assignment-2-1-code-template-4-2

November 2, 2023

Assignment 2- CNN

[1]:
```
!pip install tensorflow==2.12.0
```

```
Collecting tensorflow==2.12.0
  Downloading
tensorflow-2.12.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl
(585.9 MB)
                           585.9/585.9

MB 2.1 MB/s eta 0:00:00
Requirement already satisfied: absl-py>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (23.5.26)
Collecting gast<=0.4.0,>=0.2.1 (from tensorflow==2.12.0)
  Downloading gast-0.4.0-py3-none-any.whl (9.8 kB)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.59.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.12.0) (3.9.0)
Requirement already satisfied: jax>=0.3.15 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.12.0) (0.4.16)
Collecting keras<2.13,>=2.12.0 (from tensorflow==2.12.0)
  Downloading keras-2.12.0-py2.py3-none-any.whl (1.7 MB)
                           1.7/1.7 MB
47.5 MB/s eta 0:00:00
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (16.0.6)
Requirement already satisfied: numpy<1.24,>=1.22 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.23.5)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.12.0) (23.2)
```

```
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3
in /usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.12.0) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-
packages (from tensorflow==2.12.0) (1.16.0)
Collecting tensorboard<2.13,>=2.12 (from tensorflow==2.12.0)
  Downloading tensorboard-2.12.3-py3-none-any.whl (5.6 MB)
                           5.6/5.6 MB
66.1 MB/s eta 0:00:00
Collecting tensorflow-estimator<2.13,>=2.12.0 (from tensorflow==2.12.0)
  Downloading tensorflow_estimator-2.12.0-py2.py3-none-any.whl (440 kB)
                           440.7/440.7

kB 36.8 MB/s eta 0:00:00
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (2.3.0)
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (4.5.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.10/dist-packages (from tensorflow==2.12.0) (0.34.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.10/dist-packages (from
astunparse>=1.6.0->tensorflow==2.12.0) (0.41.2)
Requirement already satisfied: ml-dtypes>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from jax>=0.3.15->tensorflow==2.12.0)
(0.2.0)
Requirement already satisfied: scipy>=1.7 in /usr/local/lib/python3.10/dist-
packages (from jax>=0.3.15->tensorflow==2.12.0) (1.11.3)
Requirement already satisfied: google-auth<3,>=1.6.3 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.17.3)
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (1.0.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.5)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
```

```
/usr/local/lib/python3.10/dist-packages (from
tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.0.1)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in
/usr/local/lib/python3.10/dist-packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-
packages (from google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in
/usr/local/lib/python3.10/dist-packages (from google-auth-
oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
packages (from requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0)
(3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from
requests<3,>=2.21.0->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2023.7.22)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.10/dist-packages (from
werkzeug>=1.0.1->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in
/usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-
auth<3,>=1.6.3->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-
auth-oauthlib<1.1,>=0.5->tensorboard<2.13,>=2.12->tensorflow==2.12.0) (3.2.2)
Installing collected packages: tensorflow-estimator, keras, gast, tensorboard,
tensorflow
  Attempting uninstall: tensorflow-estimator
    Found existing installation: tensorflow-estimator 2.14.0
    Uninstalling tensorflow-estimator-2.14.0:
      Successfully uninstalled tensorflow-estimator-2.14.0
  Attempting uninstall: keras
    Found existing installation: keras 2.14.0
    Uninstalling keras-2.14.0:
      Successfully uninstalled keras-2.14.0
  Attempting uninstall: gast
    Found existing installation: gast 0.5.4
    Uninstalling gast-0.5.4:
      Successfully uninstalled gast-0.5.4
  Attempting uninstall: tensorboard
    Found existing installation: tensorboard 2.14.1
    Uninstalling tensorboard-2.14.1:
      Successfully uninstalled tensorboard-2.14.1
  Attempting uninstall: tensorflow
    Found existing installation: tensorflow 2.14.0
    Uninstalling tensorflow-2.14.0:
      Successfully uninstalled tensorflow-2.14.0
Successfully installed gast-0.4.0 keras-2.12.0 tensorboard-2.12.3
tensorflow-2.12.0 tensorflow-estimator-2.12.0
```

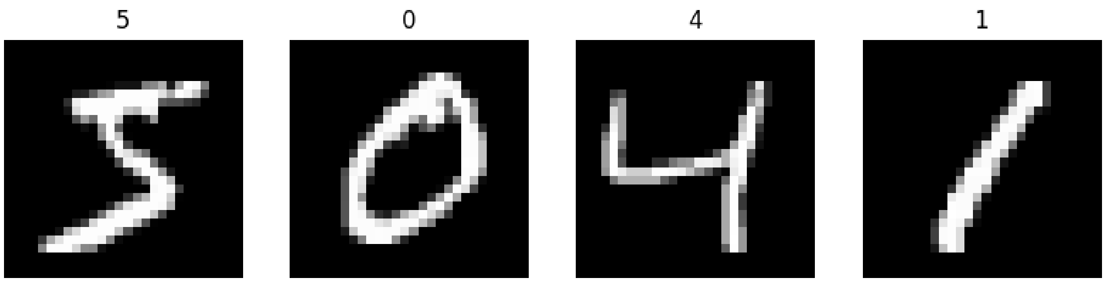
[2]:
```
import keras
import tensorflow as tf
from tensorflow.keras.layers import MaxPooling2D
from keras.datasets import mnist, cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D
from keras.layers import BatchNormalization
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.layers import Dense
from keras import optimizers
from tensorflow.keras.optimizers import SGD
from keras.preprocessing.image import ImageDataGenerator
from keras import backend as K
```

[3]:
```
# Create train and test dataset
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

1.a. start with creating a visualization of your input data

[4]:
```
#Plot the first 4 images
plt.figure(figsize=(10, 10))

for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(X_train[i], cmap='gray')
    plt.title(y_train[i])
    plt.axis('off')

plt.show()
```



[5]:
```
#preprocessing
# Kears allows us to add the number of channels either to the beggining of␣
↪shape or the end of it
img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

[6]:
```
# Normalize inputs from 0-255 to 0-1
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# Encode outputs
num_classes = 10
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

1.b. Create a CNN model with 4 convolution layers in which two of them have 32 and two of them have 64 filters. The fully connected layer has one hidden layer (512 nodes). Draw the Learning curve. What is your understanding from learning curve? Batch size=128 and epochs=20

[7]:
```
# Create model
model = Sequential()
# First 2 convolutional layers with 32 filters
model.add(Conv2D(32, kernel_size=(3,3), activation='relu',␣
↪input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))

# Next 2 convolutional layers with 64 filters
```

```
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))

# Fully connected layer with 512 nodes
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax')) # Assuming you're doing a␣
↪classification task with 'num_classes' classes
```

[8]:
```
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])

# Train the model
batch_size = 128
epochs = 20
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪epochs=epochs, batch_size=batch_size)
```

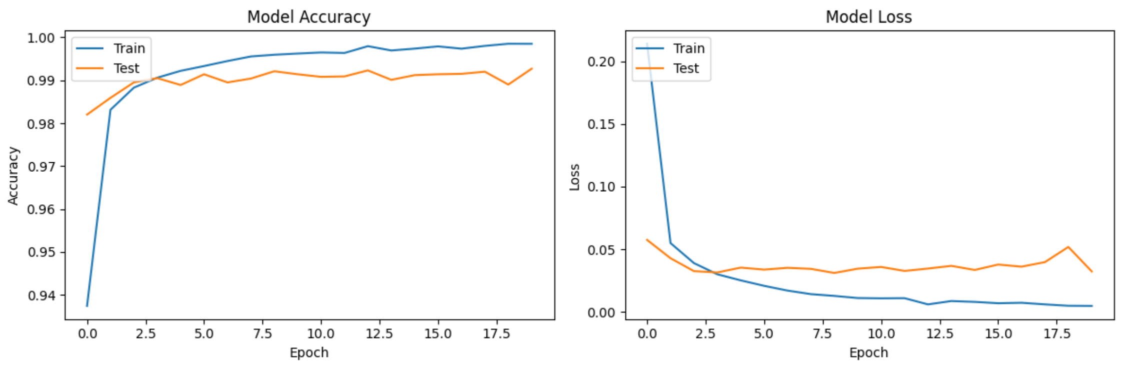
```
Epoch 1/20
469/469 [==============================] - 15s 7ms/step - loss: 0.2139 -
accuracy: 0.9374 - val_loss: 0.0574 - val_accuracy: 0.9820
Epoch 2/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0549 -
accuracy: 0.9831 - val_loss: 0.0428 - val_accuracy: 0.9859
Epoch 3/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0390 -
accuracy: 0.9883 - val_loss: 0.0326 - val_accuracy: 0.9895
Epoch 4/20
469/469 [==============================] - 3s 7ms/step - loss: 0.0300 -
accuracy: 0.9906 - val_loss: 0.0315 - val_accuracy: 0.9905
Epoch 5/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0252 -
accuracy: 0.9922 - val_loss: 0.0354 - val_accuracy: 0.9889
Epoch 6/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0208 -
accuracy: 0.9933 - val_loss: 0.0338 - val_accuracy: 0.9914
Epoch 7/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0170 -
accuracy: 0.9945 - val_loss: 0.0352 - val_accuracy: 0.9895
Epoch 8/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0142 -
accuracy: 0.9955 - val_loss: 0.0344 - val_accuracy: 0.9904
Epoch 9/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0128 -
accuracy: 0.9959 - val_loss: 0.0311 - val_accuracy: 0.9921
```

```
Epoch 10/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0111 -
accuracy: 0.9962 - val_loss: 0.0345 - val_accuracy: 0.9914
Epoch 11/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0109 -
accuracy: 0.9965 - val_loss: 0.0359 - val_accuracy: 0.9908
Epoch 12/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0110 -
accuracy: 0.9964 - val_loss: 0.0327 - val_accuracy: 0.9909
Epoch 13/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0060 -
accuracy: 0.9980 - val_loss: 0.0346 - val_accuracy: 0.9923
Epoch 14/20
469/469 [==============================] - 4s 9ms/step - loss: 0.0087 -
accuracy: 0.9969 - val_loss: 0.0367 - val_accuracy: 0.9901
Epoch 15/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0080 -
accuracy: 0.9974 - val_loss: 0.0335 - val_accuracy: 0.9912
Epoch 16/20
469/469 [==============================] - 3s 5ms/step - loss: 0.0069 -
accuracy: 0.9979 - val_loss: 0.0378 - val_accuracy: 0.9914
Epoch 17/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0073 -
accuracy: 0.9974 - val_loss: 0.0361 - val_accuracy: 0.9915
Epoch 18/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0060 -
accuracy: 0.9980 - val_loss: 0.0397 - val_accuracy: 0.9920
Epoch 19/20
469/469 [==============================] - 3s 7ms/step - loss: 0.0049 -
accuracy: 0.9985 - val_loss: 0.0518 - val_accuracy: 0.9890
Epoch 20/20
469/469 [==============================] - 3s 6ms/step - loss: 0.0048 -
accuracy: 0.9985 - val_loss: 0.0323 - val_accuracy: 0.9927
```

[9]:
```python
# Measure test accuracy
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 99.27%
```

[11]:
```python
# Draw Learning curve
import matplotlib.pyplot as plt

def learning_curve(hist):
    plt.figure(figsize=(12,4))
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['accuracy'])
    plt.plot(hist.history['val_accuracy'])
    plt.title('Model Accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')

    plt.subplot(1, 2, 2)
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('Model Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')

    plt.tight_layout()
    plt.show()

learning_curve(hist)
```



[ ]:
```
## What is your understanding from the learning curve?
# They look fine. There is a little bit of overfitting but nothing to be␣
↪worried about
```

0.1 What is your understanding from the learning curve?

1 They look fine. There is a little bit of overfitting but nothing to be worried about

From the learning curves:

```
Model Accuracy:
    The training accuracy increases steadily and approaches 1 (or 100%). This indicates that th
    The test accuracy starts off well and then plateaus. While it's close to the training accu
```

```
Model Loss:
    The training loss decreases steadily towards zero, indicating the model is fitting well to
    The test loss starts off with a sharp decline and then plateaus. Similar to the accuracy, 
```

Interpretation: The model seems to be overfitting slightly, as indicated by the gap between training and test curves in both accuracy and loss plots. Overfitting occurs when a model performs well on the training data but less so on the test data, due to memorizing the training data rather than generalizing from it.

Recommendations:

```
Regularization: Introduce techniques like dropout or L2 regularization to penalize complex mode
Data Augmentation: If applicable, augment the training dataset with techniques like rotations,
Early Stopping: Monitor the validation loss and stop training when it begins to increase or do
Review Dataset: Ensure that the training and test datasets are representative of the problem a
Model Complexity: Consider simplifying the model architecture if it's too complex for the data
```

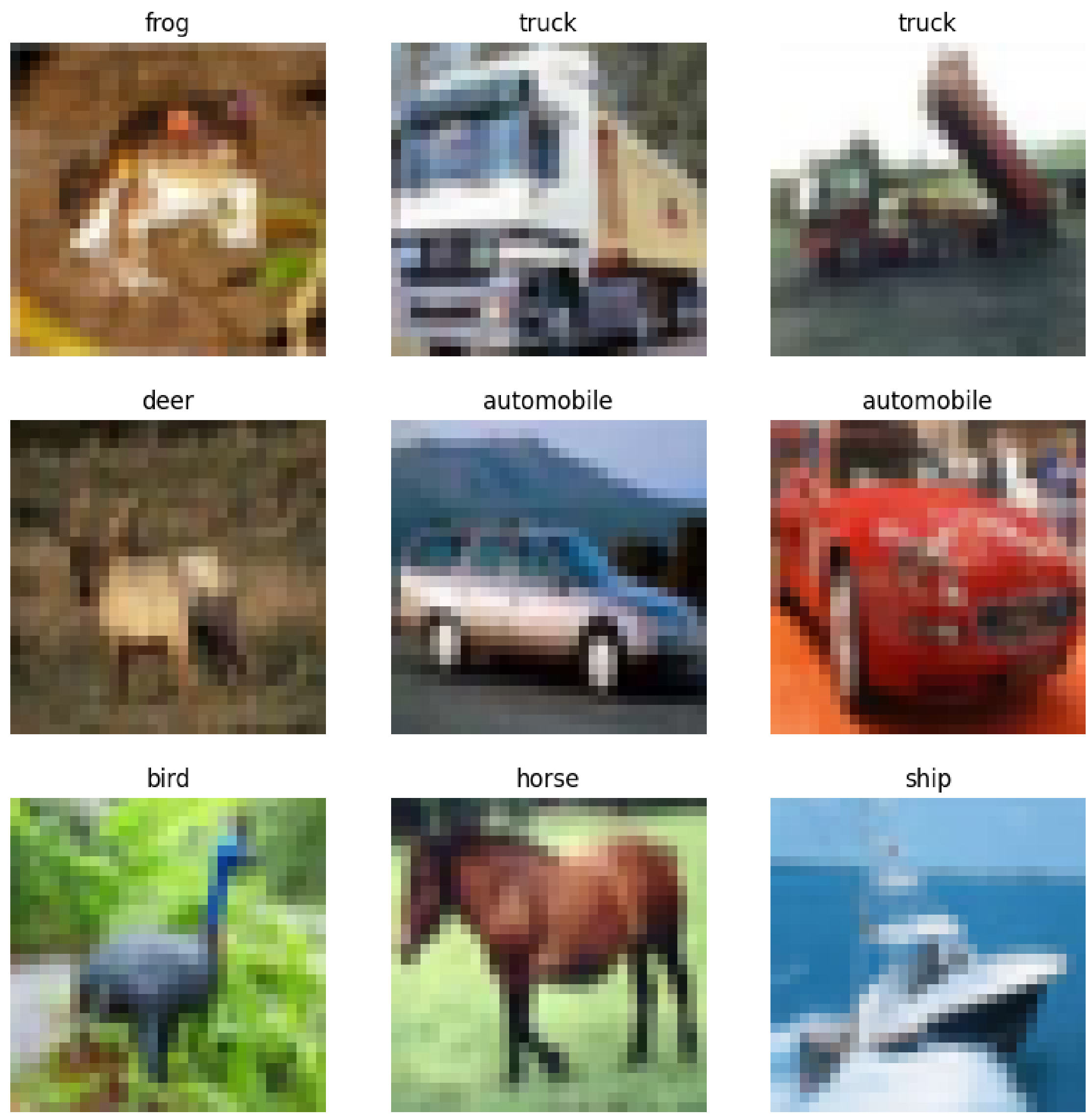
However, the overfitting observed here is not severe. If this model's performance is satisfactory for the given application, it might be acceptable to use it as is.

Part 2- CIFAR10

[12]:
```
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.cifar10.load_data() ␣
↪# Load CIFAR 10 data here
labels=␣
↪["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
print(X_train.shape)
print(X_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

[13]:
```
# 2.a. Let's look into the dataset by visualizing some data opints

# Plot the first 9 images
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.imshow(X_train[i])
    plt.title(labels[y_train[i][0]])
    plt.axis('off')
plt.show()
```



```
[ ]:
```

2.b. Apply the pre-processing algorithms that we discussed last week. The augmented images are supposed to be seared by 20%, zoomed by 20% and horizontally flipped. Now, design a CNN model with 4 convolution layers in which two of them have 32 and two of them have 64 filters. The fully connected layer has two hidden layers (512 and 256 nodes respectively). Draw the Learning curve. What is your understanding from learning curve?

```
[15]: # 2.b
      # Encoding output
      from tensorflow.keras.utils import to_categorical
      y_train_encoded = to_categorical(y_train)
```

```python
y_test_encoded = to_categorical(y_test)

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Create data generator with specified augmentations
datagen = ImageDataGenerator(
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)

# Prepare iterator
it_train = datagen.flow(X_train, y_train_encoded, batch_size=128)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Create model
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', padding='same',␣
 ↪input_shape=(32, 32, 3)))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))  # 10 classes for CIFAR-10

from tensorflow.keras.optimizers import SGD

# Compile model with SGD optimizer

optimizer = SGD(learning_rate=0.005, momentum=0.9)

model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Start training
hist = model.fit(it_train, epochs=20, validation_data=(X_test, y_test_encoded))
```

```
Epoch 1/20
391/391 [==============================] - 39s 96ms/step - loss: 9.9189 -
```

```
accuracy: 0.0984 - val_loss: 2.3027 - val_accuracy: 0.1000
Epoch 2/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0985 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 3/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0984 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 4/20
391/391 [==============================] - 29s 74ms/step - loss: 2.3027 -
accuracy: 0.0989 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 5/20
391/391 [==============================] - 28s 73ms/step - loss: 2.3027 -
accuracy: 0.0980 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 6/20
391/391 [==============================] - 31s 79ms/step - loss: 2.3027 -
accuracy: 0.0967 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 7/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.1001 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 8/20
391/391 [==============================] - 27s 69ms/step - loss: 2.3027 -
accuracy: 0.0970 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 9/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0987 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 10/20
391/391 [==============================] - 28s 73ms/step - loss: 2.3027 -
accuracy: 0.0969 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 11/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0968 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 12/20
391/391 [==============================] - 29s 73ms/step - loss: 2.3027 -
accuracy: 0.0988 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 13/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0961 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 14/20
391/391 [==============================] - 27s 69ms/step - loss: 2.3027 -
accuracy: 0.0992 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 15/20
391/391 [==============================] - 30s 76ms/step - loss: 2.3027 -
accuracy: 0.0956 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 16/20
391/391 [==============================] - 28s 70ms/step - loss: 2.3027 -
accuracy: 0.1009 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 17/20
391/391 [==============================] - 28s 71ms/step - loss: 2.3027 -
```

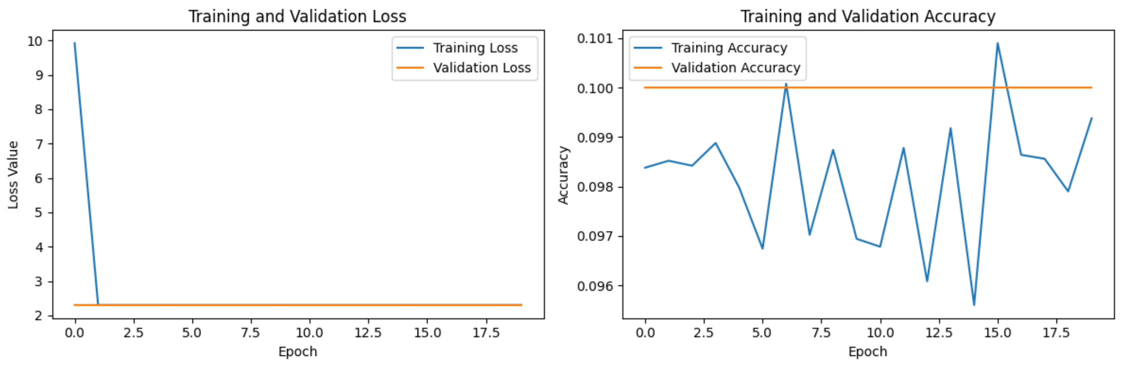
```
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 18/20
391/391 [==============================] - 28s 71ms/step - loss: 2.3027 -
accuracy: 0.0986 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 19/20
391/391 [==============================] - 28s 72ms/step - loss: 2.3027 -
accuracy: 0.0979 - val_loss: 2.3026 - val_accuracy: 0.1000
Epoch 20/20
391/391 [==============================] - 29s 73ms/step - loss: 2.3027 -
accuracy: 0.0994 - val_loss: 2.3026 - val_accuracy: 0.1000
```

[ ]:

```

```

[16]:

```python
# Draw learning curve here
import matplotlib.pyplot as plt

def learning_curve(hist):
    # Plotting the training and validation loss
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(hist.history['loss'], label='Training Loss')
    plt.plot(hist.history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss Value')
    plt.legend()

    # Plotting the training and validation accuracy
    plt.subplot(1, 2, 2)
    plt.plot(hist.history['accuracy'], label='Training Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Use the function to plot the learning curve
learning_curve(hist)
```



[ ]:

```

```

2 What is the issue and possible solution for this learning curve?

Issue with the Learning Curve:

```
The training accuracy starts off very high and remains stable through the epochs.
The validation accuracy, however, starts much lower and seems to remain constant through the ep
The training loss begins at a very low value and remains stable, whereas the validation loss st
```

These observations suggest that the model is overfitting. Overfitting occurs when the model performs very well on the training data but struggles to generalize to unseen or validation data

[ ]:

```

```

#2.c. Solution to resolve overfitting # One solution is adding drop out # Implement your solution here and train model

Solution to Resolve Overfitting:

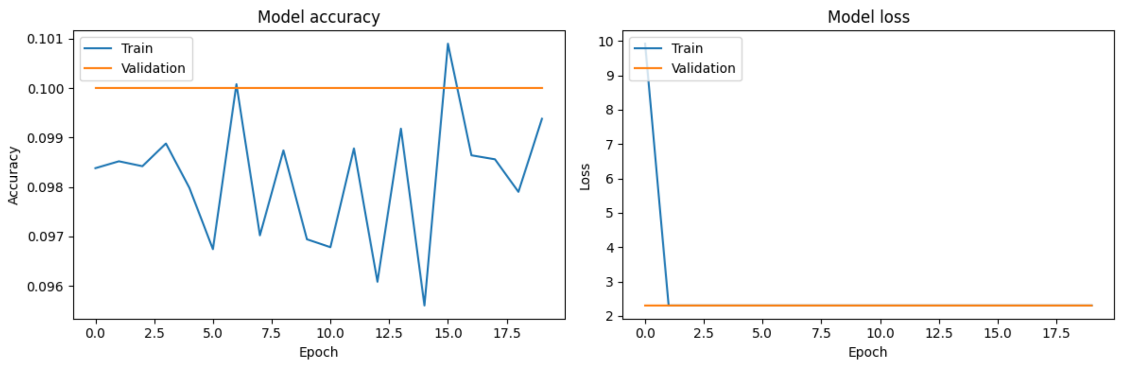
```
One of the most common techniques to mitigate overfitting is to introduce dropout in the neural
Another method is data augmentation, but it seems you've already implemented this based on the
You can also consider adding L1 or L2 regularization to the layers of the network.
```

[18]:

```

```

[19]:

```python
# Draw learning curve

import matplotlib.pyplot as plt

def learning_curve(hist):
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    plt.plot(hist.history['accuracy'])
    plt.plot(hist.history['val_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    # Plot training & validation loss values
    plt.subplot(1, 2, 2)
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'], loc='upper left')

    plt.tight_layout()
    plt.show()

# Call the function
learning_curve(hist)
```



[ ]:

[20]:
```
# 2.d- This part is up to you to choose proper pre-trained model
# I chose VGG16 and RESNet50
# Implementing VGG16


from keras.applications.vgg16 import VGG16
from keras.layers import Flatten, Dense
from keras.models import Model
from keras.optimizers import SGD

from keras.datasets import cifar10
```

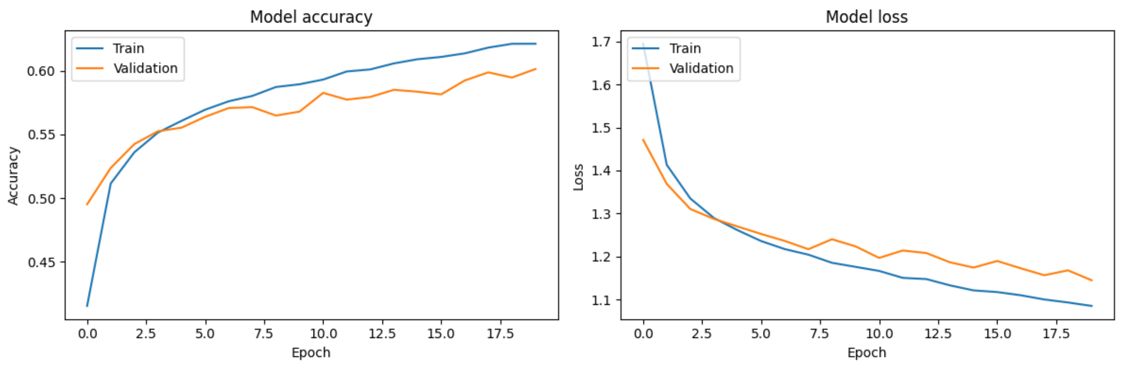
```python
from keras.utils import to_categorical

# 1. Load CIFAR-10 dataset
(X_train, Y_train), (X_test, Y_test) = cifar10.load_data()

# 2. Normalize the data
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# 3. One-hot encode the labels
Y_train = to_categorical(Y_train, 10)
Y_test = to_categorical(Y_test, 10)

# Load the VGG16 model but without the top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32,␣
 ↪3))

# Freeze the layers of the base model (optional)
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top for CIFAR-10 classification
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dense(10, activation='softmax')(x)

# Combine base model with custom layers
vgg_model = Model(inputs=base_model.input, outputs=x)

# Compile the VGG16 model
optimizer = SGD(learning_rate=0.005, momentum=0.9)
vgg_model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Train the VGG16 model on the CIFAR-10 data (assuming X_train, Y_train are the␣
 ↪training data)
hist = vgg_model.fit(X_train, Y_train, validation_split=0.2, epochs=20,␣
 ↪batch_size=128)
```

```
Epoch 1/20
313/313 [==============================] - 10s 27ms/step - loss: 1.6951 -
accuracy: 0.4155 - val_loss: 1.4714 - val_accuracy: 0.4952
Epoch 2/20
313/313 [==============================] - 6s 19ms/step - loss: 1.4132 -
accuracy: 0.5115 - val_loss: 1.3689 - val_accuracy: 0.5237
Epoch 3/20
```

```
313/313 [==============================] - 6s 19ms/step - loss: 1.3349 -
accuracy: 0.5360 - val_loss: 1.3104 - val_accuracy: 0.5424
Epoch 4/20
313/313 [==============================] - 6s 19ms/step - loss: 1.2894 -
accuracy: 0.5515 - val_loss: 1.2873 - val_accuracy: 0.5525
Epoch 5/20
313/313 [==============================] - 6s 19ms/step - loss: 1.2611 -
accuracy: 0.5606 - val_loss: 1.2698 - val_accuracy: 0.5553
Epoch 6/20
313/313 [==============================] - 6s 19ms/step - loss: 1.2358 -
accuracy: 0.5694 - val_loss: 1.2523 - val_accuracy: 0.5638
Epoch 7/20
313/313 [==============================] - 6s 20ms/step - loss: 1.2171 -
accuracy: 0.5760 - val_loss: 1.2361 - val_accuracy: 0.5707
Epoch 8/20
313/313 [==============================] - 6s 20ms/step - loss: 1.2042 -
accuracy: 0.5802 - val_loss: 1.2169 - val_accuracy: 0.5714
Epoch 9/20
313/313 [==============================] - 6s 20ms/step - loss: 1.1853 -
accuracy: 0.5872 - val_loss: 1.2401 - val_accuracy: 0.5648
Epoch 10/20
313/313 [==============================] - 6s 20ms/step - loss: 1.1758 -
accuracy: 0.5893 - val_loss: 1.2235 - val_accuracy: 0.5679
Epoch 11/20
313/313 [==============================] - 6s 19ms/step - loss: 1.1663 -
accuracy: 0.5930 - val_loss: 1.1966 - val_accuracy: 0.5826
Epoch 12/20
313/313 [==============================] - 6s 20ms/step - loss: 1.1502 -
accuracy: 0.5994 - val_loss: 1.2139 - val_accuracy: 0.5773
Epoch 13/20
313/313 [==============================] - 6s 19ms/step - loss: 1.1474 -
accuracy: 0.6010 - val_loss: 1.2079 - val_accuracy: 0.5794
Epoch 14/20
313/313 [==============================] - 6s 20ms/step - loss: 1.1327 -
accuracy: 0.6057 - val_loss: 1.1864 - val_accuracy: 0.5850
Epoch 15/20
313/313 [==============================] - 7s 21ms/step - loss: 1.1210 -
accuracy: 0.6090 - val_loss: 1.1741 - val_accuracy: 0.5835
Epoch 16/20
313/313 [==============================] - 6s 19ms/step - loss: 1.1171 -
accuracy: 0.6108 - val_loss: 1.1895 - val_accuracy: 0.5814
Epoch 17/20
313/313 [==============================] - 6s 20ms/step - loss: 1.1097 -
accuracy: 0.6136 - val_loss: 1.1724 - val_accuracy: 0.5923
Epoch 18/20
313/313 [==============================] - 6s 20ms/step - loss: 1.0999 -
accuracy: 0.6181 - val_loss: 1.1563 - val_accuracy: 0.5987
Epoch 19/20
```

```
313/313 [==============================] - 6s 20ms/step - loss: 1.0929 -
accuracy: 0.6211 - val_loss: 1.1676 - val_accuracy: 0.5946
Epoch 20/20
313/313 [==============================] - 6s 19ms/step - loss: 1.0849 -
accuracy: 0.6211 - val_loss: 1.1447 - val_accuracy: 0.6013
```

[21]:
```
learning_curve(hist)
```



[22]:
```
vgg_model.evaluate(X_test, Y_test, batch_size=256, verbose=1)
```

```
40/40 [==============================] - 2s 26ms/step - loss: 1.1581 - accuracy:
0.5970
```

[22]:
```
[1.1580777168273926, 0.597000002861023]
```

[ ]:
```

```

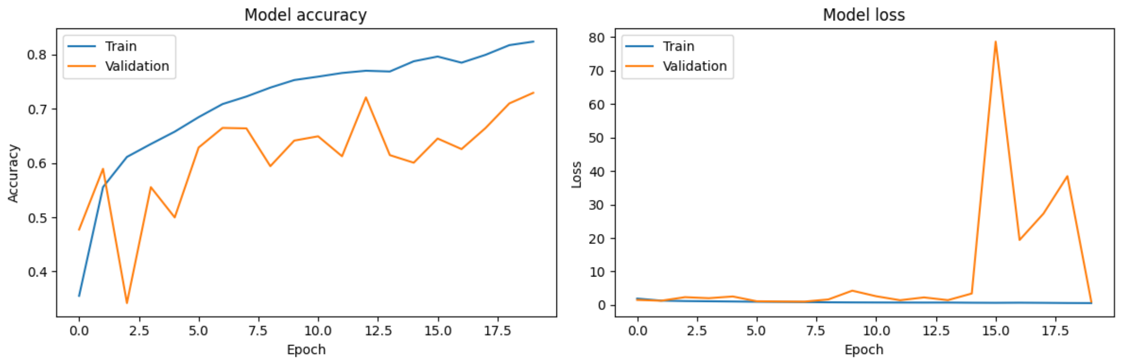
[24]:
```
# This ios my second pre-trained model
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import RMSprop

# Load the ResNet50 model without its top layers
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32,␣
 ↪32, 3))


# Add custom layers on top for CIFAR-10 classification
x = base_model.output
x = GlobalAveragePooling2D()(x) # This is more common for ResNet architectures␣
 ↪than Flatten()
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x) # You can adjust the dropout rate
```

```
x = Dense(10, activation='softmax')(x) # Assuming you have 10 classes in your␣
↪dataset

# Combine base model with custom layers
model = Model(inputs=base_model.input, outputs=x)

# Compile the model
optimizer = RMSprop(learning_rate=0.001) # You can adjust the learning rate
model.compile(optimizer=optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.
↪2, random_state=42)

# Assuming X_train, Y_train are your training data and X_val, Y_val are your␣
↪validation data

hist = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=20,␣
↪batch_size=32)
```

```
Epoch 1/20
1250/1250 [==============================] - 86s 43ms/step - loss: 1.9394 -
accuracy: 0.3550 - val_loss: 1.5365 - val_accuracy: 0.4773
Epoch 2/20
1250/1250 [==============================] - 52s 42ms/step - loss: 1.3250 -
accuracy: 0.5556 - val_loss: 1.2912 - val_accuracy: 0.5894
Epoch 3/20
1250/1250 [==============================] - 49s 39ms/step - loss: 1.1980 -
accuracy: 0.6112 - val_loss: 2.3548 - val_accuracy: 0.3416
Epoch 4/20
1250/1250 [==============================] - 50s 40ms/step - loss: 1.1278 -
accuracy: 0.6350 - val_loss: 2.0556 - val_accuracy: 0.5555
Epoch 5/20
1250/1250 [==============================] - 54s 43ms/step - loss: 1.0656 -
accuracy: 0.6579 - val_loss: 2.5652 - val_accuracy: 0.4996
Epoch 6/20
1250/1250 [==============================] - 52s 42ms/step - loss: 0.9913 -
accuracy: 0.6847 - val_loss: 1.1350 - val_accuracy: 0.6286
Epoch 7/20
1250/1250 [==============================] - 51s 41ms/step - loss: 0.9417 -
accuracy: 0.7088 - val_loss: 1.0636 - val_accuracy: 0.6648
Epoch 8/20
1250/1250 [==============================] - 54s 44ms/step - loss: 0.8871 -
accuracy: 0.7227 - val_loss: 1.0268 - val_accuracy: 0.6639
Epoch 9/20
```

```
1250/1250 [==============================] - 52s 42ms/step - loss: 0.8263 -
accuracy: 0.7392 - val_loss: 1.6777 - val_accuracy: 0.5942
Epoch 10/20
1250/1250 [==============================] - 53s 42ms/step - loss: 0.7925 -
accuracy: 0.7531 - val_loss: 4.2912 - val_accuracy: 0.6414
Epoch 11/20
1250/1250 [==============================] - 53s 43ms/step - loss: 0.7718 -
accuracy: 0.7594 - val_loss: 2.6346 - val_accuracy: 0.6493
Epoch 12/20
1250/1250 [==============================] - 52s 41ms/step - loss: 0.7573 -
accuracy: 0.7661 - val_loss: 1.4454 - val_accuracy: 0.6125
Epoch 13/20
1250/1250 [==============================] - 50s 40ms/step - loss: 0.7469 -
accuracy: 0.7703 - val_loss: 2.2829 - val_accuracy: 0.7211
Epoch 14/20
1250/1250 [==============================] - 53s 43ms/step - loss: 0.7529 -
accuracy: 0.7689 - val_loss: 1.4686 - val_accuracy: 0.6146
Epoch 15/20
1250/1250 [==============================] - 54s 43ms/step - loss: 0.7021 -
accuracy: 0.7877 - val_loss: 3.4459 - val_accuracy: 0.6006
Epoch 16/20
1250/1250 [==============================] - 53s 43ms/step - loss: 0.6749 -
accuracy: 0.7965 - val_loss: 78.5621 - val_accuracy: 0.6452
Epoch 17/20
1250/1250 [==============================] - 54s 43ms/step - loss: 0.7107 -
accuracy: 0.7851 - val_loss: 19.4073 - val_accuracy: 0.6257
Epoch 18/20
1250/1250 [==============================] - 52s 42ms/step - loss: 0.6685 -
accuracy: 0.7996 - val_loss: 27.2786 - val_accuracy: 0.6644
Epoch 19/20
1250/1250 [==============================] - 53s 42ms/step - loss: 0.6152 -
accuracy: 0.8175 - val_loss: 38.4468 - val_accuracy: 0.7100
Epoch 20/20
1250/1250 [==============================] - 53s 43ms/step - loss: 0.5919 -
accuracy: 0.8240 - val_loss: 1.0312 - val_accuracy: 0.7295
```

[25]:
```
learning_curve(hist)
```


Model accuracy
Train
Validation
Accuracy
Epoch
Model loss
Train
Validation
Loss
Epoch

```
[ ]:
```