ASSIGNMENT

Name: - Aryan Omkar Ashar
Registration Number: - 19BAI10094

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

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNor
malization
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
```

DOWNLOADING THE DATASET

In [2]:

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

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 3s 0us/step
```

In [3]:

```
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}")
```

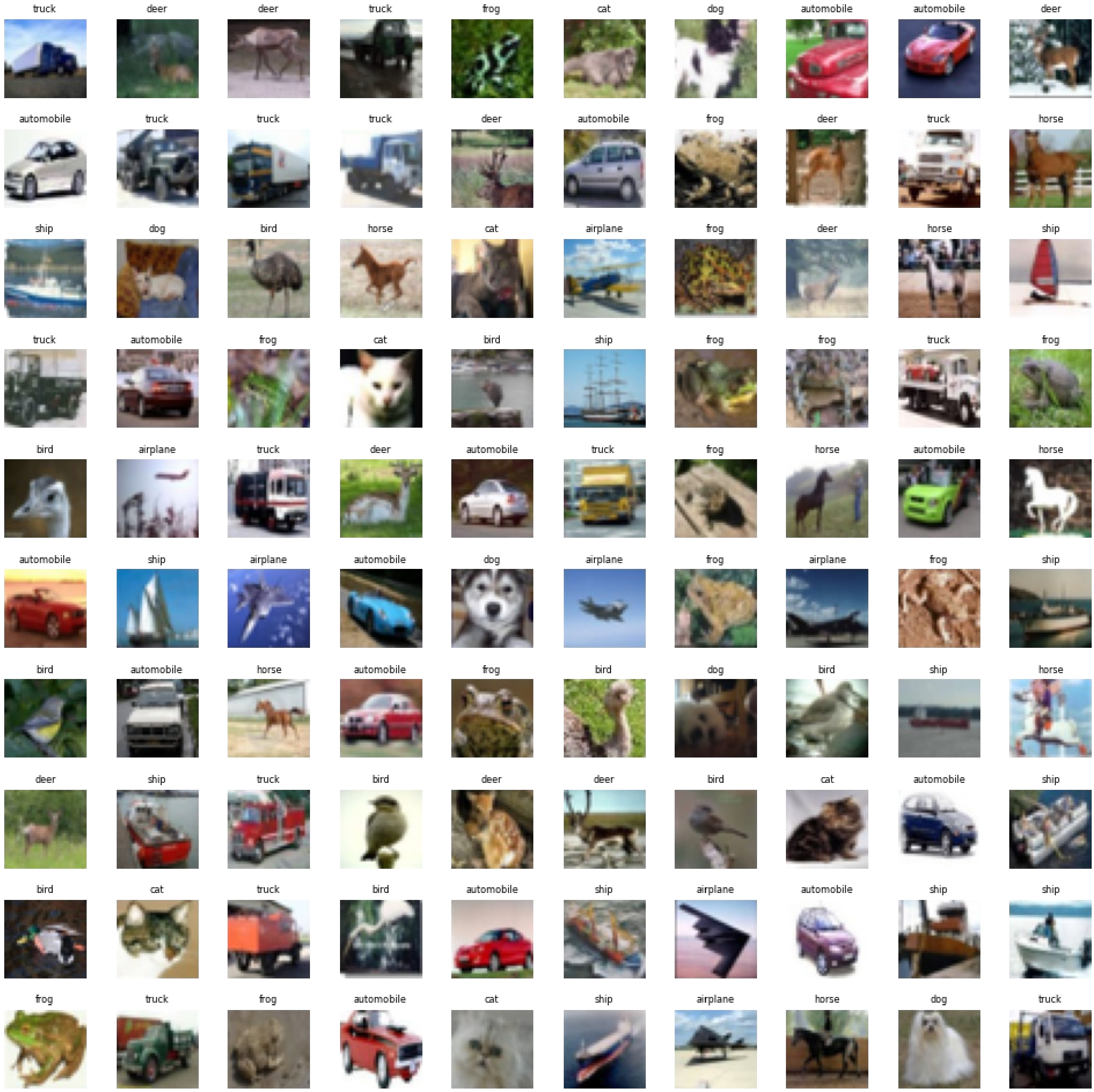
```
X_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
X_test shape: (10000, 32, 32, 3)
y_test shape: (10000, 1)
```

Data Visualization

In [4]:

```
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship
', 'truck']
W_grid = 10
L_grid = 10
fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))
axes = axes.ravel()
n_train = len(X_train)
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
    index = np.random.randint(0, n_train)
    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)
```

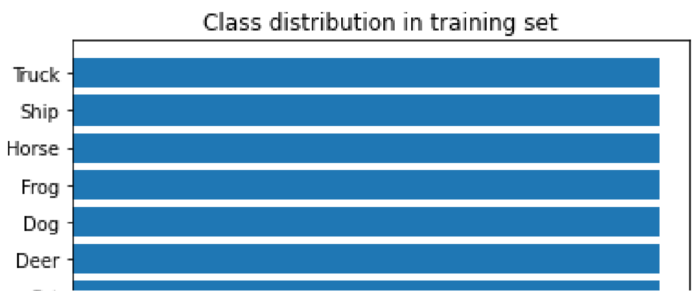


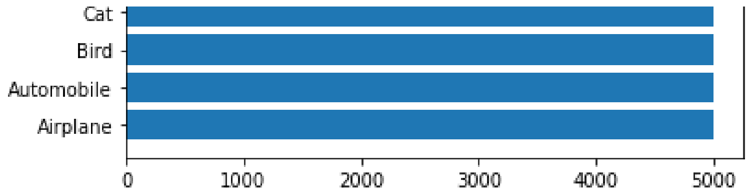
In [5]:

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

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

Out[5]:

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



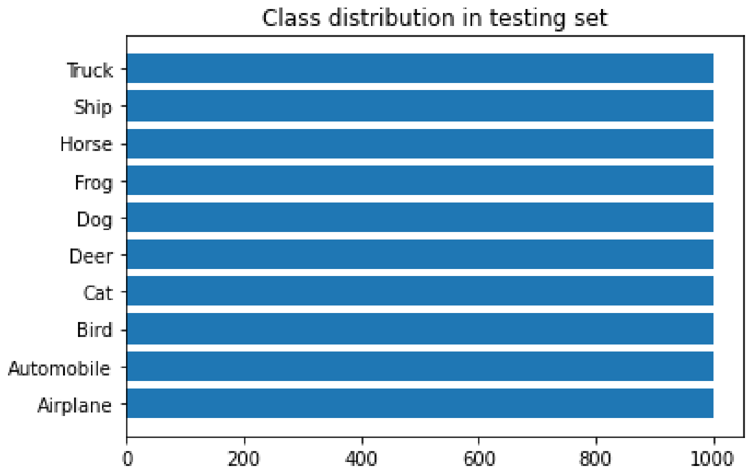


In [6]:

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

Out[6]:

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



The class are equally distributed

Data Preprocessing

In [7]:

```
# 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 [8]:

```
y_cat_train
```

Out[8]:

```
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)
```

BUILDING THE MODEL

In [9]:

```
model = Sequential()

# Convolutional Layer
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='re
```

```
lu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='re
lu', padding='same'))
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=(3, 3), input_shape=(32, 32, 3), activation='re
lu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='re
lu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='r
elu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='r
elu', padding='same'))
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 [10]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896       
_________________________________________________________________
batch_normalization (BatchNo (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 64)          0         
```

```
conv2d_4 (Conv2D)            (None, 8, 8, 128)         73856
_________________________________________________________________
batch_normalization_4 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         147584
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
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: 552,362
Trainable params: 551,466
Non-trainable params: 896
_________________________________________________________________
```

Early Stopping

In [11]:

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

Data Augmentations and Model Training

In [12]:

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

In [13]:

```
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 [==============================] - 40s 20ms/step - loss: 1.9058 - accuracy: 0.3
316 - precision: 0.5119 - recall: 0.1173 - val_loss: 1.3044 - val_accuracy: 0.5370 - val_
precision: 0.7033 - val_recall: 0.3861
Epoch 2/50
1562/1562 [==============================] - 31s 20ms/step - loss: 1.2966 - accuracy: 0.5
407 - precision: 0.7096 - recall: 0.3617 - val_loss: 0.9729 - val_accuracy: 0.6640 - val_
precision: 0.7745 - val_recall: 0.5560
Epoch 3/50
1562/1562 [==============================] - 31s 20ms/step - loss: 1.0885 - accuracy: 0.6
217 - precision: 0.7622 - recall: 0.4823 - val_loss: 0.9059 - val_accuracy: 0.6940 - val_
precision: 0.8012 - val_recall: 0.5930
Epoch 4/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.9584 - accuracy: 0.6
708 - precision: 0.7929 - recall: 0.5524 - val_loss: 0.9401 - val_accuracy: 0.6887 - val_
precision: 0.7832 - val_recall: 0.6193
Epoch 5/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.8733 - accuracy: 0.7
```

```
026 - precision: 0.8059 - recall: 0.5978 - val_loss: 1.0669 - val_accuracy: 0.6685 - val_
precision: 0.7446 - val_recall: 0.6036
Epoch 6/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.8114 - accuracy: 0.7
241 - precision: 0.8185 - recall: 0.6331 - val_loss: 0.8686 - val_accuracy: 0.7099 - val_
precision: 0.7890 - val_recall: 0.6427
Epoch 7/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.7767 - accuracy: 0.7
379 - precision: 0.8264 - recall: 0.6510 - val_loss: 0.7486 - val_accuracy: 0.7590 - val_
precision: 0.8239 - val_recall: 0.6958
Epoch 8/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.7301 - accuracy: 0.7
546 - precision: 0.8379 - recall: 0.6714 - val_loss: 0.6864 - val_accuracy: 0.7717 - val_
precision: 0.8414 - val_recall: 0.7152
Epoch 9/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.6939 - accuracy: 0.7
623 - precision: 0.8435 - recall: 0.6887 - val_loss: 0.6023 - val_accuracy: 0.7987 - val_
precision: 0.8577 - val_recall: 0.7477
Epoch 10/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.6620 - accuracy: 0.7
785 - precision: 0.8550 - recall: 0.7081 - val_loss: 0.6047 - val_accuracy: 0.7996 - val_
precision: 0.8687 - val_recall: 0.7257
Epoch 11/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.6272 - accuracy: 0.7
892 - precision: 0.8599 - recall: 0.7249 - val_loss: 0.5915 - val_accuracy: 0.8092 - val_
precision: 0.8609 - val_recall: 0.7620
Epoch 12/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.6193 - accuracy: 0.7
889 - precision: 0.8570 - recall: 0.7253 - val_loss: 0.5993 - val_accuracy: 0.8057 - val_
precision: 0.8571 - val_recall: 0.7566
Epoch 13/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.6024 - accuracy: 0.7
982 - precision: 0.8646 - recall: 0.7379 - val_loss: 0.5442 - val_accuracy: 0.8226 - val_
precision: 0.8766 - val_recall: 0.7723
Epoch 14/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5830 - accuracy: 0.8
039 - precision: 0.8681 - recall: 0.7441 - val_loss: 0.5739 - val_accuracy: 0.8097 - val_
precision: 0.8624 - val_recall: 0.7630
Epoch 15/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5641 - accuracy: 0.8
096 - precision: 0.8719 - recall: 0.7556 - val_loss: 0.6131 - val_accuracy: 0.8010 - val_
precision: 0.8531 - val_recall: 0.7588
Epoch 16/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5441 - accuracy: 0.8
156 - precision: 0.8736 - recall: 0.7624 - val_loss: 0.4915 - val_accuracy: 0.8397 - val_
precision: 0.8824 - val_recall: 0.8026
Epoch 17/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5356 - accuracy: 0.8
186 - precision: 0.8760 - recall: 0.7642 - val_loss: 0.4851 - val_accuracy: 0.8414 - val_
precision: 0.8814 - val_recall: 0.8029
Epoch 18/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5311 - accuracy: 0.8
184 - precision: 0.8757 - recall: 0.7671 - val_loss: 0.4702 - val_accuracy: 0.8462 - val_
precision: 0.8902 - val_recall: 0.8039
Epoch 19/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.5084 - accuracy: 0.8
254 - precision: 0.8791 - recall: 0.7796 - val_loss: 0.4905 - val_accuracy: 0.8387 - val_
precision: 0.8781 - val_recall: 0.8001
Epoch 20/50
1562/1562 [==============================] - 32s 21ms/step - loss: 0.4997 - accuracy: 0.8
270 - precision: 0.8797 - recall: 0.7823 - val_loss: 0.5039 - val_accuracy: 0.8325 - val_
precision: 0.8726 - val_recall: 0.7985
Epoch 21/50
1562/1562 [==============================] - 33s 21ms/step - loss: 0.4969 - accuracy: 0.8
310 - precision: 0.8831 - recall: 0.7856 - val_loss: 0.5904 - val_accuracy: 0.8082 - val_
precision: 0.8554 - val_recall: 0.7761
Epoch 22/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.4795 - accuracy: 0.8
345 - precision: 0.8825 - recall: 0.7912 - val_loss: 0.4307 - val_accuracy: 0.8564 - val_
precision: 0.8985 - val_recall: 0.8234
Epoch 23/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.4680 - accuracy: 0.8
```

```
404 - precision: 0.8889 - recall: 0.7969 - val_loss: 0.4775 - val_accuracy: 0.8481 - val_
precision: 0.8863 - val_recall: 0.8117
Epoch 24/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.4734 - accuracy: 0.8
403 - precision: 0.8890 - recall: 0.7977 - val_loss: 0.4535 - val_accuracy: 0.8480 - val_
precision: 0.8858 - val_recall: 0.8144
Epoch 25/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.4548 - accuracy: 0.8
441 - precision: 0.8924 - recall: 0.8036 - val_loss: 0.4601 - val_accuracy: 0.8521 - val_
precision: 0.8851 - val_recall: 0.8187
Epoch 26/50
1562/1562 [==============================] - 31s 20ms/step - loss: 0.4588 - accuracy: 0.8
441 - precision: 0.8905 - recall: 0.8031 - val_loss: 0.4430 - val_accuracy: 0.8516 - val_
precision: 0.8906 - val_recall: 0.8156
Epoch 27/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.4534 - accuracy: 0.8
456 - precision: 0.8916 - recall: 0.8055 - val_loss: 0.4414 - val_accuracy: 0.8556 - val_
precision: 0.8947 - val_recall: 0.8211
Epoch 28/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.4505 - accuracy: 0.8
497 - precision: 0.8908 - recall: 0.8082 - val_loss: 0.4372 - val_accuracy: 0.8554 - val_
precision: 0.8902 - val_recall: 0.8305
Epoch 29/50
1562/1562 [==============================] - 35s 22ms/step - loss: 0.4341 - accuracy: 0.8
528 - precision: 0.8952 - recall: 0.8167 - val_loss: 0.4099 - val_accuracy: 0.8634 - val_
precision: 0.8940 - val_recall: 0.8363
Epoch 30/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.4221 - accuracy: 0.8
576 - precision: 0.8995 - recall: 0.8200 - val_loss: 0.4433 - val_accuracy: 0.8519 - val_
precision: 0.8911 - val_recall: 0.8185
Epoch 31/50
1562/1562 [==============================] - 35s 22ms/step - loss: 0.4292 - accuracy: 0.8
531 - precision: 0.8949 - recall: 0.8161 - val_loss: 0.4755 - val_accuracy: 0.8442 - val_
precision: 0.8779 - val_recall: 0.8156
Epoch 32/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.4178 - accuracy: 0.8
574 - precision: 0.8978 - recall: 0.8209 - val_loss: 0.4062 - val_accuracy: 0.8637 - val_
precision: 0.8967 - val_recall: 0.8355
Epoch 33/50
1562/1562 [==============================] - 36s 23ms/step - loss: 0.4177 - accuracy: 0.8
563 - precision: 0.8987 - recall: 0.8228 - val_loss: 0.4109 - val_accuracy: 0.8617 - val_
precision: 0.8983 - val_recall: 0.8269
Epoch 34/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.4112 - accuracy: 0.8
588 - precision: 0.8978 - recall: 0.8223 - val_loss: 0.4021 - val_accuracy: 0.8683 - val_
precision: 0.8951 - val_recall: 0.8432
Epoch 35/50
1562/1562 [==============================] - 36s 23ms/step - loss: 0.4034 - accuracy: 0.8
622 - precision: 0.9011 - recall: 0.8274 - val_loss: 0.3975 - val_accuracy: 0.8700 - val_
precision: 0.8986 - val_recall: 0.8450
Epoch 36/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.3974 - accuracy: 0.8
649 - precision: 0.9007 - recall: 0.8313 - val_loss: 0.4161 - val_accuracy: 0.8640 - val_
precision: 0.8939 - val_recall: 0.8347
Epoch 37/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.3964 - accuracy: 0.8
654 - precision: 0.9020 - recall: 0.8316 - val_loss: 0.4171 - val_accuracy: 0.8598 - val_
precision: 0.8952 - val_recall: 0.8333
Epoch 38/50
1562/1562 [==============================] - 34s 22ms/step - loss: 0.3894 - accuracy: 0.8
676 - precision: 0.9029 - recall: 0.8365 - val_loss: 0.3914 - val_accuracy: 0.8723 - val_
precision: 0.9048 - val_recall: 0.8468
Epoch 39/50
1562/1562 [==============================] - 31s 20ms/step - loss: 0.3907 - accuracy: 0.8
645 - precision: 0.9020 - recall: 0.8312 - val_loss: 0.4384 - val_accuracy: 0.8659 - val_
precision: 0.8925 - val_recall: 0.8397
Epoch 40/50
1562/1562 [==============================] - 36s 23ms/step - loss: 0.3889 - accuracy: 0.8
656 - precision: 0.9027 - recall: 0.8332 - val_loss: 0.4334 - val_accuracy: 0.8619 - val_
precision: 0.8967 - val_recall: 0.8316
Epoch 41/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.3886 - accuracy: 0.8
```

```
661 - precision: 0.9023 - recall: 0.8336 - val_loss: 0.4122 - val_accuracy: 0.8633 - val_
precision: 0.8934 - val_recall: 0.8362
Epoch 42/50
1562/1562 [==============================] - 37s 23ms/step - loss: 0.3806 - accuracy: 0.8
689 - precision: 0.9049 - recall: 0.8371 - val_loss: 0.4034 - val_accuracy: 0.8689 - val_
precision: 0.8991 - val_recall: 0.8443
Epoch 43/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.3801 - accuracy: 0.8
700 - precision: 0.9053 - recall: 0.8372 - val_loss: 0.3901 - val_accuracy: 0.8740 - val_
precision: 0.9030 - val_recall: 0.8504
Epoch 44/50
1562/1562 [==============================] - 37s 24ms/step - loss: 0.3795 - accuracy: 0.8
700 - precision: 0.9045 - recall: 0.8391 - val_loss: 0.4225 - val_accuracy: 0.8619 - val_
precision: 0.8942 - val_recall: 0.8421
Epoch 45/50
1562/1562 [==============================] - 32s 20ms/step - loss: 0.3709 - accuracy: 0.8
753 - precision: 0.9078 - recall: 0.8456 - val_loss: 0.4008 - val_accuracy: 0.8679 - val_
precision: 0.8966 - val_recall: 0.8468
Epoch 46/50
1562/1562 [==============================] - 37s 24ms/step - loss: 0.3662 - accuracy: 0.8
751 - precision: 0.9103 - recall: 0.8459 - val_loss: 0.3810 - val_accuracy: 0.8723 - val_
precision: 0.9038 - val_recall: 0.8472
Epoch 47/50
1562/1562 [==============================] - 31s 20ms/step - loss: 0.3695 - accuracy: 0.8
718 - precision: 0.9048 - recall: 0.8428 - val_loss: 0.3763 - val_accuracy: 0.8804 - val_
precision: 0.9034 - val_recall: 0.8606
Epoch 48/50
1562/1562 [==============================] - 37s 24ms/step - loss: 0.3750 - accuracy: 0.8
710 - precision: 0.9062 - recall: 0.8416 - val_loss: 0.3982 - val_accuracy: 0.8713 - val_
precision: 0.9012 - val_recall: 0.8471
Epoch 49/50
1562/1562 [==============================] - 31s 20ms/step - loss: 0.3674 - accuracy: 0.8
744 - precision: 0.9081 - recall: 0.8451 - val_loss: 0.3881 - val_accuracy: 0.8754 - val_
precision: 0.9010 - val_recall: 0.8553
Epoch 50/50
1562/1562 [==============================] - 34s 21ms/step - loss: 0.3590 - accuracy: 0.8
748 - precision: 0.9087 - recall: 0.8460 - val_loss: 0.4086 - val_accuracy: 0.8699 - val_
precision: 0.8963 - val_recall: 0.8476
```

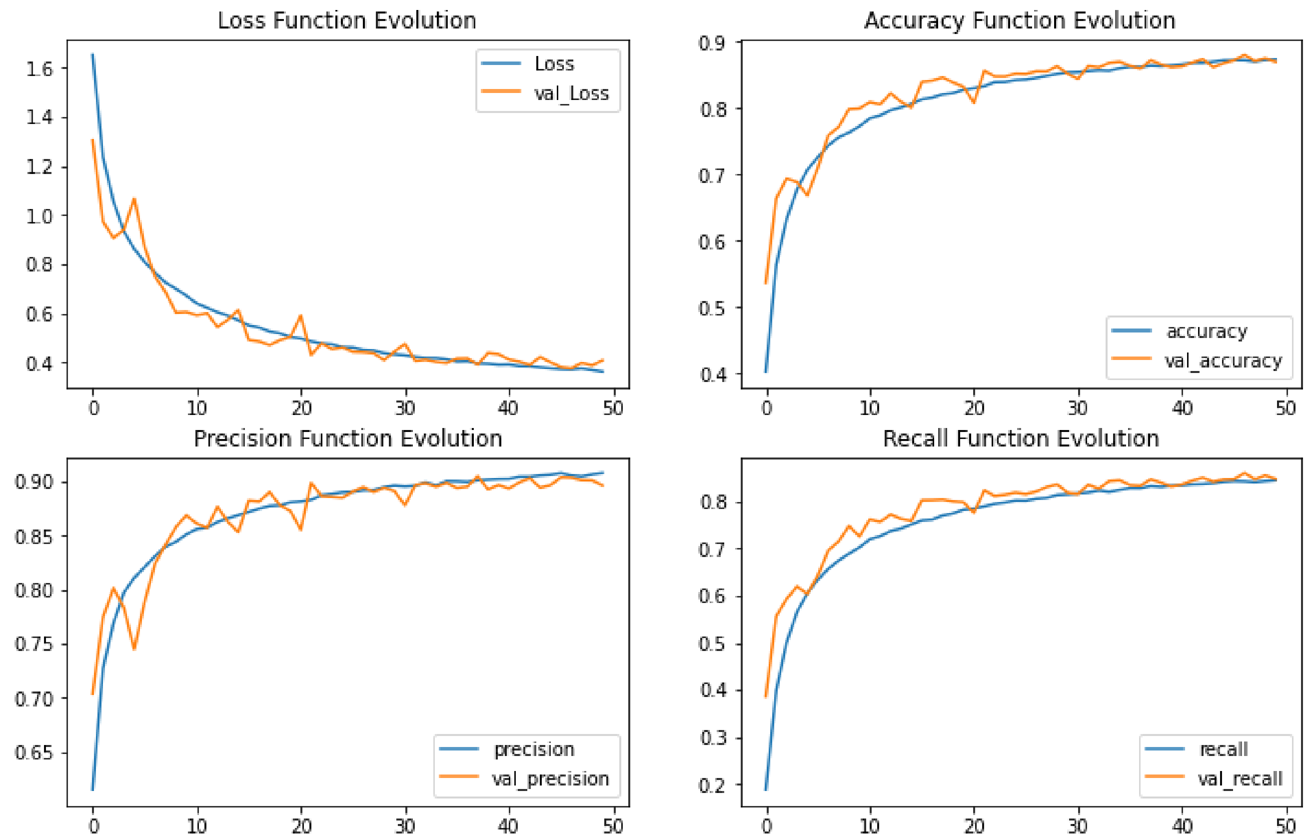
Model Evaluation

In [14]:

```
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 0x7f3660724ad0>
```



In [15]:

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

```
313/313 [==============================] - 1s 5ms/step - loss: 0.4086 - accuracy: 0.8699
- precision: 0.8963 - recall: 0.8476
Test Accuracy : 86.99%
```

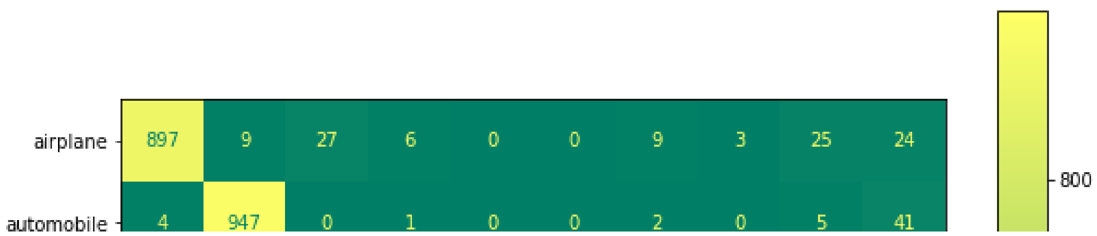
In [16]:

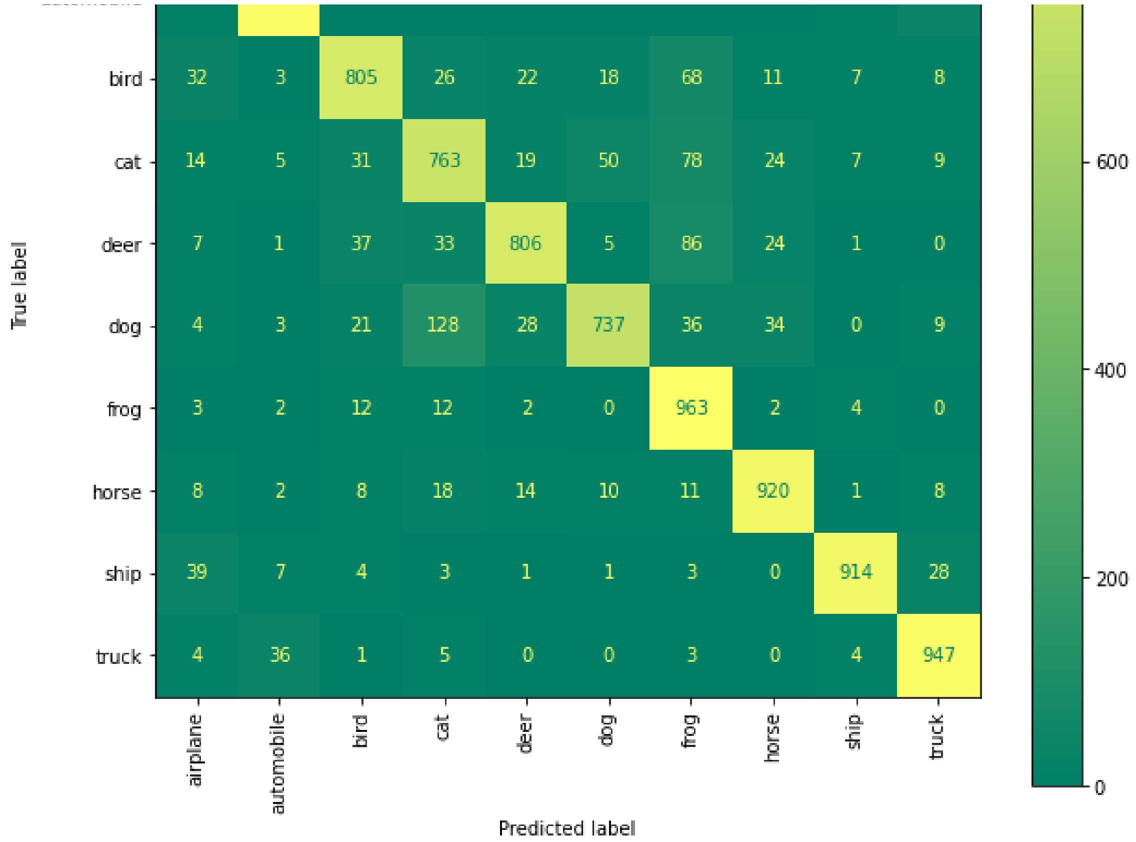
```
y_pred = model.predict_classes(X_test)

cm = confusion_matrix(y_test, y_pred)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/python/keras/engine/sequential.py:450:
UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01
. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi
-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.pre
dict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it
uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

In [17]:

```
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()
```





In [18]:

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

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      1000
           1       0.93      0.95      0.94      1000
           2       0.85      0.81      0.83      1000
           3       0.77      0.76      0.76      1000
           4       0.90      0.81      0.85      1000
           5       0.90      0.74      0.81      1000
           6       0.76      0.96      0.85      1000
           7       0.90      0.92      0.91      1000
           8       0.94      0.91      0.93      1000
           9       0.88      0.95      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
```

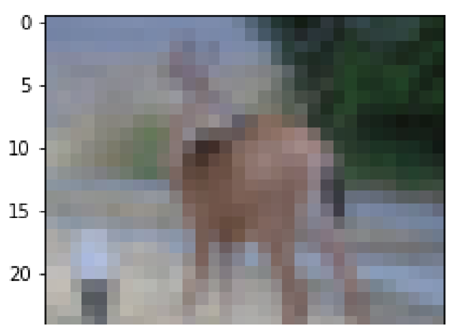
MODEL TESTING

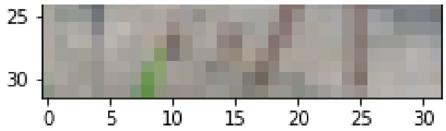
In [19]:

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

Out[19]:

```
<matplotlib.image.AxesImage at 0x7f3660454610>
```





In [20]:

```
#DEER
y_test[100]
```

Out[20]:

```
array([4], dtype=uint8)
```

In [21]:

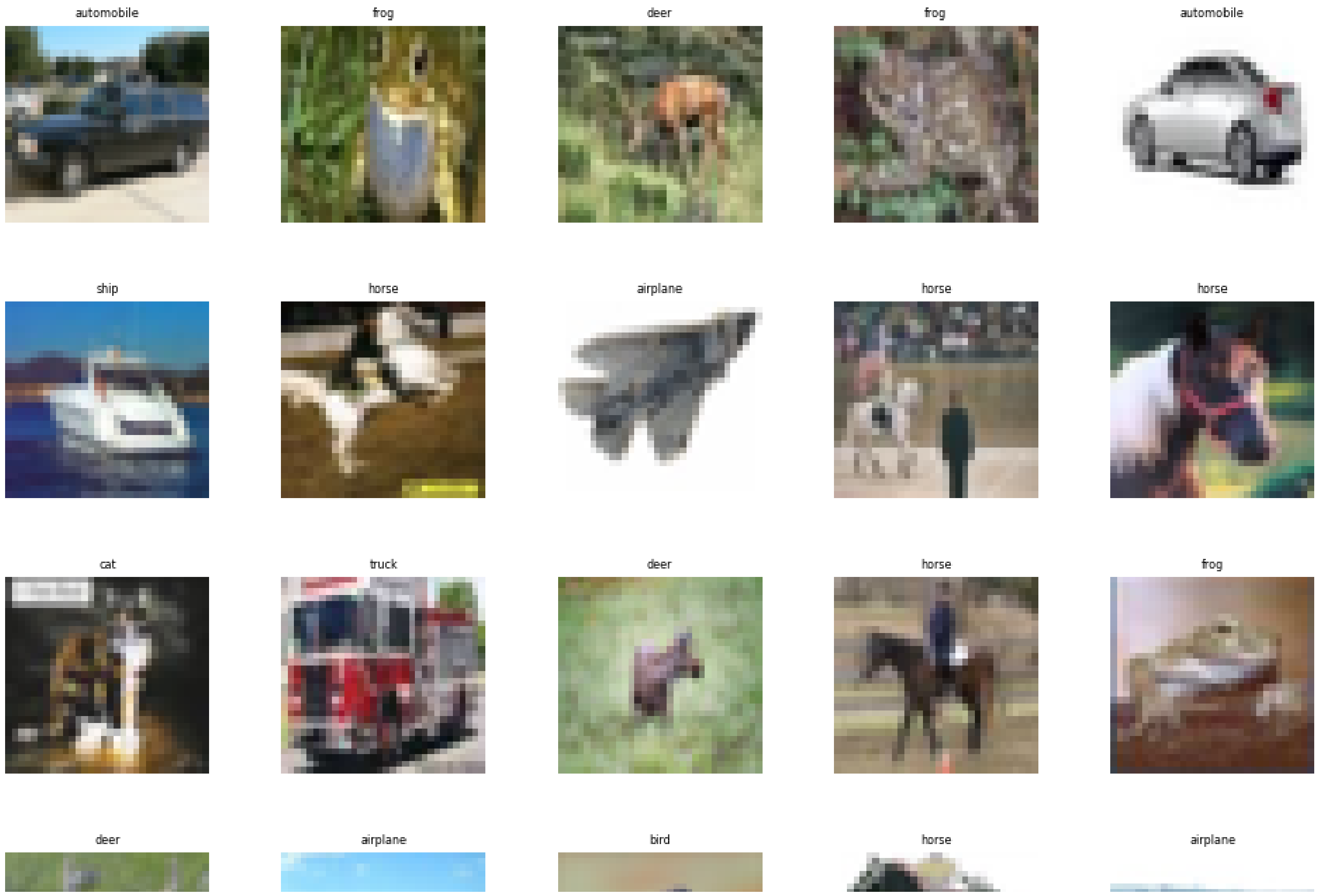
```
# correctly predicted as a Deer
model.predict_classes(my_image.reshape(1, 32, 32, 3))
```

Out[21]:

```
array([7])
```

In [22]:

```
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']
W_grid = 5
L_grid = 5
fig, axes = plt.subplots(L_grid, W_grid, figsize = (17,17))
axes = axes.ravel()
n_test = len(X_test)
for i in np.arange(0, W_grid * L_grid): # create evenly spaces variables
    index = np.random.randint(0, n_test)
    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]:

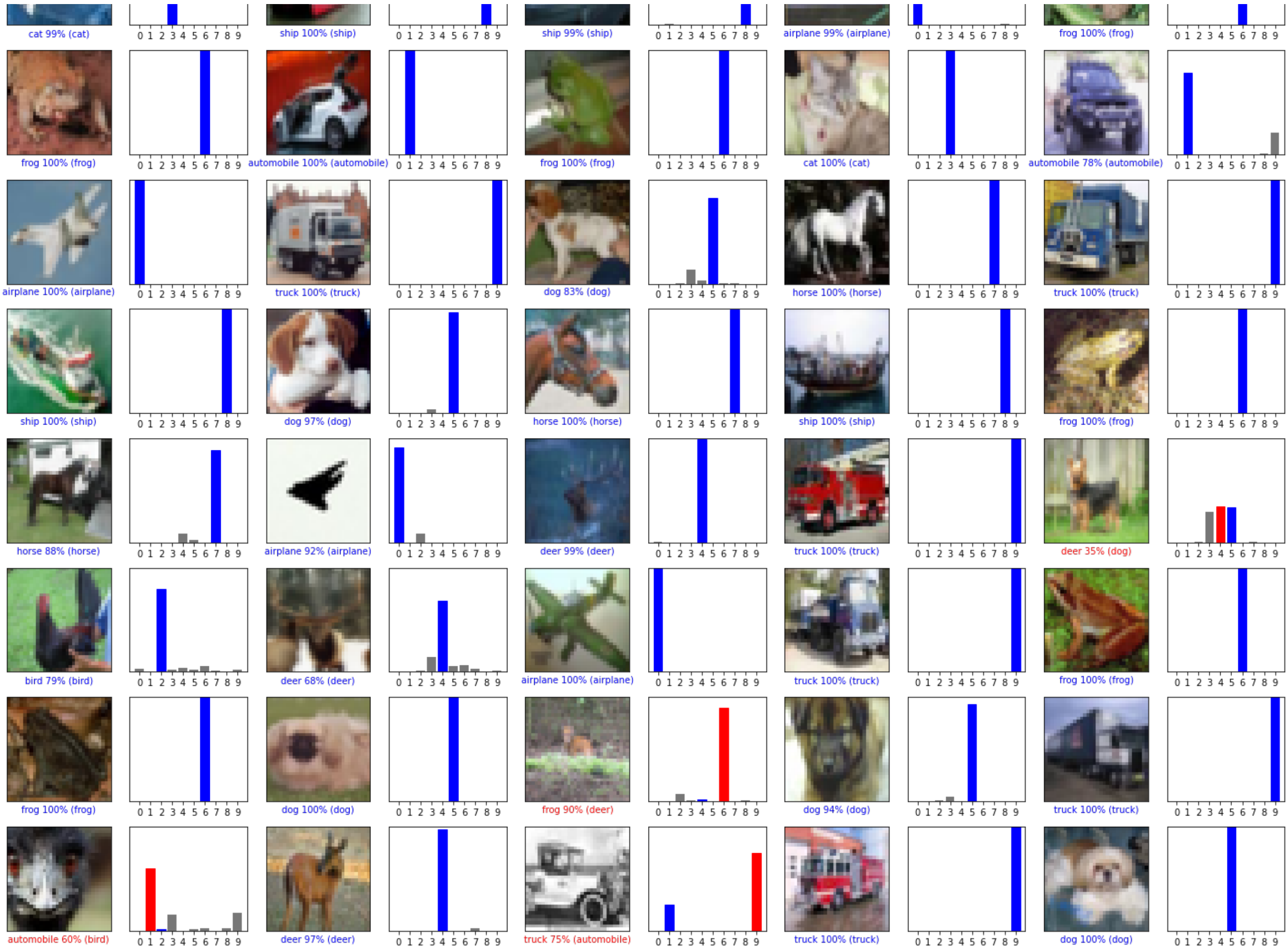
```
predictions = model.predict(X_test)
```

In [24]:

```
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}% ({labels[int(true_label)]})",
               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 [25]:

```
# 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()
```



Save the models

In [27]:

```
from tensorflow.keras.models import load_model

model.save('Assignment3.h5')
```