Machine learning & Deep learning Assignment

M.Sc Part II Computer Science

Mithun Parab 509

August 17, 2023

R.J. College of Arts, Science & Commerce
Machine learning & Deep learning
Seat number: 509

Contents

Link for GitHub or Google Colab

1 MLDL Assignment: Design a classifier using CNN

1.1 Introduction:

In the realm of modern machine learning, Convolutional Neural Networks (CNNs) stand as a cornerstone of image classification. These networks are designed to replicate the visual recognition abilities of humans by employing specialized layers that can learn and identify intricate patterns within images. This assignment delves into the development of a classifier through the implementation of a CNN architecture.

1.2 Import packages

[1]:
```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    Conv2D,
    MaxPool2D,
    Dense,
    Flatten,
    Dropout,
    Conv2D,
    BatchNormalization,
)
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.datasets import cifar10
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import cv2

%matplotlib inline
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/tensorflow_io/python/ops/__init__.py:98:
UserWarning: unable to load libtensorflow_io_plugins.so: unable to open file:
libtensorflow_io_plugins.so, from paths: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io_plugins.so: undefined symbol:
_ZN3tsl6StatusC1EN10tensorflow5error4CodeESt17basic_string_viewIcSt11char_traits
IcEENS_14SourceLocationE']
  warnings.warn(f"unable to load libtensorflow_io_plugins.so: {e}")
/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/__init__.py:104: UserWarning: file system
plugins are not loaded: unable to open file: libtensorflow_io.so, from paths:
['/opt/conda/lib/python3.10/site-
```

```
packages/tensorflow_io/python/ops/libtensorflow_io.so']
caused by: ['/opt/conda/lib/python3.10/site-
packages/tensorflow_io/python/ops/libtensorflow_io.so: undefined symbol:
_ZTVN10tensorflow13GcsFileSystemE']
  warnings.warn(f"file system plugins are not loaded: {e}")
```

1.3 Normalization and One hot encoding

Since our data is ready we now need to normalize the data, since normalizing the images in deep learning will produce very good results. Normalizing means we are bringing all the values in the data into a common scale 0-1. This will make out model converge fast and also we will not have any distrotions in the data.

For normalizing the pixel data (Image) we can simply divide the whole pixel values with 255 since pixel values ranges from 0-255. So if we divide them with 255 we automatically normalizee the data between 0-1.

1.3.1 One hot encoding.

CIFAR 10 has 10 categories, in general we should label the categorical data using the one hot encoding.

[2]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

print("Shape of x_train is {}".format(x_train.shape))
print("Shape of x_test is {}".format(x_test.shape))
print("Shape of y_train is {}".format(y_train.shape))
print("Shape of y_test is {}".format(y_test.shape))
# Normalizing
x_train = x_train / 255
x_test = x_test / 255

# One hot encoding
y_train_cat = to_categorical(y_train, 10)
y_test_cat = to_categorical(y_test, 10)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 6s 0us/step
Shape of x_train is (50000, 32, 32, 3)
Shape of x_test is (10000, 32, 32, 3)
Shape of y_train is (50000, 1)
Shape of y_test is (10000, 1)
```

1.4 Build CNN Model

Lets try to train a basic deep learning model. Any deep learning model that needs to classify images use Convolution neural network (CNN). CNN's are proven very effective on image data, also if we have enough data, we can make a deep neural network with multiple CNN layers arranged in specific design to create state of the art results.

I will start with two basic CNN layers, where each layer is attached to a maxpool layer. Max pooling is a great way to reduce the size of parameters without losing much information. As usual in any deep learning model, I need to flatten the intermediate layer results and pass them to a Dense network. Then the dense network result will be passed to a final output layer where the number of units represents the number of categories in the data, which is 10 in our case. Softmax is chosen as the final activation because we need the highest probable class out of 10.

Finally compile your model using adam optimizer.

Let us try to Sequentially build our models.

[3]:
```
model = Sequential()
model.add(
    Conv2D(
        32,
        (3, 3),
        activation="relu",
        kernel_initializer="he_uniform",
        padding="same",
        input_shape=(32, 32, 3),
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        32, (3, 3), activation="relu", kernel_initializer="he_uniform",␣
 ↪padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.2))
model.add(
    Conv2D(
        64, (3, 3), activation="relu", kernel_initializer="he_uniform",␣
 ↪padding="same"
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        64, (3, 3), activation="relu", kernel_initializer="he_uniform",␣
 ↪padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.3))
```

```
model.add(
    Conv2D(
        128, (3, 3), activation="relu", kernel_initializer="he_uniform",␣
 ↪padding="same"
    )
)
model.add(BatchNormalization())
model.add(
    Conv2D(
        128, (3, 3), activation="relu", kernel_initializer="he_uniform",␣
 ↪padding="same"
    )
)
model.add(BatchNormalization())
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.4))
model.add(Flatten())
model.add(Dense(128, activation="relu", kernel_initializer="he_uniform"))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(10, activation="softmax"))
```

1.5 Compile CNN

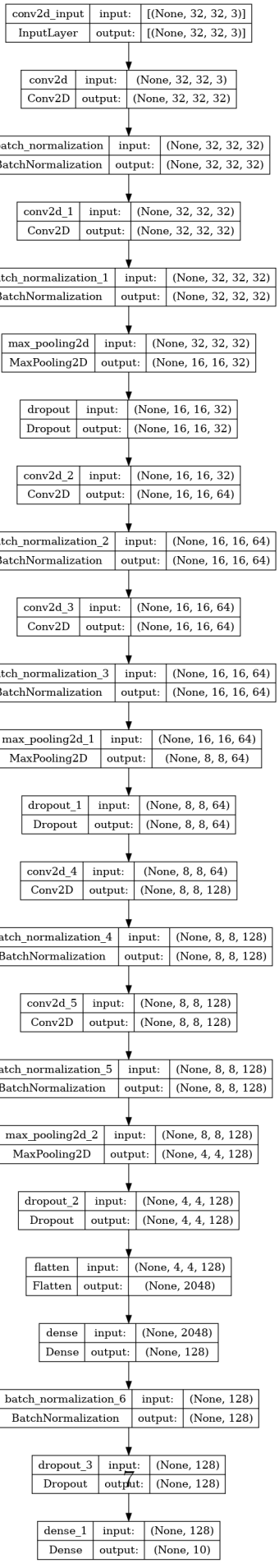
[4]:
```
# compile model
model.compile(optimizer="adam", loss="categorical_crossentropy",␣
 ↪metrics=["accuracy"])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0
```

```
 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 batch_normalization_6 (Batc  (None, 128)              512
 hNormalization)

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 552,874
Trainable params: 551,722
Non-trainable params: 1,152
_________________________________________________________________
```

[5]:
```
tf.keras.utils.plot_model(
    model,
    to_file="model.png",
    show_shapes=True,
    show_layer_names=True,
    expand_nested=False,
)
```

[5]:


conv2d_input | input: | [(None, 32, 32, 3)]
InputLayer | output: | [(None, 32, 32, 3)]
conv2d | input: | (None, 32, 32, 3)
Conv2D | output: | (None, 32, 32, 32)
batch_normalization | input: | (None, 32, 32, 32)
BatchNormalization | output: | (None, 32, 32, 32)
conv2d_1 | input: | (None, 32, 32, 32)
Conv2D | output: | (None, 32, 32, 32)
batch_normalization_1 | input: | (None, 32, 32, 32)
BatchNormalization | output: | (None, 32, 32, 32)
max_pooling2d | input: | (None, 32, 32, 32)
MaxPooling2D | output: | (None, 16, 16, 32)
dropout | input: | (None, 16, 16, 32)
Dropout | output: | (None, 16, 16, 32)
conv2d_2 | input: | (None, 16, 16, 32)
Conv2D | output: | (None, 16, 16, 64)
batch_normalization_2 | input: | (None, 16, 16, 64)
BatchNormalization | output: | (None, 16, 16, 64)
conv2d_3 | input: | (None, 16, 16, 64)
Conv2D | output: | (None, 16, 16, 64)
batch_normalization_3 | input: | (None, 16, 16, 64)
BatchNormalization | output: | (None, 16, 16, 64)
max_pooling2d_1 | input: | (None, 16, 16, 64)
MaxPooling2D | output: | (None, 8, 8, 64)
dropout_1 | input: | (None, 8, 8, 64)
Dropout | output: | (None, 8, 8, 64)
conv2d_4 | input: | (None, 8, 8, 64)
Conv2D | output: | (None, 8, 8, 128)
batch_normalization_4 | input: | (None, 8, 8, 128)
BatchNormalization | output: | (None, 8, 8, 128)
conv2d_5 | input: | (None, 8, 8, 128)
Conv2D | output: | (None, 8, 8, 128)
batch_normalization_5 | input: | (None, 8, 8, 128)
BatchNormalization | output: | (None, 8, 8, 128)
max_pooling2d_2 | input: | (None, 8, 8, 128)
MaxPooling2D | output: | (None, 4, 4, 128)
dropout_2 | input: | (None, 4, 4, 128)
Dropout | output: | (None, 4, 4, 128)
flatten | input: | (None, 4, 4, 128)
Flatten | output: | (None, 2048)
dense | input: | (None, 2048)
Dense | output: | (None, 128)
batch_normalization_6 | input: | (None, 128)
BatchNormalization | output: | (None, 128)
dropout_3 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_1 | input: | (None, 128)
Dense | output: | (None, 10)

[6]:
```
# Image Data Generator , we are shifting image accross width and height also we␣
↪are flipping the image horizantally.
datagen = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    rotation_range=20,
)

it_train = datagen.flow(x_train, y_train_cat)

steps = int(x_train.shape[0] / 64)
num_epochs = 200
```

1.6 Training CNN

[7]:
```
# Fit the model on the batches generated by datagen.flow().
history = model.fit(
    it_train,
    epochs=num_epochs,
    steps_per_epoch=steps,
    validation_data=(x_test, y_test_cat),
    verbose=2,
)
```

```
Epoch 1/200

2023-08-17 13:59:38.315391: E
tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] layout failed:
INVALID_ARGUMENT: Size of values 0 does not match size of permutation 4 @ fanin
shape insequential/dropout/dropout/SelectV2-2-TransposeNHWCToNCHW-
LayoutOptimizer

781/781 - 31s - loss: 2.0085 - accuracy: 0.3215 - val_loss: 1.4300 -
val_accuracy: 0.4833 - 31s/epoch - 40ms/step
Epoch 2/200
781/781 - 18s - loss: 1.5154 - accuracy: 0.4527 - val_loss: 1.2820 -
val_accuracy: 0.5439 - 18s/epoch - 23ms/step
Epoch 3/200
781/781 - 19s - loss: 1.3728 - accuracy: 0.5029 - val_loss: 1.2241 -
val_accuracy: 0.5710 - 19s/epoch - 24ms/step
Epoch 4/200
781/781 - 19s - loss: 1.2819 - accuracy: 0.5436 - val_loss: 1.2127 -
val_accuracy: 0.5729 - 19s/epoch - 24ms/step
Epoch 5/200
781/781 - 18s - loss: 1.2246 - accuracy: 0.5671 - val_loss: 1.1074 -
```

```
val_accuracy: 0.6051 - 18s/epoch - 23ms/step
Epoch 6/200
781/781 - 19s - loss: 1.1618 - accuracy: 0.5908 - val_loss: 0.9671 -
val_accuracy: 0.6624 - 19s/epoch - 24ms/step
Epoch 7/200
781/781 - 18s - loss: 1.1089 - accuracy: 0.6082 - val_loss: 1.0035 -
val_accuracy: 0.6519 - 18s/epoch - 23ms/step
Epoch 8/200
781/781 - 18s - loss: 1.0683 - accuracy: 0.6294 - val_loss: 0.8618 -
val_accuracy: 0.7019 - 18s/epoch - 23ms/step
Epoch 9/200
781/781 - 17s - loss: 1.0277 - accuracy: 0.6404 - val_loss: 0.9863 -
val_accuracy: 0.6669 - 17s/epoch - 22ms/step
Epoch 10/200
781/781 - 17s - loss: 1.0042 - accuracy: 0.6494 - val_loss: 1.0345 -
val_accuracy: 0.6443 - 17s/epoch - 21ms/step
Epoch 11/200
781/781 - 18s - loss: 0.9858 - accuracy: 0.6576 - val_loss: 0.9288 -
val_accuracy: 0.6852 - 18s/epoch - 23ms/step
Epoch 12/200
781/781 - 17s - loss: 0.9604 - accuracy: 0.6701 - val_loss: 0.9185 -
val_accuracy: 0.6922 - 17s/epoch - 22ms/step
Epoch 13/200
781/781 - 17s - loss: 0.9303 - accuracy: 0.6799 - val_loss: 0.8376 -
val_accuracy: 0.7151 - 17s/epoch - 22ms/step
Epoch 14/200
781/781 - 17s - loss: 0.9136 - accuracy: 0.6849 - val_loss: 0.8000 -
val_accuracy: 0.7268 - 17s/epoch - 22ms/step
Epoch 15/200
781/781 - 17s - loss: 0.8933 - accuracy: 0.6928 - val_loss: 0.7934 -
val_accuracy: 0.7323 - 17s/epoch - 21ms/step
Epoch 16/200
781/781 - 17s - loss: 0.8792 - accuracy: 0.6975 - val_loss: 0.7101 -
val_accuracy: 0.7541 - 17s/epoch - 22ms/step
Epoch 17/200
781/781 - 17s - loss: 0.8779 - accuracy: 0.7002 - val_loss: 0.8257 -
val_accuracy: 0.7201 - 17s/epoch - 22ms/step
Epoch 18/200
781/781 - 17s - loss: 0.8573 - accuracy: 0.7072 - val_loss: 0.7502 -
val_accuracy: 0.7389 - 17s/epoch - 22ms/step
Epoch 19/200
781/781 - 17s - loss: 0.8483 - accuracy: 0.7085 - val_loss: 0.6664 -
val_accuracy: 0.7742 - 17s/epoch - 22ms/step
Epoch 20/200
781/781 - 17s - loss: 0.8253 - accuracy: 0.7135 - val_loss: 0.7511 -
val_accuracy: 0.7454 - 17s/epoch - 22ms/step
Epoch 21/200
781/781 - 18s - loss: 0.8176 - accuracy: 0.7190 - val_loss: 0.7141 -
```

```
val_accuracy: 0.7566 - 18s/epoch - 23ms/step
Epoch 22/200
781/781 - 17s - loss: 0.8188 - accuracy: 0.7172 - val_loss: 0.7240 -
val_accuracy: 0.7561 - 17s/epoch - 22ms/step
Epoch 23/200
781/781 - 18s - loss: 0.8144 - accuracy: 0.7222 - val_loss: 0.6937 -
val_accuracy: 0.7687 - 18s/epoch - 23ms/step
Epoch 24/200
781/781 - 18s - loss: 0.8033 - accuracy: 0.7233 - val_loss: 0.7525 -
val_accuracy: 0.7503 - 18s/epoch - 23ms/step
Epoch 25/200
781/781 - 17s - loss: 0.7955 - accuracy: 0.7309 - val_loss: 0.6784 -
val_accuracy: 0.7694 - 17s/epoch - 21ms/step
Epoch 26/200
781/781 - 18s - loss: 0.7928 - accuracy: 0.7291 - val_loss: 0.7181 -
val_accuracy: 0.7605 - 18s/epoch - 23ms/step
Epoch 27/200
781/781 - 17s - loss: 0.7727 - accuracy: 0.7365 - val_loss: 0.6927 -
val_accuracy: 0.7650 - 17s/epoch - 22ms/step
Epoch 28/200
781/781 - 18s - loss: 0.7745 - accuracy: 0.7354 - val_loss: 0.6481 -
val_accuracy: 0.7813 - 18s/epoch - 23ms/step
Epoch 29/200
781/781 - 18s - loss: 0.7757 - accuracy: 0.7387 - val_loss: 0.6572 -
val_accuracy: 0.7777 - 18s/epoch - 23ms/step
Epoch 30/200
781/781 - 17s - loss: 0.7655 - accuracy: 0.7403 - val_loss: 0.6714 -
val_accuracy: 0.7753 - 17s/epoch - 22ms/step
Epoch 31/200
781/781 - 18s - loss: 0.7525 - accuracy: 0.7414 - val_loss: 0.6384 -
val_accuracy: 0.7847 - 18s/epoch - 23ms/step
Epoch 32/200
781/781 - 17s - loss: 0.7480 - accuracy: 0.7467 - val_loss: 0.6977 -
val_accuracy: 0.7728 - 17s/epoch - 21ms/step
Epoch 33/200
781/781 - 18s - loss: 0.7551 - accuracy: 0.7437 - val_loss: 0.5976 -
val_accuracy: 0.7974 - 18s/epoch - 23ms/step
Epoch 34/200
781/781 - 18s - loss: 0.7433 - accuracy: 0.7461 - val_loss: 0.7035 -
val_accuracy: 0.7613 - 18s/epoch - 23ms/step
Epoch 35/200
781/781 - 17s - loss: 0.7494 - accuracy: 0.7448 - val_loss: 0.6130 -
val_accuracy: 0.7970 - 17s/epoch - 22ms/step
Epoch 36/200
781/781 - 17s - loss: 0.7297 - accuracy: 0.7515 - val_loss: 0.6525 -
val_accuracy: 0.7787 - 17s/epoch - 22ms/step
Epoch 37/200
781/781 - 16s - loss: 0.7317 - accuracy: 0.7494 - val_loss: 0.6641 -
```

```
val_accuracy: 0.7763 - 16s/epoch - 21ms/step
Epoch 38/200
781/781 - 17s - loss: 0.7307 - accuracy: 0.7502 - val_loss: 0.5866 -
val_accuracy: 0.7972 - 17s/epoch - 21ms/step
Epoch 39/200
781/781 - 17s - loss: 0.7145 - accuracy: 0.7562 - val_loss: 0.6834 -
val_accuracy: 0.7672 - 17s/epoch - 22ms/step
Epoch 40/200
781/781 - 17s - loss: 0.7248 - accuracy: 0.7536 - val_loss: 0.6352 -
val_accuracy: 0.7862 - 17s/epoch - 21ms/step
Epoch 41/200
781/781 - 18s - loss: 0.7226 - accuracy: 0.7521 - val_loss: 0.6648 -
val_accuracy: 0.7773 - 18s/epoch - 23ms/step
Epoch 42/200
781/781 - 17s - loss: 0.7028 - accuracy: 0.7602 - val_loss: 0.6903 -
val_accuracy: 0.7651 - 17s/epoch - 22ms/step
Epoch 43/200
781/781 - 17s - loss: 0.7137 - accuracy: 0.7568 - val_loss: 0.5948 -
val_accuracy: 0.7996 - 17s/epoch - 22ms/step
Epoch 44/200
781/781 - 18s - loss: 0.7034 - accuracy: 0.7613 - val_loss: 0.6109 -
val_accuracy: 0.7934 - 18s/epoch - 23ms/step
Epoch 45/200
781/781 - 17s - loss: 0.7100 - accuracy: 0.7568 - val_loss: 0.6095 -
val_accuracy: 0.7943 - 17s/epoch - 22ms/step
Epoch 46/200
781/781 - 17s - loss: 0.7056 - accuracy: 0.7593 - val_loss: 0.6220 -
val_accuracy: 0.7907 - 17s/epoch - 22ms/step
Epoch 47/200
781/781 - 17s - loss: 0.6900 - accuracy: 0.7622 - val_loss: 0.5982 -
val_accuracy: 0.8017 - 17s/epoch - 22ms/step
Epoch 48/200
781/781 - 17s - loss: 0.7076 - accuracy: 0.7621 - val_loss: 0.5674 -
val_accuracy: 0.8057 - 17s/epoch - 22ms/step
Epoch 49/200
781/781 - 17s - loss: 0.6923 - accuracy: 0.7666 - val_loss: 0.6180 -
val_accuracy: 0.7901 - 17s/epoch - 22ms/step
Epoch 50/200
781/781 - 16s - loss: 0.6932 - accuracy: 0.7605 - val_loss: 0.6483 -
val_accuracy: 0.7839 - 16s/epoch - 21ms/step
Epoch 51/200
781/781 - 18s - loss: 0.6815 - accuracy: 0.7663 - val_loss: 0.5655 -
val_accuracy: 0.8075 - 18s/epoch - 23ms/step
Epoch 52/200
781/781 - 17s - loss: 0.6896 - accuracy: 0.7650 - val_loss: 0.6706 -
val_accuracy: 0.7764 - 17s/epoch - 21ms/step
Epoch 53/200
781/781 - 18s - loss: 0.6790 - accuracy: 0.7680 - val_loss: 0.5927 -
```

```
val_accuracy: 0.8022 - 18s/epoch - 23ms/step
Epoch 54/200
781/781 - 17s - loss: 0.6769 - accuracy: 0.7688 - val_loss: 0.6368 -
val_accuracy: 0.7842 - 17s/epoch - 22ms/step
Epoch 55/200
781/781 - 17s - loss: 0.6672 - accuracy: 0.7741 - val_loss: 0.5795 -
val_accuracy: 0.8043 - 17s/epoch - 21ms/step
Epoch 56/200
781/781 - 17s - loss: 0.6753 - accuracy: 0.7723 - val_loss: 0.5219 -
val_accuracy: 0.8202 - 17s/epoch - 22ms/step
Epoch 57/200
781/781 - 17s - loss: 0.6710 - accuracy: 0.7720 - val_loss: 0.5683 -
val_accuracy: 0.8085 - 17s/epoch - 21ms/step
Epoch 58/200
781/781 - 18s - loss: 0.6616 - accuracy: 0.7725 - val_loss: 0.5869 -
val_accuracy: 0.8037 - 18s/epoch - 23ms/step
Epoch 59/200
781/781 - 17s - loss: 0.6606 - accuracy: 0.7741 - val_loss: 0.5839 -
val_accuracy: 0.8068 - 17s/epoch - 22ms/step
Epoch 60/200
781/781 - 17s - loss: 0.6748 - accuracy: 0.7731 - val_loss: 0.6409 -
val_accuracy: 0.7869 - 17s/epoch - 21ms/step
Epoch 61/200
781/781 - 17s - loss: 0.6461 - accuracy: 0.7800 - val_loss: 0.6397 -
val_accuracy: 0.7840 - 17s/epoch - 22ms/step
Epoch 62/200
781/781 - 17s - loss: 0.6604 - accuracy: 0.7740 - val_loss: 0.5656 -
val_accuracy: 0.8083 - 17s/epoch - 22ms/step
Epoch 63/200
781/781 - 17s - loss: 0.6671 - accuracy: 0.7744 - val_loss: 0.5492 -
val_accuracy: 0.8133 - 17s/epoch - 22ms/step
Epoch 64/200
781/781 - 17s - loss: 0.6520 - accuracy: 0.7773 - val_loss: 0.5923 -
val_accuracy: 0.8020 - 17s/epoch - 22ms/step
Epoch 65/200
781/781 - 18s - loss: 0.6577 - accuracy: 0.7743 - val_loss: 0.6059 -
val_accuracy: 0.7959 - 18s/epoch - 23ms/step
Epoch 66/200
781/781 - 17s - loss: 0.6474 - accuracy: 0.7801 - val_loss: 0.5560 -
val_accuracy: 0.8125 - 17s/epoch - 22ms/step
Epoch 67/200
781/781 - 17s - loss: 0.6544 - accuracy: 0.7792 - val_loss: 0.5350 -
val_accuracy: 0.8219 - 17s/epoch - 21ms/step
Epoch 68/200
781/781 - 18s - loss: 0.6513 - accuracy: 0.7778 - val_loss: 0.6270 -
val_accuracy: 0.7919 - 18s/epoch - 23ms/step
Epoch 69/200
781/781 - 17s - loss: 0.6490 - accuracy: 0.7778 - val_loss: 0.4966 -
```

```
val_accuracy: 0.8307 - 17s/epoch - 22ms/step
Epoch 70/200
781/781 - 18s - loss: 0.6420 - accuracy: 0.7824 - val_loss: 0.5803 -
val_accuracy: 0.8049 - 18s/epoch - 23ms/step
Epoch 71/200
781/781 - 17s - loss: 0.6393 - accuracy: 0.7828 - val_loss: 0.6194 -
val_accuracy: 0.7953 - 17s/epoch - 22ms/step
Epoch 72/200
781/781 - 17s - loss: 0.6356 - accuracy: 0.7843 - val_loss: 0.5114 -
val_accuracy: 0.8253 - 17s/epoch - 22ms/step
Epoch 73/200
781/781 - 17s - loss: 0.6352 - accuracy: 0.7852 - val_loss: 0.5236 -
val_accuracy: 0.8225 - 17s/epoch - 22ms/step
Epoch 74/200
781/781 - 17s - loss: 0.6304 - accuracy: 0.7870 - val_loss: 0.5759 -
val_accuracy: 0.8043 - 17s/epoch - 22ms/step
Epoch 75/200
781/781 - 18s - loss: 0.6319 - accuracy: 0.7845 - val_loss: 0.5435 -
val_accuracy: 0.8170 - 18s/epoch - 23ms/step
Epoch 76/200
781/781 - 17s - loss: 0.6329 - accuracy: 0.7855 - val_loss: 0.5084 -
val_accuracy: 0.8269 - 17s/epoch - 22ms/step
Epoch 77/200
781/781 - 17s - loss: 0.6310 - accuracy: 0.7832 - val_loss: 0.5949 -
val_accuracy: 0.7987 - 17s/epoch - 22ms/step
Epoch 78/200
781/781 - 17s - loss: 0.6277 - accuracy: 0.7870 - val_loss: 0.5328 -
val_accuracy: 0.8228 - 17s/epoch - 22ms/step
Epoch 79/200
781/781 - 17s - loss: 0.6314 - accuracy: 0.7862 - val_loss: 0.5531 -
val_accuracy: 0.8148 - 17s/epoch - 22ms/step
Epoch 80/200
781/781 - 18s - loss: 0.6337 - accuracy: 0.7866 - val_loss: 0.4871 -
val_accuracy: 0.8328 - 18s/epoch - 23ms/step
Epoch 81/200
781/781 - 17s - loss: 0.6254 - accuracy: 0.7881 - val_loss: 0.5451 -
val_accuracy: 0.8200 - 17s/epoch - 21ms/step
Epoch 82/200
781/781 - 18s - loss: 0.6349 - accuracy: 0.7843 - val_loss: 0.5630 -
val_accuracy: 0.8100 - 18s/epoch - 23ms/step
Epoch 83/200
781/781 - 17s - loss: 0.6306 - accuracy: 0.7850 - val_loss: 0.5192 -
val_accuracy: 0.8241 - 17s/epoch - 21ms/step
Epoch 84/200
781/781 - 18s - loss: 0.6148 - accuracy: 0.7900 - val_loss: 0.5285 -
val_accuracy: 0.8243 - 18s/epoch - 23ms/step
Epoch 85/200
781/781 - 17s - loss: 0.6221 - accuracy: 0.7890 - val_loss: 0.5135 -
```

```
val_accuracy: 0.8264 - 17s/epoch - 22ms/step
Epoch 86/200
781/781 - 17s - loss: 0.6136 - accuracy: 0.7916 - val_loss: 0.6065 -
val_accuracy: 0.7976 - 17s/epoch - 21ms/step
Epoch 87/200
781/781 - 18s - loss: 0.6201 - accuracy: 0.7909 - val_loss: 0.5226 -
val_accuracy: 0.8212 - 18s/epoch - 23ms/step
Epoch 88/200
781/781 - 17s - loss: 0.6189 - accuracy: 0.7901 - val_loss: 0.5095 -
val_accuracy: 0.8282 - 17s/epoch - 21ms/step
Epoch 89/200
781/781 - 18s - loss: 0.6191 - accuracy: 0.7881 - val_loss: 0.5392 -
val_accuracy: 0.8170 - 18s/epoch - 23ms/step
Epoch 90/200
781/781 - 17s - loss: 0.6184 - accuracy: 0.7873 - val_loss: 0.5210 -
val_accuracy: 0.8247 - 17s/epoch - 22ms/step
Epoch 91/200
781/781 - 17s - loss: 0.6155 - accuracy: 0.7878 - val_loss: 0.5725 -
val_accuracy: 0.8102 - 17s/epoch - 22ms/step
Epoch 92/200
781/781 - 18s - loss: 0.6148 - accuracy: 0.7911 - val_loss: 0.5373 -
val_accuracy: 0.8227 - 18s/epoch - 23ms/step
Epoch 93/200
781/781 - 16s - loss: 0.6144 - accuracy: 0.7908 - val_loss: 0.4847 -
val_accuracy: 0.8343 - 16s/epoch - 21ms/step
Epoch 94/200
781/781 - 17s - loss: 0.6106 - accuracy: 0.7910 - val_loss: 0.5485 -
val_accuracy: 0.8175 - 17s/epoch - 22ms/step
Epoch 95/200
781/781 - 17s - loss: 0.5982 - accuracy: 0.7962 - val_loss: 0.5968 -
val_accuracy: 0.8056 - 17s/epoch - 22ms/step
Epoch 96/200
781/781 - 17s - loss: 0.6072 - accuracy: 0.7945 - val_loss: 0.5416 -
val_accuracy: 0.8201 - 17s/epoch - 22ms/step
Epoch 97/200
781/781 - 17s - loss: 0.6021 - accuracy: 0.7953 - val_loss: 0.5162 -
val_accuracy: 0.8249 - 17s/epoch - 22ms/step
Epoch 98/200
781/781 - 17s - loss: 0.6173 - accuracy: 0.7912 - val_loss: 0.4936 -
val_accuracy: 0.8364 - 17s/epoch - 22ms/step
Epoch 99/200
781/781 - 18s - loss: 0.6018 - accuracy: 0.7965 - val_loss: 0.5676 -
val_accuracy: 0.8113 - 18s/epoch - 23ms/step
Epoch 100/200
781/781 - 17s - loss: 0.6019 - accuracy: 0.7967 - val_loss: 0.5425 -
val_accuracy: 0.8181 - 17s/epoch - 21ms/step
Epoch 101/200
781/781 - 18s - loss: 0.5995 - accuracy: 0.7958 - val_loss: 0.5538 -
```

```
val_accuracy: 0.8157 - 18s/epoch - 23ms/step
Epoch 102/200
781/781 - 17s - loss: 0.6110 - accuracy: 0.7928 - val_loss: 0.5077 -
val_accuracy: 0.8270 - 17s/epoch - 22ms/step
Epoch 103/200
781/781 - 17s - loss: 0.6008 - accuracy: 0.7961 - val_loss: 0.5326 -
val_accuracy: 0.8202 - 17s/epoch - 22ms/step
Epoch 104/200
781/781 - 17s - loss: 0.6080 - accuracy: 0.7930 - val_loss: 0.5338 -
val_accuracy: 0.8208 - 17s/epoch - 22ms/step
Epoch 105/200
781/781 - 18s - loss: 0.5984 - accuracy: 0.7974 - val_loss: 0.4947 -
val_accuracy: 0.8338 - 18s/epoch - 23ms/step
Epoch 106/200
781/781 - 18s - loss: 0.5931 - accuracy: 0.7992 - val_loss: 0.5096 -
val_accuracy: 0.8259 - 18s/epoch - 22ms/step
Epoch 107/200
781/781 - 17s - loss: 0.6033 - accuracy: 0.7919 - val_loss: 0.4699 -
val_accuracy: 0.8408 - 17s/epoch - 22ms/step
Epoch 108/200
781/781 - 18s - loss: 0.5951 - accuracy: 0.7981 - val_loss: 0.5593 -
val_accuracy: 0.8164 - 18s/epoch - 23ms/step
Epoch 109/200
781/781 - 18s - loss: 0.6076 - accuracy: 0.7963 - val_loss: 0.5190 -
val_accuracy: 0.8248 - 18s/epoch - 23ms/step
Epoch 110/200
781/781 - 19s - loss: 0.6091 - accuracy: 0.7925 - val_loss: 0.5110 -
val_accuracy: 0.8275 - 19s/epoch - 24ms/step
Epoch 111/200
781/781 - 18s - loss: 0.5941 - accuracy: 0.7978 - val_loss: 0.5041 -
val_accuracy: 0.8285 - 18s/epoch - 23ms/step
Epoch 112/200
781/781 - 18s - loss: 0.5903 - accuracy: 0.8029 - val_loss: 0.5166 -
val_accuracy: 0.8275 - 18s/epoch - 23ms/step
Epoch 113/200
781/781 - 18s - loss: 0.5870 - accuracy: 0.8029 - val_loss: 0.4917 -
val_accuracy: 0.8345 - 18s/epoch - 23ms/step
Epoch 114/200
781/781 - 17s - loss: 0.5926 - accuracy: 0.7981 - val_loss: 0.5179 -
val_accuracy: 0.8240 - 17s/epoch - 22ms/step
Epoch 115/200
781/781 - 18s - loss: 0.5900 - accuracy: 0.8005 - val_loss: 0.5250 -
val_accuracy: 0.8251 - 18s/epoch - 23ms/step
Epoch 116/200
781/781 - 18s - loss: 0.5955 - accuracy: 0.8003 - val_loss: 0.5351 -
val_accuracy: 0.8178 - 18s/epoch - 22ms/step
Epoch 117/200
781/781 - 18s - loss: 0.5914 - accuracy: 0.8004 - val_loss: 0.5631 -
```

```
val_accuracy: 0.8141 - 18s/epoch - 24ms/step
Epoch 118/200
781/781 - 18s - loss: 0.5948 - accuracy: 0.7993 - val_loss: 0.5212 -
val_accuracy: 0.8271 - 18s/epoch - 22ms/step
Epoch 119/200
781/781 - 19s - loss: 0.5850 - accuracy: 0.8022 - val_loss: 0.5072 -
val_accuracy: 0.8251 - 19s/epoch - 24ms/step
Epoch 120/200
781/781 - 19s - loss: 0.5975 - accuracy: 0.7977 - val_loss: 0.4905 -
val_accuracy: 0.8336 - 19s/epoch - 24ms/step
Epoch 121/200
781/781 - 17s - loss: 0.5835 - accuracy: 0.8030 - val_loss: 0.4544 -
val_accuracy: 0.8436 - 17s/epoch - 22ms/step
Epoch 122/200
781/781 - 19s - loss: 0.5894 - accuracy: 0.8007 - val_loss: 0.4836 -
val_accuracy: 0.8373 - 19s/epoch - 24ms/step
Epoch 123/200
781/781 - 18s - loss: 0.5859 - accuracy: 0.8011 - val_loss: 0.5485 -
val_accuracy: 0.8168 - 18s/epoch - 23ms/step
Epoch 124/200
781/781 - 18s - loss: 0.5829 - accuracy: 0.8031 - val_loss: 0.5740 -
val_accuracy: 0.8062 - 18s/epoch - 23ms/step
Epoch 125/200
781/781 - 19s - loss: 0.5763 - accuracy: 0.8049 - val_loss: 0.5329 -
val_accuracy: 0.8228 - 19s/epoch - 24ms/step
Epoch 126/200
781/781 - 18s - loss: 0.5968 - accuracy: 0.7961 - val_loss: 0.5344 -
val_accuracy: 0.8198 - 18s/epoch - 22ms/step
Epoch 127/200
781/781 - 18s - loss: 0.5830 - accuracy: 0.8029 - val_loss: 0.5240 -
val_accuracy: 0.8244 - 18s/epoch - 24ms/step
Epoch 128/200
781/781 - 17s - loss: 0.5773 - accuracy: 0.8028 - val_loss: 0.4743 -
val_accuracy: 0.8397 - 17s/epoch - 22ms/step
Epoch 129/200
781/781 - 18s - loss: 0.5782 - accuracy: 0.8026 - val_loss: 0.5885 -
val_accuracy: 0.7990 - 18s/epoch - 23ms/step
Epoch 130/200
781/781 - 17s - loss: 0.5761 - accuracy: 0.8016 - val_loss: 0.5289 -
val_accuracy: 0.8225 - 17s/epoch - 22ms/step
Epoch 131/200
781/781 - 18s - loss: 0.5775 - accuracy: 0.8032 - val_loss: 0.4896 -
val_accuracy: 0.8338 - 18s/epoch - 23ms/step
Epoch 132/200
781/781 - 18s - loss: 0.5774 - accuracy: 0.8047 - val_loss: 0.4770 -
val_accuracy: 0.8365 - 18s/epoch - 24ms/step
Epoch 133/200
781/781 - 17s - loss: 0.5677 - accuracy: 0.8055 - val_loss: 0.4673 -
```

```
val_accuracy: 0.8393 - 17s/epoch - 22ms/step
Epoch 134/200
781/781 - 18s - loss: 0.5637 - accuracy: 0.8091 - val_loss: 0.4801 -
val_accuracy: 0.8373 - 18s/epoch - 23ms/step
Epoch 135/200
781/781 - 18s - loss: 0.5698 - accuracy: 0.8075 - val_loss: 0.5211 -
val_accuracy: 0.8257 - 18s/epoch - 23ms/step
Epoch 136/200
781/781 - 18s - loss: 0.5701 - accuracy: 0.8036 - val_loss: 0.5058 -
val_accuracy: 0.8334 - 18s/epoch - 23ms/step
Epoch 137/200
781/781 - 18s - loss: 0.5677 - accuracy: 0.8062 - val_loss: 0.5012 -
val_accuracy: 0.8318 - 18s/epoch - 23ms/step
Epoch 138/200
781/781 - 18s - loss: 0.5683 - accuracy: 0.8064 - val_loss: 0.5459 -
val_accuracy: 0.8148 - 18s/epoch - 23ms/step
Epoch 139/200
781/781 - 18s - loss: 0.5791 - accuracy: 0.8047 - val_loss: 0.5002 -
val_accuracy: 0.8311 - 18s/epoch - 23ms/step
Epoch 140/200
781/781 - 19s - loss: 0.5708 - accuracy: 0.8062 - val_loss: 0.4597 -
val_accuracy: 0.8436 - 19s/epoch - 24ms/step
Epoch 141/200
781/781 - 18s - loss: 0.5726 - accuracy: 0.8054 - val_loss: 0.5119 -
val_accuracy: 0.8279 - 18s/epoch - 23ms/step
Epoch 142/200
781/781 - 18s - loss: 0.5684 - accuracy: 0.8053 - val_loss: 0.4959 -
val_accuracy: 0.8313 - 18s/epoch - 23ms/step
Epoch 143/200
781/781 - 18s - loss: 0.5602 - accuracy: 0.8112 - val_loss: 0.4949 -
val_accuracy: 0.8332 - 18s/epoch - 23ms/step
Epoch 144/200
781/781 - 19s - loss: 0.5724 - accuracy: 0.8069 - val_loss: 0.4558 -
val_accuracy: 0.8454 - 19s/epoch - 24ms/step
Epoch 145/200
781/781 - 18s - loss: 0.5739 - accuracy: 0.8076 - val_loss: 0.4472 -
val_accuracy: 0.8491 - 18s/epoch - 22ms/step
Epoch 146/200
781/781 - 17s - loss: 0.5692 - accuracy: 0.8075 - val_loss: 0.4648 -
val_accuracy: 0.8433 - 17s/epoch - 22ms/step
Epoch 147/200
781/781 - 18s - loss: 0.5610 - accuracy: 0.8115 - val_loss: 0.4796 -
val_accuracy: 0.8397 - 18s/epoch - 24ms/step
Epoch 148/200
781/781 - 18s - loss: 0.5743 - accuracy: 0.8048 - val_loss: 0.4867 -
val_accuracy: 0.8355 - 18s/epoch - 22ms/step
Epoch 149/200
781/781 - 18s - loss: 0.5686 - accuracy: 0.8091 - val_loss: 0.4427 -
```

```
val_accuracy: 0.8499 - 18s/epoch - 24ms/step
Epoch 150/200
781/781 - 18s - loss: 0.5702 - accuracy: 0.8068 - val_loss: 0.4938 -
val_accuracy: 0.8374 - 18s/epoch - 23ms/step
Epoch 151/200
781/781 - 18s - loss: 0.5626 - accuracy: 0.8087 - val_loss: 0.5088 -
val_accuracy: 0.8310 - 18s/epoch - 23ms/step
Epoch 152/200
781/781 - 18s - loss: 0.5721 - accuracy: 0.8047 - val_loss: 0.5268 -
val_accuracy: 0.8263 - 18s/epoch - 23ms/step
Epoch 153/200
781/781 - 18s - loss: 0.5628 - accuracy: 0.8099 - val_loss: 0.4576 -
val_accuracy: 0.8423 - 18s/epoch - 23ms/step
Epoch 154/200
781/781 - 18s - loss: 0.5678 - accuracy: 0.8086 - val_loss: 0.4356 -
val_accuracy: 0.8521 - 18s/epoch - 24ms/step
Epoch 155/200
781/781 - 17s - loss: 0.5668 - accuracy: 0.8085 - val_loss: 0.4649 -
val_accuracy: 0.8442 - 17s/epoch - 22ms/step
Epoch 156/200
781/781 - 19s - loss: 0.5632 - accuracy: 0.8073 - val_loss: 0.4801 -
val_accuracy: 0.8381 - 19s/epoch - 24ms/step
Epoch 157/200
781/781 - 18s - loss: 0.5528 - accuracy: 0.8117 - val_loss: 0.4869 -
val_accuracy: 0.8384 - 18s/epoch - 24ms/step
Epoch 158/200
781/781 - 18s - loss: 0.5567 - accuracy: 0.8097 - val_loss: 0.4789 -
val_accuracy: 0.8362 - 18s/epoch - 23ms/step
Epoch 159/200
781/781 - 18s - loss: 0.5582 - accuracy: 0.8099 - val_loss: 0.5153 -
val_accuracy: 0.8285 - 18s/epoch - 24ms/step
Epoch 160/200
781/781 - 18s - loss: 0.5660 - accuracy: 0.8082 - val_loss: 0.4525 -
val_accuracy: 0.8485 - 18s/epoch - 23ms/step
Epoch 161/200
781/781 - 18s - loss: 0.5517 - accuracy: 0.8139 - val_loss: 0.4728 -
val_accuracy: 0.8435 - 18s/epoch - 24ms/step
Epoch 162/200
781/781 - 18s - loss: 0.5534 - accuracy: 0.8121 - val_loss: 0.5143 -
val_accuracy: 0.8288 - 18s/epoch - 23ms/step
Epoch 163/200
781/781 - 18s - loss: 0.5514 - accuracy: 0.8132 - val_loss: 0.4709 -
val_accuracy: 0.8435 - 18s/epoch - 22ms/step
Epoch 164/200
781/781 - 19s - loss: 0.5632 - accuracy: 0.8076 - val_loss: 0.4862 -
val_accuracy: 0.8381 - 19s/epoch - 24ms/step
Epoch 165/200
781/781 - 18s - loss: 0.5589 - accuracy: 0.8090 - val_loss: 0.4716 -
```

```
val_accuracy: 0.8408 - 18s/epoch - 23ms/step
Epoch 166/200
781/781 - 18s - loss: 0.5582 - accuracy: 0.8106 - val_loss: 0.5238 -
val_accuracy: 0.8205 - 18s/epoch - 23ms/step
Epoch 167/200
781/781 - 18s - loss: 0.5544 - accuracy: 0.8095 - val_loss: 0.4759 -
val_accuracy: 0.8430 - 18s/epoch - 23ms/step
Epoch 168/200
781/781 - 18s - loss: 0.5640 - accuracy: 0.8079 - val_loss: 0.4785 -
val_accuracy: 0.8379 - 18s/epoch - 23ms/step
Epoch 169/200
781/781 - 18s - loss: 0.5454 - accuracy: 0.8120 - val_loss: 0.4976 -
val_accuracy: 0.8383 - 18s/epoch - 24ms/step
Epoch 170/200
781/781 - 18s - loss: 0.5520 - accuracy: 0.8129 - val_loss: 0.4584 -
val_accuracy: 0.8465 - 18s/epoch - 23ms/step
Epoch 171/200
781/781 - 18s - loss: 0.5567 - accuracy: 0.8115 - val_loss: 0.4741 -
val_accuracy: 0.8436 - 18s/epoch - 23ms/step
Epoch 172/200
781/781 - 19s - loss: 0.5510 - accuracy: 0.8121 - val_loss: 0.4738 -
val_accuracy: 0.8421 - 19s/epoch - 24ms/step
Epoch 173/200
781/781 - 18s - loss: 0.5424 - accuracy: 0.8158 - val_loss: 0.4765 -
val_accuracy: 0.8403 - 18s/epoch - 23ms/step
Epoch 174/200
781/781 - 19s - loss: 0.5559 - accuracy: 0.8106 - val_loss: 0.4959 -
val_accuracy: 0.8354 - 19s/epoch - 24ms/step
Epoch 175/200
781/781 - 18s - loss: 0.5568 - accuracy: 0.8124 - val_loss: 0.4849 -
val_accuracy: 0.8377 - 18s/epoch - 23ms/step
Epoch 176/200
781/781 - 18s - loss: 0.5613 - accuracy: 0.8063 - val_loss: 0.4809 -
val_accuracy: 0.8428 - 18s/epoch - 23ms/step
Epoch 177/200
781/781 - 19s - loss: 0.5512 - accuracy: 0.8131 - val_loss: 0.4762 -
val_accuracy: 0.8426 - 19s/epoch - 24ms/step
Epoch 178/200
781/781 - 18s - loss: 0.5469 - accuracy: 0.8127 - val_loss: 0.4831 -
val_accuracy: 0.8374 - 18s/epoch - 23ms/step
Epoch 179/200
781/781 - 19s - loss: 0.5503 - accuracy: 0.8129 - val_loss: 0.4670 -
val_accuracy: 0.8452 - 19s/epoch - 24ms/step
Epoch 180/200
781/781 - 19s - loss: 0.5499 - accuracy: 0.8141 - val_loss: 0.4704 -
val_accuracy: 0.8435 - 19s/epoch - 24ms/step
Epoch 181/200
781/781 - 18s - loss: 0.5464 - accuracy: 0.8140 - val_loss: 0.4911 -
```

```
val_accuracy: 0.8388 - 18s/epoch - 23ms/step
Epoch 182/200
781/781 - 19s - loss: 0.5420 - accuracy: 0.8167 - val_loss: 0.4961 -
val_accuracy: 0.8354 - 19s/epoch - 24ms/step
Epoch 183/200
781/781 - 18s - loss: 0.5531 - accuracy: 0.8127 - val_loss: 0.4964 -
val_accuracy: 0.8356 - 18s/epoch - 23ms/step
Epoch 184/200
781/781 - 18s - loss: 0.5434 - accuracy: 0.8164 - val_loss: 0.5106 -
val_accuracy: 0.8323 - 18s/epoch - 24ms/step
Epoch 185/200
781/781 - 19s - loss: 0.5438 - accuracy: 0.8163 - val_loss: 0.5092 -
val_accuracy: 0.8290 - 19s/epoch - 24ms/step
Epoch 186/200
781/781 - 17s - loss: 0.5362 - accuracy: 0.8176 - val_loss: 0.4706 -
val_accuracy: 0.8429 - 17s/epoch - 22ms/step
Epoch 187/200
781/781 - 18s - loss: 0.5346 - accuracy: 0.8189 - val_loss: 0.4520 -
val_accuracy: 0.8495 - 18s/epoch - 24ms/step
Epoch 188/200
781/781 - 18s - loss: 0.5444 - accuracy: 0.8142 - val_loss: 0.4709 -
val_accuracy: 0.8398 - 18s/epoch - 24ms/step
Epoch 189/200
781/781 - 18s - loss: 0.5407 - accuracy: 0.8167 - val_loss: 0.5334 -
val_accuracy: 0.8246 - 18s/epoch - 24ms/step
Epoch 190/200
781/781 - 19s - loss: 0.5405 - accuracy: 0.8168 - val_loss: 0.4673 -
val_accuracy: 0.8403 - 19s/epoch - 24ms/step
Epoch 191/200
781/781 - 18s - loss: 0.5446 - accuracy: 0.8151 - val_loss: 0.5201 -
val_accuracy: 0.8284 - 18s/epoch - 23ms/step
Epoch 192/200
781/781 - 19s - loss: 0.5456 - accuracy: 0.8148 - val_loss: 0.4532 -
val_accuracy: 0.8455 - 19s/epoch - 24ms/step
Epoch 193/200
781/781 - 18s - loss: 0.5380 - accuracy: 0.8185 - val_loss: 0.4697 -
val_accuracy: 0.8433 - 18s/epoch - 23ms/step
Epoch 194/200
781/781 - 19s - loss: 0.5446 - accuracy: 0.8176 - val_loss: 0.5053 -
val_accuracy: 0.8290 - 19s/epoch - 24ms/step
Epoch 195/200
781/781 - 19s - loss: 0.5445 - accuracy: 0.8160 - val_loss: 0.4353 -
val_accuracy: 0.8521 - 19s/epoch - 24ms/step
Epoch 196/200
781/781 - 18s - loss: 0.5452 - accuracy: 0.8157 - val_loss: 0.5016 -
val_accuracy: 0.8322 - 18s/epoch - 23ms/step
Epoch 197/200
781/781 - 18s - loss: 0.5387 - accuracy: 0.8173 - val_loss: 0.4888 -
```

```
val_accuracy: 0.8342 - 18s/epoch - 24ms/step
Epoch 198/200
781/781 - 18s - loss: 0.5402 - accuracy: 0.8182 - val_loss: 0.4988 -
val_accuracy: 0.8318 - 18s/epoch - 22ms/step
Epoch 199/200
781/781 - 18s - loss: 0.5424 - accuracy: 0.8148 - val_loss: 0.4703 -
val_accuracy: 0.8444 - 18s/epoch - 23ms/step
Epoch 200/200
781/781 - 18s - loss: 0.5518 - accuracy: 0.8124 - val_loss: 0.4877 -
val_accuracy: 0.8378 - 18s/epoch - 23ms/step
```

1.7 Evaluation

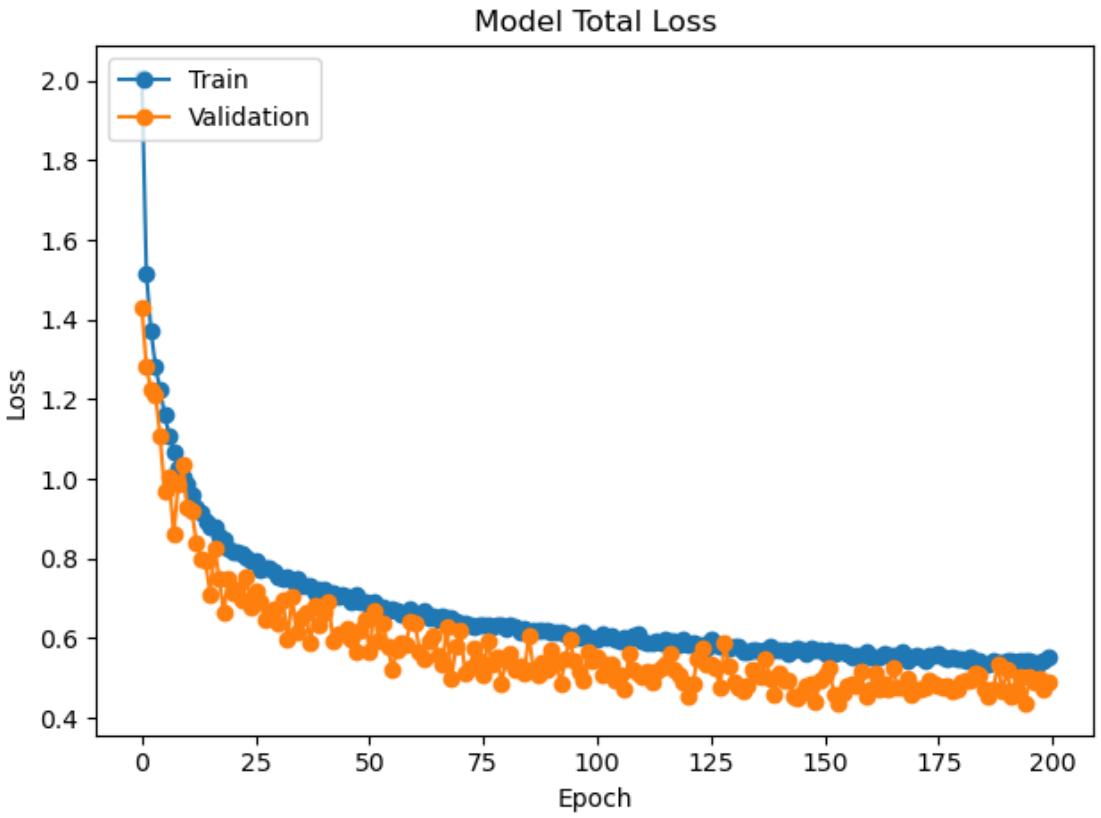
[8]:
```
evaluation_t = model.evaluate(it_train)
print(f"[Info] Train Accuracy: {evaluation_t[1]}")
```

```
1563/1563 [==============================] - 31s 20ms/step - loss: 0.3601 -
accuracy: 0.8741
[Info] Train Accuracy: 0.8741199970245361
```

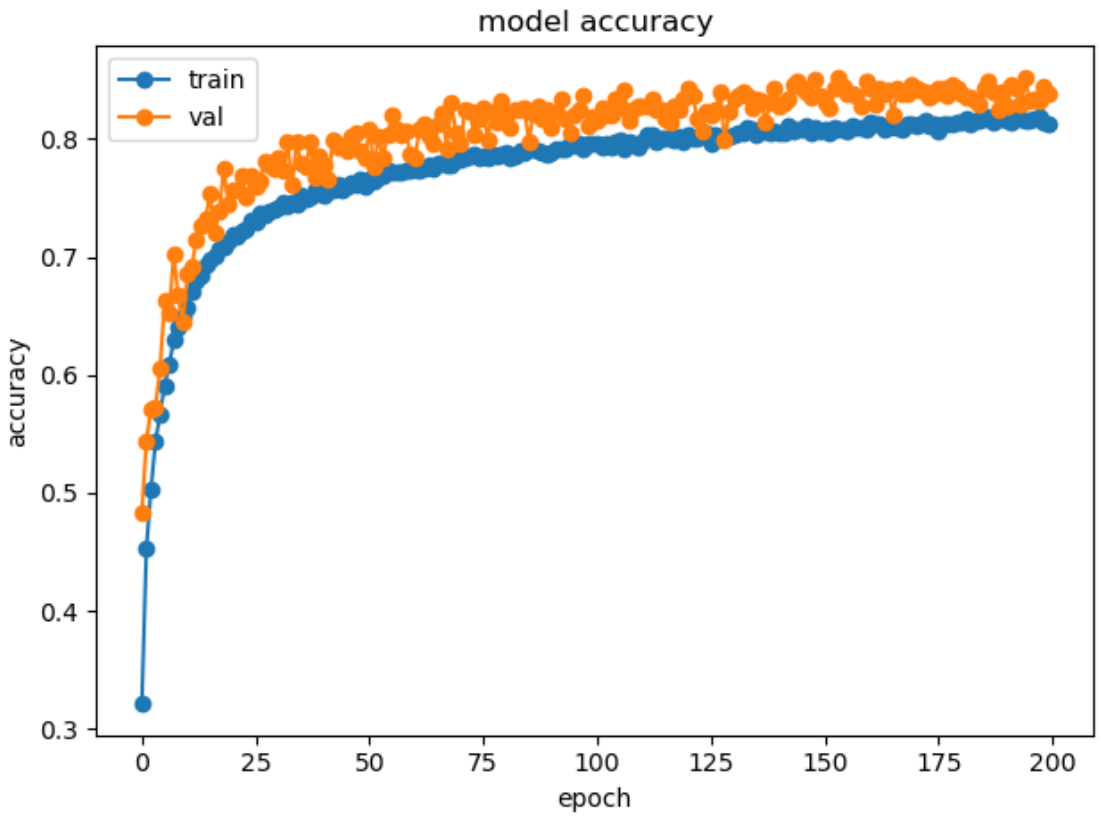
[9]:
```
evaluation = model.evaluate(x_test, y_test_cat)
print(f"[Info] Test Accuracy: {evaluation[1]}")
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.4877 -
accuracy: 0.8378
[Info] Test Accuracy: 0.8378000259399414
```

[10]:
```
try:
    plt.plot(history.history['loss'], marker='o')
    plt.plot(history.history['val_loss'], marker='o')
    plt.title('Model Total Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Adjust the rotation angle of x-axis tick labels
    plt.xticks(rotation=0)  # Set rotation angle to 0 degrees

    plt.tight_layout()  # Adjust layout to prevent clipping of labels
    plt.show()
except Exception as e:
    print(e)
```



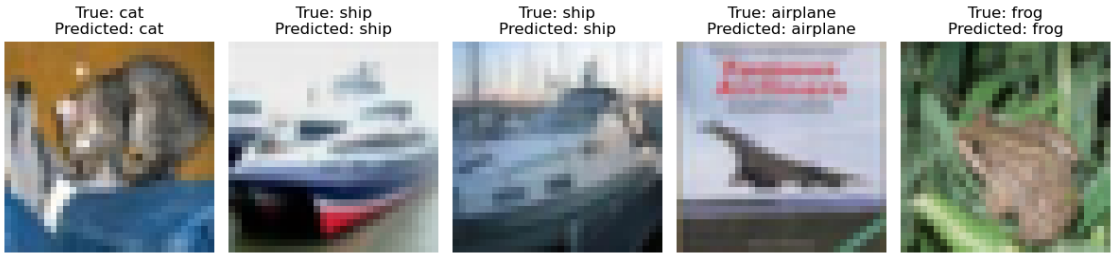
[11]:
```
try:
    plt.plot(history.history['accuracy'], marker='o',)
    plt.plot(history.history['val_accuracy'], marker='o',)
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper left')
    # Adjust the rotation angle of x-axis tick labels
    plt.xticks(rotation=0)  # Set rotation angle to 0 degrees

    plt.tight_layout()  # Adjust layout to prevent clipping of labels
    plt.show()
except Exception as e:
    print(e)
```



[12]:
```
def plot_input_vs_predictions(model: tf.keras.models.Model,
                              x_test: np.ndarray,
                              y_test: np.ndarray,
                              class_names: list,
                              num_samples=5):
    num_classes = len(class_names)
    y_pred = model.predict(x_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    plt.figure(figsize=(12, 6))
    for i in range(num_samples):
        plt.subplot(1, num_samples, i + 1)
        plt.imshow(x_test[i])
        true_label = np.argmax(y_test[i])
        predicted_label = y_pred_classes[i]
        plt.title(f'True: {class_names[true_label]}\nPredicted:␣
↪{class_names[predicted_label]}')
        plt.axis('off')

    plt.tight_layout()
```

```
    plt.show()

cifar10_class_names = [
    'airplane', 'automobile', 'bird', 'cat', 'deer',
    'dog', 'frog', 'horse', 'ship', 'truck'
]

try:
    # Plot input images vs. predictions
    plot_input_vs_predictions(model, x_test, y_test_cat,␣
 ↪class_names=cifar10_class_names, num_samples=5)
except Exception as e:
    print(e)
```

```
313/313 [==============================] - 1s 2ms/step
```



[13]:
```
def render_and_save_examples(model, example_data, true_labels, class_names,␣
 ↪image_size):
    examples_number = example_data.shape[0]
    video_output_path = 'output_video.mp4'
    codec = cv2.VideoWriter_fourcc(*'mp4v')
    vid_width_height = 1280, 720
    vw = cv2.VideoWriter(video_output_path, codec, 30, vid_width_height)

    font_face = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1.3
    thickness = 2

    for i in range(examples_number):
        image = example_data[i]
        image_disp = cv2.resize(image * 255, (720, 720))

        true_label = true_labels[i][0]

        # Generate predictions using the model
        predictions = model.predict(np.expand_dims(example_data[i], axis=0),␣
 ↪verbose=0)
```

```
        predicted_label = np.argmax(predictions)

        predicted_score = predictions[0]
        top_classes = np.argsort(predicted_score)[::-1]

        title = f"True: {class_names[true_label]}, Predicted:␣
↪{class_names[predicted_label]}"

        img = np.zeros((720, 1280, 3), dtype=np.uint8)
        img[:720, :720, :] = image_disp

        x, y = 740, 60
        is_correct = true_label == predicted_label
        txt_color = (100, 255, 0) if is_correct else (0, 0, 255)
        cv2.putText(img, text=title, org=(x, y), fontScale=font_scale,␣
↪fontFace=font_face,
                    thickness=thickness, color=txt_color, lineType=cv2.LINE_AA)

        bar_x, bar_y = 740, 130
        for j, class_index in enumerate(top_classes):
            if j < 10:
                p = predicted_score[class_index] * 100
                rect_width = int(p * 3.3)
                rect_start = 180
                color = (255, 218, 158) if class_index == true_label else (100,␣
↪100, 100)
                cv2.rectangle(img, (bar_x + rect_start, bar_y - 5), (bar_x +␣
↪rect_start + rect_width, bar_y - 20),
                              color, -1)
                text = f'{class_names[class_index]}: {int(p)}%'
                cv2.putText(img, text=text, org=(bar_x, bar_y),␣
↪fontScale=font_scale, fontFace=font_face,
                            thickness=thickness, color=color, lineType=cv2.
↪LINE_AA)
                bar_y += 60
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        vw.write(img) if true_label == predicted_label else None

    vw.release()

try:
    # Select random 60 indices
    random_indices = np.random.choice(len(x_test), size=60, replace=False)
    selected_x_test = x_test[random_indices]
    selected_y_test = y_test[random_indices]
```

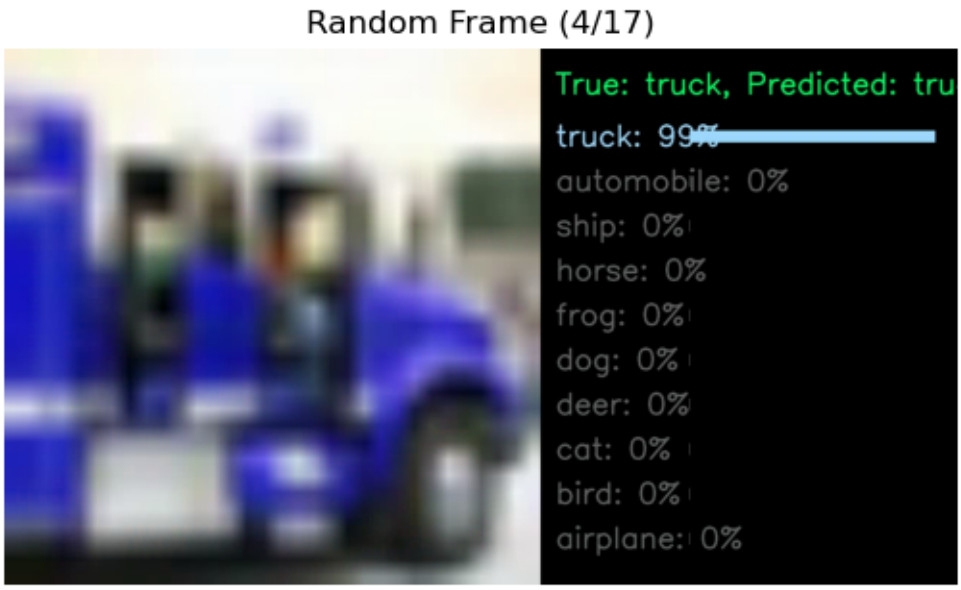
```
    # Use the function with your data and model
    render_and_save_examples(model, selected_x_test, selected_y_test,␣
 ↪cifar10_class_names,  32)
except Exception as e:
    print(e)
```

[14]:
```
# @title
from IPython.display import HTML
from base64 import b64encode

video_path = 'output_video.mp4'

def show_video(video_path, video_width = 600):
    video_file = open(video_path, "r+b").read()
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
    return HTML(f"""<video width={video_width} controls><source␣
 ↪src="{video_url}"></video>""")

show_video(video_path)
```

[14]:
```
<IPython.core.display.HTML object>
```

[15]:
```
try:
    # Open the video file
    video_path = './output_video.mp4'
    cap = cv2.VideoCapture(video_path)

    # Get the total number of frames in the video
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Select a random frame index
    random_frame_index = np.random.randint(0, total_frames)

    # Set the frame index
    cap.set(cv2.CAP_PROP_POS_FRAMES, random_frame_index)

    # Read the selected frame
    ret, frame = cap.read()

    # Check if the frame was read successfully
    if not ret:
        print("Error reading frame from the video.")
        cap.release()
    else:
        # Convert BGR to RGB for matplotlib display
        frame_rgb = frame
```

```
        # Plot the frame
        plt.imshow(frame_rgb)
        plt.title(f"Random Frame ({random_frame_index}/{total_frames})")
        plt.axis('off')
        plt.show()

    # Release the video capture object
    cap.release()
except Exception as e:
    print(e)
```



[16]:
```
try:
    model.save('cifar10-model.h5')
except Exception as e:
    print(e)
```

1.8 Conclusion:

In culmination, the designed CNN classifier has yielded promising results. Achieving an accuracy rate of 88% demonstrates the potency of CNNs in discerning patterns within images. However, this is just the beginning of the exploration. Leveraging pre-trained models and employing more complex architectures offer avenues for refinement. The augmentation of data and the optimization of parameters like batch size and learning rate can further enhance performance. With ample computational resources and the spirit of experimentation, the potential to unravel greater accuracy and capabilities within the classifier remains a tantalizing prospect.