part3-submission

July 16, 2024

MSA 2024 Phase 2 - Part 3

Welcome to the competition - in Part 3, you are encouraged to utilize neural network based models for classification.

This notebook builds a simple Multi-Layer Perceptron (MLP) model for the CIFAR-10 dataset, with the use of `keras` to define the model structure.

Before start working on the competition, please ensure all required libraries are installed and properly set up on your system:

- `python >= 3.6`,
- `tensorFlow >= 2.0`,
- `keras >= 2.3`,

and any neccassary liburaries for data manipulation and processing, e.g., `numpy`, `pandas`, etc.

[65]:
```
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.models import Sequential, Model
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Dropout, Flatten, BatchNormalization, Input
from keras.regularizers import l2
from keras.optimizers import Adam
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

1. Data loading & preprocessing

The CIFAR-10 dataset contains 60,000 images(32x32x3) in 10 different classes, with 6,000 images in each class. You can download the dataset directly from the competition webpage.

To train the model, you are expected to use the training label provided in train.csv.

[66]:
```
train_dir = './dataset/cifar10_images/train'
test_dir = './dataset/cifar10_images/test'
```

[75]:
```
def loadTrain(root_dir, csv_file):
    ids = []
    images = []
    labels = []
    annotations = np.genfromtxt(csv_file, delimiter=',', names=True)
    for idx in range(len(annotations)):
        img_id = int(annotations['id'][idx])
        img_name = os.path.join(root_dir, f"image_{img_id}.png")
        image = np.array(Image.open(img_name).convert("RGB"))
        label = int(annotations['label'][idx])

        ids.append(img_id)
        images.append(image)
        labels.append(label)
    return np.array(ids), np.array(images), np.array(labels)

def loadTest(root_dir):
    ids = []
    images = []
    for idx in range(len(os.listdir(root_dir))):
        img_name = os.path.join(root_dir, f"image_{idx}.png")
        image = np.array(Image.open(img_name).convert("RGB"))

        ids.append(idx)
        images.append(image)
    return np.array(ids), np.array(images)


# Load training, testing data and the training label provided in train.csv.
train_csv = './dataset/train.csv'
id_train, X_train, y_train = loadTrain(train_dir, train_csv)
id_test, X_test = loadTest(test_dir)

# Normalize the data
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')

X_train_split, X_val_split, y_train_split, y_val_split =␣
 ↪train_test_split(X_train, y_train, test_size=0.1, random_state=101)

# Calculate the mean and standard deviation of the training images
mean = np.mean(X_train_split)
std  = np.std(X_train_split)

# Normalize the data
# The tiny value 1e-7 is added to prevent division by zero
X_train_split = (X_train_split-mean)/(std+1e-7)
```

```
X_val_split = (X_val_split-mean)/(std+1e-7)

mean_test = np.mean(X_test)
std_test  = np.std(X_test)
X_test  = (X_test-mean_test) /(std_test+1e-7)

# Convert training labels to one-hot encoded vectors.
y_train_split = to_categorical(y_train_split, 10)
y_val_split=  to_categorical(y_val_split, 10)
```

Print the shape of loaded training and testing sets. The dataset is downloaded from the kaggle competition, it has only 55000 images, 50000 for training and 5000 for testing.

[68]:
```
print(X_train_split.shape)
print(y_train_split.shape)
print(X_val_split.shape)
print(y_val_split.shape)
print(X_test.shape)
```

```
(45000, 32, 32, 3)
(45000, 10)
(5000, 32, 32, 3)
(5000, 10)
(5000, 32, 32, 3)
```

[69]:
```
# Plot the first 20 images with their labels
def plot_images(images, labels, num_images=20):
    plt.figure(figsize=(20, 10))
    for i in range(num_images):
        plt.subplot(4, 5, i + 1)
        plt.imshow(images[i])
        plt.title(f"Label: {labels[i]}")
        plt.axis('off')
    plt.show()

y_train_labels = np.argmax(y_train_split, axis=1)
plot_images(X_train_split, y_train_labels, num_images=20)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.7583104].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.2966118..1.9141738].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7642019..1.8518285].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.8206558].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.0101662].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
```

```
floats or [0..255] for integers). Got range [-1.85772..1.8830012].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..2.1012099].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..2.0388646].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7642019..1.7583104].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.3277844..2.0388646].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.8830012].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7018566..2.0076919].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.8830012].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..1.9765192].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.6706839..1.9765192].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.3589572..2.1012099].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7018566..1.3530656].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..2.1012099].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.7330292..2.0076919].
Clipping input data to the valid range for imshow with RGB data ([0..1] for
floats or [0..255] for integers). Got range [-1.8888927..2.0388646].
```

2. Build & train the model

I am going to build and train a CNN model.

[70]:
```
# Data augmentation
data_generator = ImageDataGenerator(
    # Rotate images randomly by up to 20 degrees
    rotation_range=20,

    # Shift images horizontally by up to 15%
    width_shift_range=0.15,

    # Shift images vertically by up to 15%
    height_shift_range=0.15,

    # Randomly flip images horizontally
    horizontal_flip=True,

    # Zoom images in by up to 20%
    zoom_range=0.2,

    # Change brightness by up to 10%
    brightness_range=[0.9,1.1],

    # Shear intensity (shear angle in counter-clockwise direction in degrees)
    shear_range=10,

)
```

[71]:
```
# Initialize a sequential model
model = Sequential()

# Set the weight decay value for L2 regularization
weight_decay = 0.00008

# Add the first convolutional layer with 32 filters of size 3x3
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay),
                 input_shape=X_train.shape[1:]))
# Add batch normalization layer
model.add(BatchNormalization())

# Add the second convolutional layer similar to the first
model.add(Conv2D(filters=32, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
```

```
model.add(BatchNormalization())

# Add the first max pooling layer with pool size of 2x2
model.add(MaxPooling2D(pool_size=(2, 2)))
# Add dropout layer with 0.2 dropout rate
model.add(Dropout(rate=0.3))

# Add the third and fourth convolutional layers with 64 filters
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())

# Add the second max pooling layer and increase dropout rate to 0.3
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))

# Add the fifth and sixth convolutional layers with 128 filters
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())

# Add the third max pooling layer and increase dropout rate to 0.4
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))

# Add the seventh and eighth convolutional layers with 256 filters
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())
model.add(Conv2D(filters=256, kernel_size=(3,3), padding='same',␣
 ↪activation='relu', kernel_regularizer=l2(weight_decay)))
model.add(BatchNormalization())

# Add the fourth max pooling layer and increase dropout rate to 0.5
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.3))

# Flatten the tensor output from the previous layer
model.add(Flatten())
```

```
# Add a fully connected layer with softmax activation function for outputting␣
 ↪class probabilities
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
/Users/jxiao/Desktop/code/msa/phase2/msa2/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_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_128 (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_129 (Dropout)	(None, 8, 8, 64)	0
conv2d_4 (Conv2D)	(None, 8, 8, 128)	73,856
batch_normalization_4	(None, 8, 8, 128)	512

```
(BatchNormalization)

conv2d_5 (Conv2D)                    (None, 8, 8, 128)              147,584

batch_normalization_5                (None, 8, 8, 128)                  512
(BatchNormalization)

max_pooling2d_2 (MaxPooling2D)       (None, 4, 4, 128)                    0

dropout_130 (Dropout)                (None, 4, 4, 128)                    0

conv2d_6 (Conv2D)                    (None, 4, 4, 256)              295,168

batch_normalization_6                (None, 4, 4, 256)                1,024
(BatchNormalization)

conv2d_7 (Conv2D)                    (None, 4, 4, 256)              590,080

batch_normalization_7                (None, 4, 4, 256)                1,024
(BatchNormalization)

max_pooling2d_3 (MaxPooling2D)       (None, 2, 2, 256)                    0

dropout_131 (Dropout)                (None, 2, 2, 256)                    0

flatten (Flatten)                    (None, 1024)                         0

dense_128 (Dense)                    (None, 10)                      10,250


Total params: 1,186,346 (4.53 MB)

Trainable params: 1,184,426 (4.52 MB)

Non-trainable params: 1,920 (7.50 KB)
```

[72]:
```python
# Set the batch size for the training
batch_size = 256

# Set the maximum number of epochs for the training
epochs = 300

# Define the optimizer (Adam)
optimizer = Adam(learning_rate=0.003)
```

```
train_generator = data_generator.flow(X_train_split, y_train_split,␣
 ↪batch_size=batch_size)
val_generator = data_generator.flow(X_val_split, y_val_split,␣
 ↪batch_size=batch_size)



# Compile the model with the defined optimizer, loss function, and metrics
model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Add ReduceLROnPlateau callbackå
# Here, the learning rate will be reduced by half (factor=0.5) if no␣
 ↪improvement in validation loss is observed for 10 epochs
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=10,␣
 ↪min_lr=0.00001)

# Add EarlyStopping callback
# Here, training will be stopped if no improvement in validation loss is␣
 ↪observed for 40 epochs.
# The `restore_best_weights` parameter ensures that the model weights are reset␣
 ↪to the values from the epoch
# with the best value of the monitored quantity (in this case, 'val_loss').
early_stopping = EarlyStopping(monitor='val_loss', patience=40,␣
 ↪restore_best_weights=True, verbose=1)

# Fit the model on the training data, using the defined batch size and number␣
 ↪of epochs
# The validation data is used to evaluate the model's performance during␣
 ↪training
# The callbacks implemented are learning rate reduction when a plateau is␣
 ↪reached in validation loss and
# stopping training early if no improvement is observed
model.fit(train_generator,
          epochs=epochs,
          validation_data=val_generator,
          callbacks=[reduce_lr, early_stopping])
```

```
/Users/jxiao/Desktop/code/msa/phase2/msa2/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
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()
```

```
Epoch 1/300
```

```
176/176            0s 665ms/step -
accuracy: 0.2614 - loss: 3.0764

/Users/jxiao/Desktop/code/msa/phase2/msa2/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121:
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()

176/176            123s 683ms/step -
accuracy: 0.2617 - loss: 3.0721 - val_accuracy: 0.2318 - val_loss: 2.5065 -
learning_rate: 0.0030
Epoch 2/300
176/176            117s 653ms/step -
accuracy: 0.4121 - loss: 1.8134 - val_accuracy: 0.4464 - val_loss: 1.6447 -
learning_rate: 0.0030
Epoch 3/300
176/176            115s 651ms/step -
accuracy: 0.4617 - loss: 1.6653 - val_accuracy: 0.3220 - val_loss: 2.1884 -
learning_rate: 0.0030
Epoch 4/300
176/176            115s 652ms/step -
accuracy: 0.4537 - loss: 1.7124 - val_accuracy: 0.4394 - val_loss: 1.9262 -
learning_rate: 0.0030
Epoch 5/300
176/176            115s 655ms/step -
accuracy: 0.4541 - loss: 1.7847 - val_accuracy: 0.3496 - val_loss: 2.1553 -
learning_rate: 0.0030
Epoch 6/300
176/176            1074s 6s/step -
accuracy: 0.4411 - loss: 1.8765 - val_accuracy: 0.5070 - val_loss: 1.5949 -
learning_rate: 0.0030
Epoch 7/300
176/176            673s 4s/step -
accuracy: 0.4876 - loss: 1.7161 - val_accuracy: 0.5510 - val_loss: 1.5036 -
learning_rate: 0.0030
Epoch 8/300
176/176            1090s 6s/step -
accuracy: 0.5118 - loss: 1.6484 - val_accuracy: 0.5098 - val_loss: 1.6806 -
learning_rate: 0.0030
Epoch 9/300
176/176            636s 4s/step -
accuracy: 0.5197 - loss: 1.6971 - val_accuracy: 0.5596 - val_loss: 1.5134 -
learning_rate: 0.0030
Epoch 10/300
176/176            1107s 6s/step -
accuracy: 0.5410 - loss: 1.5998 - val_accuracy: 0.4816 - val_loss: 1.9048 -
```

```
learning_rate: 0.0030
Epoch 11/300
176/176             1067s 6s/step -
accuracy: 0.5402 - loss: 1.6073 - val_accuracy: 0.5616 - val_loss: 1.5124 -
learning_rate: 0.0030
Epoch 12/300
176/176             113s 644ms/step -
accuracy: 0.5294 - loss: 1.6760 - val_accuracy: 0.5484 - val_loss: 1.5651 -
learning_rate: 0.0030
Epoch 13/300
176/176             1038s 6s/step -
accuracy: 0.5646 - loss: 1.5629 - val_accuracy: 0.5488 - val_loss: 1.5682 -
learning_rate: 0.0030
Epoch 14/300
176/176             611s 3s/step -
accuracy: 0.5101 - loss: 1.7425 - val_accuracy: 0.5122 - val_loss: 1.9654 -
learning_rate: 0.0030
Epoch 15/300
176/176             113s 640ms/step -
accuracy: 0.5242 - loss: 1.7312 - val_accuracy: 0.5646 - val_loss: 1.5943 -
learning_rate: 0.0030
Epoch 16/300
176/176             2022s 12s/step -
accuracy: 0.5380 - loss: 1.7026 - val_accuracy: 0.5930 - val_loss: 1.4998 -
learning_rate: 0.0030
Epoch 17/300
176/176             1097s 6s/step -
accuracy: 0.5852 - loss: 1.5353 - val_accuracy: 0.5978 - val_loss: 1.4976 -
learning_rate: 0.0030
Epoch 18/300
176/176             1983s 11s/step -
accuracy: 0.5662 - loss: 1.6010 - val_accuracy: 0.6004 - val_loss: 1.4902 -
learning_rate: 0.0030
Epoch 19/300
176/176             1979s 11s/step -
accuracy: 0.6080 - loss: 1.4489 - val_accuracy: 0.6332 - val_loss: 1.3722 -
learning_rate: 0.0030
Epoch 20/300
176/176             1040s 6s/step -
accuracy: 0.6216 - loss: 1.3907 - val_accuracy: 0.6362 - val_loss: 1.3370 -
learning_rate: 0.0030
Epoch 21/300
176/176             1079s 6s/step -
accuracy: 0.6306 - loss: 1.3428 - val_accuracy: 0.6484 - val_loss: 1.2940 -
learning_rate: 0.0030
Epoch 22/300
176/176             2066s 12s/step -
accuracy: 0.6439 - loss: 1.2890 - val_accuracy: 0.6674 - val_loss: 1.2231 -
```

```
learning_rate: 0.0030
Epoch 23/300
176/176             114s 646ms/step -
accuracy: 0.6613 - loss: 1.2361 - val_accuracy: 0.6018 - val_loss: 1.3950 -
learning_rate: 0.0030
Epoch 24/300
176/176             1128s 6s/step -
accuracy: 0.6690 - loss: 1.2148 - val_accuracy: 0.6184 - val_loss: 1.3398 -
learning_rate: 0.0030
Epoch 25/300
176/176             1087s 6s/step -
accuracy: 0.6741 - loss: 1.1942 - val_accuracy: 0.6608 - val_loss: 1.2347 -
learning_rate: 0.0030
Epoch 26/300
176/176             965s 6s/step -
accuracy: 0.6828 - loss: 1.1602 - val_accuracy: 0.6794 - val_loss: 1.1813 -
learning_rate: 0.0030
Epoch 27/300
176/176             1067s 6s/step -
accuracy: 0.6839 - loss: 1.1586 - val_accuracy: 0.7186 - val_loss: 1.0698 -
learning_rate: 0.0030
Epoch 28/300
176/176             1735s 10s/step -
accuracy: 0.6945 - loss: 1.1359 - val_accuracy: 0.6750 - val_loss: 1.3363 -
learning_rate: 0.0030
Epoch 29/300
176/176             1123s 6s/step -
accuracy: 0.6850 - loss: 1.2146 - val_accuracy: 0.6892 - val_loss: 1.2452 -
learning_rate: 0.0030
Epoch 30/300
176/176             113s 640ms/step -
accuracy: 0.6982 - loss: 1.1779 - val_accuracy: 0.6866 - val_loss: 1.1966 -
learning_rate: 0.0030
Epoch 31/300
176/176             1022s 6s/step -
accuracy: 0.7152 - loss: 1.1145 - val_accuracy: 0.7366 - val_loss: 1.0422 -
learning_rate: 0.0030
Epoch 32/300
176/176             113s 641ms/step -
accuracy: 0.7237 - loss: 1.0753 - val_accuracy: 0.7502 - val_loss: 1.0111 -
learning_rate: 0.0030
Epoch 33/300
176/176             1017s 6s/step -
accuracy: 0.7308 - loss: 1.0449 - val_accuracy: 0.7032 - val_loss: 1.1216 -
learning_rate: 0.0030
Epoch 34/300
176/176             113s 639ms/step -
accuracy: 0.7283 - loss: 1.0412 - val_accuracy: 0.7298 - val_loss: 1.0457 -
```

```
learning_rate: 0.0030
Epoch 35/300
176/176             764s 4s/step -
accuracy: 0.7341 - loss: 1.0344 - val_accuracy: 0.7500 - val_loss: 0.9720 -
learning_rate: 0.0030
Epoch 36/300
176/176             175s 999ms/step -
accuracy: 0.7404 - loss: 1.0089 - val_accuracy: 0.7496 - val_loss: 0.9956 -
learning_rate: 0.0030
Epoch 37/300
176/176             1105s 6s/step -
accuracy: 0.7381 - loss: 1.0098 - val_accuracy: 0.7310 - val_loss: 1.0366 -
learning_rate: 0.0030
Epoch 38/300
176/176             883s 5s/step -
accuracy: 0.7439 - loss: 0.9980 - val_accuracy: 0.7444 - val_loss: 1.0277 -
learning_rate: 0.0030
Epoch 39/300
176/176             112s 637ms/step -
accuracy: 0.7552 - loss: 0.9745 - val_accuracy: 0.7410 - val_loss: 1.0020 -
learning_rate: 0.0030
Epoch 40/300
176/176             798s 5s/step -
accuracy: 0.7521 - loss: 0.9815 - val_accuracy: 0.7560 - val_loss: 0.9666 -
learning_rate: 0.0030
Epoch 41/300
176/176             1020s 6s/step -
accuracy: 0.7526 - loss: 0.9826 - val_accuracy: 0.7576 - val_loss: 0.9666 -
learning_rate: 0.0030
Epoch 42/300
176/176             1155s 7s/step -
accuracy: 0.7609 - loss: 0.9606 - val_accuracy: 0.7742 - val_loss: 0.9429 -
learning_rate: 0.0030
Epoch 43/300
176/176             113s 639ms/step -
accuracy: 0.7567 - loss: 0.9665 - val_accuracy: 0.7710 - val_loss: 0.9397 -
learning_rate: 0.0030
Epoch 44/300
176/176             1311s 2s/step -
accuracy: 0.7631 - loss: 0.9491 - val_accuracy: 0.7570 - val_loss: 0.9834 -
learning_rate: 0.0030
Epoch 45/300
176/176             1076s 6s/step -
accuracy: 0.7638 - loss: 0.9537 - val_accuracy: 0.7570 - val_loss: 0.9631 -
learning_rate: 0.0030
Epoch 46/300
176/176             1089s 6s/step -
accuracy: 0.7639 - loss: 0.9569 - val_accuracy: 0.7864 - val_loss: 0.9128 -
```

```
learning_rate: 0.0030
Epoch 47/300
176/176             1057s 6s/step -
accuracy: 0.7645 - loss: 0.9529 - val_accuracy: 0.7696 - val_loss: 0.9420 -
learning_rate: 0.0030
Epoch 48/300
176/176             463s 3s/step -
accuracy: 0.7700 - loss: 0.9360 - val_accuracy: 0.7812 - val_loss: 0.9263 -
learning_rate: 0.0030
Epoch 49/300
176/176             1049s 6s/step -
accuracy: 0.7737 - loss: 0.9258 - val_accuracy: 0.7608 - val_loss: 0.9529 -
learning_rate: 0.0030
Epoch 50/300
176/176             1089s 6s/step -
accuracy: 0.7753 - loss: 0.9239 - val_accuracy: 0.7848 - val_loss: 0.9058 -
learning_rate: 0.0030
Epoch 51/300
176/176             114s 645ms/step -
accuracy: 0.7720 - loss: 0.9266 - val_accuracy: 0.7714 - val_loss: 0.9716 -
learning_rate: 0.0030
Epoch 52/300
176/176             1094s 6s/step -
accuracy: 0.7791 - loss: 0.9056 - val_accuracy: 0.7688 - val_loss: 0.9773 -
learning_rate: 0.0030
Epoch 53/300
176/176             255s 1s/step -
accuracy: 0.7761 - loss: 0.9096 - val_accuracy: 0.7884 - val_loss: 0.8974 -
learning_rate: 0.0030
Epoch 54/300
176/176             1032s 6s/step -
accuracy: 0.7815 - loss: 0.9080 - val_accuracy: 0.7822 - val_loss: 0.9050 -
learning_rate: 0.0030
Epoch 55/300
176/176             115s 651ms/step -
accuracy: 0.7778 - loss: 0.9043 - val_accuracy: 0.8004 - val_loss: 0.8697 -
learning_rate: 0.0030
Epoch 56/300
176/176             115s 651ms/step -
accuracy: 0.7825 - loss: 0.8980 - val_accuracy: 0.7956 - val_loss: 0.8574 -
learning_rate: 0.0030
Epoch 57/300
176/176             115s 652ms/step -
accuracy: 0.7863 - loss: 0.9022 - val_accuracy: 0.7810 - val_loss: 0.9223 -
learning_rate: 0.0030
Epoch 58/300
176/176             115s 651ms/step -
accuracy: 0.7840 - loss: 0.9007 - val_accuracy: 0.7970 - val_loss: 0.8677 -
```

```
learning_rate: 0.0030
Epoch 59/300
176/176             118s 672ms/step -
accuracy: 0.7840 - loss: 0.9035 - val_accuracy: 0.7980 - val_loss: 0.8792 -
learning_rate: 0.0030
Epoch 60/300
176/176             1160s 7s/step -
accuracy: 0.7862 - loss: 0.9009 - val_accuracy: 0.8012 - val_loss: 0.8523 -
learning_rate: 0.0030
Epoch 61/300
176/176             114s 648ms/step -
accuracy: 0.7831 - loss: 0.8960 - val_accuracy: 0.8062 - val_loss: 0.8393 -
learning_rate: 0.0030
Epoch 62/300
176/176             124s 704ms/step -
accuracy: 0.7907 - loss: 0.8876 - val_accuracy: 0.7992 - val_loss: 0.8679 -
learning_rate: 0.0030
Epoch 63/300
176/176             660s 4s/step -
accuracy: 0.7875 - loss: 0.8872 - val_accuracy: 0.8040 - val_loss: 0.8710 -
learning_rate: 0.0030
Epoch 64/300
176/176             687s 4s/step -
accuracy: 0.7927 - loss: 0.8843 - val_accuracy: 0.8030 - val_loss: 0.8544 -
learning_rate: 0.0030
Epoch 65/300
176/176             317s 2s/step -
accuracy: 0.7909 - loss: 0.8834 - val_accuracy: 0.7990 - val_loss: 0.8685 -
learning_rate: 0.0030
Epoch 66/300
176/176             1155s 7s/step -
accuracy: 0.7917 - loss: 0.8737 - val_accuracy: 0.8038 - val_loss: 0.8456 -
learning_rate: 0.0030
Epoch 67/300
176/176             118s 670ms/step -
accuracy: 0.7879 - loss: 0.8784 - val_accuracy: 0.8076 - val_loss: 0.8333 -
learning_rate: 0.0030
Epoch 68/300
176/176             120s 684ms/step -
accuracy: 0.7921 - loss: 0.8732 - val_accuracy: 0.7914 - val_loss: 0.8923 -
learning_rate: 0.0030
Epoch 69/300
176/176             118s 670ms/step -
accuracy: 0.7923 - loss: 0.8734 - val_accuracy: 0.7940 - val_loss: 0.8550 -
learning_rate: 0.0030
Epoch 70/300
176/176             119s 678ms/step -
accuracy: 0.7986 - loss: 0.8642 - val_accuracy: 0.8042 - val_loss: 0.8415 -
```

```
learning_rate: 0.0030
Epoch 71/300
176/176             121s 688ms/step -
accuracy: 0.7976 - loss: 0.8592 - val_accuracy: 0.8084 - val_loss: 0.8320 -
learning_rate: 0.0030
Epoch 72/300
176/176             118s 672ms/step -
accuracy: 0.7969 - loss: 0.8644 - val_accuracy: 0.8180 - val_loss: 0.8066 -
learning_rate: 0.0030
Epoch 73/300
176/176             119s 678ms/step -
accuracy: 0.7935 - loss: 0.8623 - val_accuracy: 0.8186 - val_loss: 0.8169 -
learning_rate: 0.0030
Epoch 74/300
176/176             116s 660ms/step -
accuracy: 0.7987 - loss: 0.8609 - val_accuracy: 0.8162 - val_loss: 0.8259 -
learning_rate: 0.0030
Epoch 75/300
176/176             115s 651ms/step -
accuracy: 0.7987 - loss: 0.8512 - val_accuracy: 0.7986 - val_loss: 0.8750 -
learning_rate: 0.0030
Epoch 76/300
176/176             115s 652ms/step -
accuracy: 0.7965 - loss: 0.8686 - val_accuracy: 0.8096 - val_loss: 0.8446 -
learning_rate: 0.0030
Epoch 77/300
176/176             116s 659ms/step -
accuracy: 0.7979 - loss: 0.8600 - val_accuracy: 0.8126 - val_loss: 0.8180 -
learning_rate: 0.0030
Epoch 78/300
176/176             115s 654ms/step -
accuracy: 0.7988 - loss: 0.8514 - val_accuracy: 0.7964 - val_loss: 0.8769 -
learning_rate: 0.0030
Epoch 79/300
176/176             116s 656ms/step -
accuracy: 0.7989 - loss: 0.8591 - val_accuracy: 0.8270 - val_loss: 0.7851 -
learning_rate: 0.0030
Epoch 80/300
176/176             115s 650ms/step -
accuracy: 0.7989 - loss: 0.8540 - val_accuracy: 0.8218 - val_loss: 0.8105 -
learning_rate: 0.0030
Epoch 81/300
176/176             116s 657ms/step -
accuracy: 0.7967 - loss: 0.8642 - val_accuracy: 0.8170 - val_loss: 0.8140 -
learning_rate: 0.0030
Epoch 82/300
176/176             115s 654ms/step -
accuracy: 0.7963 - loss: 0.8591 - val_accuracy: 0.8092 - val_loss: 0.8357 -
```

```
learning_rate: 0.0030
Epoch 83/300
176/176             114s 646ms/step -
accuracy: 0.8064 - loss: 0.8428 - val_accuracy: 0.8014 - val_loss: 0.8553 -
learning_rate: 0.0030
Epoch 84/300
176/176             115s 652ms/step -
accuracy: 0.8015 - loss: 0.8600 - val_accuracy: 0.8130 - val_loss: 0.8117 -
learning_rate: 0.0030
Epoch 85/300
176/176             115s 652ms/step -
accuracy: 0.8021 - loss: 0.8500 - val_accuracy: 0.8128 - val_loss: 0.8413 -
learning_rate: 0.0030
Epoch 86/300
176/176             115s 651ms/step -
accuracy: 0.8030 - loss: 0.8477 - val_accuracy: 0.8164 - val_loss: 0.8211 -
learning_rate: 0.0030
Epoch 87/300
176/176             114s 649ms/step -
accuracy: 0.7970 - loss: 0.8583 - val_accuracy: 0.7926 - val_loss: 0.8799 -
learning_rate: 0.0030
Epoch 88/300
176/176             115s 655ms/step -
accuracy: 0.8042 - loss: 0.8482 - val_accuracy: 0.8092 - val_loss: 0.8178 -
learning_rate: 0.0030
Epoch 89/300
176/176             116s 656ms/step -
accuracy: 0.8043 - loss: 0.8438 - val_accuracy: 0.8138 - val_loss: 0.8174 -
learning_rate: 0.0030
Epoch 90/300
176/176             119s 660ms/step -
accuracy: 0.8164 - loss: 0.8040 - val_accuracy: 0.8450 - val_loss: 0.7149 -
learning_rate: 0.0015
Epoch 91/300
176/176             117s 665ms/step -
accuracy: 0.8301 - loss: 0.7538 - val_accuracy: 0.8348 - val_loss: 0.7337 -
learning_rate: 0.0015
Epoch 92/300
176/176             118s 668ms/step -
accuracy: 0.8304 - loss: 0.7377 - val_accuracy: 0.8574 - val_loss: 0.6635 -
learning_rate: 0.0015
Epoch 93/300
176/176             119s 676ms/step -
accuracy: 0.8298 - loss: 0.7377 - val_accuracy: 0.8510 - val_loss: 0.6783 -
learning_rate: 0.0015
Epoch 94/300
176/176             116s 659ms/step -
accuracy: 0.8318 - loss: 0.7196 - val_accuracy: 0.8486 - val_loss: 0.6872 -
```

```
learning_rate: 0.0015
Epoch 95/300
176/176             119s 675ms/step -
accuracy: 0.8308 - loss: 0.7149 - val_accuracy: 0.8496 - val_loss: 0.6794 -
learning_rate: 0.0015
Epoch 96/300
176/176             119s 676ms/step -
accuracy: 0.8328 - loss: 0.7077 - val_accuracy: 0.8558 - val_loss: 0.6453 -
learning_rate: 0.0015
Epoch 97/300
176/176             118s 668ms/step -
accuracy: 0.8322 - loss: 0.7012 - val_accuracy: 0.8506 - val_loss: 0.6649 -
learning_rate: 0.0015
Epoch 98/300
176/176             116s 660ms/step -
accuracy: 0.8362 - loss: 0.6920 - val_accuracy: 0.8582 - val_loss: 0.6567 -
learning_rate: 0.0015
Epoch 99/300
176/176             114s 649ms/step -
accuracy: 0.8338 - loss: 0.6986 - val_accuracy: 0.8398 - val_loss: 0.6761 -
learning_rate: 0.0015
Epoch 100/300
176/176             115s 652ms/step -
accuracy: 0.8380 - loss: 0.6863 - val_accuracy: 0.8482 - val_loss: 0.6589 -
learning_rate: 0.0015
Epoch 101/300
176/176             116s 661ms/step -
accuracy: 0.8352 - loss: 0.6909 - val_accuracy: 0.8516 - val_loss: 0.6455 -
learning_rate: 0.0015
Epoch 102/300
176/176             115s 655ms/step -
accuracy: 0.8354 - loss: 0.6822 - val_accuracy: 0.8534 - val_loss: 0.6359 -
learning_rate: 0.0015
Epoch 103/300
176/176             119s 677ms/step -
accuracy: 0.8353 - loss: 0.6802 - val_accuracy: 0.8498 - val_loss: 0.6536 -
learning_rate: 0.0015
Epoch 104/300
176/176             118s 673ms/step -
accuracy: 0.8369 - loss: 0.6765 - val_accuracy: 0.8574 - val_loss: 0.6360 -
learning_rate: 0.0015
Epoch 105/300
176/176             116s 660ms/step -
accuracy: 0.8399 - loss: 0.6713 - val_accuracy: 0.8604 - val_loss: 0.6129 -
learning_rate: 0.0015
Epoch 106/300
176/176             118s 669ms/step -
accuracy: 0.8416 - loss: 0.6641 - val_accuracy: 0.8532 - val_loss: 0.6507 -
```

```
learning_rate: 0.0015
Epoch 107/300
176/176            116s 660ms/step -
accuracy: 0.8381 - loss: 0.6719 - val_accuracy: 0.8558 - val_loss: 0.6336 -
learning_rate: 0.0015
Epoch 108/300
176/176            117s 667ms/step -
accuracy: 0.8428 - loss: 0.6625 - val_accuracy: 0.8522 - val_loss: 0.6389 -
learning_rate: 0.0015
Epoch 109/300
176/176            114s 645ms/step -
accuracy: 0.8388 - loss: 0.6673 - val_accuracy: 0.8578 - val_loss: 0.6292 -
learning_rate: 0.0015
Epoch 110/300
176/176            113s 644ms/step -
accuracy: 0.8364 - loss: 0.6731 - val_accuracy: 0.8478 - val_loss: 0.6400 -
learning_rate: 0.0015
Epoch 111/300
176/176            113s 644ms/step -
accuracy: 0.8395 - loss: 0.6586 - val_accuracy: 0.8536 - val_loss: 0.6219 -
learning_rate: 0.0015
Epoch 112/300
176/176            115s 655ms/step -
accuracy: 0.8385 - loss: 0.6680 - val_accuracy: 0.8618 - val_loss: 0.6088 -
learning_rate: 0.0015
Epoch 113/300
176/176            117s 663ms/step -
accuracy: 0.8436 - loss: 0.6575 - val_accuracy: 0.8516 - val_loss: 0.6358 -
learning_rate: 0.0015
Epoch 114/300
176/176            116s 657ms/step -
accuracy: 0.8400 - loss: 0.6586 - val_accuracy: 0.8392 - val_loss: 0.6656 -
learning_rate: 0.0015
Epoch 115/300
176/176            116s 657ms/step -
accuracy: 0.8406 - loss: 0.6580 - val_accuracy: 0.8604 - val_loss: 0.6264 -
learning_rate: 0.0015
Epoch 116/300
176/176            119s 677ms/step -
accuracy: 0.8408 - loss: 0.6535 - val_accuracy: 0.8606 - val_loss: 0.6200 -
learning_rate: 0.0015
Epoch 117/300
176/176            122s 693ms/step -
accuracy: 0.8399 - loss: 0.6603 - val_accuracy: 0.8622 - val_loss: 0.6036 -
learning_rate: 0.0015
Epoch 118/300
176/176            121s 688ms/step -
accuracy: 0.8387 - loss: 0.6622 - val_accuracy: 0.8614 - val_loss: 0.6188 -
```

```
learning_rate: 0.0015
Epoch 119/300
176/176            122s 692ms/step -
accuracy: 0.8382 - loss: 0.6637 - val_accuracy: 0.8574 - val_loss: 0.6215 -
learning_rate: 0.0015
Epoch 120/300
176/176            118s 671ms/step -
accuracy: 0.8451 - loss: 0.6498 - val_accuracy: 0.8554 - val_loss: 0.6363 -
learning_rate: 0.0015
Epoch 121/300
176/176            118s 669ms/step -
accuracy: 0.8421 - loss: 0.6442 - val_accuracy: 0.8532 - val_loss: 0.6290 -
learning_rate: 0.0015
Epoch 122/300
176/176            123s 697ms/step -
accuracy: 0.8470 - loss: 0.6445 - val_accuracy: 0.8620 - val_loss: 0.6039 -
learning_rate: 0.0015
Epoch 123/300
176/176            120s 679ms/step -
accuracy: 0.8435 - loss: 0.6474 - val_accuracy: 0.8508 - val_loss: 0.6292 -
learning_rate: 0.0015
Epoch 124/300
176/176            120s 680ms/step -
accuracy: 0.8436 - loss: 0.6423 - val_accuracy: 0.8488 - val_loss: 0.6503 -
learning_rate: 0.0015
Epoch 125/300
176/176            120s 681ms/step -
accuracy: 0.8409 - loss: 0.6546 - val_accuracy: 0.8436 - val_loss: 0.6588 -
learning_rate: 0.0015
Epoch 126/300
176/176            120s 680ms/step -
accuracy: 0.8441 - loss: 0.6394 - val_accuracy: 0.8596 - val_loss: 0.6253 -
learning_rate: 0.0015
Epoch 127/300
176/176            118s 669ms/step -
accuracy: 0.8404 - loss: 0.6521 - val_accuracy: 0.8578 - val_loss: 0.6184 -
learning_rate: 0.0015
Epoch 128/300
176/176            121s 675ms/step -
accuracy: 0.8502 - loss: 0.6240 - val_accuracy: 0.8696 - val_loss: 0.5725 -
learning_rate: 7.5000e-04
Epoch 129/300
176/176            116s 661ms/step -
accuracy: 0.8602 - loss: 0.5943 - val_accuracy: 0.8726 - val_loss: 0.5624 -
learning_rate: 7.5000e-04
Epoch 130/300
176/176            117s 662ms/step -
accuracy: 0.8599 - loss: 0.5875 - val_accuracy: 0.8676 - val_loss: 0.5701 -
```

```
learning_rate: 7.5000e-04
Epoch 131/300
176/176             118s 668ms/step -
accuracy: 0.8619 - loss: 0.5729 - val_accuracy: 0.8716 - val_loss: 0.5625 -
learning_rate: 7.5000e-04
Epoch 132/300
176/176             119s 673ms/step -
accuracy: 0.8638 - loss: 0.5724 - val_accuracy: 0.8680 - val_loss: 0.5837 -
learning_rate: 7.5000e-04
Epoch 133/300
176/176             119s 674ms/step -
accuracy: 0.8686 - loss: 0.5610 - val_accuracy: 0.8766 - val_loss: 0.5455 -
learning_rate: 7.5000e-04
Epoch 134/300
176/176             119s 677ms/step -
accuracy: 0.8664 - loss: 0.5613 - val_accuracy: 0.8692 - val_loss: 0.5530 -
learning_rate: 7.5000e-04
Epoch 135/300
176/176             129s 733ms/step -
accuracy: 0.8623 - loss: 0.5656 - val_accuracy: 0.8780 - val_loss: 0.5294 -
learning_rate: 7.5000e-04
Epoch 136/300
176/176             126s 715ms/step -
accuracy: 0.8657 - loss: 0.5557 - val_accuracy: 0.8738 - val_loss: 0.5387 -
learning_rate: 7.5000e-04
Epoch 137/300
176/176             114s 649ms/step -
accuracy: 0.8660 - loss: 0.5492 - val_accuracy: 0.8826 - val_loss: 0.5166 -
learning_rate: 7.5000e-04
Epoch 138/300
176/176             114s 646ms/step -
accuracy: 0.8669 - loss: 0.5515 - val_accuracy: 0.8816 - val_loss: 0.5224 -
learning_rate: 7.5000e-04
Epoch 139/300
176/176             114s 644ms/step -
accuracy: 0.8696 - loss: 0.5376 - val_accuracy: 0.8748 - val_loss: 0.5349 -
learning_rate: 7.5000e-04
Epoch 140/300
176/176             113s 643ms/step -
accuracy: 0.8679 - loss: 0.5458 - val_accuracy: 0.8760 - val_loss: 0.5303 -
learning_rate: 7.5000e-04
Epoch 141/300
176/176             113s 643ms/step -
accuracy: 0.8659 - loss: 0.5485 - val_accuracy: 0.8810 - val_loss: 0.5155 -
learning_rate: 7.5000e-04
Epoch 142/300
176/176             114s 646ms/step -
accuracy: 0.8652 - loss: 0.5431 - val_accuracy: 0.8706 - val_loss: 0.5445 -
```

```
learning_rate: 7.5000e-04
Epoch 143/300
176/176             113s 644ms/step -
accuracy: 0.8657 - loss: 0.5492 - val_accuracy: 0.8828 - val_loss: 0.5163 -
learning_rate: 7.5000e-04
Epoch 144/300
176/176             114s 645ms/step -
accuracy: 0.8703 - loss: 0.5323 - val_accuracy: 0.8802 - val_loss: 0.5131 -
learning_rate: 7.5000e-04
Epoch 145/300
176/176             119s 677ms/step -
accuracy: 0.8706 - loss: 0.5324 - val_accuracy: 0.8792 - val_loss: 0.5135 -
learning_rate: 7.5000e-04
Epoch 146/300
176/176             119s 676ms/step -
accuracy: 0.8680 - loss: 0.5356 - val_accuracy: 0.8824 - val_loss: 0.5158 -
learning_rate: 7.5000e-04
Epoch 147/300
176/176             117s 665ms/step -
accuracy: 0.8678 - loss: 0.5392 - val_accuracy: 0.8684 - val_loss: 0.5381 -
learning_rate: 7.5000e-04
Epoch 148/300
176/176             116s 658ms/step -
accuracy: 0.8720 - loss: 0.5285 - val_accuracy: 0.8766 - val_loss: 0.5230 -
learning_rate: 7.5000e-04
Epoch 149/300
176/176             116s 660ms/step -
accuracy: 0.8684 - loss: 0.5243 - val_accuracy: 0.8798 - val_loss: 0.5260 -
learning_rate: 7.5000e-04
Epoch 150/300
176/176             116s 661ms/step -
accuracy: 0.8670 - loss: 0.5356 - val_accuracy: 0.8746 - val_loss: 0.5287 -
learning_rate: 7.5000e-04
Epoch 151/300
176/176             119s 678ms/step -
accuracy: 0.8744 - loss: 0.5134 - val_accuracy: 0.8844 - val_loss: 0.5029 -
learning_rate: 7.5000e-04
Epoch 152/300
176/176             117s 664ms/step -
accuracy: 0.8704 - loss: 0.5260 - val_accuracy: 0.8802 - val_loss: 0.5025 -
learning_rate: 7.5000e-04
Epoch 153/300
176/176             117s 666ms/step -
accuracy: 0.8736 - loss: 0.5226 - val_accuracy: 0.8782 - val_loss: 0.5099 -
learning_rate: 7.5000e-04
Epoch 154/300
176/176             122s 691ms/step -
accuracy: 0.8709 - loss: 0.5202 - val_accuracy: 0.8780 - val_loss: 0.5123 -
```

```
learning_rate: 7.5000e-04
Epoch 155/300
176/176             120s 681ms/step -
accuracy: 0.8719 - loss: 0.5169 - val_accuracy: 0.8744 - val_loss: 0.5248 -
learning_rate: 7.5000e-04
Epoch 156/300
176/176             126s 718ms/step -
accuracy: 0.8683 - loss: 0.5292 - val_accuracy: 0.8802 - val_loss: 0.5011 -
learning_rate: 7.5000e-04
Epoch 157/300
176/176             126s 716ms/step -
accuracy: 0.8703 - loss: 0.5248 - val_accuracy: 0.8876 - val_loss: 0.5072 -
learning_rate: 7.5000e-04
Epoch 158/300
176/176             126s 718ms/step -
accuracy: 0.8735 - loss: 0.5137 - val_accuracy: 0.8766 - val_loss: 0.5107 -
learning_rate: 7.5000e-04
Epoch 159/300
176/176             121s 689ms/step -
accuracy: 0.8711 - loss: 0.5201 - val_accuracy: 0.8754 - val_loss: 0.5098 -
learning_rate: 7.5000e-04
Epoch 160/300
176/176             118s 669ms/step -
accuracy: 0.8708 - loss: 0.5154 - val_accuracy: 0.8786 - val_loss: 0.5090 -
learning_rate: 7.5000e-04
Epoch 161/300
176/176             119s 677ms/step -
accuracy: 0.8730 - loss: 0.5105 - val_accuracy: 0.8728 - val_loss: 0.5171 -
learning_rate: 7.5000e-04
Epoch 162/300
176/176             121s 688ms/step -
accuracy: 0.8679 - loss: 0.5211 - val_accuracy: 0.8772 - val_loss: 0.5075 -
learning_rate: 7.5000e-04
Epoch 163/300
176/176             123s 697ms/step -
accuracy: 0.8728 - loss: 0.5133 - val_accuracy: 0.8826 - val_loss: 0.4944 -
learning_rate: 7.5000e-04
Epoch 164/300
176/176             119s 675ms/step -
accuracy: 0.8744 - loss: 0.5033 - val_accuracy: 0.8778 - val_loss: 0.5087 -
learning_rate: 7.5000e-04
Epoch 165/300
176/176             119s 678ms/step -
accuracy: 0.8724 - loss: 0.5129 - val_accuracy: 0.8816 - val_loss: 0.5076 -
learning_rate: 7.5000e-04
Epoch 166/300
176/176             117s 666ms/step -
accuracy: 0.8706 - loss: 0.5141 - val_accuracy: 0.8782 - val_loss: 0.5114 -
```

```
learning_rate: 7.5000e-04
Epoch 167/300
176/176             117s 665ms/step -
accuracy: 0.8697 - loss: 0.5170 - val_accuracy: 0.8772 - val_loss: 0.5134 -
learning_rate: 7.5000e-04
Epoch 168/300
176/176             118s 670ms/step -
accuracy: 0.8715 - loss: 0.5101 - val_accuracy: 0.8844 - val_loss: 0.4899 -
learning_rate: 7.5000e-04
Epoch 169/300
176/176             118s 673ms/step -
accuracy: 0.8734 - loss: 0.5057 - val_accuracy: 0.8804 - val_loss: 0.5100 -
learning_rate: 7.5000e-04
Epoch 170/300
176/176             117s 667ms/step -
accuracy: 0.8694 - loss: 0.5140 - val_accuracy: 0.8770 - val_loss: 0.4905 -
learning_rate: 7.5000e-04
Epoch 171/300
176/176             122s 694ms/step -
accuracy: 0.8732 - loss: 0.5049 - val_accuracy: 0.8804 - val_loss: 0.4989 -
learning_rate: 7.5000e-04
Epoch 172/300
176/176             118s 672ms/step -
accuracy: 0.8713 - loss: 0.5089 - val_accuracy: 0.8840 - val_loss: 0.4916 -
learning_rate: 7.5000e-04
Epoch 173/300
176/176             114s 650ms/step -
accuracy: 0.8711 - loss: 0.5092 - val_accuracy: 0.8834 - val_loss: 0.4915 -
learning_rate: 7.5000e-04
Epoch 174/300
176/176             122s 693ms/step -
accuracy: 0.8733 - loss: 0.5077 - val_accuracy: 0.8786 - val_loss: 0.5119 -
learning_rate: 7.5000e-04
Epoch 175/300
176/176             120s 683ms/step -
accuracy: 0.8702 - loss: 0.5151 - val_accuracy: 0.8772 - val_loss: 0.5060 -
learning_rate: 7.5000e-04
Epoch 176/300
176/176             118s 670ms/step -
accuracy: 0.8707 - loss: 0.5148 - val_accuracy: 0.8806 - val_loss: 0.4964 -
learning_rate: 7.5000e-04
Epoch 177/300
176/176             117s 664ms/step -
accuracy: 0.8718 - loss: 0.5090 - val_accuracy: 0.8776 - val_loss: 0.5053 -
learning_rate: 7.5000e-04
Epoch 178/300
176/176             119s 676ms/step -
accuracy: 0.8714 - loss: 0.5038 - val_accuracy: 0.8696 - val_loss: 0.5154 -
```

```
learning_rate: 7.5000e-04
Epoch 179/300
176/176             118s 668ms/step -
accuracy: 0.8772 - loss: 0.4834 - val_accuracy: 0.8914 - val_loss: 0.4775 -
learning_rate: 3.7500e-04
Epoch 180/300
176/176             114s 648ms/step -
accuracy: 0.8831 - loss: 0.4706 - val_accuracy: 0.8898 - val_loss: 0.4708 -
learning_rate: 3.7500e-04
Epoch 181/300
176/176             116s 657ms/step -
accuracy: 0.8835 - loss: 0.4742 - val_accuracy: 0.8886 - val_loss: 0.4733 -
learning_rate: 3.7500e-04
Epoch 182/300
176/176             119s 675ms/step -
accuracy: 0.8880 - loss: 0.4584 - val_accuracy: 0.8870 - val_loss: 0.4708 -
learning_rate: 3.7500e-04
Epoch 183/300
176/176             121s 689ms/step -
accuracy: 0.8883 - loss: 0.4617 - val_accuracy: 0.8910 - val_loss: 0.4626 -
learning_rate: 3.7500e-04
Epoch 184/300
176/176             122s 691ms/step -
accuracy: 0.8821 - loss: 0.4634 - val_accuracy: 0.9008 - val_loss: 0.4565 -
learning_rate: 3.7500e-04
Epoch 185/300
176/176             119s 675ms/step -
accuracy: 0.8874 - loss: 0.4543 - val_accuracy: 0.8852 - val_loss: 0.4758 -
learning_rate: 3.7500e-04
Epoch 186/300
176/176             122s 691ms/step -
accuracy: 0.8862 - loss: 0.4640 - val_accuracy: 0.8950 - val_loss: 0.4641 -
learning_rate: 3.7500e-04
Epoch 187/300
176/176             121s 685ms/step -
accuracy: 0.8877 - loss: 0.4510 - val_accuracy: 0.8908 - val_loss: 0.4685 -
learning_rate: 3.7500e-04
Epoch 188/300
176/176             119s 679ms/step -
accuracy: 0.8900 - loss: 0.4458 - val_accuracy: 0.8966 - val_loss: 0.4504 -
learning_rate: 3.7500e-04
Epoch 189/300
176/176             120s 683ms/step -
accuracy: 0.8922 - loss: 0.4413 - val_accuracy: 0.8930 - val_loss: 0.4528 -
learning_rate: 3.7500e-04
Epoch 190/300
176/176             118s 671ms/step -
accuracy: 0.8868 - loss: 0.4556 - val_accuracy: 0.8934 - val_loss: 0.4560 -
```

```
learning_rate: 3.7500e-04
Epoch 191/300
176/176             117s 664ms/step -
accuracy: 0.8887 - loss: 0.4496 - val_accuracy: 0.8924 - val_loss: 0.4664 -
learning_rate: 3.7500e-04
Epoch 192/300
176/176             118s 668ms/step -
accuracy: 0.8848 - loss: 0.4511 - val_accuracy: 0.8914 - val_loss: 0.4549 -
learning_rate: 3.7500e-04
Epoch 193/300
176/176             118s 669ms/step -
accuracy: 0.8887 - loss: 0.4474 - val_accuracy: 0.8904 - val_loss: 0.4645 -
learning_rate: 3.7500e-04
Epoch 194/300
176/176             121s 686ms/step -
accuracy: 0.8855 - loss: 0.4446 - val_accuracy: 0.8836 - val_loss: 0.4791 -
learning_rate: 3.7500e-04
Epoch 195/300
176/176             118s 672ms/step -
accuracy: 0.8907 - loss: 0.4364 - val_accuracy: 0.8870 - val_loss: 0.4630 -
learning_rate: 3.7500e-04
Epoch 196/300
176/176             121s 686ms/step -
accuracy: 0.8886 - loss: 0.4449 - val_accuracy: 0.8896 - val_loss: 0.4583 -
learning_rate: 3.7500e-04
Epoch 197/300
176/176             117s 664ms/step -
accuracy: 0.8911 - loss: 0.4329 - val_accuracy: 0.8900 - val_loss: 0.4632 -
learning_rate: 3.7500e-04
Epoch 198/300
176/176             115s 654ms/step -
accuracy: 0.8913 - loss: 0.4377 - val_accuracy: 0.8904 - val_loss: 0.4427 -
learning_rate: 3.7500e-04
Epoch 199/300
176/176             114s 644ms/step -
accuracy: 0.8902 - loss: 0.4327 - val_accuracy: 0.8880 - val_loss: 0.4659 -
learning_rate: 3.7500e-04
Epoch 200/300
176/176             113s 644ms/step -
accuracy: 0.8880 - loss: 0.4418 - val_accuracy: 0.8964 - val_loss: 0.4311 -
learning_rate: 3.7500e-04
Epoch 201/300
176/176             115s 655ms/step -
accuracy: 0.8950 - loss: 0.4229 - val_accuracy: 0.8868 - val_loss: 0.4580 -
learning_rate: 3.7500e-04
Epoch 202/300
176/176             115s 650ms/step -
accuracy: 0.8929 - loss: 0.4304 - val_accuracy: 0.8874 - val_loss: 0.4624 -
```

```
learning_rate: 3.7500e-04
Epoch 203/300
176/176             113s 643ms/step -
accuracy: 0.8918 - loss: 0.4288 - val_accuracy: 0.8938 - val_loss: 0.4350 -
learning_rate: 3.7500e-04
Epoch 204/300
176/176             116s 658ms/step -
accuracy: 0.8933 - loss: 0.4281 - val_accuracy: 0.8904 - val_loss: 0.4494 -
learning_rate: 3.7500e-04
Epoch 205/300
176/176             117s 664ms/step -
accuracy: 0.8934 - loss: 0.4258 - val_accuracy: 0.8858 - val_loss: 0.4653 -
learning_rate: 3.7500e-04
Epoch 206/300
176/176             118s 671ms/step -
accuracy: 0.8898 - loss: 0.4347 - val_accuracy: 0.8840 - val_loss: 0.4601 -
learning_rate: 3.7500e-04
Epoch 207/300
176/176             116s 659ms/step -
accuracy: 0.8915 - loss: 0.4277 - val_accuracy: 0.8958 - val_loss: 0.4351 -
learning_rate: 3.7500e-04
Epoch 208/300
176/176             116s 659ms/step -
accuracy: 0.8897 - loss: 0.4322 - val_accuracy: 0.8952 - val_loss: 0.4384 -
learning_rate: 3.7500e-04
Epoch 209/300
176/176             121s 685ms/step -
accuracy: 0.8987 - loss: 0.4136 - val_accuracy: 0.8832 - val_loss: 0.4649 -
learning_rate: 3.7500e-04
Epoch 210/300
176/176             119s 675ms/step -
accuracy: 0.8944 - loss: 0.4193 - val_accuracy: 0.8892 - val_loss: 0.4454 -
learning_rate: 3.7500e-04
Epoch 211/300
176/176             122s 696ms/step -
accuracy: 0.8924 - loss: 0.4228 - val_accuracy: 0.8954 - val_loss: 0.4332 -
learning_rate: 1.8750e-04
Epoch 212/300
176/176             118s 669ms/step -
accuracy: 0.8973 - loss: 0.4129 - val_accuracy: 0.9012 - val_loss: 0.4234 -
learning_rate: 1.8750e-04
Epoch 213/300
176/176             118s 671ms/step -
accuracy: 0.8972 - loss: 0.4086 - val_accuracy: 0.8912 - val_loss: 0.4329 -
learning_rate: 1.8750e-04
Epoch 214/300
176/176             118s 672ms/step -
accuracy: 0.8995 - loss: 0.4108 - val_accuracy: 0.9022 - val_loss: 0.4075 -
```

```
learning_rate: 1.8750e-04
Epoch 215/300
176/176             118s 667ms/step -
accuracy: 0.8989 - loss: 0.4063 - val_accuracy: 0.8964 - val_loss: 0.4364 -
learning_rate: 1.8750e-04
Epoch 216/300
176/176             116s 661ms/step -
accuracy: 0.8976 - loss: 0.4023 - val_accuracy: 0.8948 - val_loss: 0.4224 -
learning_rate: 1.8750e-04
Epoch 217/300
176/176             117s 667ms/step -
accuracy: 0.9013 - loss: 0.3972 - val_accuracy: 0.8942 - val_loss: 0.4329 -
learning_rate: 1.8750e-04
Epoch 218/300
176/176             118s 673ms/step -
accuracy: 0.8935 - loss: 0.4127 - val_accuracy: 0.8974 - val_loss: 0.4235 -
learning_rate: 1.8750e-04
Epoch 219/300
176/176             116s 656ms/step -
accuracy: 0.8991 - loss: 0.4042 - val_accuracy: 0.8976 - val_loss: 0.4305 -
learning_rate: 1.8750e-04
Epoch 220/300
176/176             116s 657ms/step -
accuracy: 0.9006 - loss: 0.3997 - val_accuracy: 0.8932 - val_loss: 0.4266 -
learning_rate: 1.8750e-04
Epoch 221/300
176/176             117s 665ms/step -
accuracy: 0.8976 - loss: 0.4026 - val_accuracy: 0.9022 - val_loss: 0.4215 -
learning_rate: 1.8750e-04
Epoch 222/300
176/176             117s 665ms/step -
accuracy: 0.9022 - loss: 0.3927 - val_accuracy: 0.8968 - val_loss: 0.4230 -
learning_rate: 1.8750e-04
Epoch 223/300
176/176             117s 665ms/step -
accuracy: 0.8996 - loss: 0.3949 - val_accuracy: 0.8974 - val_loss: 0.4366 -
learning_rate: 1.8750e-04
Epoch 224/300
176/176             118s 670ms/step -
accuracy: 0.9007 - loss: 0.3971 - val_accuracy: 0.8926 - val_loss: 0.4350 -
learning_rate: 1.8750e-04
Epoch 225/300
176/176             120s 680ms/step -
accuracy: 0.9026 - loss: 0.3935 - val_accuracy: 0.8982 - val_loss: 0.4194 -
learning_rate: 9.3750e-05
Epoch 226/300
176/176             119s 675ms/step -
accuracy: 0.9012 - loss: 0.3958 - val_accuracy: 0.8986 - val_loss: 0.4239 -
```

```
learning_rate: 9.3750e-05
Epoch 227/300
176/176             117s 662ms/step -
accuracy: 0.9052 - loss: 0.3811 - val_accuracy: 0.9004 - val_loss: 0.4154 -
learning_rate: 9.3750e-05
Epoch 228/300
176/176             273s 2s/step -
accuracy: 0.9046 - loss: 0.3851 - val_accuracy: 0.9018 - val_loss: 0.4043 -
learning_rate: 9.3750e-05
Epoch 229/300
176/176             972s 6s/step -
accuracy: 0.9006 - loss: 0.3918 - val_accuracy: 0.8990 - val_loss: 0.4202 -
learning_rate: 9.3750e-05
Epoch 230/300
176/176             119s 676ms/step -
accuracy: 0.9047 - loss: 0.3820 - val_accuracy: 0.9008 - val_loss: 0.4148 -
learning_rate: 9.3750e-05
Epoch 231/300
176/176             117s 666ms/step -
accuracy: 0.9041 - loss: 0.3801 - val_accuracy: 0.9042 - val_loss: 0.4105 -
learning_rate: 9.3750e-05
Epoch 232/300
176/176             117s 666ms/step -
accuracy: 0.8999 - loss: 0.3907 - val_accuracy: 0.9036 - val_loss: 0.4011 -
learning_rate: 9.3750e-05
Epoch 233/300
176/176             116s 659ms/step -
accuracy: 0.9069 - loss: 0.3742 - val_accuracy: 0.9016 - val_loss: 0.4097 -
learning_rate: 9.3750e-05
Epoch 234/300
176/176             116s 661ms/step -
accuracy: 0.9049 - loss: 0.3775 - val_accuracy: 0.8976 - val_loss: 0.4294 -
learning_rate: 9.3750e-05
Epoch 235/300
176/176             117s 662ms/step -
accuracy: 0.9039 - loss: 0.3820 - val_accuracy: 0.9022 - val_loss: 0.4096 -
learning_rate: 9.3750e-05
Epoch 236/300
176/176             115s 653ms/step -
accuracy: 0.9079 - loss: 0.3771 - val_accuracy: 0.8960 - val_loss: 0.4230 -
learning_rate: 9.3750e-05
Epoch 237/300
176/176             119s 677ms/step -
accuracy: 0.9016 - loss: 0.3812 - val_accuracy: 0.9022 - val_loss: 0.4220 -
learning_rate: 9.3750e-05
Epoch 238/300
176/176             118s 668ms/step -
accuracy: 0.9040 - loss: 0.3771 - val_accuracy: 0.9006 - val_loss: 0.4126 -
```

```
learning_rate: 9.3750e-05
Epoch 239/300
176/176             118s 672ms/step -
accuracy: 0.9054 - loss: 0.3742 - val_accuracy: 0.9016 - val_loss: 0.4055 -
learning_rate: 9.3750e-05
Epoch 240/300
176/176             120s 679ms/step -
accuracy: 0.9060 - loss: 0.3764 - val_accuracy: 0.8990 - val_loss: 0.4143 -
learning_rate: 9.3750e-05
Epoch 241/300
176/176             119s 673ms/step -
accuracy: 0.9068 - loss: 0.3768 - val_accuracy: 0.8994 - val_loss: 0.4159 -
learning_rate: 9.3750e-05
Epoch 242/300
176/176             118s 667ms/step -
accuracy: 0.9060 - loss: 0.3736 - val_accuracy: 0.8994 - val_loss: 0.4166 -
learning_rate: 9.3750e-05
Epoch 243/300
176/176             120s 681ms/step -
accuracy: 0.9070 - loss: 0.3718 - val_accuracy: 0.9038 - val_loss: 0.4137 -
learning_rate: 4.6875e-05
Epoch 244/300
176/176             115s 655ms/step -
accuracy: 0.9088 - loss: 0.3700 - val_accuracy: 0.9038 - val_loss: 0.3989 -
learning_rate: 4.6875e-05
Epoch 245/300
176/176             121s 687ms/step -
accuracy: 0.9045 - loss: 0.3750 - val_accuracy: 0.9014 - val_loss: 0.4194 -
learning_rate: 4.6875e-05
Epoch 246/300
176/176             120s 680ms/step -
accuracy: 0.9076 - loss: 0.3722 - val_accuracy: 0.9028 - val_loss: 0.4110 -
learning_rate: 4.6875e-05
Epoch 247/300
176/176             120s 680ms/step -
accuracy: 0.9058 - loss: 0.3730 - val_accuracy: 0.9020 - val_loss: 0.3983 -
learning_rate: 4.6875e-05
Epoch 248/300
176/176             117s 666ms/step -
accuracy: 0.9097 - loss: 0.3692 - val_accuracy: 0.9054 - val_loss: 0.3962 -
learning_rate: 4.6875e-05
Epoch 249/300
176/176             120s 679ms/step -
accuracy: 0.9077 - loss: 0.3711 - val_accuracy: 0.9066 - val_loss: 0.3968 -
learning_rate: 4.6875e-05
Epoch 250/300
176/176             118s 672ms/step -
accuracy: 0.9068 - loss: 0.3697 - val_accuracy: 0.8992 - val_loss: 0.4093 -
```

```
learning_rate: 4.6875e-05
Epoch 251/300
176/176             117s 665ms/step -
accuracy: 0.9060 - loss: 0.3735 - val_accuracy: 0.9032 - val_loss: 0.4090 -
learning_rate: 4.6875e-05
Epoch 252/300
176/176             118s 670ms/step -
accuracy: 0.9098 - loss: 0.3616 - val_accuracy: 0.9014 - val_loss: 0.4024 -
learning_rate: 4.6875e-05
Epoch 253/300
176/176             115s 653ms/step -
accuracy: 0.9058 - loss: 0.3746 - val_accuracy: 0.8988 - val_loss: 0.4080 -
learning_rate: 4.6875e-05
Epoch 254/300
176/176             120s 684ms/step -
accuracy: 0.9084 - loss: 0.3687 - val_accuracy: 0.9028 - val_loss: 0.4031 -
learning_rate: 4.6875e-05
Epoch 255/300
176/176             123s 697ms/step -
accuracy: 0.9077 - loss: 0.3696 - val_accuracy: 0.8992 - val_loss: 0.4139 -
learning_rate: 4.6875e-05
Epoch 256/300
176/176             122s 690ms/step -
accuracy: 0.9078 - loss: 0.3696 - val_accuracy: 0.9044 - val_loss: 0.4083 -
learning_rate: 4.6875e-05
Epoch 257/300
176/176             121s 687ms/step -
accuracy: 0.9066 - loss: 0.3731 - val_accuracy: 0.9024 - val_loss: 0.4054 -
learning_rate: 4.6875e-05
Epoch 258/300
176/176             114s 645ms/step -
accuracy: 0.9085 - loss: 0.3641 - val_accuracy: 0.8972 - val_loss: 0.4175 -
learning_rate: 4.6875e-05
Epoch 259/300
176/176             114s 645ms/step -
accuracy: 0.9062 - loss: 0.3721 - val_accuracy: 0.9036 - val_loss: 0.4080 -
learning_rate: 2.3438e-05
Epoch 260/300
176/176             113s 643ms/step -
accuracy: 0.9102 - loss: 0.3579 - val_accuracy: 0.9070 - val_loss: 0.3953 -
learning_rate: 2.3438e-05
Epoch 261/300
176/176             113s 642ms/step -
accuracy: 0.9079 - loss: 0.3621 - val_accuracy: 0.9032 - val_loss: 0.4122 -
learning_rate: 2.3438e-05
Epoch 262/300
176/176             113s 643ms/step -
accuracy: 0.9117 - loss: 0.3603 - val_accuracy: 0.9044 - val_loss: 0.3978 -
```

```
learning_rate: 2.3438e-05
Epoch 263/300
176/176             113s 643ms/step -
accuracy: 0.9080 - loss: 0.3683 - val_accuracy: 0.9014 - val_loss: 0.4073 -
learning_rate: 2.3438e-05
Epoch 264/300
176/176             113s 643ms/step -
accuracy: 0.9131 - loss: 0.3557 - val_accuracy: 0.9048 - val_loss: 0.4060 -
learning_rate: 2.3438e-05
Epoch 265/300
176/176             113s 644ms/step -
accuracy: 0.9083 - loss: 0.3638 - val_accuracy: 0.9030 - val_loss: 0.4101 -
learning_rate: 2.3438e-05
Epoch 266/300
176/176             113s 642ms/step -
accuracy: 0.9121 - loss: 0.3606 - val_accuracy: 0.9020 - val_loss: 0.3957 -
learning_rate: 2.3438e-05
Epoch 267/300
176/176             113s 643ms/step -
accuracy: 0.9120 - loss: 0.3590 - val_accuracy: 0.9056 - val_loss: 0.3988 -
learning_rate: 2.3438e-05
Epoch 268/300
176/176             113s 642ms/step -
accuracy: 0.9103 - loss: 0.3582 - val_accuracy: 0.9028 - val_loss: 0.4132 -
learning_rate: 2.3438e-05
Epoch 269/300
176/176             113s 643ms/step -
accuracy: 0.9118 - loss: 0.3540 - val_accuracy: 0.9008 - val_loss: 0.3981 -
learning_rate: 2.3438e-05
Epoch 270/300
176/176             113s 643ms/step -
accuracy: 0.9070 - loss: 0.3675 - val_accuracy: 0.9010 - val_loss: 0.3971 -
learning_rate: 2.3438e-05
Epoch 271/300
176/176             113s 642ms/step -
accuracy: 0.9071 - loss: 0.3720 - val_accuracy: 0.9026 - val_loss: 0.4023 -
learning_rate: 1.1719e-05
Epoch 272/300
176/176             113s 642ms/step -
accuracy: 0.9107 - loss: 0.3556 - val_accuracy: 0.8966 - val_loss: 0.4061 -
learning_rate: 1.1719e-05
Epoch 273/300
176/176             113s 642ms/step -
accuracy: 0.9103 - loss: 0.3599 - val_accuracy: 0.9000 - val_loss: 0.4094 -
learning_rate: 1.1719e-05
Epoch 274/300
176/176             113s 643ms/step -
accuracy: 0.9090 - loss: 0.3640 - val_accuracy: 0.9018 - val_loss: 0.4156 -
```

```
learning_rate: 1.1719e-05
Epoch 275/300
176/176             113s 642ms/step -
accuracy: 0.9109 - loss: 0.3603 - val_accuracy: 0.9064 - val_loss: 0.4040 -
learning_rate: 1.1719e-05
Epoch 276/300
176/176             113s 643ms/step -
accuracy: 0.9107 - loss: 0.3583 - val_accuracy: 0.9044 - val_loss: 0.4030 -
learning_rate: 1.1719e-05
Epoch 277/300
176/176             113s 642ms/step -
accuracy: 0.9097 - loss: 0.3612 - val_accuracy: 0.9028 - val_loss: 0.4061 -
learning_rate: 1.1719e-05
Epoch 278/300
176/176             113s 642ms/step -
accuracy: 0.9099 - loss: 0.3584 - val_accuracy: 0.9050 - val_loss: 0.3968 -
learning_rate: 1.1719e-05
Epoch 279/300
176/176             113s 642ms/step -
accuracy: 0.9114 - loss: 0.3610 - val_accuracy: 0.9008 - val_loss: 0.4104 -
learning_rate: 1.1719e-05
Epoch 280/300
176/176             113s 643ms/step -
accuracy: 0.9121 - loss: 0.3539 - val_accuracy: 0.9008 - val_loss: 0.3975 -
learning_rate: 1.1719e-05
Epoch 281/300
176/176             113s 642ms/step -
accuracy: 0.9112 - loss: 0.3577 - val_accuracy: 0.9048 - val_loss: 0.4091 -
learning_rate: 1.0000e-05
Epoch 282/300
176/176             113s 643ms/step -
accuracy: 0.9094 - loss: 0.3638 - val_accuracy: 0.9054 - val_loss: 0.3964 -
learning_rate: 1.0000e-05
Epoch 283/300
176/176             113s 645ms/step -
accuracy: 0.9094 - loss: 0.3577 - val_accuracy: 0.9074 - val_loss: 0.4030 -
learning_rate: 1.0000e-05
Epoch 284/300
176/176             113s 642ms/step -
accuracy: 0.9129 - loss: 0.3559 - val_accuracy: 0.8956 - val_loss: 0.4153 -
learning_rate: 1.0000e-05
Epoch 285/300
176/176             113s 641ms/step -
accuracy: 0.9109 - loss: 0.3560 - val_accuracy: 0.9000 - val_loss: 0.4125 -
learning_rate: 1.0000e-05
Epoch 286/300
176/176             113s 643ms/step -
accuracy: 0.9093 - loss: 0.3604 - val_accuracy: 0.9034 - val_loss: 0.4075 -
```

```
learning_rate: 1.0000e-05
Epoch 287/300
176/176             113s 642ms/step -
accuracy: 0.9125 - loss: 0.3572 - val_accuracy: 0.8998 - val_loss: 0.4072 -
learning_rate: 1.0000e-05
Epoch 288/300
176/176             113s 642ms/step -
accuracy: 0.9111 - loss: 0.3556 - val_accuracy: 0.9002 - val_loss: 0.4098 -
learning_rate: 1.0000e-05
Epoch 289/300
176/176             113s 643ms/step -
accuracy: 0.9090 - loss: 0.3605 - val_accuracy: 0.9010 - val_loss: 0.4145 -
learning_rate: 1.0000e-05
Epoch 290/300
176/176             113s 642ms/step -
accuracy: 0.9099 - loss: 0.3592 - val_accuracy: 0.9046 - val_loss: 0.3992 -
learning_rate: 1.0000e-05
Epoch 291/300
176/176             113s 642ms/step -
accuracy: 0.9103 - loss: 0.3628 - val_accuracy: 0.9074 - val_loss: 0.4047 -
learning_rate: 1.0000e-05
Epoch 292/300
176/176             113s 643ms/step -
accuracy: 0.9105 - loss: 0.3573 - val_accuracy: 0.9012 - val_loss: 0.4066 -
learning_rate: 1.0000e-05
Epoch 293/300
176/176             113s 641ms/step -
accuracy: 0.9097 - loss: 0.3600 - val_accuracy: 0.8988 - val_loss: 0.4075 -
learning_rate: 1.0000e-05
Epoch 294/300
176/176             113s 642ms/step -
accuracy: 0.9093 - loss: 0.3600 - val_accuracy: 0.9004 - val_loss: 0.4030 -
learning_rate: 1.0000e-05
Epoch 295/300
176/176             113s 642ms/step -
accuracy: 0.9125 - loss: 0.3546 - val_accuracy: 0.9036 - val_loss: 0.3989 -
learning_rate: 1.0000e-05
Epoch 296/300
176/176             113s 644ms/step -
accuracy: 0.9133 - loss: 0.3519 - val_accuracy: 0.9100 - val_loss: 0.3933 -
learning_rate: 1.0000e-05
Epoch 297/300
176/176             113s 643ms/step -
accuracy: 0.9080 - loss: 0.3622 - val_accuracy: 0.9022 - val_loss: 0.4066 -
learning_rate: 1.0000e-05
Epoch 298/300
176/176             113s 642ms/step -
accuracy: 0.9117 - loss: 0.3521 - val_accuracy: 0.9084 - val_loss: 0.3927 -
```

```
learning_rate: 1.0000e-05
Epoch 299/300
176/176             113s 643ms/step -
accuracy: 0.9077 - loss: 0.3651 - val_accuracy: 0.9022 - val_loss: 0.4009 -
learning_rate: 1.0000e-05
Epoch 300/300
176/176             113s 642ms/step -
accuracy: 0.9105 - loss: 0.3563 - val_accuracy: 0.9034 - val_loss: 0.3997 -
learning_rate: 1.0000e-05
Restoring model weights from the end of the best epoch: 298.
```

[72]: `<keras.src.callbacks.history.History at 0x300f01010>`

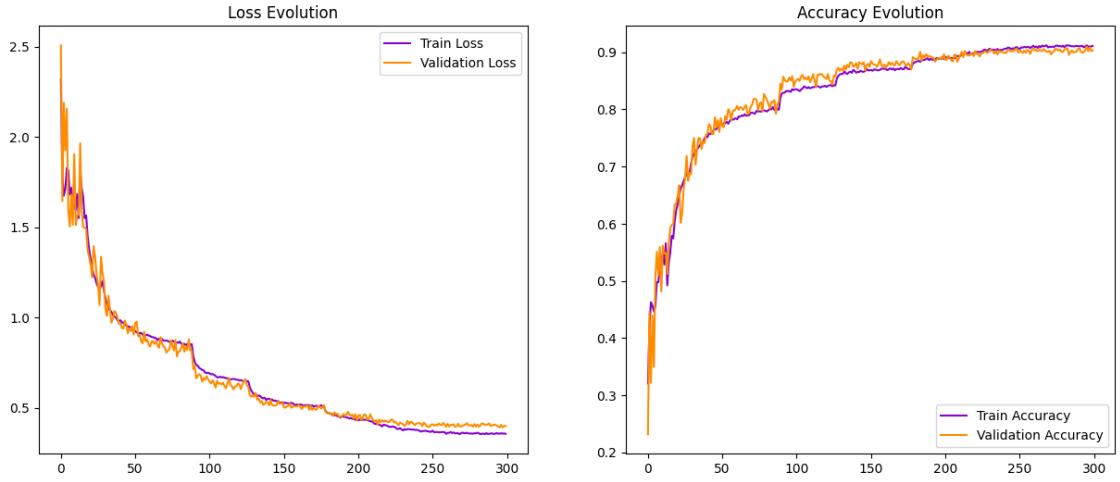
3.Evaluation

[73]:
```
plt.figure(figsize=(15,6))

# Plotting the training and validation loss
plt.subplot(1, 2, 1)
plt.plot(model.history.history['loss'], label='Train Loss', color='#8502d1')
plt.plot(model.history.history['val_loss'], label='Validation Loss',␣
 ↪color='darkorange')
plt.legend()
plt.title('Chart 1. Loss Evolution')

# Plotting the training and validation accuracy
plt.subplot(1, 2, 2)
plt.plot(model.history.history['accuracy'], label='Train Accuracy',␣
 ↪color='#8502d1')
plt.plot(model.history.history['val_accuracy'], label='Validation Accuracy',␣
 ↪color='darkorange')
plt.legend()
plt.title('Chart 2. Accuracy Evolution')

plt.show()
```



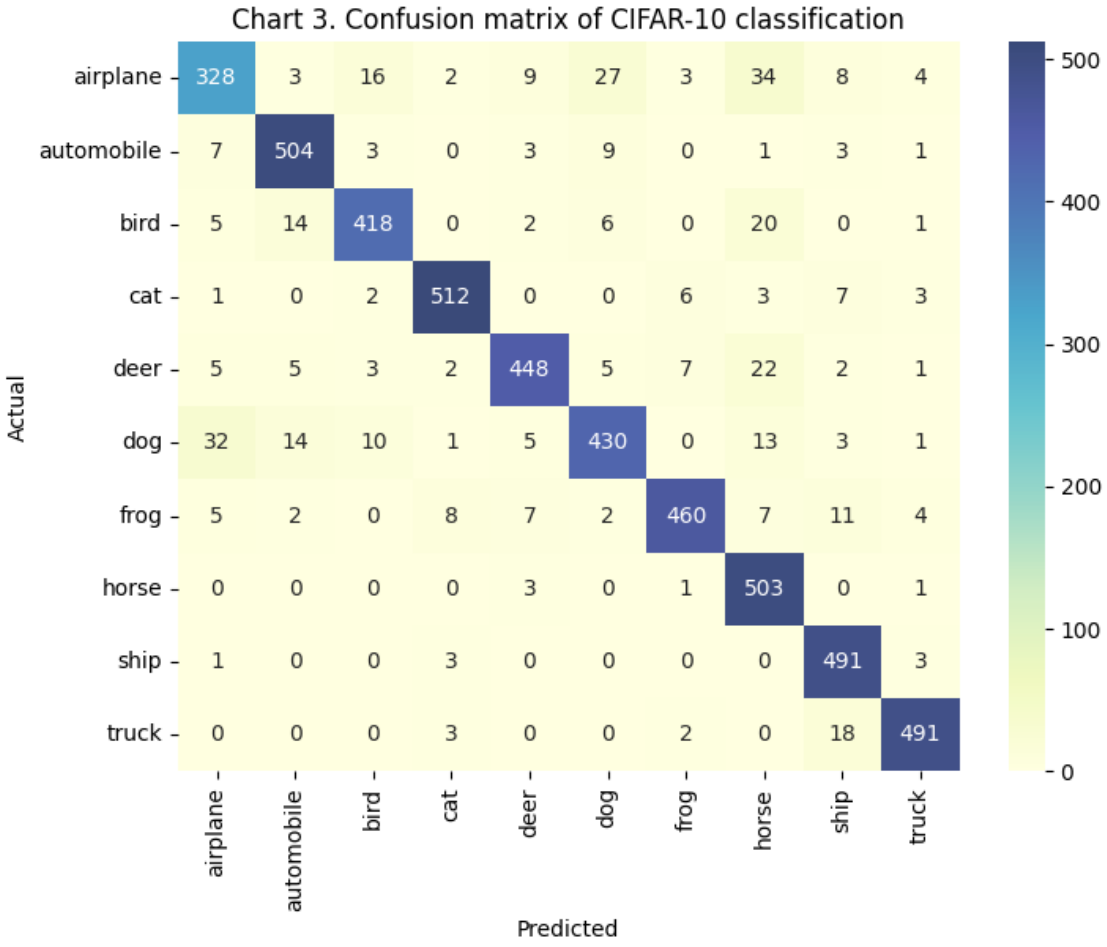
[85]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import pandas as pd

def plot_confusion_matrix_multiclass(y_pred, y, class_names, normalize=False):
    plt.figure(figsize=(8, 6))
    # Create confusion matrix
    cm = confusion_matrix(y, y_pred)


    if (normalize):
        # Normalise
        cmn = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        cm_df = pd.DataFrame(cmn, index=class_names, columns=class_names)
        sns.heatmap(cm_df, annot=True, fmt='.2%', cmap='YlGnBu', alpha=0.8,␣
 ↪vmin=0)
    else:
        cm_df = pd.DataFrame(cm, index=class_names, columns=class_names)
        sns.heatmap(cm_df, annot=True, fmt='d', cmap='YlGnBu', alpha=0.8,␣
 ↪vmin=0)

    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Chart 3. Confusion matrix of CIFAR-10 classification')
y_pred = np.argmax(model.predict(X_val_split), axis=1)
y_true = np.argmax(y_val_split, axis=1)
plot_confusion_matrix_multiclass(y_pred, y_true, class_names=['airplane',␣
 ↪'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',␣
 ↪'truck'], normalize=False)
```

```
157/157             3s 20ms/step
```



[86]:
```
# Calculate the accuracy, precision, f1-score, and recall on the validation data
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.76      0.80       434
           1       0.93      0.95      0.94       531
           2       0.92      0.90      0.91       466
           3       0.96      0.96      0.96       534
           4       0.94      0.90      0.92       500
           5       0.90      0.84      0.87       509
           6       0.96      0.91      0.93       506
           7       0.83      0.99      0.91       508
           8       0.90      0.99      0.94       498
           9       0.96      0.96      0.96       514

    accuracy                           0.92      5000
```

```
   macro avg       0.92      0.91      0.91      5000
weighted avg       0.92      0.92      0.92      5000
```

4. Export result and model

[76]:
```
# Make predictions.
predictions = model.predict(X_test)
predicted_labels = np.argmax(predictions, axis=1)

# Prepare your submission file.
submission = np.column_stack((id_test, predicted_labels))
np.savetxt('submission.csv', submission, delimiter=',', header='id, Label',␣
 ↪comments='', fmt='%d')
```

```
157/157             3s 16ms/step
```

[77]:
```
model.save('cifar_cnn.h5')
```

```
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')`.
```

5. Summary

In Part3, it focused on building and training a Convolutional Neural Network (CNN) model for the CIFAR-10 dataset using Keras and TensorFlow. Below are the key steps and findings:

Data Loading & Preprocessing

- Dataset: The downloaded CIFAR-10 dataset contains 55,000 images (32x32x3) in 10 different classes. 45,000 training images, 5,000 validation images (for hypertuning), 5,000 testing images. All the data are stored under ./dataset/cifa10_images

- Normalization: Normalized the image data by computing the mean and standard deviation, and then scaling the pixel values.
- One-Hot Encoding: Converted training/testing labels to one-hot encoded vectors for use in classification tasks.
- Data Split: Split the training data into training and validation sets.

Modeling and training

- Data Augmentation: Used ImageDataGenerator to apply various transformations to make the model to be more generalizing such as rotation, shift, flip, zoom, and brightness adjustments to augment the training data.
- Model Architecture: Built a CNN model(inspired by kaggle), also applied batch normalization, dropout, L2 regularization to prevent overfitting.

- ReduceLROnPlateau and earlystopping are used, the previous one is significatly useful for the training.

Performance

- Achieved 92% accuracy and weighted recall rate on validation set, and 90.7% accuracy for the kaggle submission.

Disccusion

Despite achieving an overall accuracy of over 90%, there is still room for improvement. Recent advancements, such as the Vision Transformer (ViT) model, can easily achieve 95% accuracy. Even more advanced Transformer models have surpassed 99.5% accuracy. With recent advancements in models and hardware, image classification is becoming a less challenging problem for deep neural networks.