01_model_s_baseline_sgd

June 10, 2024

---

1 Model S - Baseline Model with Stochastic Gradient Descent (SGD) Optimizer

- **32 x 32 x 3** Image size.
- **64** Batch size.
- Stochastic Gradient Descent **(SGD)** optimizer with **0.9** momentum.
- **0.01** Initial Learning rate.
- **Sparse Categorical Cross-Entropy** loss function.
- **Reduce Learning Rate on Plateau** callback with a **0.1** factor and **3** patience.
- **Early Stopping** callback with patience of **6** and restore best weights.
- **Model Checkpoint** callback to save the best model based on validation loss.
- 3 **Convolutional** layers with **16**, **32** and **64** filters, with **ReLU** activation.
- 3 **MaxPooling** layers with **2 x 2** pool size.
- **3 x 3** Convolutional kernel size.
- **Padding** is **valid**, in this case **1**.
- **4 x 4 x 64** Tensor before the **Flatten** layer.
- **128** Dense layer with **ReLU** activation.
- **10** Dense output layer with **Softmax** activation.
- **156 074** Trainable Parameters.
- **30** Epochs.
- Evaluate the model on the **Validation** dataset.
- Test the model on the **Test** dataset.

Imports and Setup

[1]:
```python
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
print(f'TensorFlow version: {tf.__version__}')
tf.get_logger().setLevel('ERROR')
tf.autograph.set_verbosity(3)
import matplotlib.pyplot as plt
import pickle
import numpy as np
from tensorflow.keras.utils import image_dataset_from_directory
```

```python
from tensorflow import keras
from tensorflow.keras import callbacks, layers, models
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay␣
↪,accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.preprocessing import label_binarize
from itertools import cycle
```

```
TensorFlow version: 2.15.0
```

Group Datasets

[2]:
```python
train_dirs = ['../data/train1', '../data/train3', '../data/train4', '../data/
↪train5']
validation_dir = '../data/train2'
test_dir = '../data/test'
```

Create Datasets

[3]:
```python
IMG_SIZE = 32
BATCH_SIZE = 64
NUM_CLASSES = 10

train_datasets = [image_dataset_from_directory(directory, image_size=(IMG_SIZE,␣
↪IMG_SIZE), batch_size=BATCH_SIZE) for directory in train_dirs]

train_dataset = train_datasets[0]
for dataset in train_datasets[1:]:
    train_dataset = train_dataset.concatenate(dataset)

train_dataset = train_dataset.shuffle(buffer_size=1000).prefetch(buffer_size=tf.
↪data.AUTOTUNE)
validation_dataset = image_dataset_from_directory(validation_dir,␣
↪image_size=(IMG_SIZE, IMG_SIZE), batch_size=BATCH_SIZE).
↪prefetch(buffer_size=tf.data.AUTOTUNE)
test_dataset = image_dataset_from_directory(test_dir, image_size=(IMG_SIZE,␣
↪IMG_SIZE), batch_size=BATCH_SIZE).prefetch(buffer_size=tf.data.AUTOTUNE)

class_names = train_datasets[0].class_names

for data_batch, labels_batch in train_dataset.take(1):
    print('data batch shape:', data_batch.shape)
    print('labels batch shape:', labels_batch.shape)
```

```
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
```

```
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
Found 10000 files belonging to 10 classes.
data batch shape: (64, 32, 32, 3)
labels batch shape: (64,)
```

- We define the image size of 32 x 32 x 3, batch size of 64 and create an array with the label's names.
- We create the train dataset by concatenating them, we **shuffle** the samples before each epoch and **prefetch** them to memory.
- We do the same for the validation and test dataset except **shuffling** which is **unwanted** for these datasets.

Model Architecture

[4]:
```python
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=16, kernel_size=3, activation="relu",␣
↪padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=32, kernel_size=3,␣
↪activation="relu",padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3,␣
↪activation="relu",padding="same")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
x = layers.Dense(128, activation="relu")(x)
outputs = layers.Dense(NUM_CLASSES, activation="softmax")(x)
model = models.Model(inputs=inputs, outputs=outputs)
model.summary()
```

```
Model: "model"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 rescaling (Rescaling)       (None, 32, 32, 3)         0

 conv2d (Conv2D)             (None, 32, 32, 16)        448

 max_pooling2d (MaxPooling2  (None, 16, 16, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 16, 16, 32)        4640
```

```
 max_pooling2d_1 (MaxPoolin  (None, 8, 8, 32)          0
 g2D)

 conv2d_2 (Conv2D)           (None, 8, 8, 64)          18496

 max_pooling2d_2 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 128)               131200

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 156074 (609.66 KB)
Trainable params: 156074 (609.66 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Model Compilation

[5]:
```
initial_learning_rate = 0.01
optimizer = tf.keras.optimizers.SGD(learning_rate=initial_learning_rate,␣
 ↪momentum=0.9)
loss_function = keras.losses.SparseCategoricalCrossentropy()

lr_scheduler = callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,␣
 ↪patience=3, verbose=1)
early_stopping = callbacks.EarlyStopping(monitor='val_loss', patience=6,␣
 ↪restore_best_weights=True, verbose=1)
save_best_model = callbacks.ModelCheckpoint(filepath='../models/
 ↪01_model_s_baseline_sgd.keras', save_best_only=True, monitor='val_loss',␣
 ↪verbose=1)

callbacks = [lr_scheduler, early_stopping, save_best_model]

model.compile(optimizer=optimizer,
              loss=loss_function,
              metrics=['accuracy'])
```

Model Training

[6]:
```python
history = model.fit(train_dataset,
                    validation_data=validation_dataset,
                    epochs=30,
                    callbacks=callbacks)
```

```
Epoch 1/30
623/628 [============================>.] - ETA: 0s - loss: 1.8155 - accuracy:
0.3367
Epoch 1: val_loss improved from inf to 1.43607, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 9s 13ms/step - loss: 1.8122 -
accuracy: 0.3379 - val_loss: 1.4361 - val_accuracy: 0.4827 - lr: 0.0100
Epoch 2/30
627/628 [============================>.] - ETA: 0s - loss: 1.3289 - accuracy:
0.5243
Epoch 2: val_loss improved from 1.43607 to 1.24996, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 1.3285 -
accuracy: 0.5245 - val_loss: 1.2500 - val_accuracy: 0.5588 - lr: 0.0100
Epoch 3/30
628/628 [==============================] - ETA: 0s - loss: 1.1355 - accuracy:
0.5965
Epoch 3: val_loss improved from 1.24996 to 1.15490, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 6s 10ms/step - loss: 1.1355 -
accuracy: 0.5965 - val_loss: 1.1549 - val_accuracy: 0.5999 - lr: 0.0100
Epoch 4/30
628/628 [==============================] - ETA: 0s - loss: 0.9946 - accuracy:
0.6498
Epoch 4: val_loss improved from 1.15490 to 1.01155, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 0.9946 -
accuracy: 0.6498 - val_loss: 1.0115 - val_accuracy: 0.6460 - lr: 0.0100
Epoch 5/30
627/628 [============================>.] - ETA: 0s - loss: 0.9019 - accuracy:
0.6837
Epoch 5: val_loss improved from 1.01155 to 1.00895, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 0.9017 -
accuracy: 0.6837 - val_loss: 1.0089 - val_accuracy: 0.6457 - lr: 0.0100
Epoch 6/30
626/628 [============================>.] - ETA: 0s - loss: 0.8196 - accuracy:
0.7103
Epoch 6: val_loss improved from 1.00895 to 0.94372, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 0.8196 -
accuracy: 0.7102 - val_loss: 0.9437 - val_accuracy: 0.6727 - lr: 0.0100
```

```
Epoch 7/30
623/628 [============================>.] - ETA: 0s - loss: 0.7525 - accuracy:
0.7355
Epoch 7: val_loss improved from 0.94372 to 0.91953, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 0.7524 -
accuracy: 0.7357 - val_loss: 0.9195 - val_accuracy: 0.6858 - lr: 0.0100
Epoch 8/30
625/628 [============================>.] - ETA: 0s - loss: 0.6812 - accuracy:
0.7616
Epoch 8: val_loss did not improve from 0.91953
628/628 [==============================] - 7s 10ms/step - loss: 0.6816 -
accuracy: 0.7612 - val_loss: 0.9825 - val_accuracy: 0.6777 - lr: 0.0100
Epoch 9/30
628/628 [==============================] - ETA: 0s - loss: 0.6217 - accuracy:
0.7818
Epoch 9: val_loss did not improve from 0.91953
628/628 [==============================] - 6s 10ms/step - loss: 0.6217 -
accuracy: 0.7818 - val_loss: 0.9320 - val_accuracy: 0.6883 - lr: 0.0100
Epoch 10/30
625/628 [============================>.] - ETA: 0s - loss: 0.5586 - accuracy:
0.8029
Epoch 10: ReduceLROnPlateau reducing learning rate to 0.0009999999776482583.

Epoch 10: val_loss did not improve from 0.91953
628/628 [==============================] - 7s 10ms/step - loss: 0.5591 -
accuracy: 0.8028 - val_loss: 0.9420 - val_accuracy: 0.6889 - lr: 0.0100
Epoch 11/30
626/628 [============================>.] - ETA: 0s - loss: 0.3697 - accuracy:
0.8748
Epoch 11: val_loss improved from 0.91953 to 0.91586, saving model to
../models/01_model_s_baseline_sgd.keras
628/628 [==============================] - 7s 10ms/step - loss: 0.3698 -
accuracy: 0.8748 - val_loss: 0.9159 - val_accuracy: 0.7222 - lr: 1.0000e-03
Epoch 12/30
628/628 [==============================] - ETA: 0s - loss: 0.3101 - accuracy:
0.8973
Epoch 12: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.3101 -
accuracy: 0.8973 - val_loss: 0.9378 - val_accuracy: 0.7199 - lr: 1.0000e-03
Epoch 13/30
625/628 [============================>.] - ETA: 0s - loss: 0.2867 - accuracy:
0.9048
Epoch 13: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.2865 -
accuracy: 0.9049 - val_loss: 0.9595 - val_accuracy: 0.7226 - lr: 1.0000e-03
Epoch 14/30
623/628 [============================>.] - ETA: 0s - loss: 0.2677 - accuracy:
```

```
0.9133
Epoch 14: ReduceLROnPlateau reducing learning rate to 9.999999310821295e-05.

Epoch 14: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.2679 -
accuracy: 0.9133 - val_loss: 0.9820 - val_accuracy: 0.7218 - lr: 1.0000e-03
Epoch 15/30
628/628 [==============================] - ETA: 0s - loss: 0.2400 - accuracy:
0.9254
Epoch 15: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.2400 -
accuracy: 0.9254 - val_loss: 0.9786 - val_accuracy: 0.7217 - lr: 1.0000e-04
Epoch 16/30
625/628 [============================>.] - ETA: 0s - loss: 0.2360 - accuracy:
0.9273
Epoch 16: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.2360 -
accuracy: 0.9272 - val_loss: 0.9842 - val_accuracy: 0.7212 - lr: 1.0000e-04
Epoch 17/30
627/628 [============================>.] - ETA: 0s - loss: 0.2339 - accuracy:
0.9276
Epoch 17: ReduceLROnPlateau reducing learning rate to 9.999999019782991e-06.
Restoring model weights from the end of the best epoch: 11.

Epoch 17: val_loss did not improve from 0.91586
628/628 [==============================] - 7s 10ms/step - loss: 0.2338 -
accuracy: 0.9276 - val_loss: 0.9868 - val_accuracy: 0.7202 - lr: 1.0000e-04
Epoch 17: early stopping
```

Save Model History

[7]:
```python
with open("../history/01_model_s_baseline_sgd.pkl", "wb") as file:
    pickle.dump(history.history, file)
```

Model Evaluation

[8]:
```python
val_loss, val_acc = model.evaluate(validation_dataset)
print(f'Model Validation Loss: {val_loss:.2f}')
print(f'Model Validation Accuracy: {val_acc:.2%}')
```

```
157/157 [==============================] - 1s 3ms/step - loss: 0.9159 -
accuracy: 0.7222
Model Validation Loss: 0.92
Model Validation Accuracy: 72.22%
```

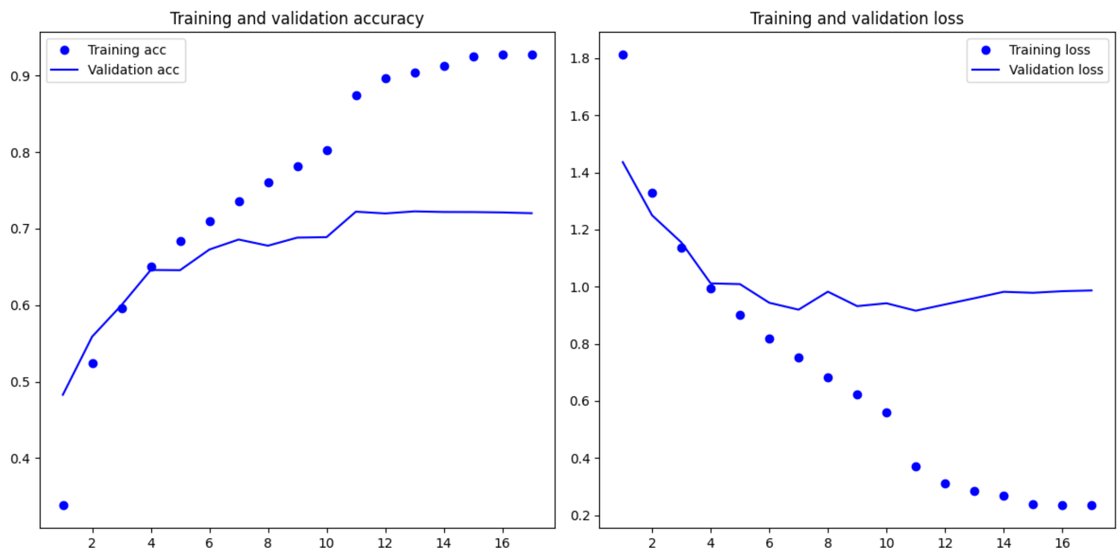
Model Training Visualization

[9]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.tight_layout()
plt.show()
```



- Analyzing the training and validation, accuracy and loss over the epochs:
  - We see that the model begins overfitting after the **6th** epoch.
  - The validation accuracy stops improving significantly after the **11th** epoch while the training accuracy keeps improving.
  - The validation loss stops improving significantly after the **7th** epoch while the training loss keeps improving.

- The best model, based on validation loss, is saved on the **11th** epoch.
- The training stops after the **17th** epoch because of the **Early Stopping** callback.

Model Testing

[10]:
```
test_labels = []
test_predictions = []
test_probabilities = []

for images, labels in test_dataset:
    test_labels.extend(labels.numpy())
    predictions = model.predict(images)
    test_predictions.extend(np.argmax(predictions, axis=-1))
    test_probabilities.extend(predictions)

test_labels = np.array(test_labels)
test_predictions = np.array(test_predictions)
test_probabilities = np.array(test_probabilities)
```

```
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 6ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
```

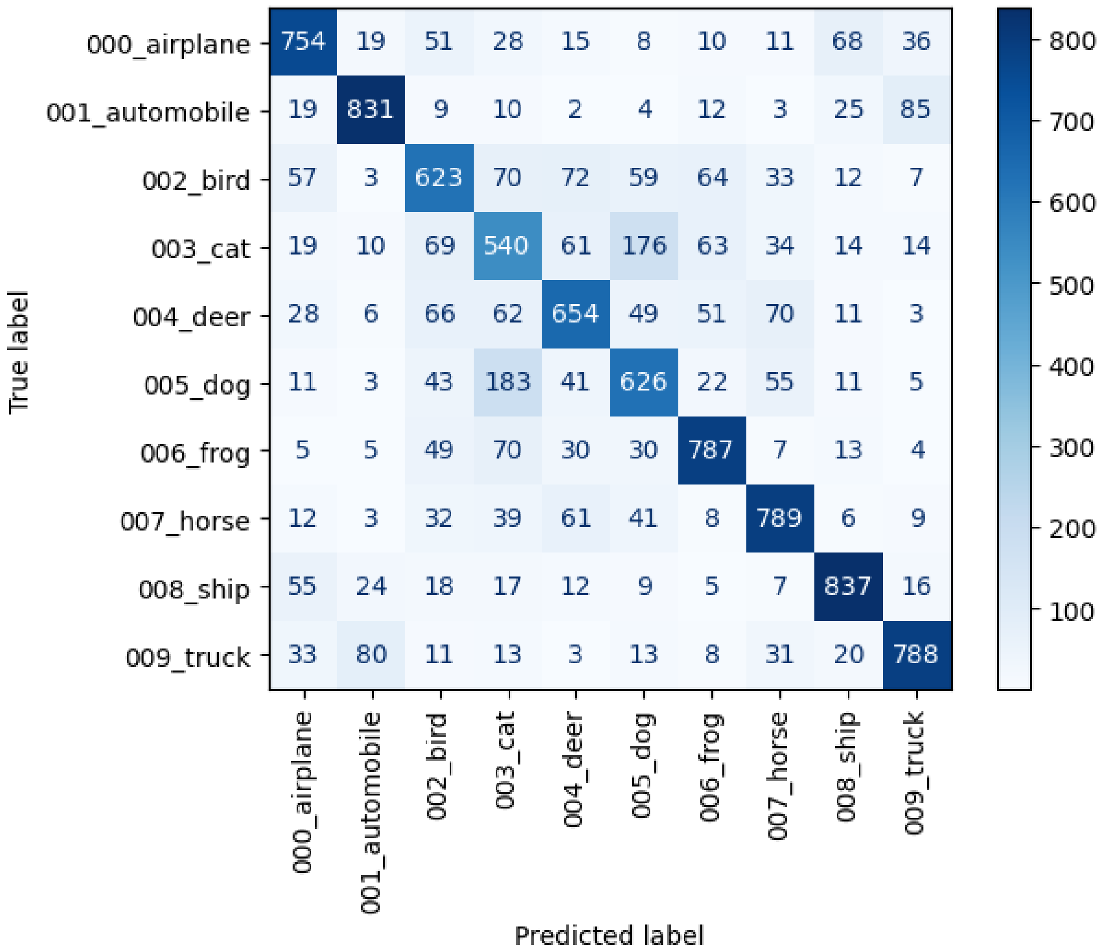
```
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
```

```
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
```

```
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 2ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
2/2 [==============================] - 0s 3ms/step
1/1 [==============================] - 0s 45ms/step
```

Confusion Matrix

[11]:
```python
cm = confusion_matrix(test_labels, test_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues, xticks_rotation=90)
plt.show()
```



- Looking at the confusion matrix, we see that:
  - The model has a hard time distinguishing the categories 003_cat and 005_dog.
  - The model has a very low performance on the category 003_cat.
  - The model performs better on the vehicle categories than on the animal categories.
  - The model has a below average performance on the categories 003_cat, 005_dog, 002_bird and 004_deer, in which we see a very high false positive rate.
  - The model also has a hard time distinguishing between some other categories but the deviation is not as significant.

  - The model has an above average performance on the categories 000_airplane, 001_automobile, 006_frog, 008_ship and 009_truck.

ROC Curve Analysis

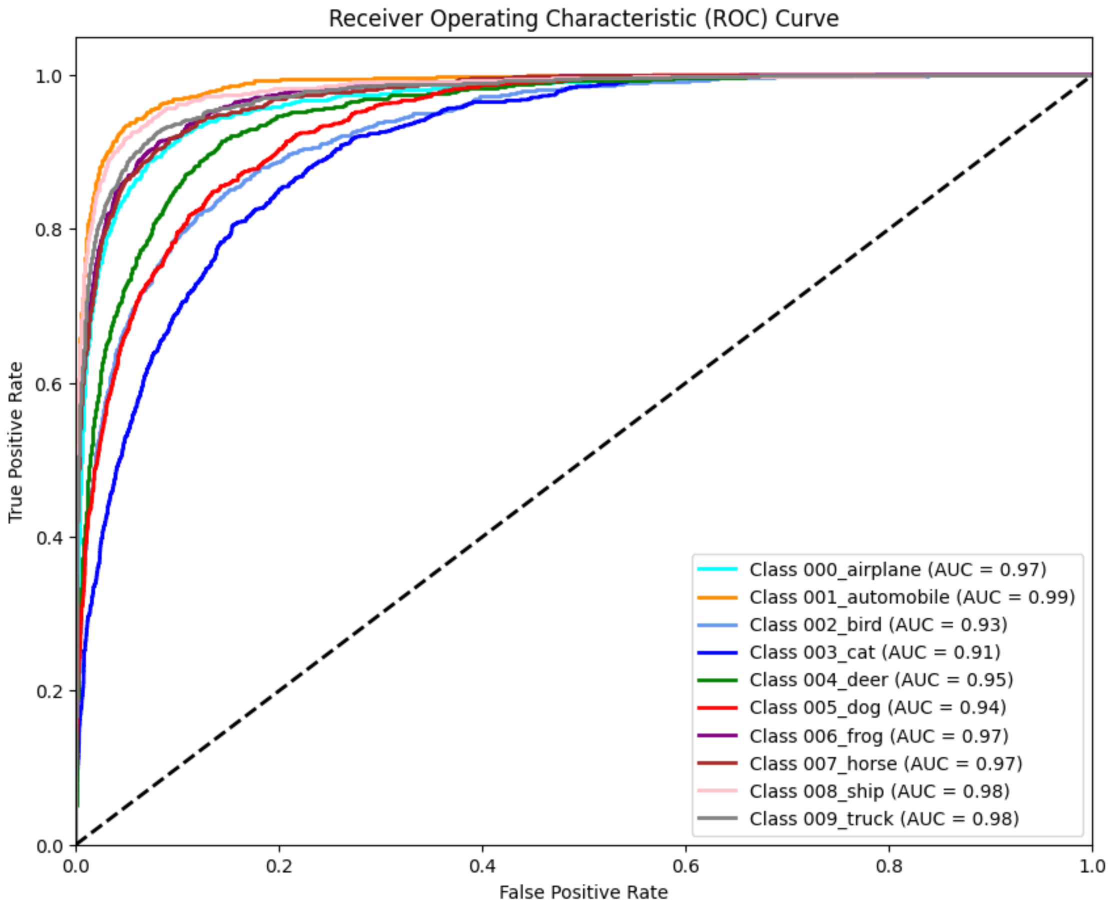
[12]:
```
test_labels_bin = label_binarize(test_labels, classes=range(NUM_CLASSES))
```

```python
false_positive_rate = dict()
true_positive_rate = dict()
roc_auc = dict()

for i in range(NUM_CLASSES):
    false_positive_rate[i], true_positive_rate[i], _ = roc_curve(test_labels_bin[:, i], test_probabilities[:, i])
    roc_auc[i] = auc(false_positive_rate[i], true_positive_rate[i])

plt.figure(figsize=(10, 8))
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'blue', 'green', 'red', 'purple', 'brown', 'pink', 'grey'])
for i, color in zip(range(NUM_CLASSES), colors):
    plt.plot(false_positive_rate[i], true_positive_rate[i], color=color, lw=2, label=f'Class {class_names[i]} (AUC = {roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```



- Looking at the ROC curve:
  - We see that the model has a mediocre performance on the ROC curve for most categories.
  - The categories 003_cat, 002_bird, 005_dog and 004_deer have the worst AUC (Area Under Curve) performance.
  - The categories 001_automobile, 006_frog, 008_ship and 009_truck have the best AUC performance.
  - **A perfect AUC of 1.0 would mean that the model classifies all images either true positives or true negatives**.

Performance Metrics

- **Accuracy** is the proportion of correctly predicted instances out of the total instances.
- **Precision** is the ratio of true positive predictions to the total predicted positives. Macro precision calculates this for each class independently and then averages them.
- **Weighted precision** calculates the precision for each class, then averages them, weighted

by the number of true instances for each class.

- **Recall** is the ratio of true positive predictions to the total actual positives. Macro recall calculates this for each class independently and then averages them.

- **Weighted recall** calculates the recall for each class, then averages them, weighted by the number of true instances for each class.

- The **F1-score** is the harmonic mean of precision and recall. Macro F1-score calculates this for each class independently and then averages them.

- **Weighted F1-score** calculates the F1-score for each class, then averages them, weighted by the number of true instances for each class.

[13]:
```
acc = accuracy_score(y_true =  test_labels, y_pred = test_predictions)
print(f'Accuracy : {np.round(acc*100,2)}%')
precision = precision_score(y_true =  test_labels, y_pred = test_predictions,␣
 ↪average='macro')
print(f'Precision - Macro: {np.round(precision*100,2)}%')
recall = recall_score(y_true =  test_labels, y_pred = test_predictions,␣
 ↪average='macro')
print(f'Recall - Macro: {np.round(recall*100,2)}%')
f1 = f1_score(y_true =  test_labels, y_pred = test_predictions, average='macro')
print(f'F1-score - Macro: {np.round(f1*100,2)}%')
precision = precision_score(y_true =  test_labels, y_pred = test_predictions,␣
 ↪average='weighted')
print(f'Precision - Weighted: {np.round(precision*100,2)}%')
recall = recall_score(y_true =  test_labels, y_pred = test_predictions,␣
 ↪average='weighted')
print(f'Recall - Weighted: {np.round(recall*100,2)}%')
f1 = f1_score(y_true =  test_labels, y_pred = test_predictions,␣
 ↪average='weighted')
print(f'F1-score - Weighted: {np.round(f1*100,2)}%')
```

```
Accuracy : 72.29%
Precision - Macro: 72.34%
Recall - Macro: 72.29%
F1-score - Macro: 72.3%
Precision - Weighted: 72.34%
Recall - Weighted: 72.29%
F1-score - Weighted: 72.3%
```

- **Since the dataset is balanced, the** MACRO** average is a good metric to evaluate the model.**

2 Conclusion

2.0.1 Summary

- In this notebook:
  - We trained a model with minimal complexity to serve as a **baseline** for future improvements.
  - We used the **Stochastic Gradient Descent (SGD)** optimizer with a **0.9** momentum and an initial learning rate of **0.01**.
  - Training during **30 epochs** with a **batch size of 64**.
  - We evaluated the model on the validation dataset:
    - Overfitting was observed after **6 epochs**, but the best model was saved at the **11th epoch**.
    - Training was intended for 30 epochs but stopped early due to the **Early Stopping** callback.
  - We evaluated the model on the test dataset:
    - We evaluated the model using a confusion matrix to analyze its performance on each category.
    - We evaluated the model using ROC curves for a deeper performance analysis.
    - The model achieved an accuracy of **72.29%** on the test set, which will serve as a baseline for future improvements.

2.0.2 Future Work

- In the next notebook:
  - We will deepen, widen and enhance the architecture by:
    - We will add 3 more convolutional layers and upscale the filters to 32, 64 and 128.
    - We will upscale the dense layer to 256.
    - We will add batch normalization after each convolution.
    - We will add dropout before each dense layer.
  - We will use the Root Mean Square Propagation (RMSprop) optimizer.
  - We will keep the same 30 epochs and batch size of 64.