.8640 - precision: 0.9004 - recall: 0.8304 - val_loss: 0.4260 - val_accurac
y: 0.8610 - val_precision: 0.8933 - val_recall: 0.8334
Epoch 41/50
1562/1562 [==============================] - 333s 213ms/step - loss: 0.3924 - accu
racy: 0.8657 - precision: 0.9027 - recall: 0.8325 - val_loss: 0.4133 - val_accurac
y: 0.8671 - val_precision: 0.8990 - val_recall: 0.8411
Epoch 42/50
1562/1562 [==============================] - 343s 220ms/step - loss: 0.3862 - accu
racy: 0.8663 - precision: 0.9032 - recall: 0.8347 - val_loss: 0.4003 - val_accurac
y: 0.8696 - val_precision: 0.9005 - val_recall: 0.8456
Epoch 43/50
1562/1562 [==============================] - 344s 220ms/step - loss: 0.3810 - accu
racy: 0.8688 - precision: 0.9062 - recall: 0.8379 - val_loss: 0.4208 - val_accurac
y: 0.8623 - val_precision: 0.8935 - val_recall: 0.8357
Epoch 44/50
1562/1562 [==============================] - 362s 232ms/step - loss: 0.3801 - accu
racy: 0.8704 - precision: 0.9047 - recall: 0.8384 - val_loss: 0.4189 - val_accurac
y: 0.8683 - val_precision: 0.8942 - val_recall: 0.8477
Epoch 45/50
1562/1562 [==============================] - 350s 224ms/step - loss: 0.3790 - accu
racy: 0.8701 - precision: 0.9063 - recall: 0.8381 - val_loss: 0.4316 - val_accurac
y: 0.8636 - val_precision: 0.8881 - val_recall: 0.8405
Epoch 46/50
1562/1562 [==============================] - 316s 203ms/step - loss: 0.3820 - accu
racy: 0.8692 - precision: 0.9040 - recall: 0.8375 - val_loss: 0.3955 - val_accurac
y: 0.8707 - val_precision: 0.9020 - val_recall: 0.8456
Epoch 47/50
1562/1562 [==============================] - 335s 214ms/step - loss: 0.3776 - accu
racy: 0.8702 - precision: 0.9059 - recall: 0.8389 - val_loss: 0.3845 - val_accurac
y: 0.8708 - val_precision: 0.9013 - val_recall: 0.8470
Epoch 48/50
1562/1562 [==============================] - 347s 222ms/step - loss: 0.3664 - accu
racy: 0.8722 - precision: 0.9064 - recall: 0.8422 - val_loss: 0.3815 - val_accurac
y: 0.8758 - val_precision: 0.9054 - val_recall: 0.8524
```

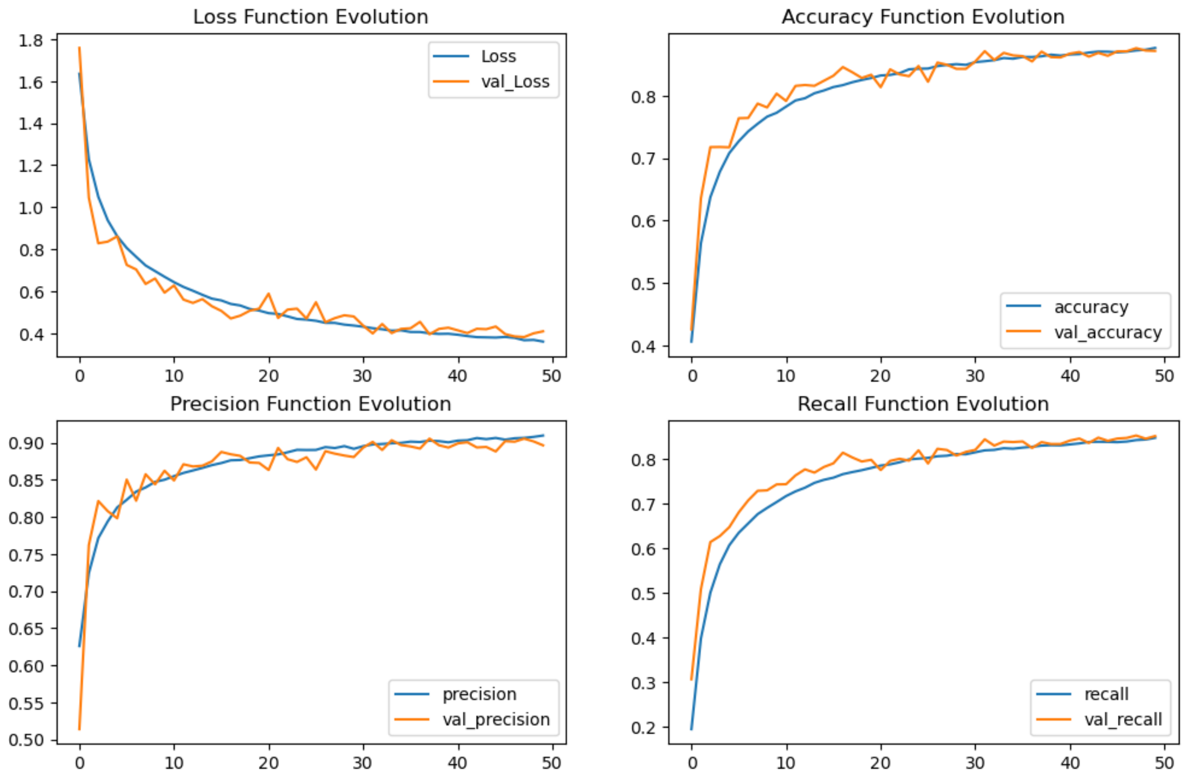
```
Epoch 49/50
1562/1562 [==============================] - 338s 216ms/step - loss: 0.3679 - accu
racy: 0.8733 - precision: 0.9077 - recall: 0.8434 - val_loss: 0.3991 - val_accurac
y: 0.8720 - val_precision: 0.9017 - val_recall: 0.8453
Epoch 50/50
1562/1562 [==============================] - 316s 202ms/step - loss: 0.3600 - accu
racy: 0.8762 - precision: 0.9096 - recall: 0.8473 - val_loss: 0.4087 - val_accurac
y: 0.8714 - val_precision: 0.8964 - val_recall: 0.8512
```

In [14]:

```python
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()
```

Out[14]:

```
<matplotlib.legend.Legend at 0x1203a724d10>
```

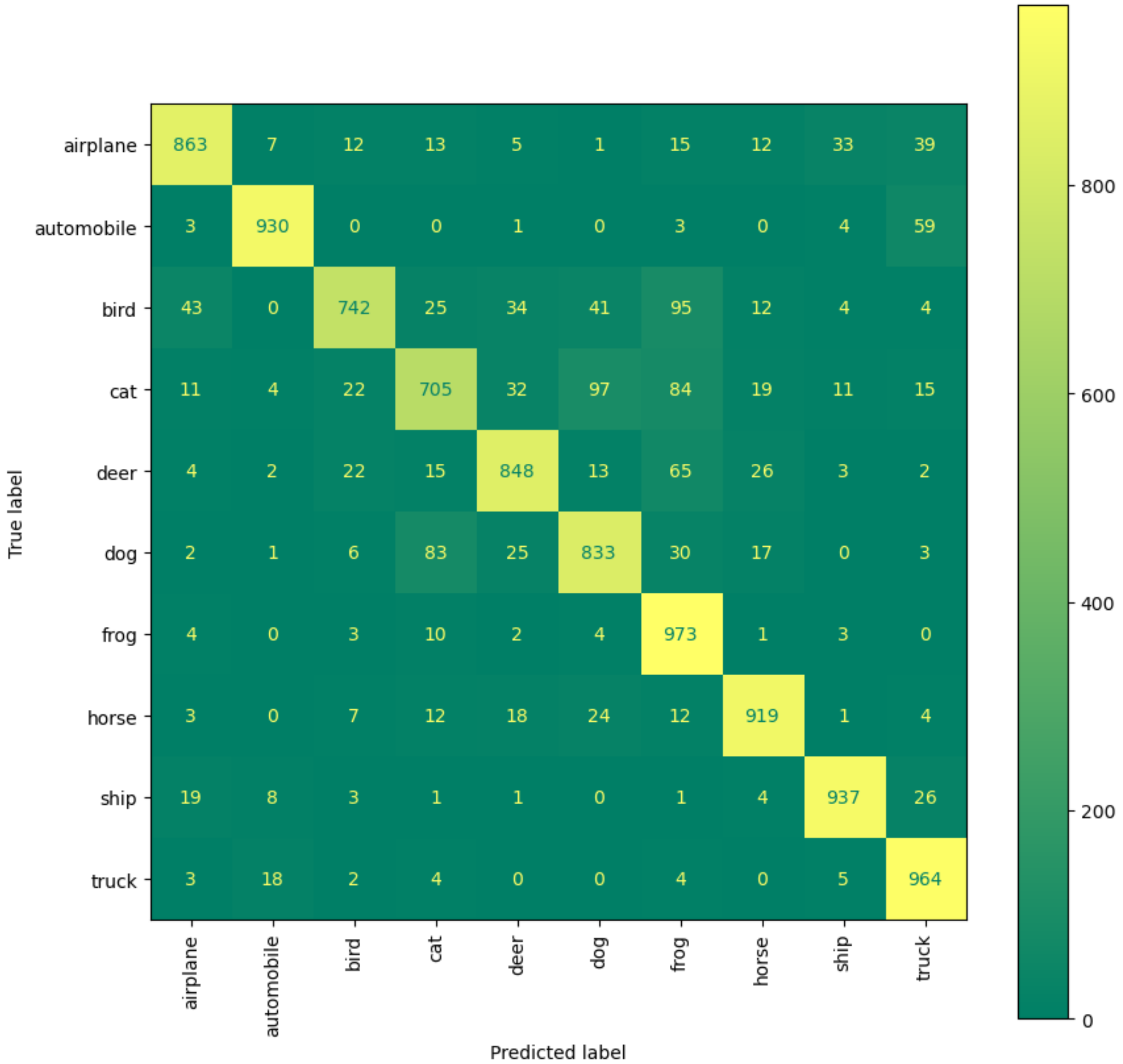


In [15]:

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

```python
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 [==============================] - 15s 46ms/step - loss: 0.4087 - accurac
y: 0.8714 - precision: 0.8964 - recall: 0.8512
Test Accuracy : 87.14%
313/313 [==============================] - 14s 42ms/step
```



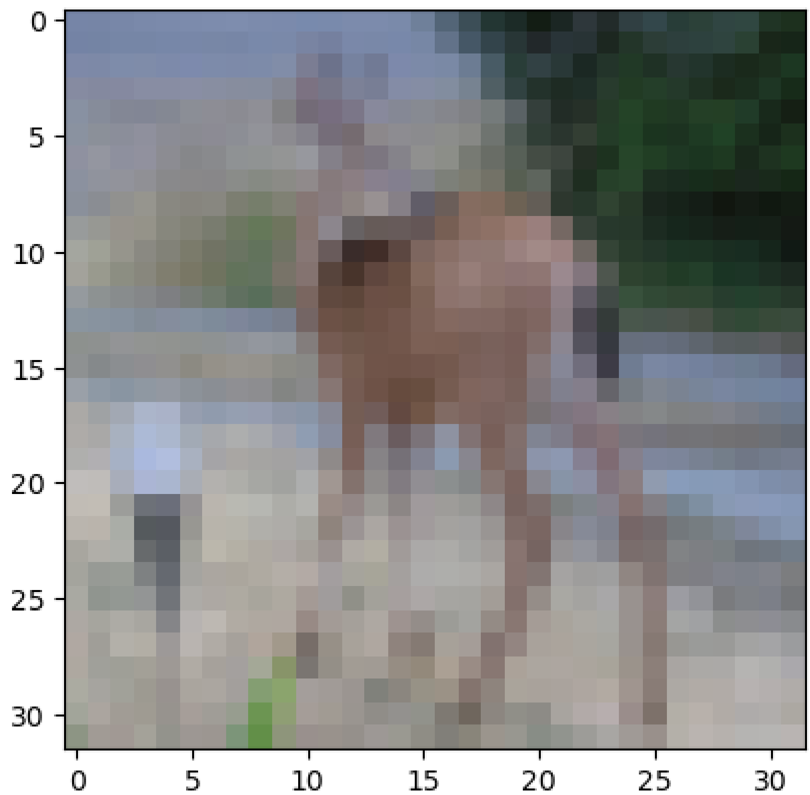
In [16]:

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

```
              precision    recall  f1-score   support

           0       0.90      0.86      0.88      1000
           1       0.96      0.93      0.94      1000
           2       0.91      0.74      0.82      1000
           3       0.81      0.70      0.75      1000
           4       0.88      0.85      0.86      1000
           5       0.82      0.83      0.83      1000
           6       0.76      0.97      0.85      1000
           7       0.91      0.92      0.91      1000
           8       0.94      0.94      0.94      1000
           9       0.86      0.96      0.91      1000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

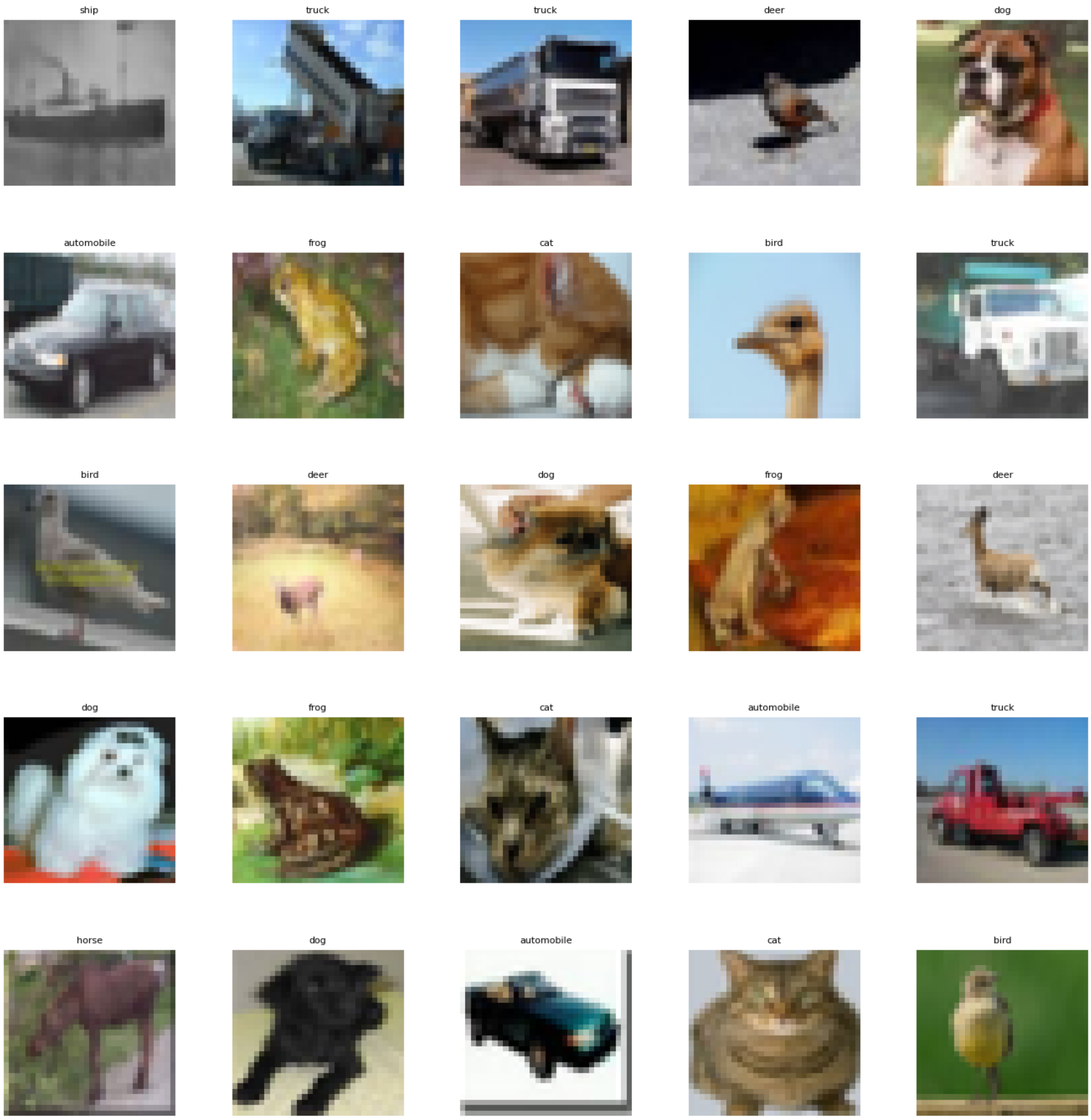
In [21]:
```
my_image = X_test[100]
plt.imshow(my_image)

# 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 61ms/step
The model predict that image 100 is 4
```



In [22]:
```
# 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)
```



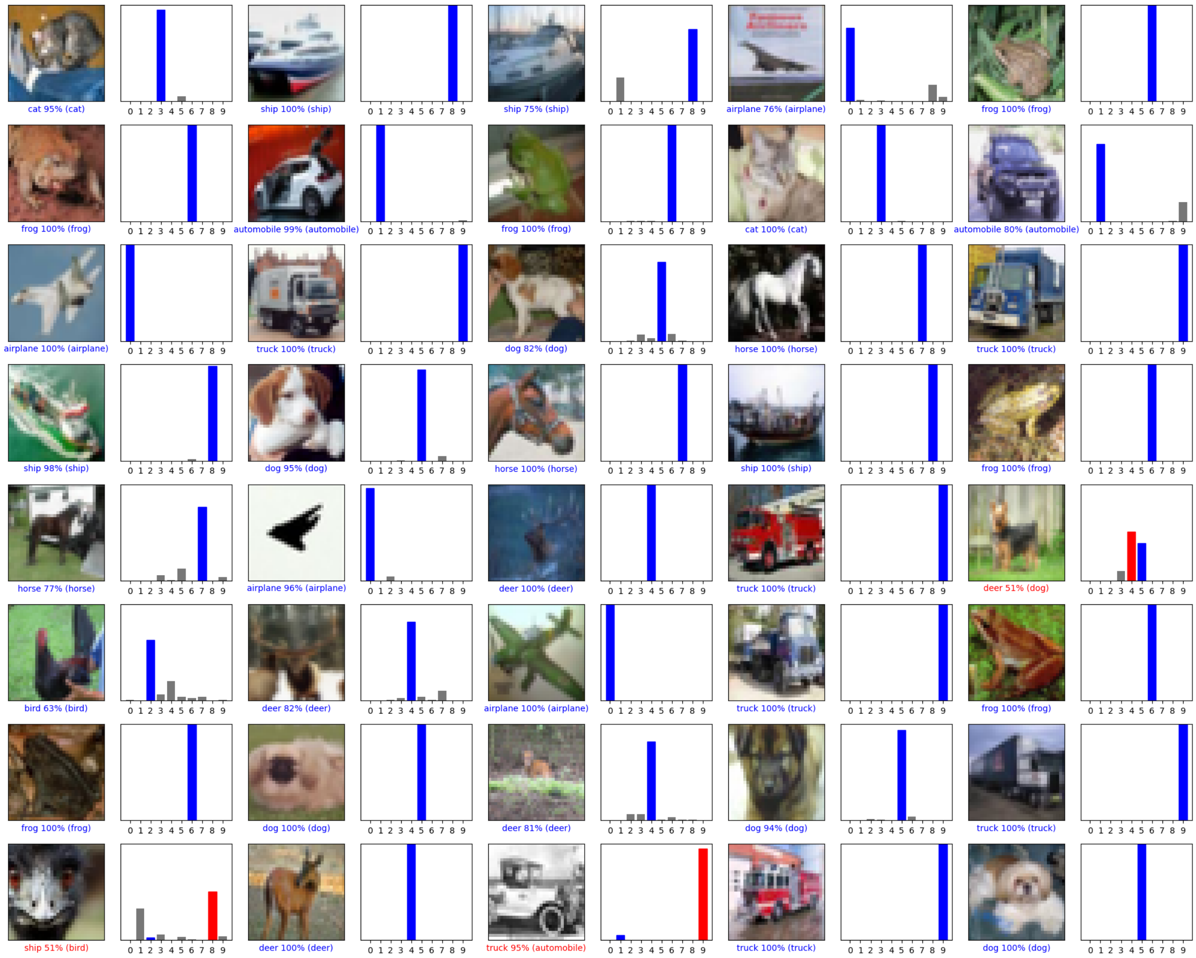
In [23]:

```
def plot_image(i, predictions_array, true_label, img):
    predictions_array, true_label, img = predictions_array, true_label[i], img[i]
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])

    plt.imshow(img, cmap=plt.cm.binary)

    predicted_label = np.argmax(predictions_array)
    if predicted_label == true_label:
        color = 'blue'
    else:
        color = 'red'

    plt.xlabel(f"{labels[int(predicted_label)]} {100*np.max(predictions_array):2.0f
               color=color)

def plot_value_array(i, predictions_array, true_label):
    predictions_array, true_label = predictions_array, int(true_label[i])
    plt.grid(False)
    plt.xticks(range(10))
    plt.yticks([])
    thisplot = plt.bar(range(10), predictions_array, color="#777777")
    plt.ylim([0, 1])
    predicted_label = np.argmax(predictions_array)

    thisplot[predicted_label].set_color('red')
    thisplot[true_label].set_color('blue')
```

In [24]:

```
predictions = model.predict(X_test)

# Plot the first X test images, their predicted labels, and the true labels.
# Color correct predictions in blue and incorrect predictions in red.
num_rows = 8
num_cols = 5
num_images = num_rows * num_cols
plt.figure(figsize=(2 * 2 * num_cols, 2 * num_rows))
for i in range(num_images):
    plt.subplot(num_rows, 2 * num_cols, 2 * i + 1)
    plot_image(i, predictions[i], y_test, X_test)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions[i], y_test)
plt.tight_layout()
plt.show()
```

```
313/313 [==============================] - 14s 44ms/step
```



In [26]:

```
from keras.applications.densenet import DenseNet121
from keras.layers import Dense
from keras.models import Sequential

model = Sequential()
base_model = DenseNet121(input_shape=(32, 32, 3), include_top=False, weights='image
model.add(base_model)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy

r = model.fit(train_generator,
              epochs=10,
              steps_per_epoch=steps_per_epoch,
              validation_data=(X_test, y_cat_test),
#               callbacks=[early_stop],
             )
```

```
Epoch 1/10
1562/1562 [==============================] - 899s 547ms/step - loss: 1.4403 - accu
racy: 0.5159 - val_loss: 1.9765 - val_accuracy: 0.4508
Epoch 2/10
1562/1562 [==============================] - 11397s 7s/step - loss: 1.1748 - accur
acy: 0.6064 - val_loss: 0.9715 - val_accuracy: 0.6597
Epoch 3/10
1562/1562 [==============================] - 1031s 660ms/step - loss: 1.0042 - acc
uracy: 0.6640 - val_loss: 0.9630 - val_accuracy: 0.6686
Epoch 4/10
1562/1562 [==============================] - 1160s 742ms/step - loss: 0.8990 - acc
uracy: 0.6968 - val_loss: 1.3524 - val_accuracy: 0.5796
Epoch 5/10
1562/1562 [==============================] - 1083s 693ms/step - loss: 1.0670 - acc
uracy: 0.6435 - val_loss: 1.1454 - val_accuracy: 0.5905
Epoch 6/10
1562/1562 [==============================] - 1084s 694ms/step - loss: 1.0414 - acc
uracy: 0.6476 - val_loss: 1.0944 - val_accuracy: 0.6206
Epoch 7/10
1562/1562 [==============================] - 1016s 651ms/step - loss: 0.9275 - acc
uracy: 0.6830 - val_loss: 0.8930 - val_accuracy: 0.6917
Epoch 8/10
1562/1562 [==============================] - 1073s 687ms/step - loss: 0.7666 - acc
uracy: 0.7411 - val_loss: 0.7846 - val_accuracy: 0.7616
Epoch 9/10
1562/1562 [==============================] - 1116s 715ms/step - loss: 0.9112 - acc
uracy: 0.7021 - val_loss: 1.1693 - val_accuracy: 0.7348
Epoch 10/10
1204/1562 [======================>.......] - ETA: 3:44 - loss: 0.6925 - accuracy:
0.7642
```

In [28]:

```
from tensorflow.keras.models import load_model

model.save('cnn_20_epochs.h5')
```

In [ ]: