IMAGE CLASSIFICATION - A CNN APPROACH

In [ ]:
```
project by

Hitaishi Bairapureddy
Nikhila Reddy Vantari
Archana Gadicharla
```

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
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, classification_report, confusion_matrix

# Load the CIFAR-10 dataset
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

print(f"Train images shape: {train_images.shape}")
print(f"Train labels shape: {train_labels.shape}")
print(f"Test images shape: {test_images.shape}")
print(f"Test labels shape: {test_labels.shape}")

# Define class labels
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Visualize the training data
rows, cols = 10, 10
fig, axes = plt.subplots(rows, cols, figsize=(15, 15))
```

```
axes = axes.ravel()

num_train = len(train_images)
for i in np.arange(0, rows * cols):
    idx = np.random.randint(0, num_train)
    axes[i].imshow(train_images[idx])
    label_idx = int(train_labels[idx])
    axes[i].set_title(class_names[label_idx], fontsize=8)
    axes[i].axis('off')

plt.subplots_adjust(hspace=0.5)
plt.show()

# Plot class distribution
unique_classes, counts = np.unique(train_labels, return_counts=True)
plt.barh(class_names, counts)
plt.title('Class Distribution in Training Set')
plt.show()

unique_classes, counts = np.unique(test_labels, return_counts=True)
plt.barh(class_names, counts)
plt.title('Class Distribution in Testing Set')
plt.show()

# Data Preprocessing: Scale and one-hot encode the labels
train_images = train_images / 255.0
test_images = test_images / 255.0

train_labels_ohe = to_categorical(train_labels, 10)
test_labels_ohe = to_categorical(test_labels, 10)

# Model Building
input_shape = (32, 32, 3)
kernel_size = (3, 3)

cnn_model = Sequential()

# First Convolutional Block
cnn_model.add(Conv2D(32, kernel_size=kernel_size, activation='relu', padding='same', input_shape=input_shape))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(32, kernel_size=kernel_size, activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))
```

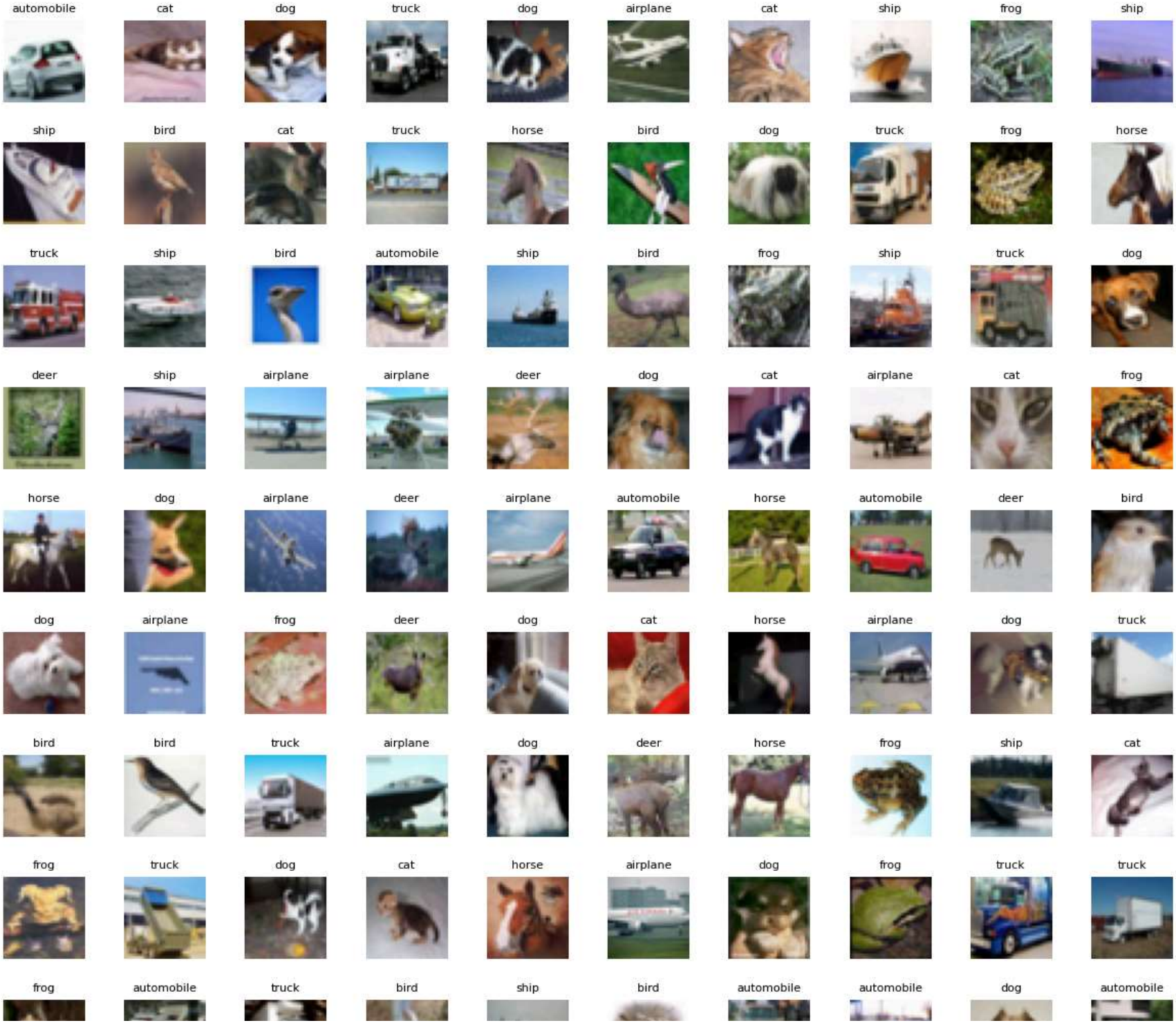
```python
# Second Convolutional Block
cnn_model.add(Conv2D(64, kernel_size=kernel_size, activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(64, kernel_size=kernel_size, activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))

# Third Convolutional Block
cnn_model.add(Conv2D(128, kernel_size=kernel_size, activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(Conv2D(128, kernel_size=kernel_size, activation='relu', padding='same'))
cnn_model.add(BatchNormalization())
cnn_model.add(MaxPool2D(pool_size=(2, 2)))
cnn_model.add(Dropout(0.25))

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

# Compile the model
metrics_list = [
    'accuracy',
    tf.keras.metrics.Precision(name='precision'),
    tf.keras.metrics.Recall(name='recall')
]

cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=metrics_list)
cnn_model.summary()

# Early Stopping Callback
early_stop = EarlyStopping(monitor='val_loss', patience=2)

# Data Augmentation
batch_size = 32
data_augmentor = ImageDataGenerator(width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
train_data_gen = data_augmentor.flow(train_images, train_labels_ohe, batch_size=batch_size)

steps_per_epoch = train_images.shape[0] // batch_size

# Train the model
history = cnn_model.fit(
    train_data_gen,
```

```
    epochs=50,
    steps_per_epoch=steps_per_epoch,
    validation_data=(test_images, test_labels_ohe),
    callbacks=[early_stop]
)

# Model Evaluation
plt.figure(figsize=(12, 16))

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

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

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

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

plt.show()

# Evaluate on test set
evaluation = cnn_model.evaluate(test_images, test_labels_ohe)
print(f'Test Accuracy: {evaluation[1] * 100:.2f}%')

# Confusion Matrix
predicted_labels = np.argmax(cnn_model.predict(test_images), axis=1)
confusion_mtx = confusion_matrix(test_labels, predicted_labels)

disp = ConfusionMatrixDisplay(confusion_matrix=confusion_mtx, display_labels=class_names)
fig, ax = plt.subplots(figsize=(10, 10))
```

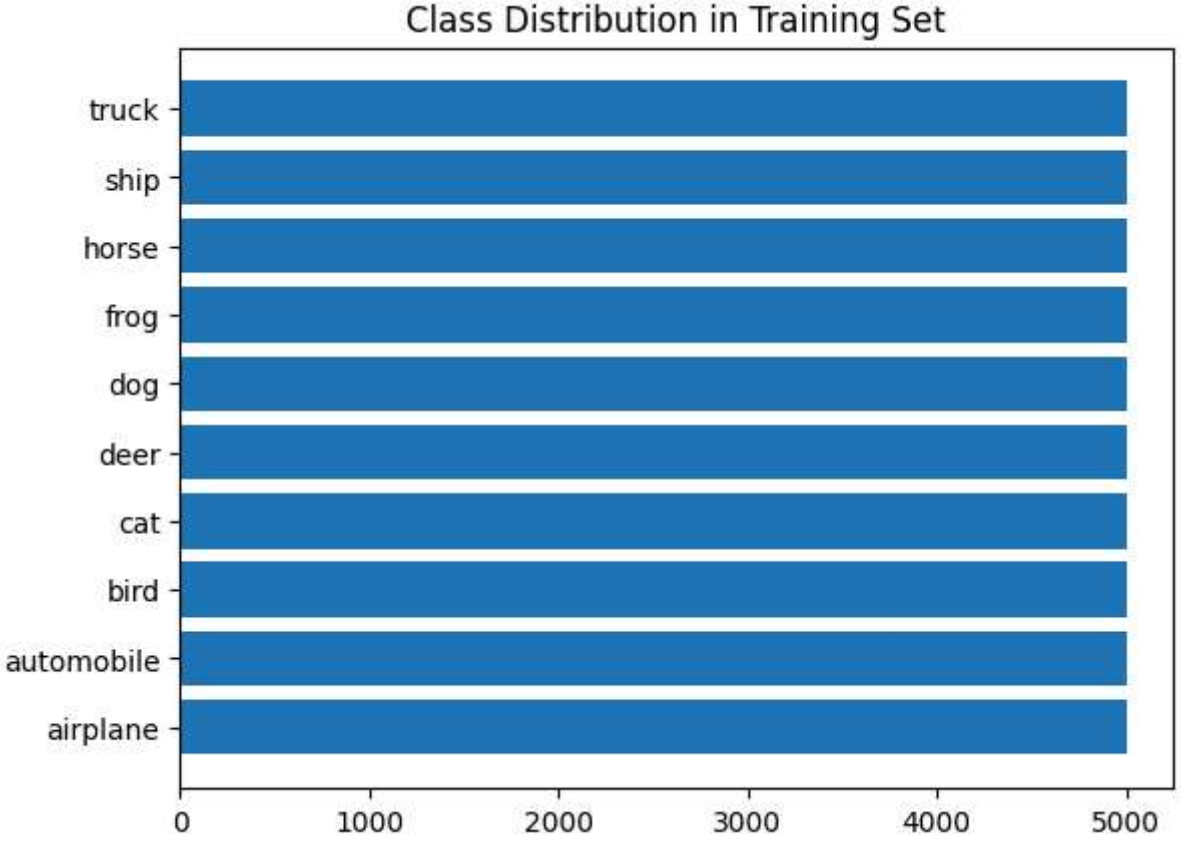
```
disp.plot(xticks_rotation='vertical', ax=ax, cmap='viridis')
plt.show()

print(classification_report(test_labels, predicted_labels))

# Save the model
cnn_model.save('cnn_model_cifar10.h5')
```

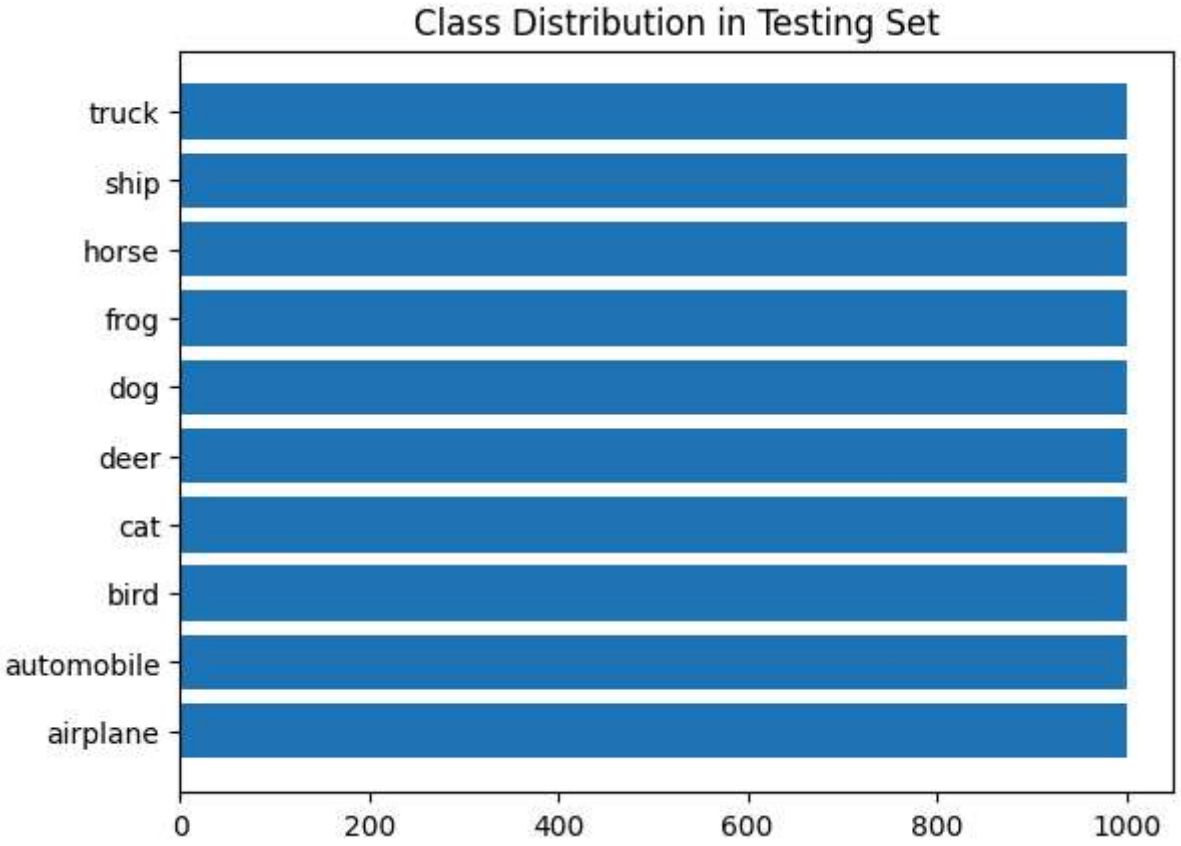
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 6s 0us/step
Train images shape: (50000, 32, 32, 3)
Train labels shape: (50000, 1)
Test images shape: (10000, 32, 32, 3)
Test labels shape: (10000, 1)
```

```
<ipython-input-1-c32f621ee113>:40: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated,
and will error in future. Ensure you extract a single element from your array before performing this operation. (Deprec
ated NumPy 1.25.)
  label_idx = int(train_labels[idx])
```


automobile cat dog truck dog airplane cat ship frog ship
ship bird cat truck horse bird dog truck frog horse
truck ship bird automobile ship bird frog ship truck dog
deer ship airplane airplane deer dog cat airplane cat frog
horse dog airplane deer airplane automobile horse automobile deer bird
dog airplane frog deer dog cat horse airplane dog truck
bird bird truck airplane dog deer horse frog ship cat
frog truck dog cat horse airplane dog frog truck truck
frog automobile truck bird ship bird automobile automobile dog automobile


truck
automobile
dog
cat
automobile
bird
bird
airplane
horse
deer


Class Distribution in Training Set
truck
ship
horse
frog
dog
deer
cat
bird
automobile
airplane
0
1000
2000
3000
4000
5000




```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `i
nput_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the
first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
batch_normalization (BatchNormalization)	(None, 32, 32, 32)	128
conv2d_1 (Conv2D)	(None, 32, 32, 32)	9,248
batch_normalization_1 (BatchNormalization)	(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)	18,496
batch_normalization_2 (BatchNormalization)	(None, 16, 16, 64)	256
conv2d_3 (Conv2D)	(None, 16, 16, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 16, 16, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 64)	0
dropout_1 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4 (BatchNormalization)	(None, 8, 8, 128)	512
conv2d_5 (Conv2D)	(None, 8, 8, 128)	147,584
batch_normalization_5 (BatchNormalization)	(None, 8, 8, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 128)	0

dropout_2 (Dropout)	(None, 4, 4, 128)	0
flatten (Flatten)	(None, 2048)	0
dense (Dense)	(None, 128)	262,272
dropout_3 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

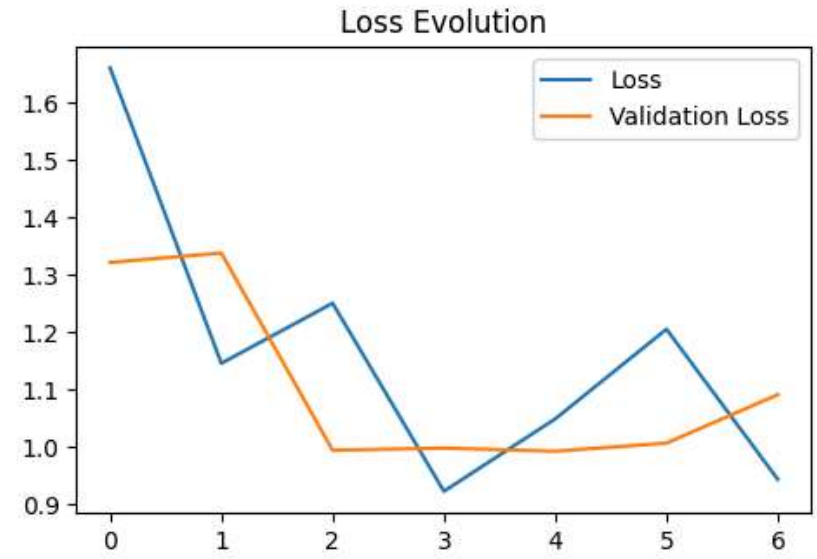
Total params: 552,362 (2.11 MB)

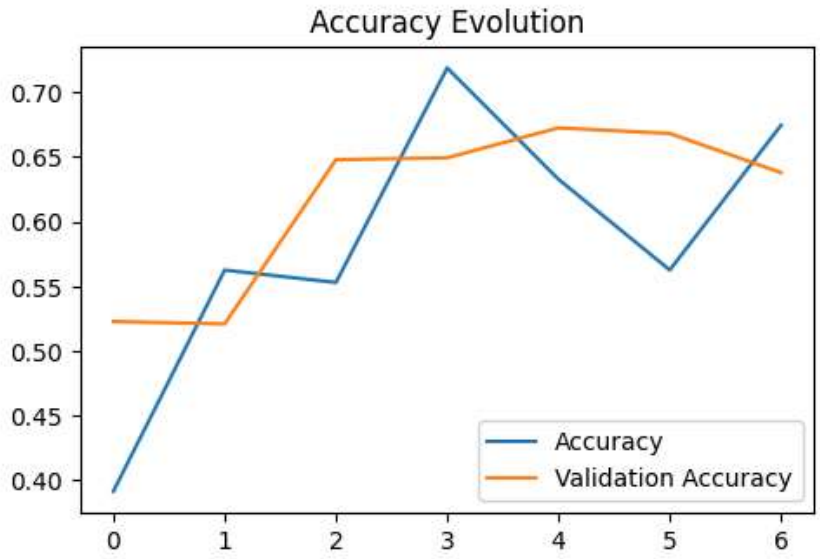
Trainable params: 551,466 (2.10 MB)

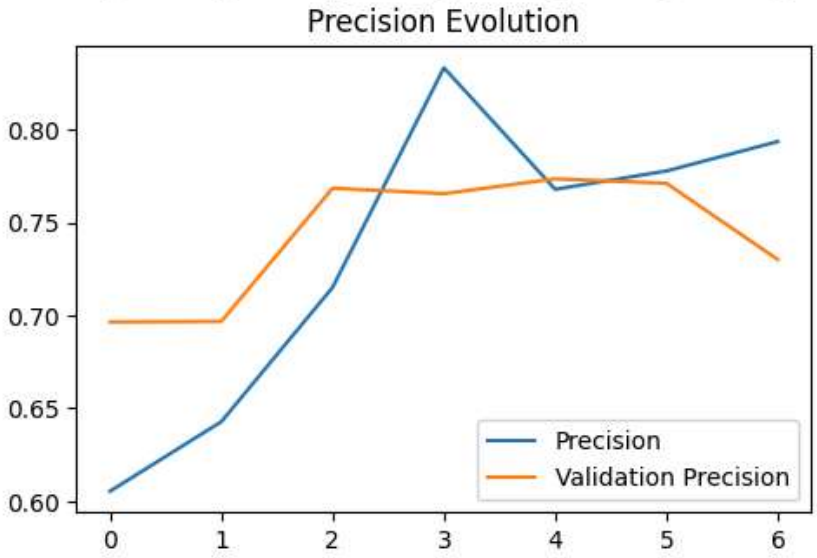
Non-trainable params: 896 (3.50 KB)

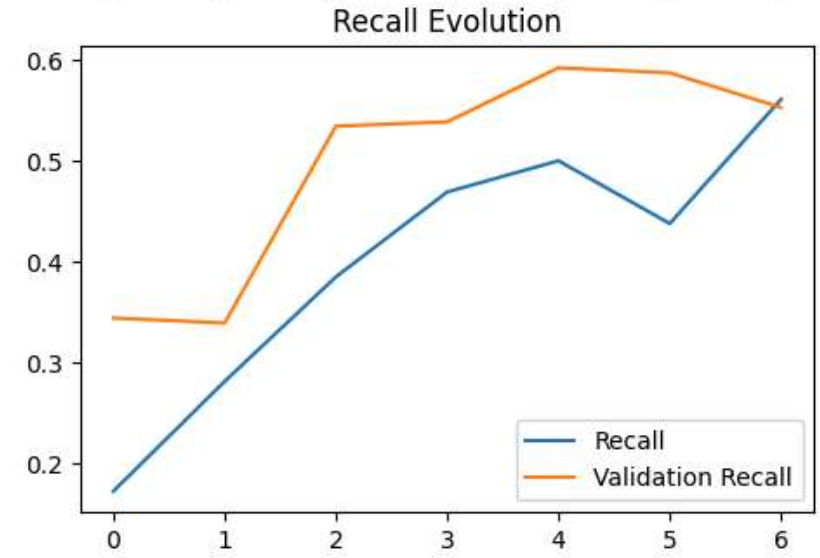
```
Epoch 1/50
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 68s 36ms/step - accuracy: 0.3202 - loss: 1.8887 - precision: 0.5107 - recall: 0.1076 - val_accuracy: 0.5227 - val_loss: 1.3207 - val_precision: 0.6965 - val_recall: 0.3440
Epoch 2/50
   1/1562 ━━━━━━━━━━━━━━━━━━━━ 16s 11ms/step - accuracy: 0.5625 - loss: 1.1448 - precision: 0.6429 - recall: 0.2812
/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out of data; interrupting training. Make sure that your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.repeat()` function when building your dataset.
  self.gen.throw(typ, value, traceback)
```

```
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 1s 423us/step - accuracy: 0.5625 - loss: 1.1448 - precision: 0.6429 - recall: 0.2812 - v
al_accuracy: 0.5209 - val_loss: 1.3370 - val_precision: 0.6968 - val_recall: 0.3390
Epoch 3/50
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 65s 30ms/step - accuracy: 0.5259 - loss: 1.3199 - precision: 0.6953 - recall: 0.3445 - v
al_accuracy: 0.6478 - val_loss: 0.9933 - val_precision: 0.7685 - val_recall: 0.5339
Epoch 4/50
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 1s 490us/step - accuracy: 0.7188 - loss: 0.9217 - precision: 0.8333 - recall: 0.4688 - v
al_accuracy: 0.6493 - val_loss: 0.9970 - val_precision: 0.7656 - val_recall: 0.5382
Epoch 5/50
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 82s 31ms/step - accuracy: 0.6181 - loss: 1.0760 - precision: 0.7604 - recall: 0.4843 - v
al_accuracy: 0.6724 - val_loss: 0.9915 - val_precision: 0.7736 - val_recall: 0.5917
Epoch 6/50
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 1s 493us/step - accuracy: 0.5625 - loss: 1.2044 - precision: 0.7778 - recall: 0.4375 - v
al_accuracy: 0.6681 - val_loss: 1.0056 - val_precision: 0.7712 - val_recall: 0.5867
Epoch 7/50
1562/1562 ━━━━━━━━━━━━━━━━━━━━ 80s 30ms/step - accuracy: 0.6689 - loss: 0.9513 - precision: 0.7894 - recall: 0.5541 - v
al_accuracy: 0.6378 - val_loss: 1.0902 - val_precision: 0.7301 - val_recall: 0.5524
```

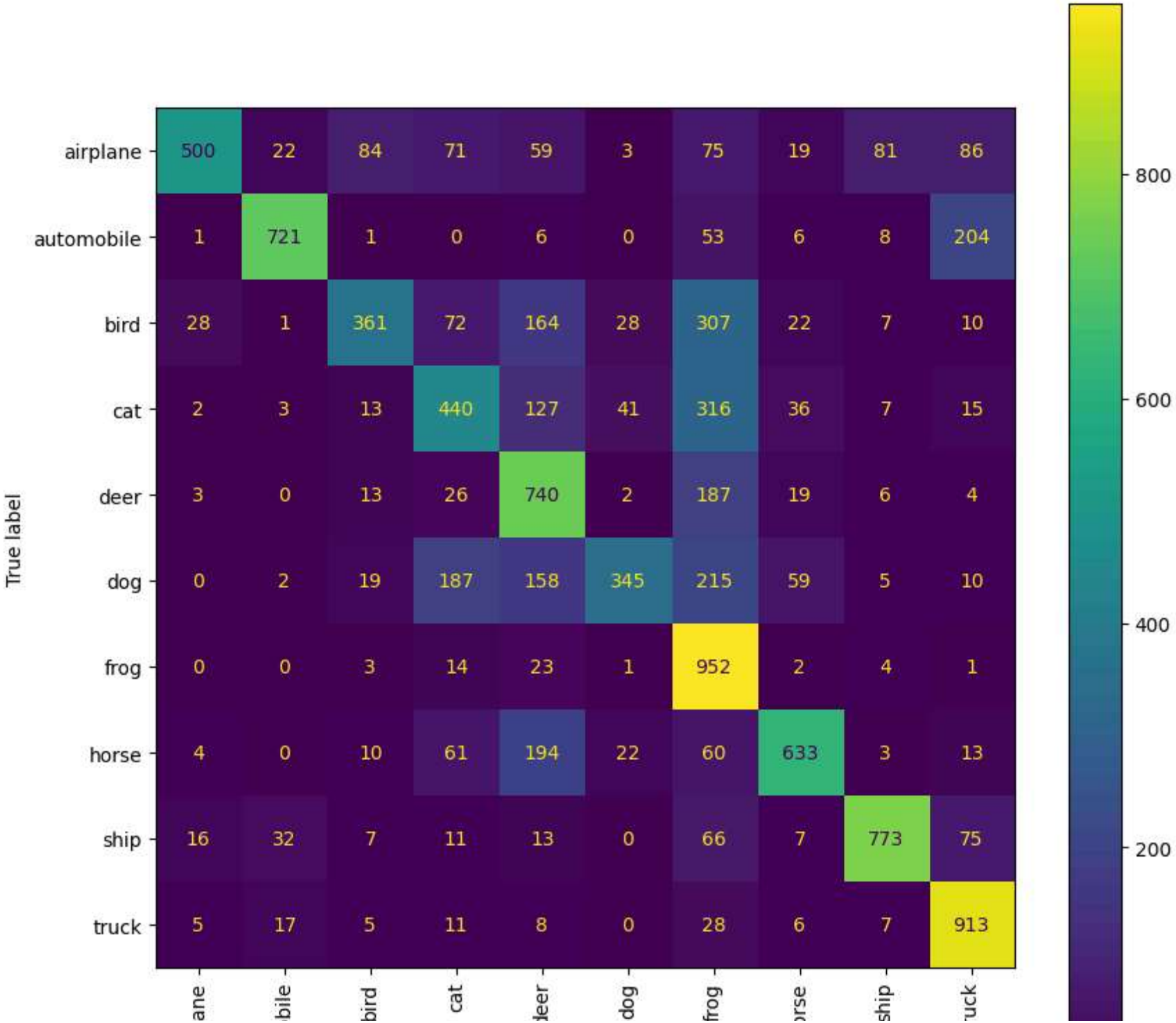









```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.6388 - loss: 1.0792 - precision: 0.7322 - recall: 0.5518
Test Accuracy: 63.78%
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step
```


True label
airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck
500 22 84 71 59 3 75 19 81 86
1 721 1 0 6 0 53 6 8 204
28 1 361 72 164 28 307 22 7 10
2 3 13 440 127 41 316 36 7 15
3 0 13 26 740 2 187 19 6 4
0 2 19 187 158 345 215 59 5 10
0 0 3 14 23 1 952 2 4 1
4 0 10 61 194 22 60 633 3 13
16 32 7 11 13 0 66 7 773 75
5 17 5 11 8 0 28 6 7 913
ane
bile
bird
cat
deer
dog
frog
orse
ship
uck
800
600
400
200

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.

	precision	recall	f1-score	support
0	0.89	0.50	0.64	1000
1	0.90	0.72	0.80	1000
2	0.70	0.36	0.48	1000
3	0.49	0.44	0.46	1000
4	0.50	0.74	0.59	1000
5	0.78	0.34	0.48	1000
6	0.42	0.95	0.58	1000
7	0.78	0.63	0.70	1000
8	0.86	0.77	0.81	1000
9	0.69	0.91	0.78	1000
accuracy			0.64	10000
macro avg	0.70	0.64	0.63	10000
weighted avg	0.70	0.64	0.63	10000