Assignment2_template

November 4, 2024

Assignment 2- CNN

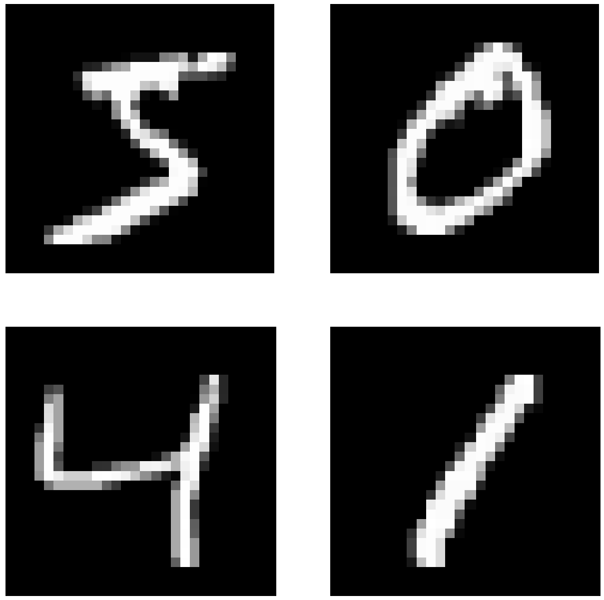
[1]:
```
import keras
import tensorflow as tf
from tensorflow.keras.layers import MaxPooling2D
from keras.datasets import mnist, cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Activation
from keras.layers import Conv2D
from keras.layers import BatchNormalization
import matplotlib.pyplot as plt
from keras.utils import to_categorical
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) = 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]:
```
#1.a. Create the visualization here
# Let's look into the dataset by visualizing some data opints
plt.figure(figsize=(4, 4))
for i in range(4):
    plt.subplot(2, 2, i + 1)
    plt.imshow(X_train[i], cmap='gray')
    plt.axis('off')
plt.show()
```



[5]:
```python
#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]:
```python
# You need to apply some preprocessing on X and y

# Normalize inputs from 0-255 to 0-1
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Encode outputs (y) as categorical data
# Since we're dealing with digits (0-9), we have 10 classes
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
```

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

[8]:
```
#1.b.
# Create model here
# 1.b. Create the CNN model

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout

# Define the CNN model with fewer pooling layers to prevent excessive␣
↪downsampling
model = Sequential()

# First convolutional layer with 32 filters
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',␣
↪input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolutional layer with 32 filters
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu'))

# Third conv layer with 64 filters
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Fourth conv layer with 64 filters
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(Dropout(0.5))

# Output layer with 10 nodes (one for each class)
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_3 (MaxPoolin  (None, 13, 13, 32)        0
 g2D)

 conv2d_5 (Conv2D)           (None, 11, 11, 32)        9248
```

```
 conv2d_6 (Conv2D)           (None, 9, 9, 64)          18496

 max_pooling2d_4 (MaxPoolin  (None, 4, 4, 64)          0
 g2D)

 conv2d_7 (Conv2D)           (None, 2, 2, 64)          36928

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 512)               131584

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 201706 (787.91 KB)
Trainable params: 201706 (787.91 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[9]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam', ␣
↪metrics=['accuracy'])

# Start training the model
hist = model.fit(X_train, y_train, validation_data=(X_test, y_test),␣
↪batch_size=128, epochs=20, verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 7s 14ms/step - loss: 0.2559 -
accuracy: 0.9200 - val_loss: 0.0654 - val_accuracy: 0.9797
Epoch 2/20
469/469 [==============================] - 7s 15ms/step - loss: 0.0664 -
accuracy: 0.9800 - val_loss: 0.0391 - val_accuracy: 0.9885
Epoch 3/20
469/469 [==============================] - 7s 16ms/step - loss: 0.0470 -
accuracy: 0.9854 - val_loss: 0.0363 - val_accuracy: 0.9890
Epoch 4/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0357 -
accuracy: 0.9887 - val_loss: 0.0308 - val_accuracy: 0.9902
Epoch 5/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0297 -
accuracy: 0.9908 - val_loss: 0.0244 - val_accuracy: 0.9923
Epoch 6/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0244 -
accuracy: 0.9923 - val_loss: 0.0251 - val_accuracy: 0.9918
Epoch 7/20
```

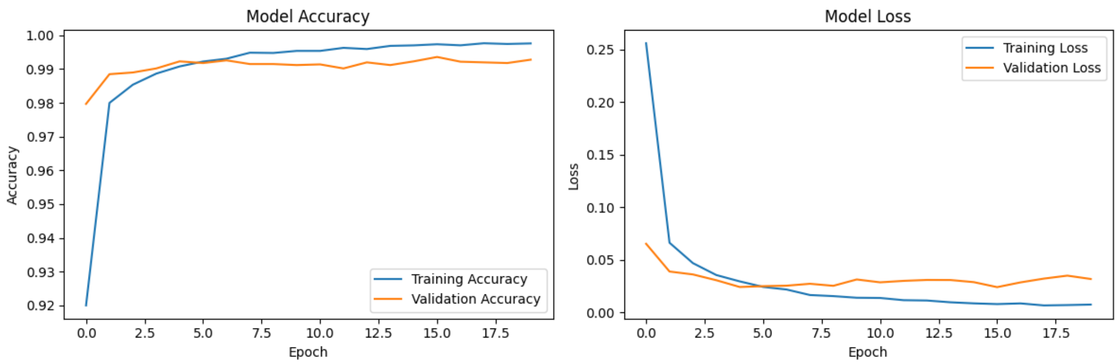
```
469/469 [==============================] - 8s 17ms/step - loss: 0.0220 -
accuracy: 0.9931 - val_loss: 0.0256 - val_accuracy: 0.9926
Epoch 8/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0167 -
accuracy: 0.9949 - val_loss: 0.0274 - val_accuracy: 0.9915
Epoch 9/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0157 -
accuracy: 0.9948 - val_loss: 0.0255 - val_accuracy: 0.9915
Epoch 10/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0141 -
accuracy: 0.9954 - val_loss: 0.0315 - val_accuracy: 0.9912
Epoch 11/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0139 -
accuracy: 0.9954 - val_loss: 0.0288 - val_accuracy: 0.9914
Epoch 12/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0118 -
accuracy: 0.9963 - val_loss: 0.0302 - val_accuracy: 0.9902
Epoch 13/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0115 -
accuracy: 0.9959 - val_loss: 0.0311 - val_accuracy: 0.9920
Epoch 14/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0098 -
accuracy: 0.9969 - val_loss: 0.0310 - val_accuracy: 0.9912
Epoch 15/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0088 -
accuracy: 0.9970 - val_loss: 0.0290 - val_accuracy: 0.9923
Epoch 16/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0081 -
accuracy: 0.9974 - val_loss: 0.0242 - val_accuracy: 0.9936
Epoch 17/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0088 -
accuracy: 0.9970 - val_loss: 0.0287 - val_accuracy: 0.9922
Epoch 18/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0068 -
accuracy: 0.9977 - val_loss: 0.0323 - val_accuracy: 0.9920
Epoch 19/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0072 -
accuracy: 0.9974 - val_loss: 0.0351 - val_accuracy: 0.9918
Epoch 20/20
469/469 [==============================] - 8s 17ms/step - loss: 0.0077 -
accuracy: 0.9976 - val_loss: 0.0320 - val_accuracy: 0.9928
```

[ ]:
```python
# Measure test accuracy
scores = #Measure test accuracy
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0337 -
accuracy: 0.9921
```

```
Accuracy: 99.21%
```

[10]:
```python
# Draw Learning Curve
def learning_curve(hist):
    # Create a function to draw learning curves
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 4))

    # Accuracy plot
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['accuracy'], label='Training Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(hist.history['loss'], label='Training Loss')
    plt.plot(hist.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right')

    plt.tight_layout()
    plt.show()

learning_curve(hist)
```



What is your understanding from the learning curve?

The learning curve shows that both training and validation accuracy improve quickly in the early epochs, then stabilize around 98-99%, which is good model performance. The loss decreases rapidly initially and then flattens, suggesting that the model has converged well. The close alignment between training and validation curves shows pretty minimal overfitting, meaning the model generalizes effectively on unseen data.

Part 2- CIFAR10

[11]:
```
# CIFAR-10 data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

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 [==============================] - 7s 0us/step
(50000, 32, 32, 3)
(10000, 32, 32, 3)
```

[12]:
```
# 2.a. Let's look into the dataset by visualizing some data opints
# Plot the first 9 images in the training set
plt.figure(figsize=(6, 6))
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?

[16]:
```python
# Create model with padding='same' in Conv2D layers to avoid excessive␣
  ↪downsampling
model = Sequential()

# First convolutional layer with 32 filters
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',␣
  ↪input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolutional layer with 32 filters
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))

# Third convolutional layer with 64 filters
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Fourth convolutional layer with 64 filters
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flatten and add fully connected layers
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 32, 32, 32)        896

 max_pooling2d_16 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 conv2d_21 (Conv2D)          (None, 16, 16, 32)        9248

 max_pooling2d_17 (MaxPooli  (None, 8, 8, 32)          0
 ng2D)

 conv2d_22 (Conv2D)          (None, 8, 8, 64)          18496

 max_pooling2d_18 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 conv2d_23 (Conv2D)          (None, 4, 4, 64)          36928

 max_pooling2d_19 (MaxPooli  (None, 2, 2, 64)          0
 ng2D)

 flatten_3 (Flatten)         (None, 256)               0

 dense_8 (Dense)             (None, 512)               131584
```

```
 dropout_5 (Dropout)         (None, 512)              0

 dense_9 (Dense)             (None, 256)              131328

 dropout_6 (Dropout)         (None, 256)              0

 dense_10 (Dense)            (None, 10)               2570

=================================================================
Total params: 331050 (1.26 MB)
Trainable params: 331050 (1.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[17]:
```
from tensorflow.keras.optimizers.legacy import SGD

# Compile model using the legacy SGD optimizer to support decay
model.compile(optimizer=SGD(learning_rate=0.005, decay=1e-6, momentum=0.9),
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

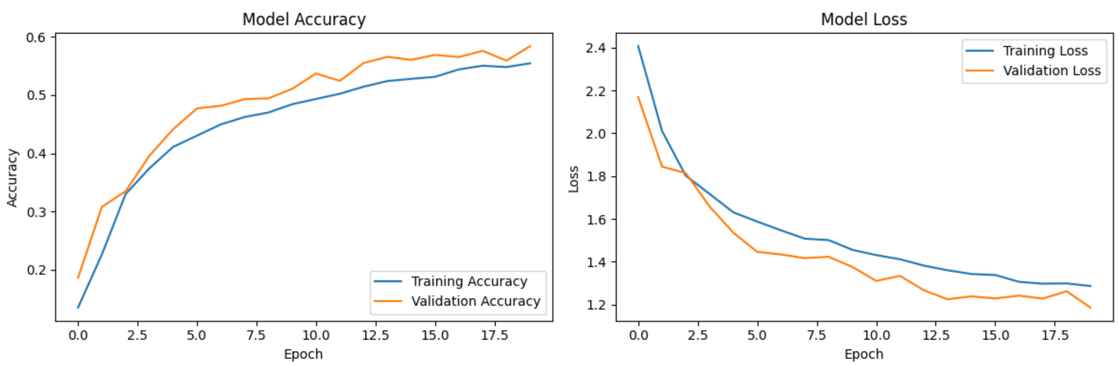
[18]:
```
# Training
hist = model.fit(it_train,
                 steps_per_epoch=len(X_train) // 128,
                epochs=20,
                validation_data=(X_test, y_test),
                verbose=1)
```

```
Epoch 1/20
390/390 [==============================] - 10s 26ms/step - loss: 2.4073 -
accuracy: 0.1353 - val_loss: 2.1689 - val_accuracy: 0.1863
Epoch 2/20
390/390 [==============================] - 10s 24ms/step - loss: 2.0111 -
accuracy: 0.2262 - val_loss: 1.8441 - val_accuracy: 0.3080
Epoch 3/20
390/390 [==============================] - 10s 25ms/step - loss: 1.8017 -
accuracy: 0.3301 - val_loss: 1.8140 - val_accuracy: 0.3350
Epoch 4/20
390/390 [==============================] - 10s 25ms/step - loss: 1.7165 -
accuracy: 0.3743 - val_loss: 1.6575 - val_accuracy: 0.3959
Epoch 5/20
390/390 [==============================] - 10s 25ms/step - loss: 1.6305 -
accuracy: 0.4112 - val_loss: 1.5353 - val_accuracy: 0.4415
Epoch 6/20
390/390 [==============================] - 10s 25ms/step - loss: 1.5875 -
accuracy: 0.4304 - val_loss: 1.4459 - val_accuracy: 0.4771
Epoch 7/20
390/390 [==============================] - 10s 26ms/step - loss: 1.5465 -
```

```
accuracy: 0.4497 - val_loss: 1.4338 - val_accuracy: 0.4815
Epoch 8/20
390/390 [==============================] - 10s 25ms/step - loss: 1.5075 -
accuracy: 0.4623 - val_loss: 1.4167 - val_accuracy: 0.4930
Epoch 9/20
390/390 [==============================] - 10s 26ms/step - loss: 1.5009 -
accuracy: 0.4701 - val_loss: 1.4230 - val_accuracy: 0.4944
Epoch 10/20
390/390 [==============================] - 10s 26ms/step - loss: 1.4553 -
accuracy: 0.4843 - val_loss: 1.3756 - val_accuracy: 0.5107
Epoch 11/20
390/390 [==============================] - 10s 25ms/step - loss: 1.4308 -
accuracy: 0.4933 - val_loss: 1.3100 - val_accuracy: 0.5371
Epoch 12/20
390/390 [==============================] - 10s 25ms/step - loss: 1.4114 -
accuracy: 0.5023 - val_loss: 1.3337 - val_accuracy: 0.5244
Epoch 13/20
390/390 [==============================] - 10s 26ms/step - loss: 1.3817 -
accuracy: 0.5144 - val_loss: 1.2665 - val_accuracy: 0.5549
Epoch 14/20
390/390 [==============================] - 10s 26ms/step - loss: 1.3598 -
accuracy: 0.5241 - val_loss: 1.2244 - val_accuracy: 0.5656
Epoch 15/20
390/390 [==============================] - 10s 26ms/step - loss: 1.3425 -
accuracy: 0.5279 - val_loss: 1.2380 - val_accuracy: 0.5604
Epoch 16/20
390/390 [==============================] - 10s 26ms/step - loss: 1.3377 -
accuracy: 0.5313 - val_loss: 1.2281 - val_accuracy: 0.5689
Epoch 17/20
390/390 [==============================] - 10s 26ms/step - loss: 1.3058 -
accuracy: 0.5440 - val_loss: 1.2414 - val_accuracy: 0.5653
Epoch 18/20
390/390 [==============================] - 10s 26ms/step - loss: 1.2972 -
accuracy: 0.5504 - val_loss: 1.2274 - val_accuracy: 0.5759
Epoch 19/20
390/390 [==============================] - 10s 26ms/step - loss: 1.2986 -
accuracy: 0.5480 - val_loss: 1.2620 - val_accuracy: 0.5591
Epoch 20/20
390/390 [==============================] - 10s 26ms/step - loss: 1.2862 -
accuracy: 0.5546 - val_loss: 1.1855 - val_accuracy: 0.5840
```

[19]:
```
def learning_curve(hist):
    # Plot training & validation accuracy values
    plt.figure(figsize=(12, 4))

    # Accuracy plot
    plt.subplot(1, 2, 1)
```

```
    plt.plot(hist.history['accuracy'], label='Training Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')

    # Loss plot
    plt.subplot(1, 2, 2)
    plt.plot(hist.history['loss'], label='Training Loss')
    plt.plot(hist.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend(loc='upper right')

    plt.tight_layout()
    plt.show()

learning_curve(hist)
```



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

The learning curve shows underfitting, as both training and validation accuracy remain low with minimal separation. The model struggles to capture the complexity of the CIFAR-10 dataset. The model may be too simple for CIFAR-10's complex images, and 20 epochs could honestly not be enough for convergence.

To address this, I could increase the model complexity by adding more layers or filters and train for additional epochs. Also, using a pre-trained model with transfer learning could significantly improve performance on this dataset.

[22]:
```
from keras.applications.vgg16 import VGG16
from keras.layers import Dense, Flatten
from keras.models import Model
from keras.optimizers import SGD

# Load the VGG16 model with pre-trained weights, excluding the top layer
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32,␣
↪3))

# Add custom layers on top of VGG16
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Define the new model
vgg_model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of VGG16 to prevent them from being trained
for layer in base_model.layers:
    layer.trainable = False

# Compile without decay
vgg_model.compile(optimizer=SGD(learning_rate=0.005, momentum=0.9),
                loss='categorical_crossentropy',
                metrics=['accuracy'])

model.summary()
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.SGD`.

Model: "sequential_5"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 32, 32, 32)        896

 max_pooling2d_16 (MaxPooli  (None, 16, 16, 32)        0
 ng2D)

 conv2d_21 (Conv2D)          (None, 16, 16, 32)        9248

 max_pooling2d_17 (MaxPooli  (None, 8, 8, 32)          0
 ng2D)
```

```
 conv2d_22 (Conv2D)          (None, 8, 8, 64)          18496

 max_pooling2d_18 (MaxPooli  (None, 4, 4, 64)          0
 ng2D)

 conv2d_23 (Conv2D)          (None, 4, 4, 64)          36928

 max_pooling2d_19 (MaxPooli  (None, 2, 2, 64)          0
 ng2D)

 flatten_3 (Flatten)         (None, 256)               0

 dense_8 (Dense)             (None, 512)               131584

 dropout_5 (Dropout)         (None, 512)               0

 dense_9 (Dense)             (None, 256)               131328

 dropout_6 (Dropout)         (None, 256)               0

 dense_10 (Dense)            (None, 10)                2570

=================================================================
Total params: 331050 (1.26 MB)
Trainable params: 331050 (1.26 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[23]:

```python
# Start training the VGG16 model
hist = vgg_model.fit(it_train, steps_per_epoch=len(X_train) // 128, epochs=20,
                    validation_data=(X_test, y_test), verbose=1)
```

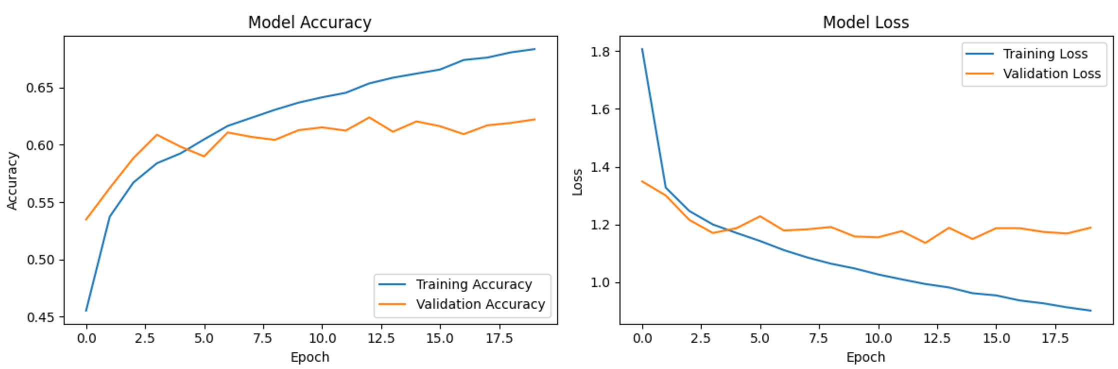
```
Epoch 1/20
390/390 [==============================] - 70s 180ms/step - loss: 1.8067 -
accuracy: 0.4554 - val_loss: 1.3489 - val_accuracy: 0.5348
Epoch 2/20
390/390 [==============================] - 70s 180ms/step - loss: 1.3274 -
accuracy: 0.5372 - val_loss: 1.3002 - val_accuracy: 0.5622
Epoch 3/20
390/390 [==============================] - 72s 183ms/step - loss: 1.2464 -
accuracy: 0.5670 - val_loss: 1.2155 - val_accuracy: 0.5882
Epoch 4/20
390/390 [==============================] - 69s 177ms/step - loss: 1.1996 -
accuracy: 0.5838 - val_loss: 1.1703 - val_accuracy: 0.6087
Epoch 5/20
390/390 [==============================] - 69s 177ms/step - loss: 1.1705 -
accuracy: 0.5924 - val_loss: 1.1871 - val_accuracy: 0.5983
Epoch 6/20
```

```
390/390 [==============================] - 69s 177ms/step - loss: 1.1425 -
accuracy: 0.6046 - val_loss: 1.2282 - val_accuracy: 0.5898
Epoch 7/20
390/390 [==============================] - 68s 175ms/step - loss: 1.1113 -
accuracy: 0.6164 - val_loss: 1.1791 - val_accuracy: 0.6107
Epoch 8/20
390/390 [==============================] - 68s 175ms/step - loss: 1.0856 -
accuracy: 0.6234 - val_loss: 1.1832 - val_accuracy: 0.6068
Epoch 9/20
390/390 [==============================] - 68s 176ms/step - loss: 1.0641 -
accuracy: 0.6304 - val_loss: 1.1912 - val_accuracy: 0.6042
Epoch 10/20
390/390 [==============================] - 68s 175ms/step - loss: 1.0477 -
accuracy: 0.6366 - val_loss: 1.1584 - val_accuracy: 0.6127
Epoch 11/20
390/390 [==============================] - 69s 176ms/step - loss: 1.0268 -
accuracy: 0.6413 - val_loss: 1.1555 - val_accuracy: 0.6151
Epoch 12/20
390/390 [==============================] - 69s 176ms/step - loss: 1.0099 -
accuracy: 0.6452 - val_loss: 1.1770 - val_accuracy: 0.6123
Epoch 13/20
390/390 [==============================] - 68s 176ms/step - loss: 0.9938 -
accuracy: 0.6535 - val_loss: 1.1360 - val_accuracy: 0.6238
Epoch 14/20
390/390 [==============================] - 69s 176ms/step - loss: 0.9819 -
accuracy: 0.6584 - val_loss: 1.1884 - val_accuracy: 0.6113
Epoch 15/20
390/390 [==============================] - 69s 176ms/step - loss: 0.9619 -
accuracy: 0.6620 - val_loss: 1.1495 - val_accuracy: 0.6203
Epoch 16/20
390/390 [==============================] - 505s 1s/step - loss: 0.9544 -
accuracy: 0.6655 - val_loss: 1.1870 - val_accuracy: 0.6161
Epoch 17/20
390/390 [==============================] - 72s 185ms/step - loss: 0.9368 -
accuracy: 0.6738 - val_loss: 1.1870 - val_accuracy: 0.6092
Epoch 18/20
390/390 [==============================] - 72s 184ms/step - loss: 0.9270 -
accuracy: 0.6759 - val_loss: 1.1740 - val_accuracy: 0.6169
Epoch 19/20
390/390 [==============================] - 72s 184ms/step - loss: 0.9131 -
accuracy: 0.6805 - val_loss: 1.1687 - val_accuracy: 0.6190
Epoch 20/20
390/390 [==============================] - 72s 184ms/step - loss: 0.9018 -
accuracy: 0.6833 - val_loss: 1.1887 - val_accuracy: 0.6220
```

[24]:

```
def learning_curve(hist):
```

```
    plt.figure(figsize=(12, 4))

    # Plot training & val accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['accuracy'], label='Training Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')

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

    plt.tight_layout()
    plt.show()

learning_curve(hist)
```



The learning curve for the ResNet50 model shows an improvement in both training and validation accuracy over the epochs. However, the validation accuracy plateaus around 60%, which is significantly lower than the training accuracy. This discrepancy shows that the model is overfitting, as it performs well on the training data but does poorly trying to generalize to the validation set.

[25]:
```
# Evaluate the VGG16 model on the test data
test_loss, test_accuracy = vgg_model.evaluate(X_test, y_test, batch_size=256,␣
 ↪verbose=1)
```

```python
# Print the test loss and accuracy
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
```

```
40/40 [==============================] - 11s 271ms/step - loss: 1.1887 -
accuracy: 0.6220
Test Loss: 1.1887129545211792
Test Accuracy: 0.621999979019165
```

[26]:
```python
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import RMSprop

# Load the ResNet50 model with pre-trained weights, excluding the top layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32,␣
 ↪32, 3))

# Add custom layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Define the new model
resnet_model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of ResNet50 to retain pre-trained weights
for layer in base_model.layers:
    layer.trainable = False

# Compile the model with RMSprop optimizer
resnet_model.compile(optimizer=RMSprop(learning_rate=0.001),
                    loss='categorical_crossentropy',
                    metrics=['accuracy'])

hist = resnet_model.fit(it_train, steps_per_epoch=len(X_train) // 128,␣
 ↪epochs=20,
                        validation_data=(X_test, y_test), verbose=1)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 3s 0us/step
```

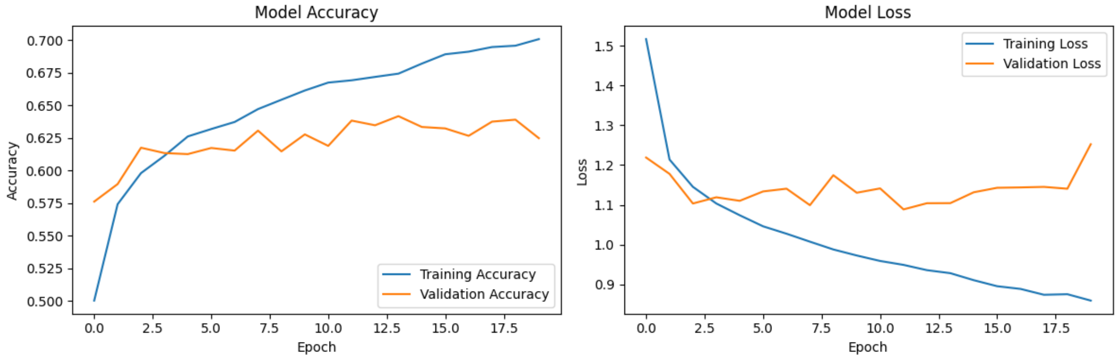
```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.RMSprop`
runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead,
located at `tf.keras.optimizers.legacy.RMSprop`.
```

```
Epoch 1/20
390/390 [==============================] - 44s 110ms/step - loss: 1.5162 -
accuracy: 0.5004 - val_loss: 1.2187 - val_accuracy: 0.5763
Epoch 2/20
390/390 [==============================] - 43s 110ms/step - loss: 1.2138 -
accuracy: 0.5743 - val_loss: 1.1777 - val_accuracy: 0.5897
Epoch 3/20
390/390 [==============================] - 45s 116ms/step - loss: 1.1451 -
accuracy: 0.5981 - val_loss: 1.1032 - val_accuracy: 0.6176
Epoch 4/20
390/390 [==============================] - 46s 119ms/step - loss: 1.1033 -
accuracy: 0.6114 - val_loss: 1.1189 - val_accuracy: 0.6135
Epoch 5/20
390/390 [==============================] - 47s 120ms/step - loss: 1.0737 -
accuracy: 0.6262 - val_loss: 1.1100 - val_accuracy: 0.6127
Epoch 6/20
390/390 [==============================] - 46s 119ms/step - loss: 1.0459 -
accuracy: 0.6319 - val_loss: 1.1336 - val_accuracy: 0.6174
Epoch 7/20
390/390 [==============================] - 48s 124ms/step - loss: 1.0274 -
accuracy: 0.6373 - val_loss: 1.1405 - val_accuracy: 0.6154
Epoch 8/20
390/390 [==============================] - 49s 127ms/step - loss: 1.0074 -
accuracy: 0.6472 - val_loss: 1.0990 - val_accuracy: 0.6307
Epoch 9/20
390/390 [==============================] - 50s 128ms/step - loss: 0.9878 -
accuracy: 0.6544 - val_loss: 1.1743 - val_accuracy: 0.6148
Epoch 10/20
390/390 [==============================] - 51s 131ms/step - loss: 0.9726 -
accuracy: 0.6615 - val_loss: 1.1303 - val_accuracy: 0.6278
Epoch 11/20
390/390 [==============================] - 49s 125ms/step - loss: 0.9589 -
accuracy: 0.6675 - val_loss: 1.1412 - val_accuracy: 0.6190
Epoch 12/20
390/390 [==============================] - 45s 116ms/step - loss: 0.9491 -
accuracy: 0.6693 - val_loss: 1.0885 - val_accuracy: 0.6384
Epoch 13/20
390/390 [==============================] - 44s 113ms/step - loss: 0.9358 -
accuracy: 0.6719 - val_loss: 1.1039 - val_accuracy: 0.6348
Epoch 14/20
390/390 [==============================] - 45s 116ms/step - loss: 0.9282 -
accuracy: 0.6744 - val_loss: 1.1042 - val_accuracy: 0.6418
Epoch 15/20
390/390 [==============================] - 46s 119ms/step - loss: 0.9106 -
accuracy: 0.6821 - val_loss: 1.1315 - val_accuracy: 0.6335
Epoch 16/20
390/390 [==============================] - 49s 127ms/step - loss: 0.8955 -
accuracy: 0.6892 - val_loss: 1.1428 - val_accuracy: 0.6324
```

```
Epoch 17/20
390/390 [==============================] - 49s 125ms/step - loss: 0.8887 -
accuracy: 0.6912 - val_loss: 1.1437 - val_accuracy: 0.6267
Epoch 18/20
390/390 [==============================] - 49s 127ms/step - loss: 0.8740 -
accuracy: 0.6948 - val_loss: 1.1450 - val_accuracy: 0.6376
Epoch 19/20
390/390 [==============================] - 49s 126ms/step - loss: 0.8754 -
accuracy: 0.6958 - val_loss: 1.1404 - val_accuracy: 0.6391
Epoch 20/20
390/390 [==============================] - 47s 121ms/step - loss: 0.8594 -
accuracy: 0.7009 - val_loss: 1.2521 - val_accuracy: 0.6249
```

[27]:

```
def learning_curve(hist):

    plt.figure(figsize=(12, 4))

    # Plot training & val accuracy values
    plt.subplot(1, 2, 1)
    plt.plot(hist.history['accuracy'], label='Training Accuracy')
    plt.plot(hist.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')

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

    plt.tight_layout()
    plt.show()

learning_curve(hist)
```



The updated learning curve for the ResNet50 model still shows a noticeable gap between training and validation accuracy, with validation accuracy plateauing around 63% while training accuracy continues to increase. The model is still badly overfitting

[28]:
```
%shell jupyter nbconvert --to pdf 'filename.ipynb'
```

```
zsh:1: command not found: apt-get
zsh:1: command not found: apt-get
```