EXPLORING IMAGE CLASSIFICATION WITH THE CIFAR-10 DATASET

TEAM 4

1. SHUBHAM SINGH
2. A.AKHILA
3. CH.RAHUL
4. A.HARI HARAN

Bhavan's Vivekananda College
BSc Hons Data Science

DESCRIPTION:

The CIFAR-10 data consists of 60,000 32x32 color images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images in the official data. We have preserved the train/test split from the original dataset.

OBJECTIVE:

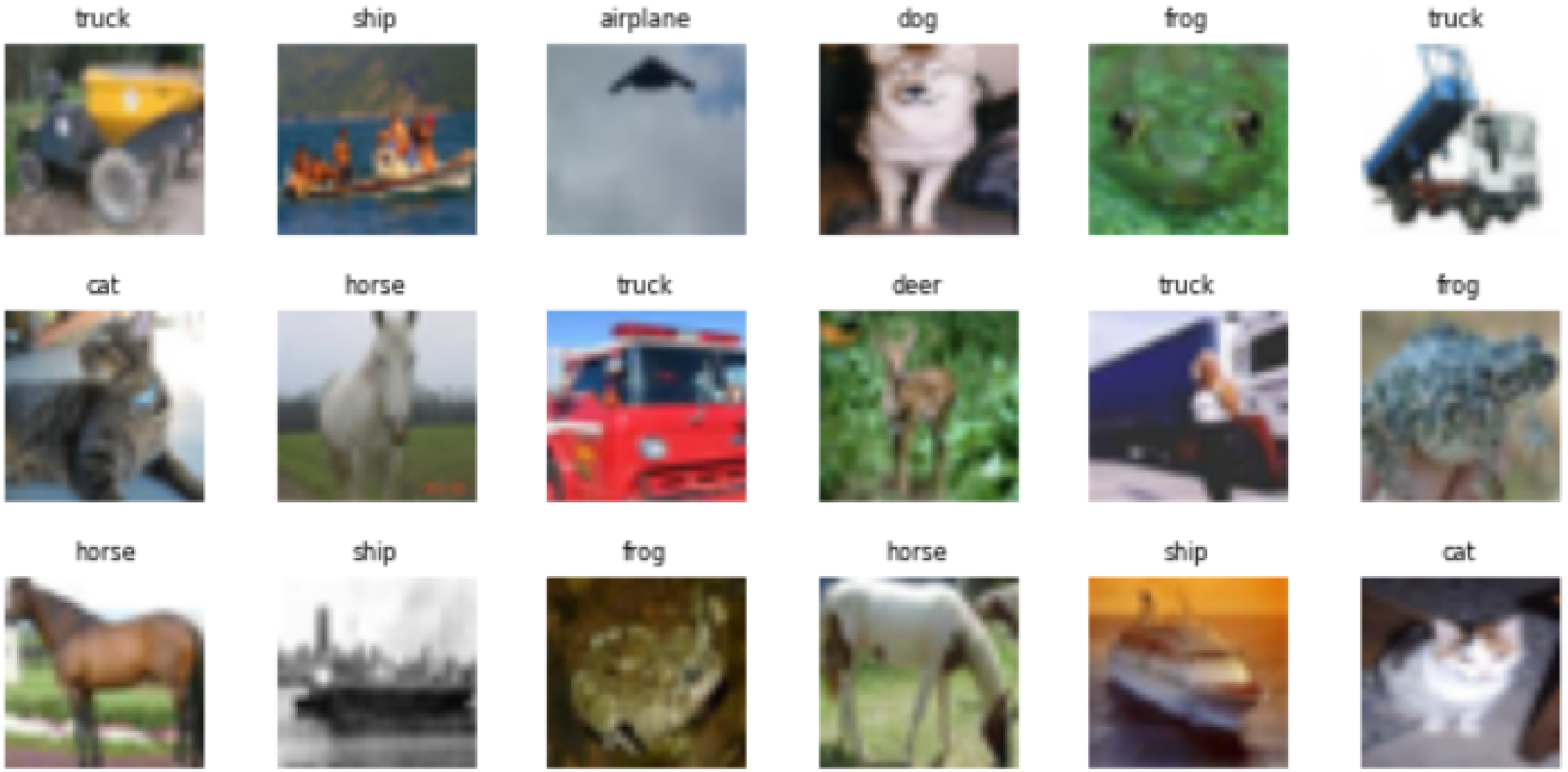
The objective is to determine the class to which the image belong to using CNN(Convulational Neural Network)

CLASSES:

airplane	cat	deer	
automobile	dog	bird	truck
ship	horse	frog	

Data Preparation:

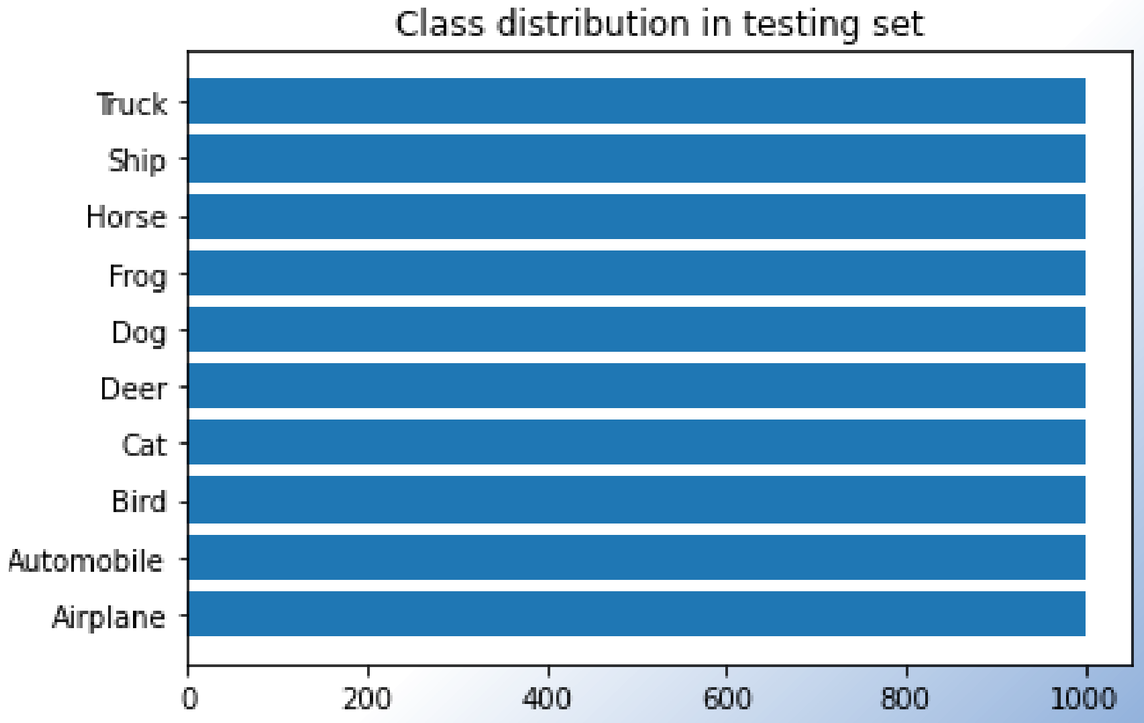
DATA VISUALATION



Shape of the image - 32x32 pixels

TRAINING DATA

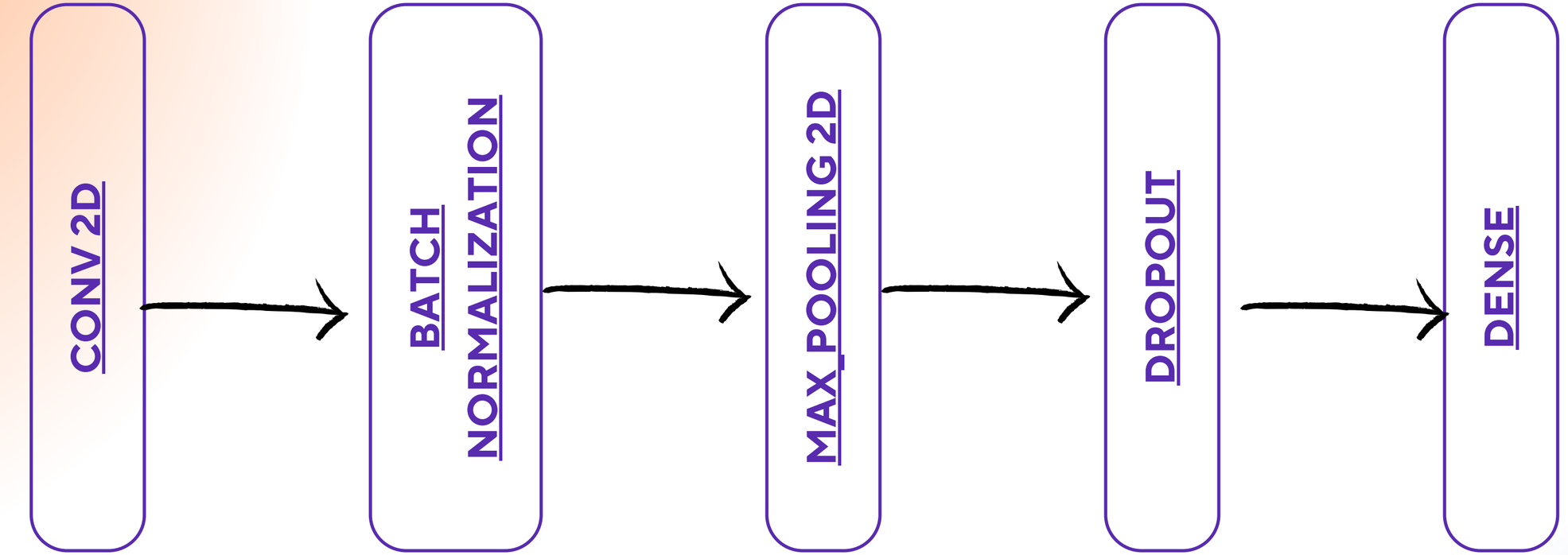
TESTING DATA



Parameters:

Optimizers	Adam	SGD	
Loss	binary cross entropy	categorical cross entropy	
Epochs	min- 20	max - 300	
Batch size	min-32	max-64	
Metrics	*Accuracy*	*Precision*	*Recall*

CNN Summary:



Total params	Trainable params	non trainable params	activation
552362	551466	896	relu,softmax

batch_size	Epoch	optimizer	Loss_function	Accuracy	Precision	Recall	Loss_val
Adam							
64	10	Adam	categorical_crossentropy	0.8999	0.9244	0.879	0.394
46	60	Adam	binary_crossentropy	0.9008	0.9205	0.8864	0.0523
64	50	Adam	binary_crossentropy	0.9176	0.9344	0.9026	0.3358
64	**300**	**adam**	**binary_crossentropy**	**0.9418**	**0.9515**	**0.9351**	**0.0328**
SGD							
40	90	SGD	binary_crossentropy	0.8057	0.8466	0.7744	0.0869
50	60	SGD	categorical_crossentropy	0.8396	0.8864	0.7996	0.4622
32	**60**	**SGD**	**categorical_crossentropy**	**0.8508**	**0.8913**	**0.8144**	**0.4323**

binary cross entropy

batch_size	Epoch	optimizer	Loss_function	Accuracy	Precision	Recall	Loss_val
64	60	Adam	binary_crossentropy	0.8924	0.9142	0.8763	0.0565
46	60	Adam	binary_crossentropy	0.9008	0.9205	0.8864	0.0523
64	50	Adam	binary_crossentropy	0.9176	0.9344	0.9026	0.3358
64	**300**	**adam**	**binary_crossentropy**	**0.9418**	**0.9515**	**0.9351**	**0.0328**

categorical cross entropy

40	40	adam	categorical_crossentropy	0.8963	0.9218	0.8739	0.3744
50	**60**	**Adam**	**categorical_crossentropy**	**0.8986**	**0.9232**	**0.878**	**0.2932**
64	10	Adam	categorical_crossentropy	0.8999	0.9244	0.879	0.394

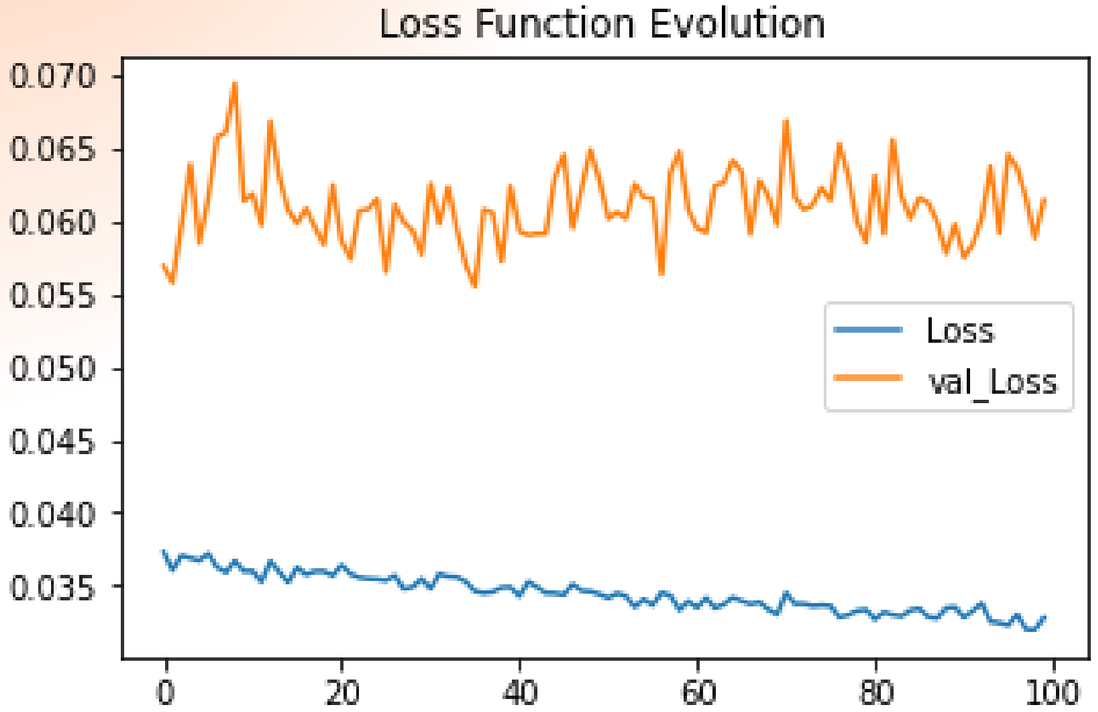
batch_size

batch_size	Epoch	optimizer	Loss_function	Accuracy	Precision	Recall	Loss_val
64	10	Adam	categorical_crossentropy	0.8999	0.9244	0.879	0.394
46	60	Adam	binary_crossentropy	0.9008	0.9205	0.8864	0.0523
64	50	Adam	binary_crossentropy	0.9176	0.9344	0.9026	0.3358
64	**300**	**Adam**	**binary_crossentropy**	**0.9418**	**0.9515**	**0.9351**	**0.0328**

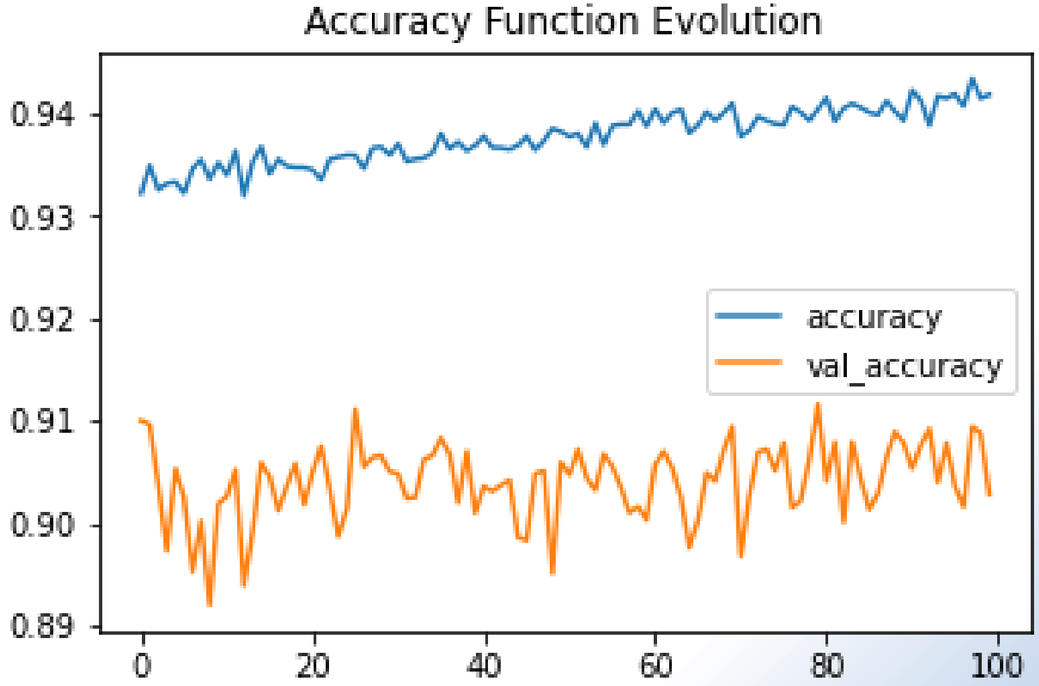
	parameter	Accuracy	loss
Optimizers	Adam	0.9418	0.0328
Loss function	binary cross entropy	0.9418	0.0328
Epochs	300	0.9418	0.0328
Batch size	64	0.9418	0.0328

< Comparision

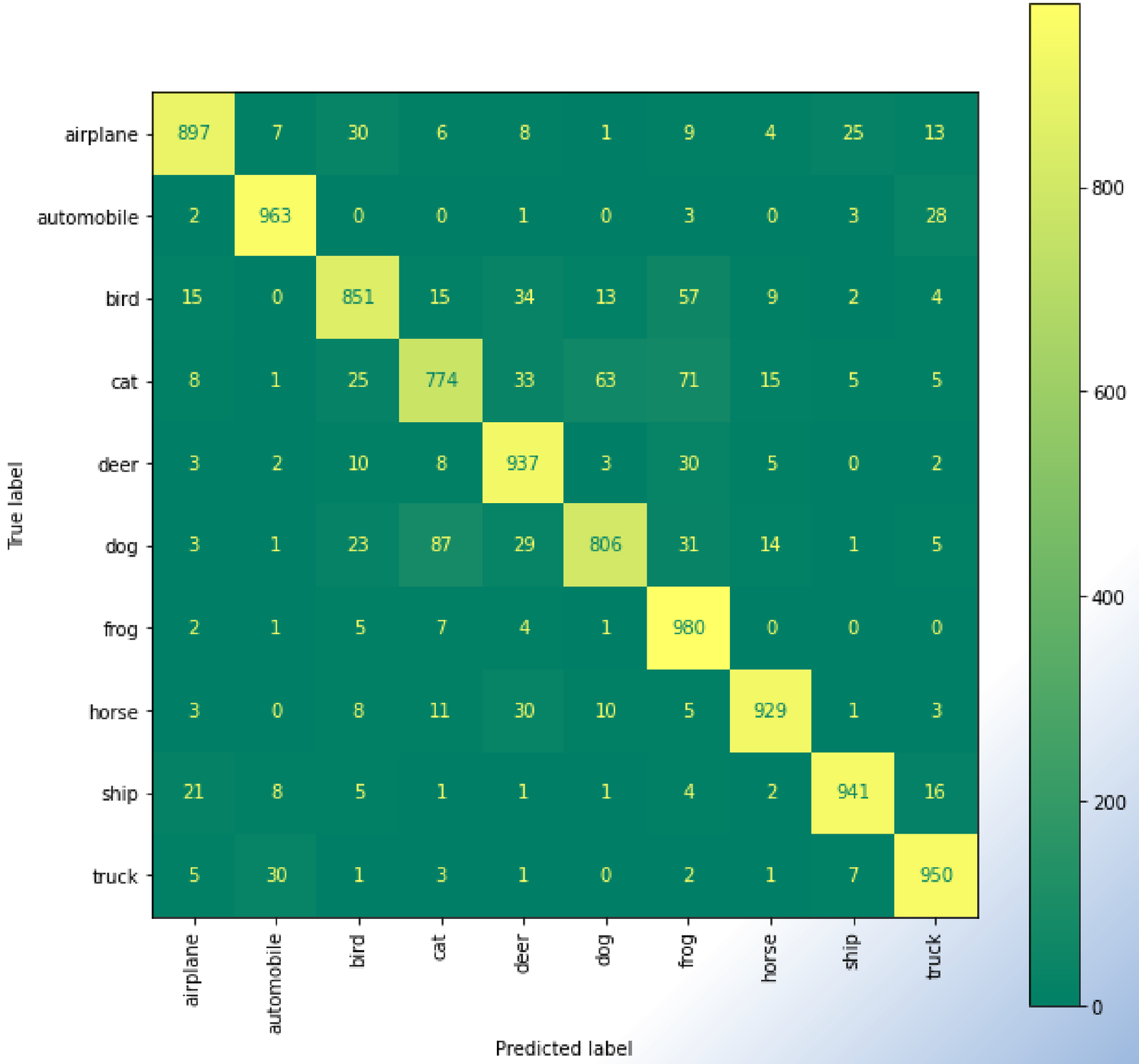
Loss curve



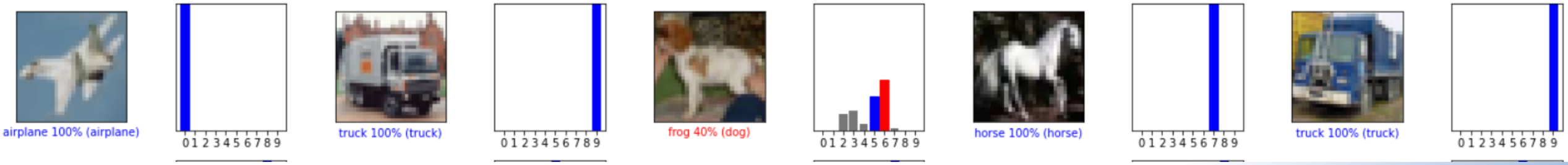
Accuracy curve



Heatmap


airplane 897 7 30 6 8 1 9 4 25 13
automobile 2 963 0 0 1 0 3 0 3 28
bird 15 0 851 15 34 13 57 9 2 4
cat 8 1 25 774 33 63 71 15 5 5
deer 3 2 10 8 937 3 30 5 0 2
dog 3 1 23 87 29 806 31 14 1 5
frog 2 1 5 7 4 1 980 0 0 0
horse 3 0 8 11 30 10 5 929 1 3
ship 21 8 5 1 1 1 4 2 941 16
truck 5 30 1 3 1 0 2 1 7 950
True label
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
Predicted label
800
600
400
200
0

Comparision of Actual And predicted



Conclusion:

- Optimizer- Adam
- Loss- Binary cross entropy
- Epochs- 300
- Batch size- 64

All the Images are equal in proportion in our dataset.

- Highest Accuracy-0.9418
- Least loss- 0.0328

Majority Correctly classified are Frog.

Minority correctly classified are cat.

Thank You

QR code of our code

APPENDIX

```
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
```

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

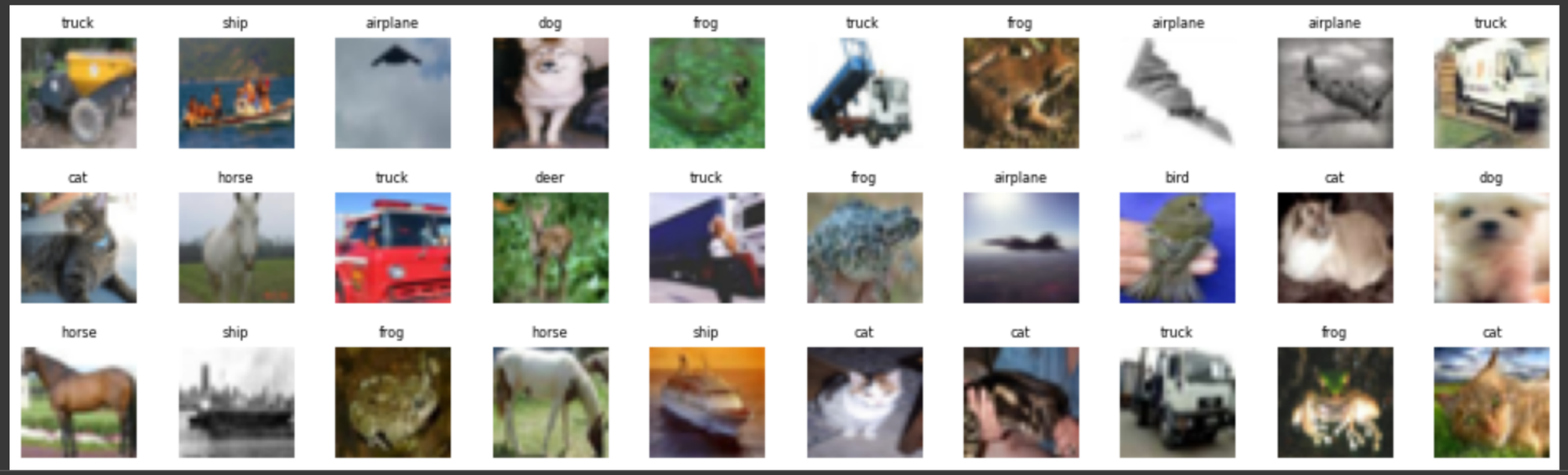
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
```

```
[ ]  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)
```

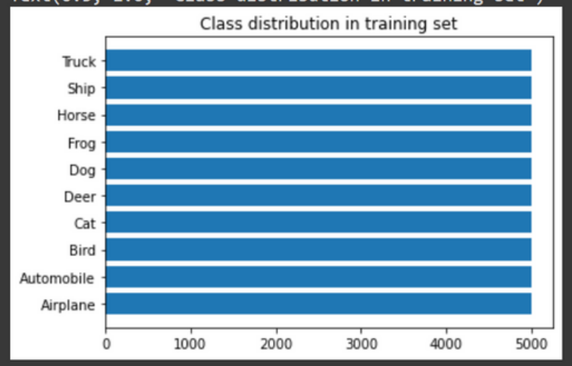
```
# Define the labels of the dataset
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer',
          'dog', 'frog', 'horse', 'ship', 'truck']

# 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
```

```
# 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)
```



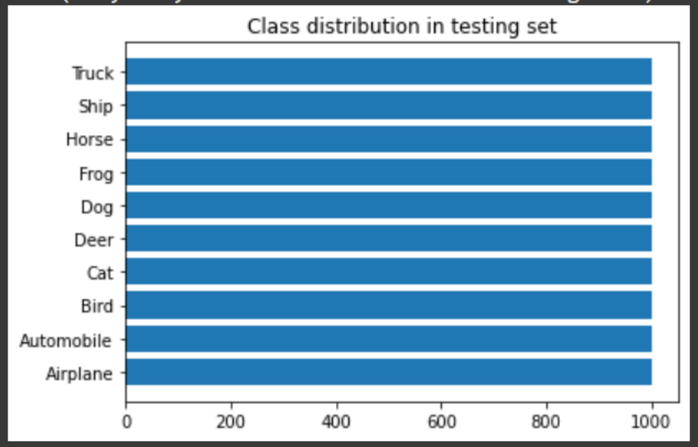
```
classes_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

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')
```




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

```
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 = Sequential()

    # Convolutional Layer
    model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', 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='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(2, 2)))
```

```
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', 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='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), input_shape=(32, 32, 3), activation='relu', 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='binary_crossentropy', optimizer='Adam', metrics=METRICS)
```

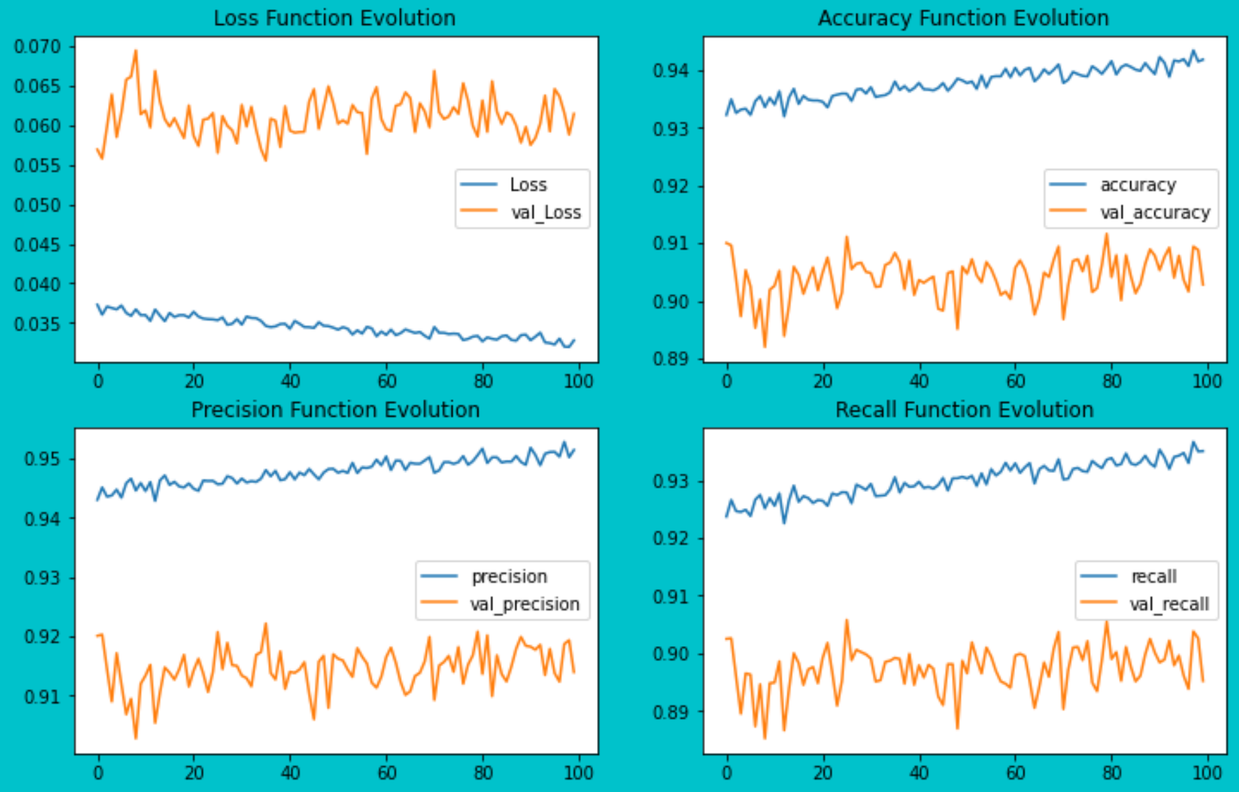
```
[ ]  early_stop = EarlyStopping(monitor='val_loss', patience=2)
```

```
[ ]  batch_size = 64
     data_generator = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
     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=100,
                   steps_per_epoch=steps_per_epoch,
                   validation_data=(X_test, y_cat_test),
     #               callbacks=[early_stop],
                     batch_size=batch_size,
                  )
```

```
Epoch 1/100
781/781 [==============================] - 26s 34ms/step - loss: 0.0373 - accuracy: 0.9322 - precision: 0.9431 - recall: 0.9237 - val_loss: 0.0569 - val_accuracy: 0.91
Epoch 2/100
781/781 [==============================] - 26s 33ms/step - loss: 0.0360 - accuracy: 0.9349 - precision: 0.9452 - recall: 0.9266 - val_loss: 0.0558 - val_accuracy: 0.90
Epoch 3/100
781/781 [==============================] - 26s 33ms/step - loss: 0.0371 - accuracy: 0.9325 - precision: 0.9436 - recall: 0.9247 - val_loss: 0.0598 - val_accuracy: 0.90
Epoch 4/100
781/781 [==============================] - 26s 34ms/step - loss: 0.0369 - accuracy: 0.9331 - precision: 0.9438 - recall: 0.9245 - val_loss: 0.0639 - val_accuracy: 0.89
Epoch 5/100
781/781 [==============================] - 26s 33ms/step - loss: 0.0367 - accuracy: 0.9333 - precision: 0.9448 - recall: 0.9249 - val_loss: 0.0585 - val_accuracy: 0.90
Epoch 6/100
781/781 [==============================] - 27s 34ms/step - loss: 0.0372 - accuracy: 0.9322 - precision: 0.9434 - recall: 0.9238 - val_loss: 0.0616 - val_accuracy: 0.90
Epoch 7/100
781/781 [==============================] - 26s 33ms/step - loss: 0.0363 - accuracy: 0.9346 - precision: 0.9459 - recall: 0.9267 - val_loss: 0.0657 - val_accuracy: 0.89
Epoch 8/100
```

```python
from matplotlib import pyplot as plt
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()
```



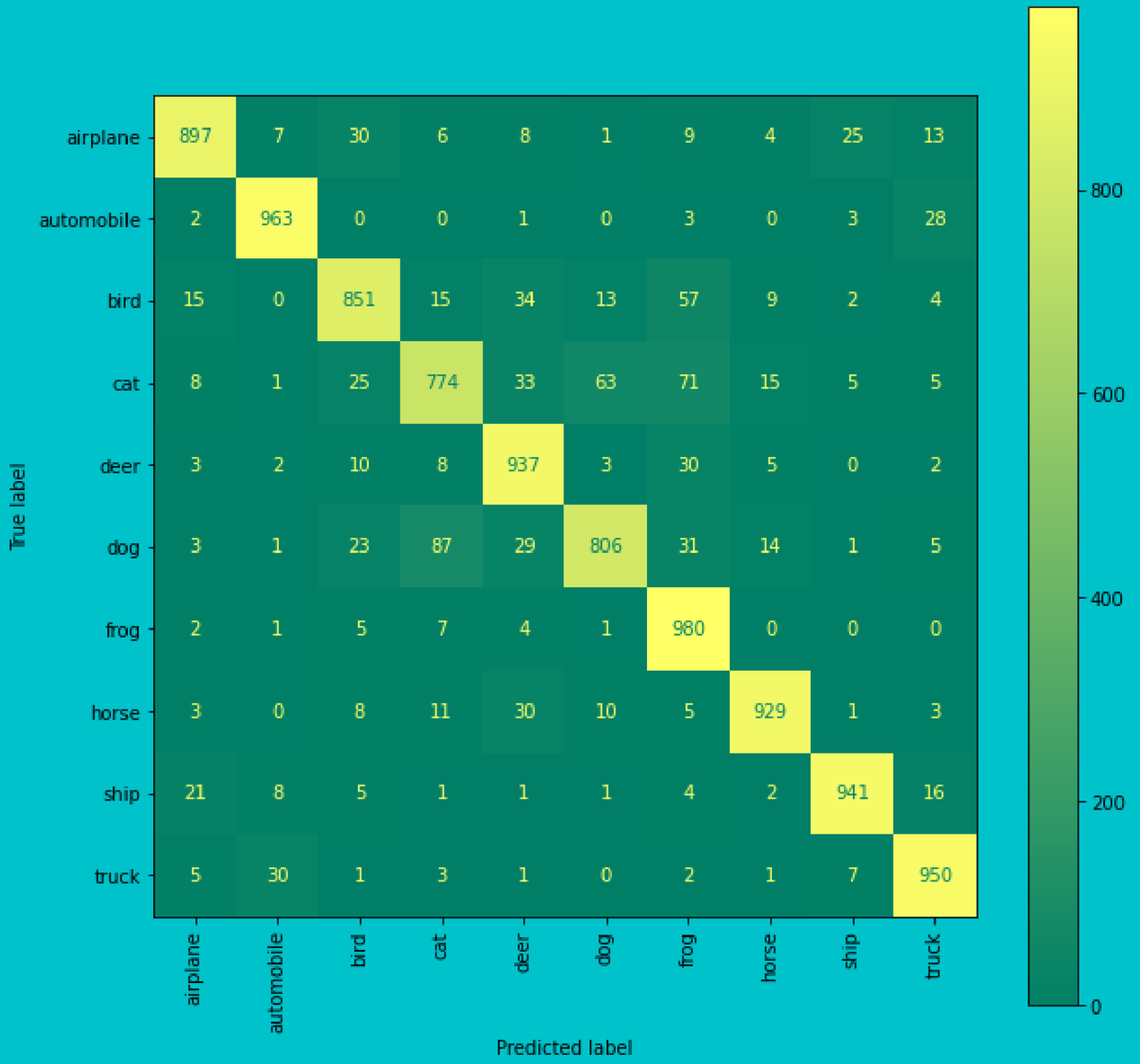
```
[ ]  evaluation = model.evaluate(X_test, y_cat_test)
     print(f'Test Accuracy : {evaluation[1] * 100:.2f}%')
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0614 - accuracy: 0.9028 - precision: 0.9139 - recall: 0.8952
Test Accuracy : 90.28%
```

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

```
313/313 [==============================] - 1s 3ms/step
```

```
[ ]  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()
```


airplane 897 7 30 6 8 1 9 4 25 13
automobile 2 963 0 0 1 0 3 0 3 28
bird 15 0 851 15 34 13 57 9 2 4
cat 8 1 25 774 33 63 71 15 5 5
deer 3 2 10 8 937 3 30 5 0 2
dog 3 1 23 87 29 806 31 14 1 5
frog 2 1 5 7 4 1 980 0 0 0
horse 3 0 8 11 30 10 5 929 1 3
ship 21 8 5 1 1 1 4 2 941 16
truck 5 30 1 3 1 0 2 1 7 950
True label
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
Predicted label
800
600
400
200
0

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

	precision	recall	f1-score	support
0	0.94	0.90	0.92	1000
1	0.95	0.96	0.96	1000
2	0.89	0.85	0.87	1000
3	0.85	0.77	0.81	1000
4	0.87	0.94	0.90	1000
5	0.90	0.81	0.85	1000
6	0.82	0.98	0.89	1000
7	0.95	0.93	0.94	1000
8	0.96	0.94	0.95	1000
9	0.93	0.95	0.94	1000
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

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

```
# 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()
```


cat 100% (cat)
ship 100% (ship)
ship 100% (ship)
airplane 100% (airplane)
frog 100% (frog)
frog 100% (frog)
automobile 100% (automobile)
frog 100% (frog)
cat 100% (cat)
automobile 100% (automobile)
airplane 100% (airplane)
truck 100% (truck)
frog 40% (dog)
horse 100% (horse)
truck 100% (truck)
ship 100% (ship)
dog 100% (dog)
horse 100% (horse)
ship 100% (ship)
frog 100% (frog)
horse 100% (horse)
airplane 100% (airplane)
deer 100% (deer)
truck 100% (truck)
deer 99% (dog)
bird 100% (bird)
deer 100% (deer)
airplane 100% (airplane)
truck 100% (truck)
frog 100% (frog)
frog 100% (frog)
dog 100% (dog)
deer 100% (deer)
dog 100% (dog)
truck 100% (truck)
ship 41% (bird)
deer 100% (deer)
automobile 53% (automobile)
truck 100% (truck)
dog 100% (dog)
0 1 2 3 4 5 6 7 8 9