02_model_s_deepen_widen_enhance_rmsprop

June 10, 2024

---

1 Model S - Deepened, Widened, Enhanced with RMSprop Optimizer

- **32 x 32 x 3** Image size.
- **64** Batch size.
- Root Mean Squared **(RMSprop)** optimizer.
- **0.001** Initial Learning rate.
- **Sparse Categorical Cross-Entropy** loss function.
- **Reduce Learning Rate on Plateau** callback with a **0.1** factor and patience of **3**.
- **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** blocks with 2 layers each of **32**, **64** and **128** filters, with **ReLU** activation.
- **Batch Normalization** after each Convolutional layer.
- 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 128** Tensor before the **Flatten** layer.
- **256** Dense layer with **ReLU** activation.
- **10** Dense output layer with **Softmax** activation.
- **Dropout** layers with **0.5** rate after the Flatten and Dense layers.
- **815 018** 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
from tensorflow import keras
from tensorflow.keras import callbacks, layers, models, regularizers
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]:
```
train_dirs = ['../data/train1', '../data/train3', '../data/train4', '../data/
↪train5']
validation_dir = '../data/train2'
test_dir = '../data/test'
```

Create Datasets

[3]:
```
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]:
```
inputs = keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = layers.Rescaling(1./255)(inputs)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu",␣
 ↪padding="same")(x)
x = layers.BatchNormalization()(x)
x = layers.MaxPooling2D(pool_size=2)(x)
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(256, activation="relu")(x)
x = layers.Dropout(0.5)(x)
```

```python
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, 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)

 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)

 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)

 flatten (Flatten)           (None, 2048)              0

 dropout (Dropout)           (None, 2048)              0

 dense (Dense)               (None, 256)               524544

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 10)                2570

=================================================================
Total params: 815914 (3.11 MB)
Trainable params: 815018 (3.11 MB)
Non-trainable params: 896 (3.50 KB)
_________________________________________________________________
```

Model Compilation

[5]:
```python
initial_learning_rate = 0.001
optimizer = tf.keras.optimizers.RMSprop(learning_rate=initial_learning_rate)
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/
 ↪02_model_s_deepen_widen_enhance_rmsprop.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
628/628 [==============================] - ETA: 0s - loss: 1.7873 - accuracy:
0.3832
Epoch 1: val_loss improved from inf to 1.36833, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 59s 92ms/step - loss: 1.7873 -
accuracy: 0.3832 - val_loss: 1.3683 - val_accuracy: 0.5059 - lr: 0.0010
Epoch 2/30
628/628 [==============================] - ETA: 0s - loss: 1.3130 - accuracy:
0.5365
Epoch 2: val_loss improved from 1.36833 to 1.29688, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 56s 89ms/step - loss: 1.3130 -
accuracy: 0.5365 - val_loss: 1.2969 - val_accuracy: 0.5659 - lr: 0.0010
Epoch 3/30
628/628 [==============================] - ETA: 0s - loss: 1.1322 - accuracy:
0.6149
Epoch 3: val_loss improved from 1.29688 to 1.06904, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 56s 89ms/step - loss: 1.1322 -
accuracy: 0.6149 - val_loss: 1.0690 - val_accuracy: 0.6288 - lr: 0.0010
Epoch 4/30
628/628 [==============================] - ETA: 0s - loss: 1.0178 - accuracy:
0.6607
Epoch 4: val_loss did not improve from 1.06904
628/628 [==============================] - 56s 89ms/step - loss: 1.0178 -
accuracy: 0.6607 - val_loss: 1.6596 - val_accuracy: 0.5828 - lr: 0.0010
Epoch 5/30
628/628 [==============================] - ETA: 0s - loss: 0.9234 - accuracy:
0.6895
Epoch 5: val_loss improved from 1.06904 to 0.86893, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 57s 90ms/step - loss: 0.9234 -
accuracy: 0.6895 - val_loss: 0.8689 - val_accuracy: 0.7061 - lr: 0.0010
Epoch 6/30
628/628 [==============================] - ETA: 0s - loss: 0.8560 - accuracy:
0.7151
Epoch 6: val_loss did not improve from 0.86893
628/628 [==============================] - 57s 90ms/step - loss: 0.8560 -
accuracy: 0.7151 - val_loss: 1.0259 - val_accuracy: 0.6498 - lr: 0.0010
Epoch 7/30
628/628 [==============================] - ETA: 0s - loss: 0.7856 - accuracy:
0.7387
Epoch 7: val_loss did not improve from 0.86893
628/628 [==============================] - 57s 90ms/step - loss: 0.7856 -
```

```
accuracy: 0.7387 - val_loss: 0.9757 - val_accuracy: 0.6763 - lr: 0.0010
Epoch 8/30
628/628 [==============================] - ETA: 0s - loss: 0.7346 - accuracy:
0.7562
Epoch 8: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.

Epoch 8: val_loss did not improve from 0.86893
628/628 [==============================] - 57s 90ms/step - loss: 0.7346 -
accuracy: 0.7562 - val_loss: 0.8880 - val_accuracy: 0.7353 - lr: 0.0010
Epoch 9/30
628/628 [==============================] - ETA: 0s - loss: 0.5625 - accuracy:
0.8088
Epoch 9: val_loss improved from 0.86893 to 0.60316, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 57s 90ms/step - loss: 0.5625 -
accuracy: 0.8088 - val_loss: 0.6032 - val_accuracy: 0.8002 - lr: 1.0000e-04
Epoch 10/30
628/628 [==============================] - ETA: 0s - loss: 0.5149 - accuracy:
0.8235
Epoch 10: val_loss did not improve from 0.60316
628/628 [==============================] - 57s 89ms/step - loss: 0.5149 -
accuracy: 0.8235 - val_loss: 0.6111 - val_accuracy: 0.8058 - lr: 1.0000e-04
Epoch 11/30
628/628 [==============================] - ETA: 0s - loss: 0.4914 - accuracy:
0.8288
Epoch 11: val_loss did not improve from 0.60316
628/628 [==============================] - 57s 91ms/step - loss: 0.4914 -
accuracy: 0.8288 - val_loss: 0.6275 - val_accuracy: 0.8072 - lr: 1.0000e-04
Epoch 12/30
628/628 [==============================] - ETA: 0s - loss: 0.4745 - accuracy:
0.8357
Epoch 12: ReduceLROnPlateau reducing learning rate to 1.0000000474974514e-05.

Epoch 12: val_loss did not improve from 0.60316
628/628 [==============================] - 57s 89ms/step - loss: 0.4745 -
accuracy: 0.8357 - val_loss: 0.6083 - val_accuracy: 0.8092 - lr: 1.0000e-04
Epoch 13/30
628/628 [==============================] - ETA: 0s - loss: 0.4492 - accuracy:
0.8409
Epoch 13: val_loss improved from 0.60316 to 0.59494, saving model to
../models/02_model_s_deepen_widen_enhance_rmsprop.keras
628/628 [==============================] - 57s 90ms/step - loss: 0.4492 -
accuracy: 0.8409 - val_loss: 0.5949 - val_accuracy: 0.8109 - lr: 1.0000e-05
Epoch 14/30
628/628 [==============================] - ETA: 0s - loss: 0.4424 - accuracy:
0.8459
Epoch 14: val_loss did not improve from 0.59494
628/628 [==============================] - 56s 88ms/step - loss: 0.4424 -
```

```
accuracy: 0.8459 - val_loss: 0.5965 - val_accuracy: 0.8118 - lr: 1.0000e-05
Epoch 15/30
628/628 [==============================] - ETA: 0s - loss: 0.4368 - accuracy:
0.8472
Epoch 15: val_loss did not improve from 0.59494
628/628 [==============================] - 55s 87ms/step - loss: 0.4368 -
accuracy: 0.8472 - val_loss: 0.5969 - val_accuracy: 0.8109 - lr: 1.0000e-05
Epoch 16/30
628/628 [==============================] - ETA: 0s - loss: 0.4400 - accuracy:
0.8483
Epoch 16: ReduceLROnPlateau reducing learning rate to 1.0000000656873453e-06.

Epoch 16: val_loss did not improve from 0.59494
628/628 [==============================] - 55s 87ms/step - loss: 0.4400 -
accuracy: 0.8483 - val_loss: 0.5996 - val_accuracy: 0.8120 - lr: 1.0000e-05
Epoch 17/30
628/628 [==============================] - ETA: 0s - loss: 0.4391 - accuracy:
0.8448
Epoch 17: val_loss did not improve from 0.59494
628/628 [==============================] - 56s 88ms/step - loss: 0.4391 -
accuracy: 0.8448 - val_loss: 0.5983 - val_accuracy: 0.8130 - lr: 1.0000e-06
Epoch 18/30
628/628 [==============================] - ETA: 0s - loss: 0.4306 - accuracy:
0.8497
Epoch 18: val_loss did not improve from 0.59494
628/628 [==============================] - 55s 87ms/step - loss: 0.4306 -
accuracy: 0.8497 - val_loss: 0.5988 - val_accuracy: 0.8133 - lr: 1.0000e-06
Epoch 19/30
628/628 [==============================] - ETA: 0s - loss: 0.4344 - accuracy:
0.8494
Epoch 19: ReduceLROnPlateau reducing learning rate to 1.0000001111620805e-07.
Restoring model weights from the end of the best epoch: 13.

Epoch 19: val_loss did not improve from 0.59494
628/628 [==============================] - 56s 89ms/step - loss: 0.4344 -
accuracy: 0.8494 - val_loss: 0.5999 - val_accuracy: 0.8125 - lr: 1.0000e-06
Epoch 19: early stopping
```

Save Model History

[7]:
```
with open("../history/02_model_s_deepen_widen_enhance_rmsprop.pkl", "wb") as file:
    pickle.dump(history.history, file)
```

Model Evaluation

[8]:
```
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 [==============================] - 3s 16ms/step - loss: 0.5949 -
accuracy: 0.8109
Model Validation Loss: 0.59
Model Validation Accuracy: 81.09%
```

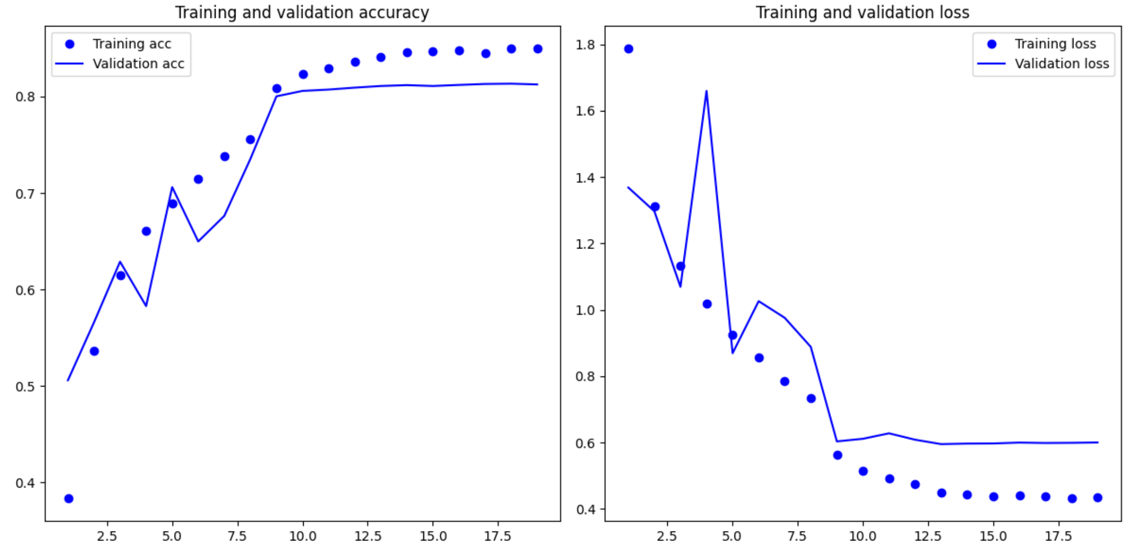
Model Training Visualization

[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 **10th** epoch.
  - The validation accuracy stops improving significantly after the **10th** epoch while the training accuracy keeps improving.
  - The validation loss stops improving significantly after the **9th** epoch while the training loss keeps improving.
  - The best model, based on validation loss, is saved on the **16th** epoch.
  - The training stops after the **19th** 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 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
```

```
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 8ms/step
```

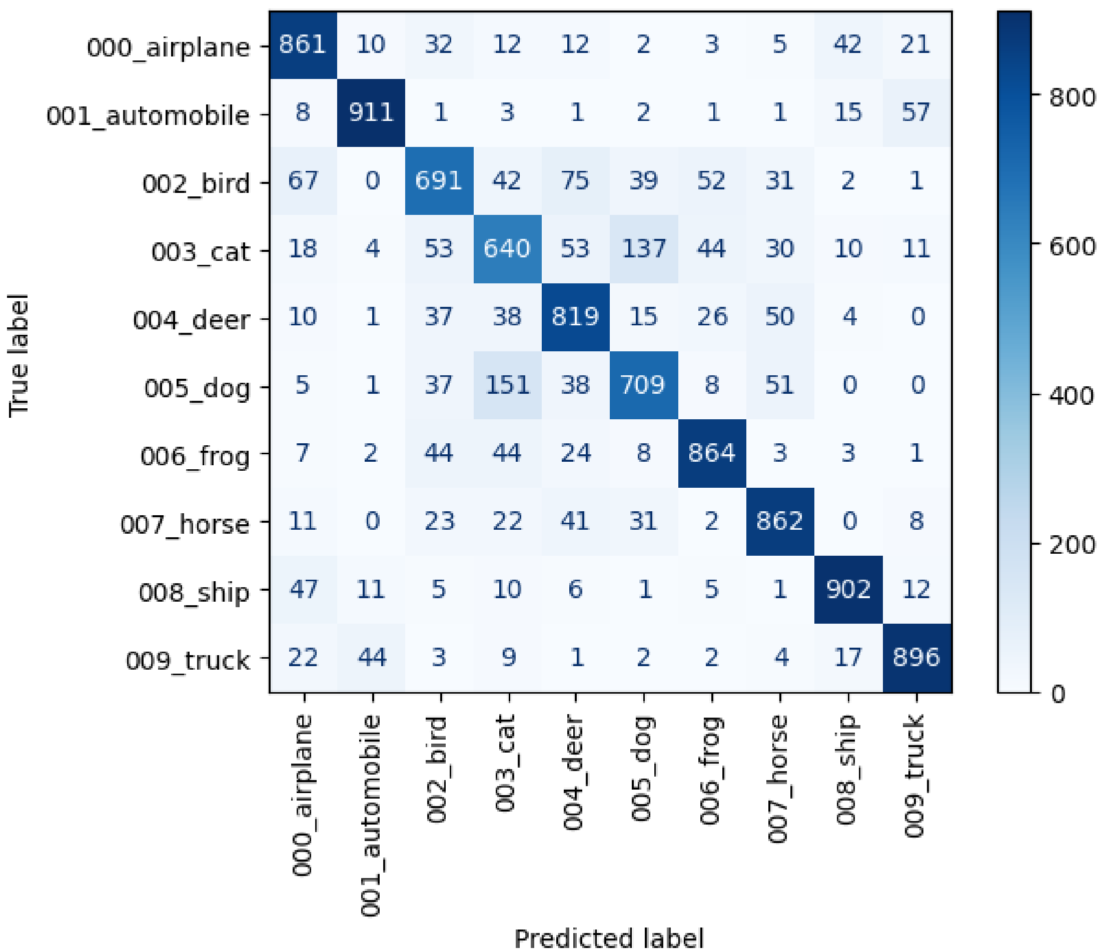
```
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
```

```
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
```

```
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 9ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 10ms/step
2/2 [==============================] - 0s 11ms/step
2/2 [==============================] - 0s 9ms/step
1/1 [==============================] - 0s 101ms/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 005_dog, 002_bird and 003_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.
  - **Basically, the model has the same error distribution but with higher accuracy**.

ROC Curve Analysis

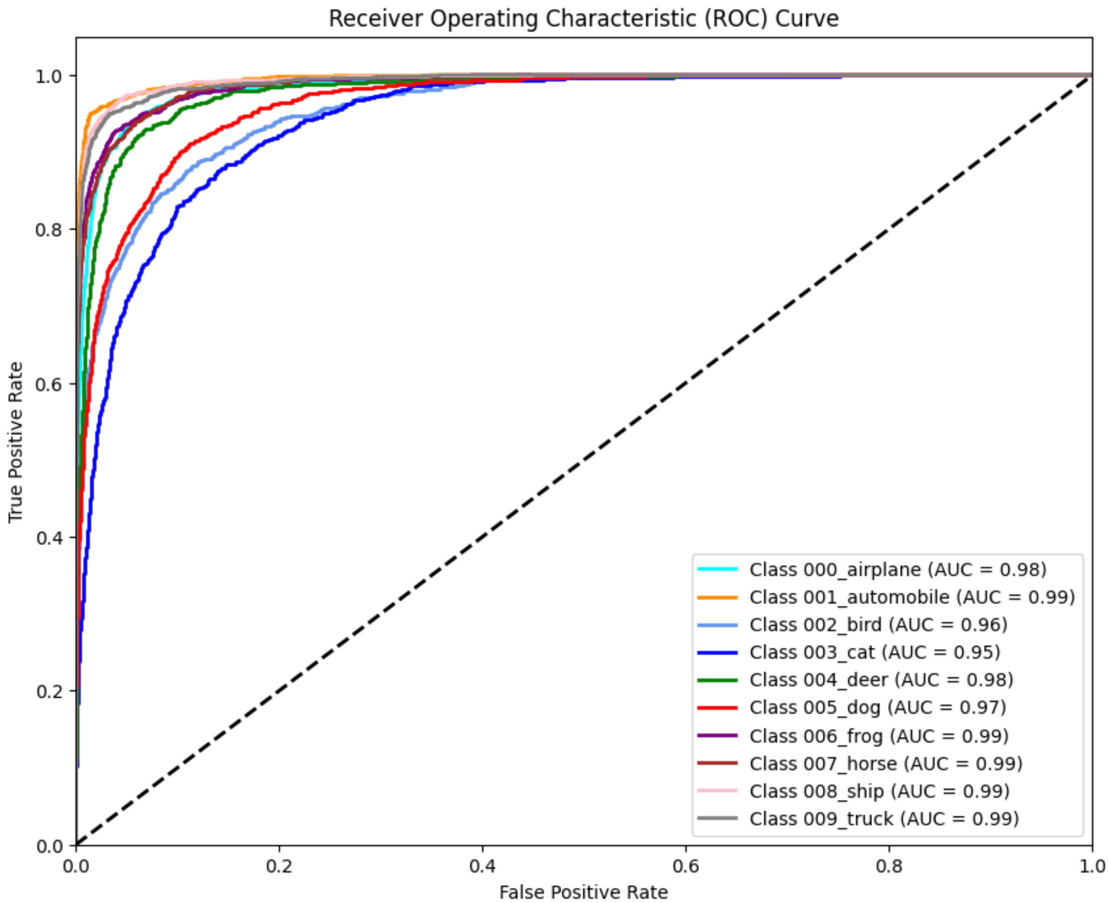
[12]:
```
test_labels_bin = label_binarize(test_labels, classes=range(NUM_CLASSES))

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 decent performance on the ROC curve for most of the categories.

  - The categories 003_cat, 002_bird, 005_dog and 004_deer have the worst AUC (Area Under Curve) performance.
  - The other categories have the same performance but with higher AUC.
  - The category 001_automobile has the best AUC performance.
  - The overall AUC performance increases as the false positive rate decreases and the true positive rate increases.
  - **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 : 81.55%
Precision - Macro: 81.46%
Recall - Macro: 81.55%
F1-score - Macro: 81.47%
Precision - Weighted: 81.46%
Recall - Weighted: 81.55%
F1-score - Weighted: 81.47%
```

- **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 deepened, widened and enhanced the architecture by:
    * We added 3 more convolutional layers.
    * We used a 256 unit dense layer.
    * We added batch normalization after each convolution.
    * We used a dropout rate of 0.5 after the flatten and dense layers.
  - We used the Root Mean Squared Propagation (RMSProp) optimizer with an initial learning rate of 0.001.
  - We kept the same 30 epochs with a batch size of 64.
  - We evaluated the model on the validation dataset:
    * Overfitting was observed after **10 epochs**, but the best model was saved at the **16th 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 **81.55%** on the test set, which is a noticeable improvement.

2.0.2 Future Work

- In the next notebook:
  - We will implement a few measures to improve the model's performance:
    * We will use data augmentation techniques.
    * We will use L2 regularization to both the convolutional and dense layers.
  - We will use the Adaptive Moment Estimation (Adam) optimizer.
  - We will train for 40 epochs this time, with a batch size of 64.