uwzo0krou

August 12, 2024

[14]:
```python
import tensorflow as tf
import os
import numpy as np
from PIL import Image
from tensorflow.keras.preprocessing.image import img_to_array
def load_flower_dataset(dataset_path):
    data = []
    labels = []
    class_names = os.listdir(dataset_path)
    class_indices = {class_name: idx for idx, class_name in␣
 ↪enumerate(class_names)}

    for class_name in class_names:
        class_path = os.path.join(dataset_path, class_name)
        if not os.path.isdir(class_path):
            continue
        for file in os.listdir(class_path):
            if file.endswith(('jpg', 'jpeg', 'png')):
                file_path = os.path.join(class_path, file)
                try:
                    img = Image.open(file_path).convert('RGB')
                    img = img.resize((128, 128))  # Resize to a fixed size
                    img_array = img_to_array(img)
                    data.append(img_array)
                    labels.append(class_indices[class_name])
                except Exception as e:
                    print(f"Error loading image {file_path}: {e}")

    data = np.array(data, dtype='float32') / 255.0  # Normalize images
    labels = np.array(labels)
    return data, labels, class_names

# Load the dataset
dataset_path = 'flower_photos'
data, labels, class_names = load_flower_dataset(dataset_path)
```

[16]:
```
from sklearn.model_selection import train_test_split

# Splitting the data into training and test set
x_train, x_test, y_train, y_test = train_test_split(
    data, labels, test_size=0.2, random_state=42  # 20% for test set
)

# Print shapes of the datasets
print(f"Training data shape: {x_train.shape}") # Prints the shape of training␣
↪data
print(f"Testing data shape: {x_test.shape}") # Prints the shape of Test data
print(f"Training labels shape: {y_train.shape}") # Prints the shape of training␣
↪lebels
print(f"Testing labels shape: {y_test.shape}") # Prints the shape of testing␣
↪labels
print(f"Class names: {class_names}") # Prints the types of flowers in the␣
↪dataset
```

```
Training data shape: (2936, 128, 128, 3)
Testing data shape: (734, 128, 128, 3)
Training labels shape: (2936,)
Testing labels shape: (734,)
Class names: ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

[17]:
```
import keras
# Converting label encoding to one hot encoding
y_train_cat = keras.utils.to_categorical(y_train)
y_test_cat = keras.utils.to_categorical(y_test)
```

[18]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Initializing an ImageDataGenerator for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rotation_range=20,          # Randomly rotate images by up to 20 degrees
    width_shift_range=0.1,      # Randomly shift images horizontally by up to␣
↪10% of the width
    height_shift_range=0.1,     # Randomly shift images vertically by up to␣
↪10% of the height
    horizontal_flip=True,       # Randomly flip images horizontally
    vertical_flip=False,        # Do not flip images vertically
    shear_range=0.10,           # Apply a random shear transformation of up␣
↪to 10%
    zoom_range=0.10,            # Randomly zoom in on images by up to 10%
    validation_split=0.2        # Reserve 20% of the data for validation
)
```

1 Model 1

1.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 32

[20]:
```
model_1 = Sequential()

# First Convolutional and pooling layer block
model_1.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_1.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_1.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_1.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_1.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_1.add(MaxPooling2D(pool_size = (2, 2)))

# Adding Flatten Layer
model_1.add(Flatten())
# Dense layer with 256 neurons
model_1.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_1.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_1.add(Dense(5, activation = 'softmax'))

model_1.summary()
```

```
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_1"

```
 Layer (type)                       Output Shape                         ␣
↪Param #

 conv2d_3 (Conv2D)                  (None, 126, 126, 32)                  ␣
↪896

 max_pooling2d_3 (MaxPooling2D)     (None, 63, 63, 32)                    ␣
↪  0
```

```
conv2d_4 (Conv2D)                    (None, 61, 61, 64)                 ␣
↪18,496

max_pooling2d_4 (MaxPooling2D)       (None, 30, 30, 64)                   ␣
↪  0

conv2d_5 (Conv2D)                    (None, 28, 28, 128)                ␣
↪73,856

max_pooling2d_5 (MaxPooling2D)       (None, 14, 14, 128)                  ␣
↪  0

flatten_1 (Flatten)                  (None, 25088)                        ␣
↪  0

dense_2 (Dense)                      (None, 256)                      ␣
↪6,422,784

dropout_1 (Dropout)                  (None, 256)                          ␣
↪  0

dense_3 (Dense)                      (None, 5)                           ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

[22]:

```
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
 ↪0.001

model_1.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

batch_size = 32 # Batch size set to 32
history_1 = model_1.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 20, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
```

```python
                                        batch_size = batch_size,
                                        subset = "validation")) # Epochs set to 20
```

```
Epoch 1/20
```

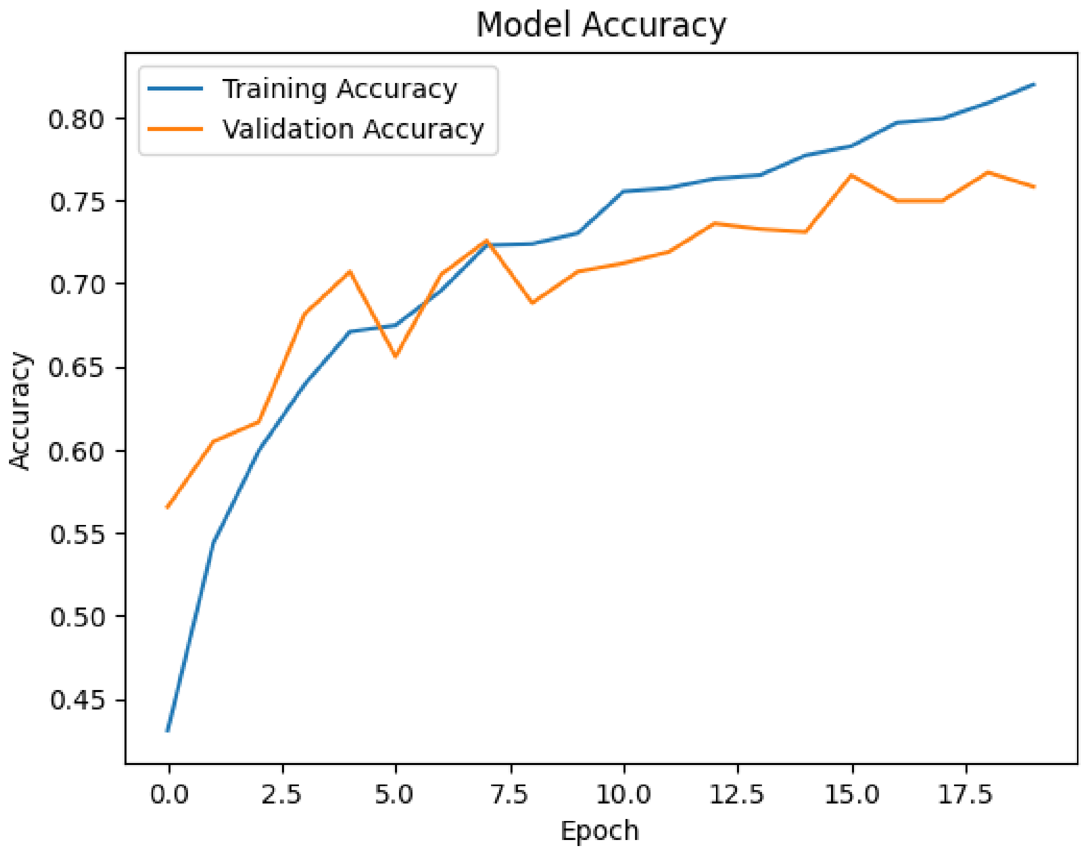
```
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

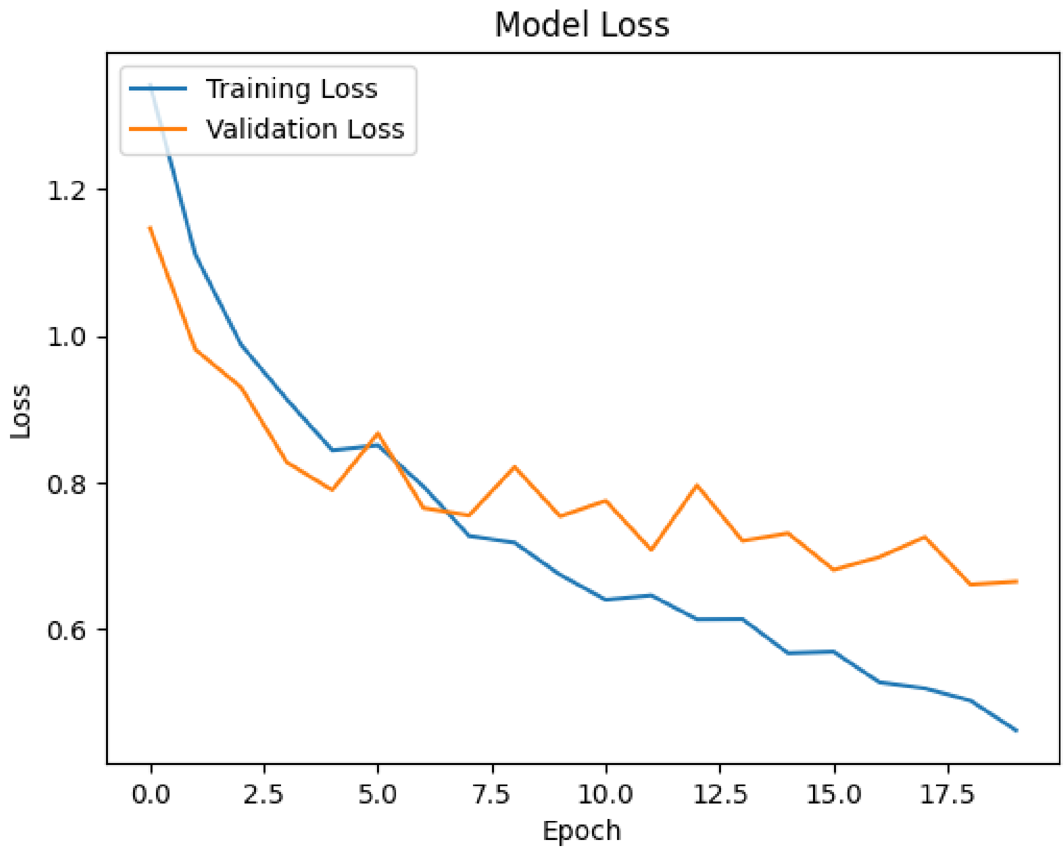
```
74/74                 45s 466ms/step -
accuracy: 0.3540 - loss: 1.5293 - val_accuracy: 0.5656 - val_loss: 1.1471
Epoch 2/20
74/74                 35s 457ms/step -
accuracy: 0.5176 - loss: 1.1505 - val_accuracy: 0.6048 - val_loss: 0.9812
Epoch 3/20
74/74                 34s 455ms/step -
accuracy: 0.5995 - loss: 0.9785 - val_accuracy: 0.6167 - val_loss: 0.9298
Epoch 4/20
74/74                 34s 453ms/step -
accuracy: 0.6247 - loss: 0.9302 - val_accuracy: 0.6814 - val_loss: 0.8277
Epoch 5/20
74/74                 34s 451ms/step -
accuracy: 0.6531 - loss: 0.8678 - val_accuracy: 0.7070 - val_loss: 0.7897
Epoch 6/20
74/74                 35s 463ms/step -
accuracy: 0.6809 - loss: 0.8204 - val_accuracy: 0.6559 - val_loss: 0.8670
Epoch 7/20
74/74                 34s 453ms/step -
accuracy: 0.6881 - loss: 0.7963 - val_accuracy: 0.7053 - val_loss: 0.7653
Epoch 8/20
74/74                 35s 464ms/step -
accuracy: 0.7330 - loss: 0.7261 - val_accuracy: 0.7257 - val_loss: 0.7551
Epoch 9/20
74/74                 35s 463ms/step -
accuracy: 0.7361 - loss: 0.7011 - val_accuracy: 0.6882 - val_loss: 0.8213
Epoch 10/20
74/74                 34s 453ms/step -
accuracy: 0.7327 - loss: 0.6556 - val_accuracy: 0.7070 - val_loss: 0.7540
Epoch 11/20
74/74                 34s 453ms/step -
accuracy: 0.7737 - loss: 0.6247 - val_accuracy: 0.7121 - val_loss: 0.7751
Epoch 12/20
74/74                 35s 458ms/step -
```

```
accuracy: 0.7658 - loss: 0.6152 - val_accuracy: 0.7189 - val_loss: 0.7080
Epoch 13/20
74/74           37s 487ms/step -
accuracy: 0.7506 - loss: 0.6293 - val_accuracy: 0.7359 - val_loss: 0.7965
Epoch 14/20
74/74           34s 452ms/step -
accuracy: 0.7653 - loss: 0.6242 - val_accuracy: 0.7325 - val_loss: 0.7205
Epoch 15/20
74/74           35s 460ms/step -
accuracy: 0.7781 - loss: 0.5701 - val_accuracy: 0.7308 - val_loss: 0.7309
Epoch 16/20
74/74           35s 466ms/step -
accuracy: 0.7937 - loss: 0.5381 - val_accuracy: 0.7649 - val_loss: 0.6809
Epoch 17/20
74/74           35s 459ms/step -
accuracy: 0.7980 - loss: 0.5314 - val_accuracy: 0.7496 - val_loss: 0.6983
Epoch 18/20
74/74           35s 457ms/step -
accuracy: 0.8038 - loss: 0.5150 - val_accuracy: 0.7496 - val_loss: 0.7256
Epoch 19/20
74/74           35s 463ms/step -
accuracy: 0.8178 - loss: 0.4740 - val_accuracy: 0.7666 - val_loss: 0.6609
Epoch 20/20
74/74           42s 561ms/step -
accuracy: 0.8164 - loss: 0.4658 - val_accuracy: 0.7581 - val_loss: 0.6649
```

[24]:

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_1.history['accuracy'], label='Training Accuracy')
plt.plot(history_1.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_1.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_1.history['loss'], label='Training Loss')
plt.plot(history_1.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_1.png')
plt.show()
```


Model Accuracy
Training Accuracy
Validation Accuracy
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch



2 Model 2

2.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.0001, and batch size 32

[25]:
```
model_2 = Sequential()

# First Convolutional and pooling layer block
model_2.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_2.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_2.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_2.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_2.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_2.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding Flatten Layer
model_2.add(Flatten())
# Dense layer with 256 neurons
model_2.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_2.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_2.add(Dense(5, activation = 'softmax'))

model_2.summary()
```

```
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_2"

```
 Layer (type)                         Output Shape                         ␣
↪Param #

 conv2d_6 (Conv2D)                    (None, 126, 126, 32)                    ␣
↪896

 max_pooling2d_6 (MaxPooling2D)       (None, 63, 63, 32)                      ␣
↪   0

 conv2d_7 (Conv2D)                    (None, 61, 61, 64)                    ␣
↪18,496

 max_pooling2d_7 (MaxPooling2D)       (None, 30, 30, 64)                      ␣
↪   0

 conv2d_8 (Conv2D)                    (None, 28, 28, 128)                   ␣
↪73,856

 max_pooling2d_8 (MaxPooling2D)       (None, 14, 14, 128)                     ␣
↪   0

 flatten_2 (Flatten)                  (None, 25088)                           ␣
↪   0

 dense_4 (Dense)                      (None, 256)                         ␣
↪6,422,784
```

```
 dropout_2 (Dropout)                  (None, 256)                                    ␣
↪  0

 dense_5 (Dense)                      (None, 5)                                    ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

[26]:
```
adam_optimizer = Adam(learning_rate = 0.0001) # Learning rate dropped of 0.0001

model_2.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

batch_size = 32 # Batch size set to 32
history_2 = model_2.fit(train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "training"),
                        epochs = 20, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "validation")) # Epochs set to␣
 ↪20
```

```
Epoch 1/20
```

```
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121:
UserWarning: Your `PyDataset` class should call `super().__init__(**kwargs)` in
its constructor. `**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will be
ignored.
  self._warn_if_super_not_called()
```

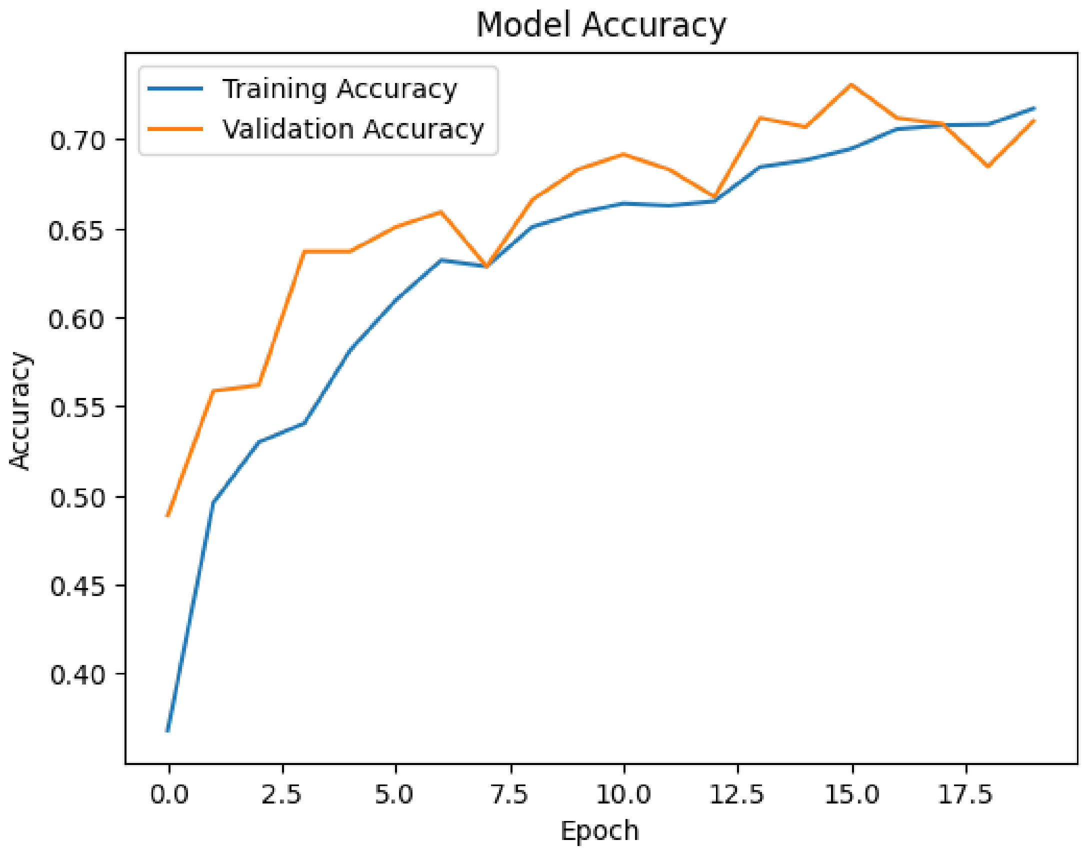
```
74/74                 40s 496ms/step -
accuracy: 0.3183 - loss: 1.5270 - val_accuracy: 0.4889 - val_loss: 1.2492
Epoch 2/20
74/74                 38s 503ms/step -
accuracy: 0.4727 - loss: 1.2358 - val_accuracy: 0.5588 - val_loss: 1.1030
Epoch 3/20
74/74                 40s 534ms/step -
```

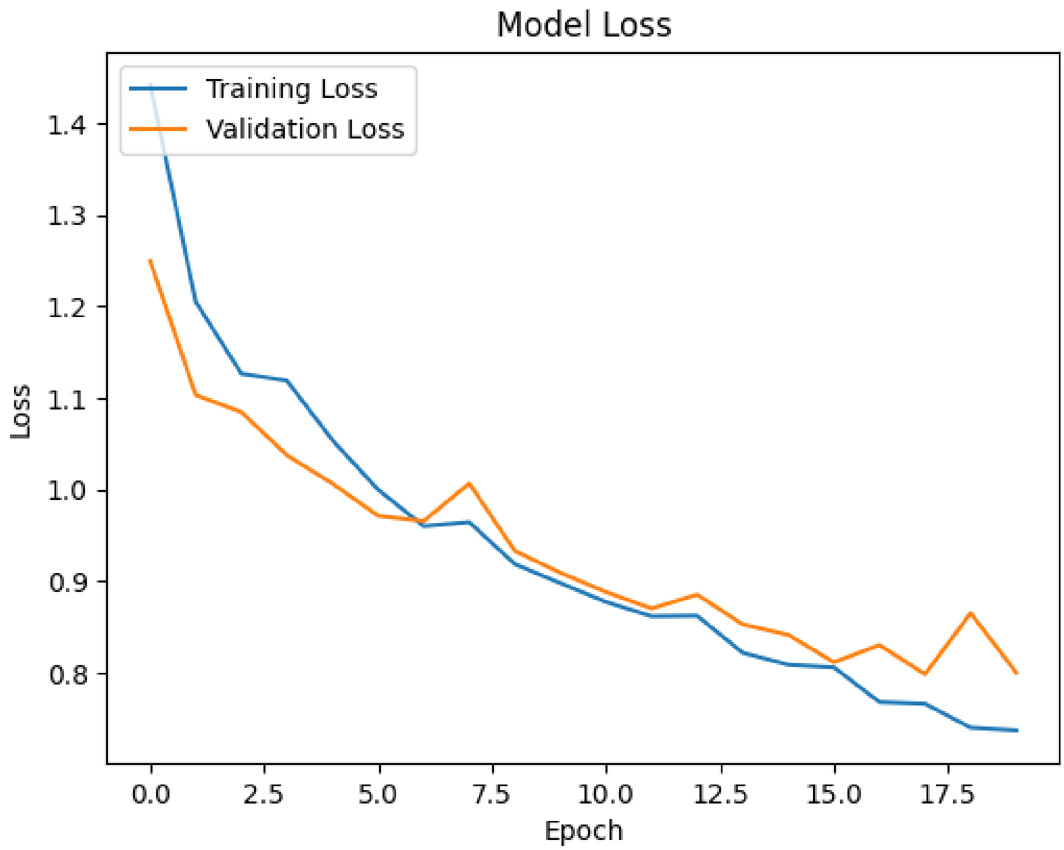
```
accuracy: 0.5301 - loss: 1.1428 - val_accuracy: 0.5622 - val_loss: 1.0846
Epoch 4/20
74/74            34s 450ms/step -
accuracy: 0.5348 - loss: 1.1307 - val_accuracy: 0.6371 - val_loss: 1.0373
Epoch 5/20
74/74            41s 541ms/step -
accuracy: 0.5823 - loss: 1.0483 - val_accuracy: 0.6371 - val_loss: 1.0065
Epoch 6/20
74/74            41s 524ms/step -
accuracy: 0.5936 - loss: 1.0224 - val_accuracy: 0.6508 - val_loss: 0.9713
Epoch 7/20
74/74            39s 520ms/step -
accuracy: 0.6420 - loss: 0.9482 - val_accuracy: 0.6593 - val_loss: 0.9657
Epoch 8/20
74/74            40s 528ms/step -
accuracy: 0.6403 - loss: 0.9516 - val_accuracy: 0.6286 - val_loss: 1.0066
Epoch 9/20
74/74            36s 478ms/step -
accuracy: 0.6515 - loss: 0.9361 - val_accuracy: 0.6661 - val_loss: 0.9330
Epoch 10/20
74/74            43s 568ms/step -
accuracy: 0.6543 - loss: 0.9028 - val_accuracy: 0.6831 - val_loss: 0.9092
Epoch 11/20
74/74            45s 598ms/step -
accuracy: 0.6576 - loss: 0.8780 - val_accuracy: 0.6917 - val_loss: 0.8882
Epoch 12/20
74/74            41s 536ms/step -
accuracy: 0.6760 - loss: 0.8291 - val_accuracy: 0.6831 - val_loss: 0.8703
Epoch 13/20
74/74            37s 495ms/step -
accuracy: 0.6596 - loss: 0.8723 - val_accuracy: 0.6678 - val_loss: 0.8853
Epoch 14/20
74/74            37s 494ms/step -
accuracy: 0.6754 - loss: 0.8436 - val_accuracy: 0.7121 - val_loss: 0.8529
Epoch 15/20
74/74            38s 490ms/step -
accuracy: 0.6943 - loss: 0.8037 - val_accuracy: 0.7070 - val_loss: 0.8415
Epoch 16/20
74/74            38s 500ms/step -
accuracy: 0.6840 - loss: 0.8108 - val_accuracy: 0.7308 - val_loss: 0.8116
Epoch 17/20
74/74            35s 467ms/step -
accuracy: 0.7098 - loss: 0.7733 - val_accuracy: 0.7121 - val_loss: 0.8304
Epoch 18/20
74/74            36s 470ms/step -
accuracy: 0.7024 - loss: 0.7880 - val_accuracy: 0.7087 - val_loss: 0.7986
Epoch 19/20
74/74            35s 464ms/step -
```

```
accuracy: 0.7072 - loss: 0.7363 - val_accuracy: 0.6848 - val_loss: 0.8653
Epoch 20/20
74/74               35s 466ms/step -
accuracy: 0.7013 - loss: 0.7708 - val_accuracy: 0.7104 - val_loss: 0.8002
```

[29]:

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_2.history['accuracy'], label='Training Accuracy')
plt.plot(history_2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_2.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_2.history['loss'], label='Training Loss')
plt.plot(history_2.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_2.png')
plt.show()
```


Model Accuracy
Training Accuracy
Validation Accuracy
0.70
0.65
0.60
0.55
0.50
0.45
0.40
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch



3 Model 3

3.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 64

[32]:
```
model_3 = Sequential()

# First Convolutional and pooling layer block
model_3.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_3.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_3.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_3.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_3.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_3.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding Flatten Layer
model_3.add(Flatten())
# Dense layer with 256 neurons
model_3.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_3.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_3.add(Dense(5, activation = 'softmax'))

model_3.summary()
```

```
Model: "sequential_5"


 Layer (type)                    Output Shape                           ␣
↪Param #

 conv2d_15 (Conv2D)              (None, 126, 126, 32)                    ␣
↪896

 max_pooling2d_15 (MaxPooling2D)  (None, 63, 63, 32)                     ␣
↪  0

 conv2d_16 (Conv2D)              (None, 61, 61, 64)                     ␣
↪18,496

 max_pooling2d_16 (MaxPooling2D)  (None, 30, 30, 64)                     ␣
↪  0

 conv2d_17 (Conv2D)              (None, 28, 28, 128)                    ␣
↪73,856

 max_pooling2d_17 (MaxPooling2D)  (None, 14, 14, 128)                    ␣
↪  0

 flatten_5 (Flatten)             (None, 25088)                           ␣
↪  0

 dense_10 (Dense)                (None, 256)                           ␣
↪6,422,784

 dropout_5 (Dropout)             (None, 256)                             ␣
↪  0

 dense_11 (Dense)                (None, 5)                              ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

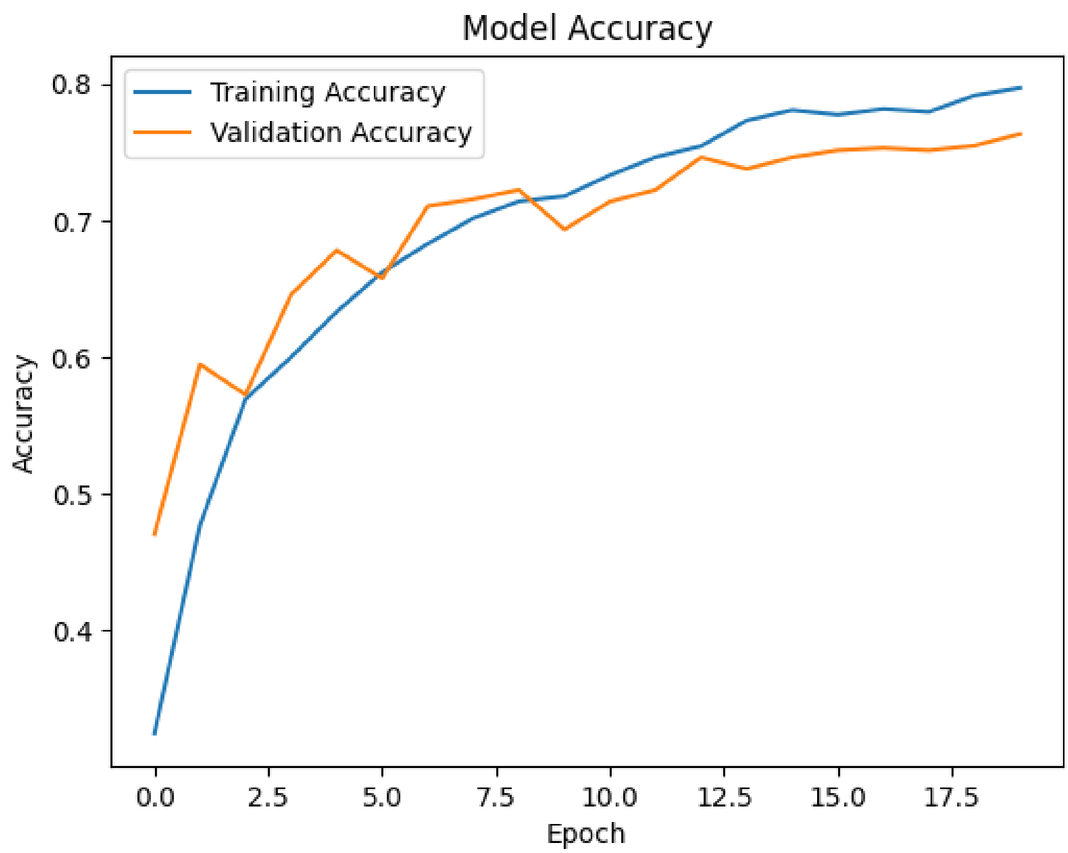
[33]:
```
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
↪0.01

model_3.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

batch_size = 64 # Batch size set to 64
history_3 = model_3.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 20, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation")) # Epochs set to␣
↪20
```

```
Epoch 1/20
37/37                39s 962ms/step -
accuracy: 0.2415 - loss: 1.6806 - val_accuracy: 0.4702 - val_loss: 1.2553
Epoch 2/20
37/37                36s 929ms/step -
accuracy: 0.4407 - loss: 1.2361 - val_accuracy: 0.5945 - val_loss: 1.0731
Epoch 3/20
37/37                36s 943ms/step -
accuracy: 0.5850 - loss: 1.0682 - val_accuracy: 0.5724 - val_loss: 1.0482
Epoch 4/20
37/37                36s 943ms/step -
accuracy: 0.5976 - loss: 1.0210 - val_accuracy: 0.6457 - val_loss: 0.9734
Epoch 5/20
37/37                36s 933ms/step -
accuracy: 0.6343 - loss: 0.9325 - val_accuracy: 0.6780 - val_loss: 0.8652
Epoch 6/20
37/37                35s 904ms/step -
accuracy: 0.6499 - loss: 0.8771 - val_accuracy: 0.6576 - val_loss: 0.8892
Epoch 7/20
37/37                33s 861ms/step -
accuracy: 0.6786 - loss: 0.8273 - val_accuracy: 0.7104 - val_loss: 0.7833
```

```
Epoch 8/20
37/37            33s 853ms/step -
accuracy: 0.7067 - loss: 0.7757 - val_accuracy: 0.7155 - val_loss: 0.7595
Epoch 9/20
37/37            33s 852ms/step -
accuracy: 0.7154 - loss: 0.7131 - val_accuracy: 0.7223 - val_loss: 0.7564
Epoch 10/20
37/37            33s 850ms/step -
accuracy: 0.7088 - loss: 0.7295 - val_accuracy: 0.6934 - val_loss: 0.8716
Epoch 11/20
37/37            34s 876ms/step -
accuracy: 0.7551 - loss: 0.6726 - val_accuracy: 0.7138 - val_loss: 0.7493
Epoch 12/20
37/37            33s 850ms/step -
accuracy: 0.7418 - loss: 0.6528 - val_accuracy: 0.7223 - val_loss: 0.7339
Epoch 13/20
37/37            33s 865ms/step -
accuracy: 0.7535 - loss: 0.6495 - val_accuracy: 0.7462 - val_loss: 0.6879
Epoch 14/20
37/37            34s 873ms/step -
accuracy: 0.7812 - loss: 0.5829 - val_accuracy: 0.7376 - val_loss: 0.7269
Epoch 15/20
37/37            33s 857ms/step -
accuracy: 0.7767 - loss: 0.5892 - val_accuracy: 0.7462 - val_loss: 0.6802
Epoch 16/20
37/37            33s 863ms/step -
accuracy: 0.7738 - loss: 0.5685 - val_accuracy: 0.7513 - val_loss: 0.6788
Epoch 17/20
37/37            33s 853ms/step -
accuracy: 0.7854 - loss: 0.5721 - val_accuracy: 0.7530 - val_loss: 0.6644
Epoch 18/20
37/37            34s 868ms/step -
accuracy: 0.7880 - loss: 0.5451 - val_accuracy: 0.7513 - val_loss: 0.6877
Epoch 19/20
37/37            33s 855ms/step -
accuracy: 0.7891 - loss: 0.5534 - val_accuracy: 0.7547 - val_loss: 0.7163
Epoch 20/20
37/37            35s 897ms/step -
accuracy: 0.8000 - loss: 0.5298 - val_accuracy: 0.7632 - val_loss: 0.6373
```

[34]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_3.history['accuracy'], label='Training Accuracy')
plt.plot(history_3.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_3.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_3.history['loss'], label='Training Loss')
plt.plot(history_3.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_3.png')
plt.show()
```



4 Model 4

4.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 128

[35]:
```
model_4 = Sequential()

# First Convolutional and pooling layer block
model_4.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_4.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_4.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_4.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_4.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_4.add(MaxPooling2D(pool_size = (2, 2)))
```

```
# Adding Flatten Layer
model_4.add(Flatten())
# Dense layer with 256 neurons
model_4.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_4.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_4.add(Dense(5, activation = 'softmax'))

model_4.summary()
```

```
Model: "sequential_6"


 Layer (type)                        Output Shape                         ␣
↪Param #

 conv2d_18 (Conv2D)                  (None, 126, 126, 32)                     ␣
↪896

 max_pooling2d_18 (MaxPooling2D)     (None, 63, 63, 32)                       ␣
↪  0

 conv2d_19 (Conv2D)                  (None, 61, 61, 64)                    ␣
↪18,496

 max_pooling2d_19 (MaxPooling2D)     (None, 30, 30, 64)                       ␣
↪  0

 conv2d_20 (Conv2D)                  (None, 28, 28, 128)                   ␣
↪73,856

 max_pooling2d_20 (MaxPooling2D)     (None, 14, 14, 128)                      ␣
↪  0

 flatten_6 (Flatten)                 (None, 25088)                            ␣
↪  0

 dense_12 (Dense)                    (None, 256)                        ␣
↪6,422,784

 dropout_6 (Dropout)                 (None, 256)                              ␣
↪  0

 dense_13 (Dense)                    (None, 5)                              ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

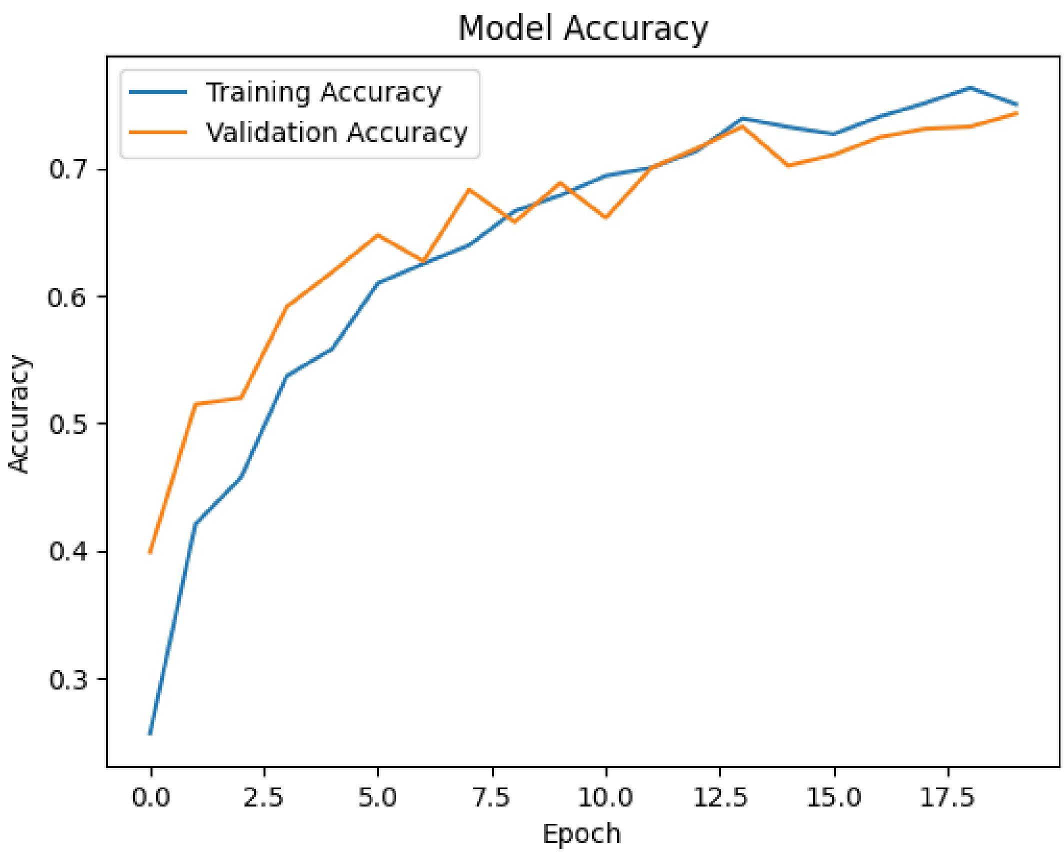
[36]:
```
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
↪0.01

model_4.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

batch_size = 128 # Batch size set to 128
history_4 = model_4.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 20, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation")) # Epochs set to␣
↪20
```

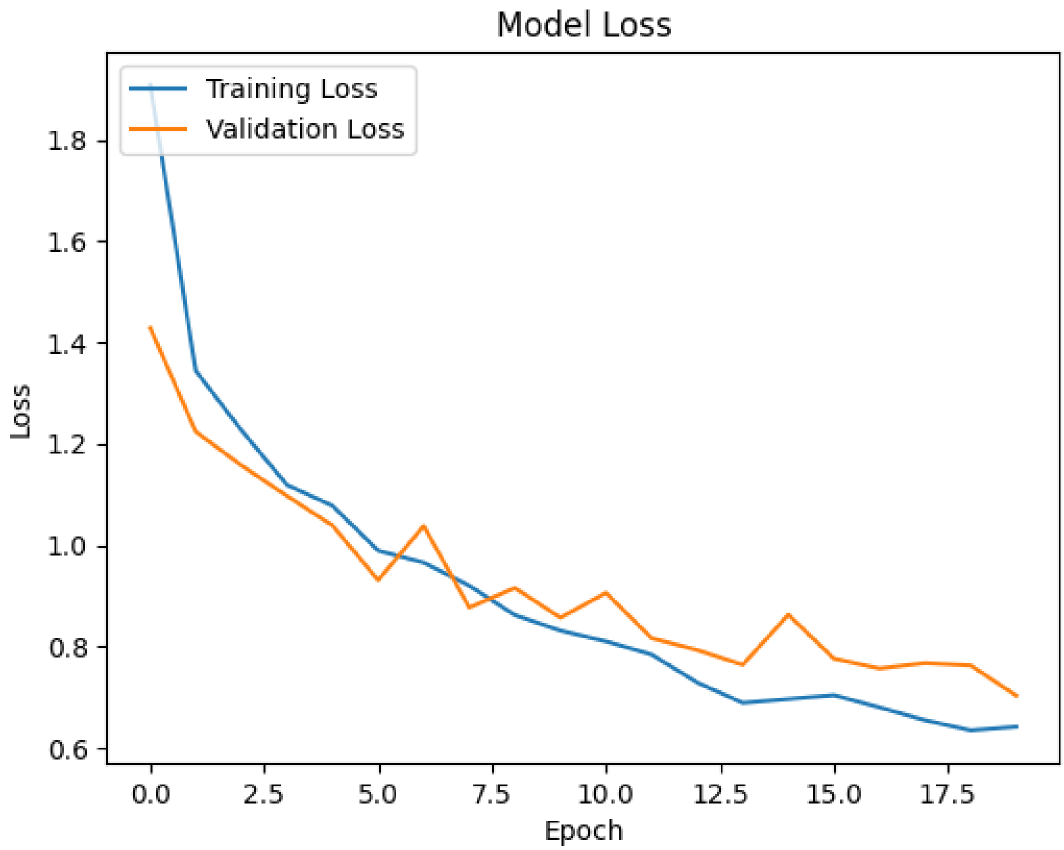
```
Epoch 1/20
19/19             40s 2s/step -
accuracy: 0.2280 - loss: 2.4422 - val_accuracy: 0.3986 - val_loss: 1.4285
Epoch 2/20
19/19             33s 2s/step -
accuracy: 0.4036 - loss: 1.3857 - val_accuracy: 0.5145 - val_loss: 1.2235
Epoch 3/20
19/19             35s 2s/step -
accuracy: 0.4410 - loss: 1.2559 - val_accuracy: 0.5196 - val_loss: 1.1572
Epoch 4/20
19/19             33s 2s/step -
accuracy: 0.5288 - loss: 1.1049 - val_accuracy: 0.5911 - val_loss: 1.0968
Epoch 5/20
19/19             34s 2s/step -
accuracy: 0.5522 - loss: 1.0910 - val_accuracy: 0.6184 - val_loss: 1.0384
Epoch 6/20
19/19             33s 2s/step -
accuracy: 0.6014 - loss: 1.0171 - val_accuracy: 0.6474 - val_loss: 0.9306
Epoch 7/20
19/19             38s 2s/step -
accuracy: 0.6270 - loss: 0.9738 - val_accuracy: 0.6269 - val_loss: 1.0374
```

```
Epoch 8/20
19/19            37s 2s/step -
accuracy: 0.6365 - loss: 0.9206 - val_accuracy: 0.6831 - val_loss: 0.8767
Epoch 9/20
19/19            33s 2s/step -
accuracy: 0.6875 - loss: 0.8234 - val_accuracy: 0.6576 - val_loss: 0.9152
Epoch 10/20
19/19            33s 2s/step -
accuracy: 0.6567 - loss: 0.8579 - val_accuracy: 0.6882 - val_loss: 0.8566
Epoch 11/20
19/19            33s 2s/step -
accuracy: 0.6993 - loss: 0.7944 - val_accuracy: 0.6610 - val_loss: 0.9055
Epoch 12/20
19/19            33s 2s/step -
accuracy: 0.6785 - loss: 0.8076 - val_accuracy: 0.7002 - val_loss: 0.8162
Epoch 13/20
19/19            33s 2s/step -
accuracy: 0.7228 - loss: 0.7137 - val_accuracy: 0.7155 - val_loss: 0.7926
Epoch 14/20
19/19            33s 2s/step -
accuracy: 0.7405 - loss: 0.6864 - val_accuracy: 0.7325 - val_loss: 0.7637
Epoch 15/20
19/19            33s 2s/step -
accuracy: 0.7321 - loss: 0.6989 - val_accuracy: 0.7019 - val_loss: 0.8624
Epoch 16/20
19/19            33s 2s/step -
accuracy: 0.7114 - loss: 0.7397 - val_accuracy: 0.7104 - val_loss: 0.7758
Epoch 17/20
19/19            33s 2s/step -
accuracy: 0.7475 - loss: 0.6627 - val_accuracy: 0.7240 - val_loss: 0.7563
Epoch 18/20
19/19            33s 2s/step -
accuracy: 0.7561 - loss: 0.6445 - val_accuracy: 0.7308 - val_loss: 0.7672
Epoch 19/20
19/19            33s 2s/step -
accuracy: 0.7590 - loss: 0.6403 - val_accuracy: 0.7325 - val_loss: 0.7623
Epoch 20/20
19/19            33s 2s/step -
accuracy: 0.7563 - loss: 0.6362 - val_accuracy: 0.7428 - val_loss: 0.7028
```

[37]:
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_4.history['accuracy'], label='Training Accuracy')
plt.plot(history_4.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
```

```
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_4.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_4.history['loss'], label='Training Loss')
plt.plot(history_4.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_4.png')
plt.show()
```





5 Model 5

5.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, batch size 64 and epochs 50

[38]:
```
model_5 = Sequential()

# First Convolutional and pooling layer block
model_5.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_5.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_5.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_5.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_5.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_5.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding Flatten Layer
model_5.add(Flatten())
# Dense layer with 256 neurons
model_5.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_5.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_5.add(Dense(5, activation = 'softmax'))

model_5.summary()
```

```
Model: "sequential_7"


 Layer (type)                    Output Shape
 ↪Param #

 conv2d_21 (Conv2D)              (None, 126, 126, 32)
 ↪896

 max_pooling2d_21 (MaxPooling2D)  (None, 63, 63, 32)
 ↪  0

 conv2d_22 (Conv2D)              (None, 61, 61, 64)
 ↪18,496

 max_pooling2d_22 (MaxPooling2D)  (None, 30, 30, 64)
 ↪  0

 conv2d_23 (Conv2D)              (None, 28, 28, 128)
 ↪73,856

 max_pooling2d_23 (MaxPooling2D)  (None, 14, 14, 128)
 ↪  0

 flatten_7 (Flatten)             (None, 25088)
 ↪  0

 dense_14 (Dense)                (None, 256)
 ↪6,422,784

 dropout_7 (Dropout)             (None, 256)
 ↪  0
```

```
 dense_15 (Dense)                    (None, 5)                    ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

[39]:
```python
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
↪0.01

model_5.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

batch_size = 64 # Batch size set to 64
history_5 = model_5.fit(train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "training"),
                        epochs = 50, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "validation")) # Epochs set to␣
↪50
```

```
Epoch 1/50
37/37                 34s 816ms/step -
accuracy: 0.2982 - loss: 1.7881 - val_accuracy: 0.4736 - val_loss: 1.2011
Epoch 2/50
37/37                 33s 842ms/step -
accuracy: 0.4768 - loss: 1.1839 - val_accuracy: 0.5417 - val_loss: 1.1242
Epoch 3/50
37/37                 41s 1s/step -
accuracy: 0.5371 - loss: 1.0886 - val_accuracy: 0.6252 - val_loss: 1.0247
Epoch 4/50
37/37                 33s 842ms/step -
accuracy: 0.6027 - loss: 1.0091 - val_accuracy: 0.6593 - val_loss: 0.9494
Epoch 5/50
37/37                 33s 854ms/step -
accuracy: 0.6457 - loss: 0.9130 - val_accuracy: 0.6508 - val_loss: 0.9131
Epoch 6/50
37/37                 33s 843ms/step -
accuracy: 0.6612 - loss: 0.8960 - val_accuracy: 0.6405 - val_loss: 0.9689
Epoch 7/50
```

```
37/37 33s 858ms/step -
accuracy: 0.6707 - loss: 0.8731 - val_accuracy: 0.7019 - val_loss: 0.8489
Epoch 8/50
37/37 33s 845ms/step -
accuracy: 0.6770 - loss: 0.8283 - val_accuracy: 0.6917 - val_loss: 0.8304
Epoch 9/50
37/37 33s 853ms/step -
accuracy: 0.7012 - loss: 0.7844 - val_accuracy: 0.6899 - val_loss: 0.8331
Epoch 10/50
37/37 33s 843ms/step -
accuracy: 0.7220 - loss: 0.7234 - val_accuracy: 0.7308 - val_loss: 0.7559
Epoch 11/50
37/37 33s 859ms/step -
accuracy: 0.7348 - loss: 0.6921 - val_accuracy: 0.7172 - val_loss: 0.7612
Epoch 12/50
37/37 33s 857ms/step -
accuracy: 0.7315 - loss: 0.6959 - val_accuracy: 0.7325 - val_loss: 0.7430
Epoch 13/50
37/37 33s 852ms/step -
accuracy: 0.7418 - loss: 0.6565 - val_accuracy: 0.7070 - val_loss: 0.7982
Epoch 14/50
37/37 33s 845ms/step -
accuracy: 0.7633 - loss: 0.6072 - val_accuracy: 0.7530 - val_loss: 0.7082
Epoch 15/50
37/37 33s 844ms/step -
accuracy: 0.7662 - loss: 0.5904 - val_accuracy: 0.7325 - val_loss: 0.7469
Epoch 16/50
37/37 33s 858ms/step -
accuracy: 0.7711 - loss: 0.5745 - val_accuracy: 0.7257 - val_loss: 0.7383
Epoch 17/50
37/37 33s 842ms/step -
accuracy: 0.7750 - loss: 0.5673 - val_accuracy: 0.7223 - val_loss: 0.7691
Epoch 18/50
37/37 33s 843ms/step -
accuracy: 0.7812 - loss: 0.5712 - val_accuracy: 0.7394 - val_loss: 0.7166
Epoch 19/50
37/37 33s 846ms/step -
accuracy: 0.7793 - loss: 0.5405 - val_accuracy: 0.7547 - val_loss: 0.7263
Epoch 20/50
37/37 33s 858ms/step -
accuracy: 0.8071 - loss: 0.4908 - val_accuracy: 0.7019 - val_loss: 0.7757
Epoch 21/50
37/37 33s 864ms/step -
accuracy: 0.8200 - loss: 0.5106 - val_accuracy: 0.7683 - val_loss: 0.6652
Epoch 22/50
37/37 33s 846ms/step -
accuracy: 0.8170 - loss: 0.4650 - val_accuracy: 0.7462 - val_loss: 0.7107
Epoch 23/50
```

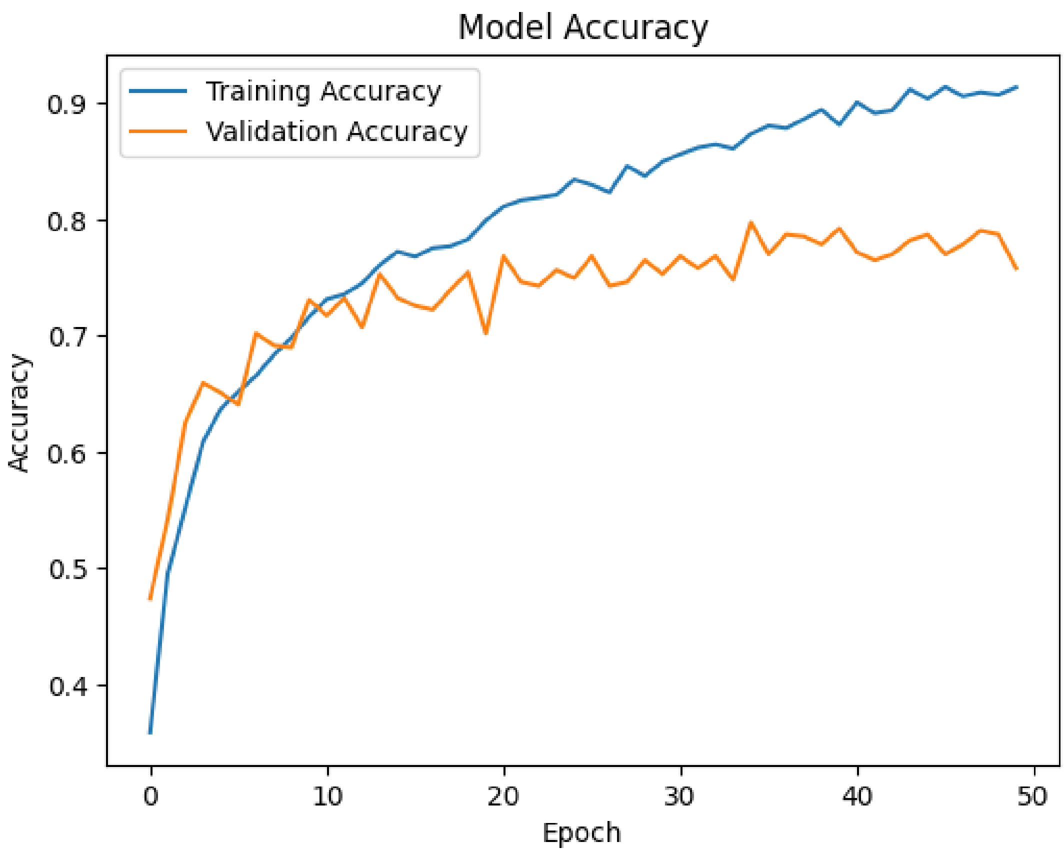
```
37/37              33s 859ms/step -
accuracy: 0.8369 - loss: 0.4573 - val_accuracy: 0.7428 - val_loss: 0.7648
Epoch 24/50
37/37              33s 844ms/step -
accuracy: 0.8169 - loss: 0.4527 - val_accuracy: 0.7564 - val_loss: 0.6826
Epoch 25/50
37/37              33s 848ms/step -
accuracy: 0.8407 - loss: 0.4017 - val_accuracy: 0.7496 - val_loss: 0.7367
Epoch 26/50
37/37              33s 844ms/step -
accuracy: 0.8473 - loss: 0.4236 - val_accuracy: 0.7683 - val_loss: 0.6776
Epoch 27/50
37/37              33s 860ms/step -
accuracy: 0.8247 - loss: 0.4690 - val_accuracy: 0.7428 - val_loss: 0.6986
Epoch 28/50
37/37              33s 843ms/step -
accuracy: 0.8573 - loss: 0.3942 - val_accuracy: 0.7462 - val_loss: 0.7653
Epoch 29/50
37/37              33s 856ms/step -
accuracy: 0.8370 - loss: 0.4271 - val_accuracy: 0.7649 - val_loss: 0.7018
Epoch 30/50
37/37              33s 848ms/step -
accuracy: 0.8508 - loss: 0.3872 - val_accuracy: 0.7530 - val_loss: 0.7745
Epoch 31/50
37/37              33s 850ms/step -
accuracy: 0.8422 - loss: 0.3898 - val_accuracy: 0.7683 - val_loss: 0.8007
Epoch 32/50
37/37              33s 845ms/step -
accuracy: 0.8620 - loss: 0.3551 - val_accuracy: 0.7581 - val_loss: 0.7035
Epoch 33/50
37/37              33s 854ms/step -
accuracy: 0.8539 - loss: 0.3845 - val_accuracy: 0.7683 - val_loss: 0.7195
Epoch 34/50
37/37              34s 865ms/step -
accuracy: 0.8651 - loss: 0.3465 - val_accuracy: 0.7479 - val_loss: 0.7472
Epoch 35/50
37/37              33s 843ms/step -
accuracy: 0.8768 - loss: 0.3352 - val_accuracy: 0.7973 - val_loss: 0.6937
Epoch 36/50
37/37              33s 847ms/step -
accuracy: 0.8806 - loss: 0.3184 - val_accuracy: 0.7700 - val_loss: 0.7901
Epoch 37/50
37/37              33s 844ms/step -
accuracy: 0.8749 - loss: 0.3233 - val_accuracy: 0.7871 - val_loss: 0.6626
Epoch 38/50
37/37              33s 858ms/step -
accuracy: 0.8815 - loss: 0.3238 - val_accuracy: 0.7853 - val_loss: 0.7137
Epoch 39/50
```

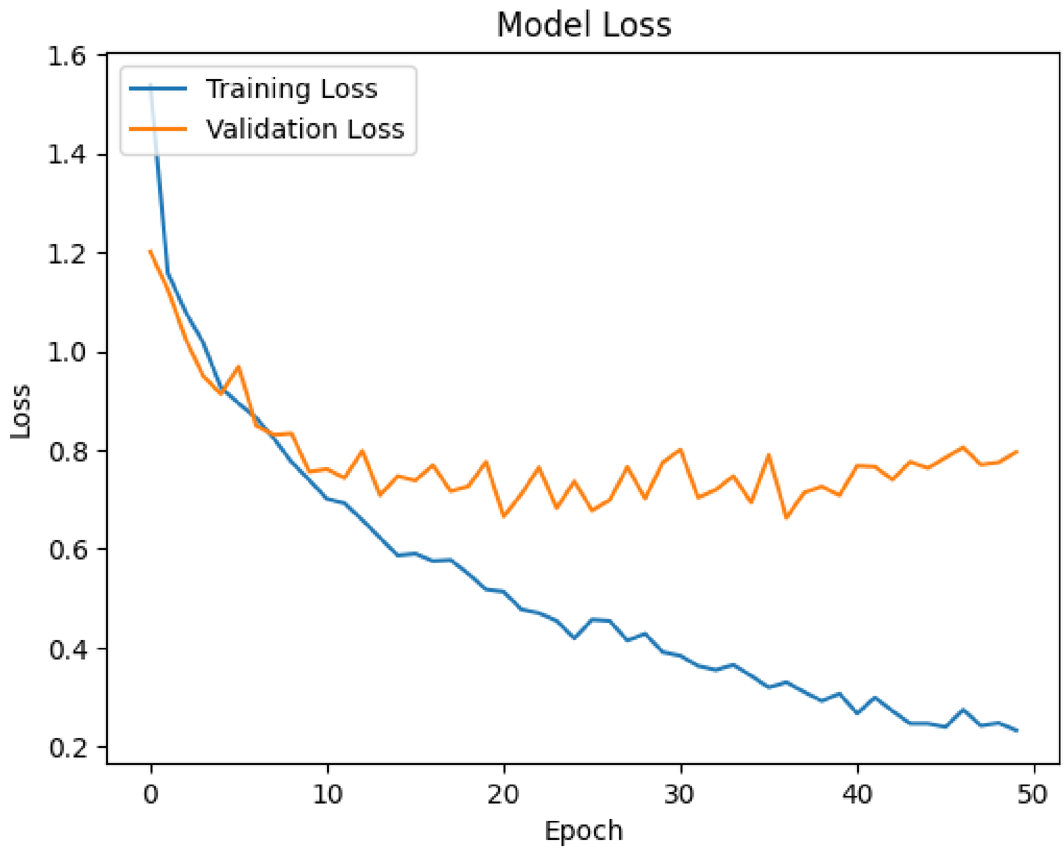
```
37/37          33s 854ms/step -
accuracy: 0.8922 - loss: 0.3033 - val_accuracy: 0.7785 - val_loss: 0.7258
Epoch 40/50
37/37          33s 849ms/step -
accuracy: 0.8867 - loss: 0.2944 - val_accuracy: 0.7922 - val_loss: 0.7083
Epoch 41/50
37/37          33s 854ms/step -
accuracy: 0.9037 - loss: 0.2579 - val_accuracy: 0.7717 - val_loss: 0.7678
Epoch 42/50
37/37          33s 852ms/step -
accuracy: 0.8826 - loss: 0.3201 - val_accuracy: 0.7649 - val_loss: 0.7659
Epoch 43/50
37/37          33s 848ms/step -
accuracy: 0.8967 - loss: 0.2732 - val_accuracy: 0.7700 - val_loss: 0.7399
Epoch 44/50
37/37          33s 843ms/step -
accuracy: 0.9096 - loss: 0.2494 - val_accuracy: 0.7819 - val_loss: 0.7758
Epoch 45/50
37/37          33s 860ms/step -
accuracy: 0.9046 - loss: 0.2477 - val_accuracy: 0.7871 - val_loss: 0.7631
Epoch 46/50
37/37          33s 845ms/step -
accuracy: 0.9153 - loss: 0.2334 - val_accuracy: 0.7700 - val_loss: 0.7843
Epoch 47/50
37/37          33s 865ms/step -
accuracy: 0.9100 - loss: 0.2571 - val_accuracy: 0.7785 - val_loss: 0.8053
Epoch 48/50
37/37          33s 847ms/step -
accuracy: 0.9020 - loss: 0.2554 - val_accuracy: 0.7905 - val_loss: 0.7700
Epoch 49/50
37/37          33s 862ms/step -
accuracy: 0.9168 - loss: 0.2318 - val_accuracy: 0.7871 - val_loss: 0.7747
Epoch 50/50
37/37          33s 853ms/step -
accuracy: 0.9089 - loss: 0.2354 - val_accuracy: 0.7581 - val_loss: 0.7957
```

[40]:

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_5.history['accuracy'], label='Training Accuracy')
plt.plot(history_5.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_5.png')
plt.show()
```

```
# Plot training & validation loss values
plt.plot(history_5.history['loss'], label='Training Loss')
plt.plot(history_5.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_5.png')
plt.show()
```





6 Model 6

6.1 Kernel Size (5,5), Dropout of 0.2, learning rate of 0.001, batch size 64 and epochs 30

[41]:
```
model_6 = Sequential()

# First Convolutional and pooling layer block
model_6.add(Conv2D(filters = 32, kernel_size = (5, 5), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_6.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_6.add(Conv2D(filters = 64, kernel_size = (5, 5), activation = 'relu'))
model_6.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_6.add(Conv2D(filters = 128, kernel_size = (5, 5), activation = 'relu'))
model_6.add(MaxPooling2D(pool_size = (2, 2)))
```

```
# Adding Flatten Layer
model_6.add(Flatten())
# Dense layer with 256 neurons
model_6.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 20%
model_6.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_6.add(Dense(5, activation = 'softmax'))

model_6.summary()
```

```
Model: "sequential_8"


 Layer (type)                          Output Shape                          ␣
↪Param #

 conv2d_24 (Conv2D)                    (None, 124, 124, 32)                    ␣
↪2,432

 max_pooling2d_24 (MaxPooling2D)       (None, 62, 62, 32)                        ␣
↪   0

 conv2d_25 (Conv2D)                    (None, 58, 58, 64)                      ␣
↪51,264

 max_pooling2d_25 (MaxPooling2D)       (None, 29, 29, 64)                        ␣
↪   0

 conv2d_26 (Conv2D)                    (None, 25, 25, 128)                   ␣
↪204,928

 max_pooling2d_26 (MaxPooling2D)       (None, 12, 12, 128)                       ␣
↪   0

 flatten_8 (Flatten)                   (None, 18432)                             ␣
↪   0

 dense_16 (Dense)                      (None, 256)                         ␣
↪4,718,848

 dropout_8 (Dropout)                   (None, 256)                               ␣
↪   0
```

```
 dense_17 (Dense)                      (None, 5)                                  ␣
↪1,285
```

Total params: 4,978,757 (18.99 MB)

Trainable params: 4,978,757 (18.99 MB)

Non-trainable params: 0 (0.00 B)

[42]:
```python
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
↪0.01

model_6.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

batch_size = 64 # Batch size set to 64
history_6 = model_6.fit(train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "training"),
                        epochs = 30, validation_data =
                        train_datagen.flow(x_train, y_train_cat,
                                           batch_size = batch_size,
                                           subset = "validation")) # Epochs set to␣
↪30
```

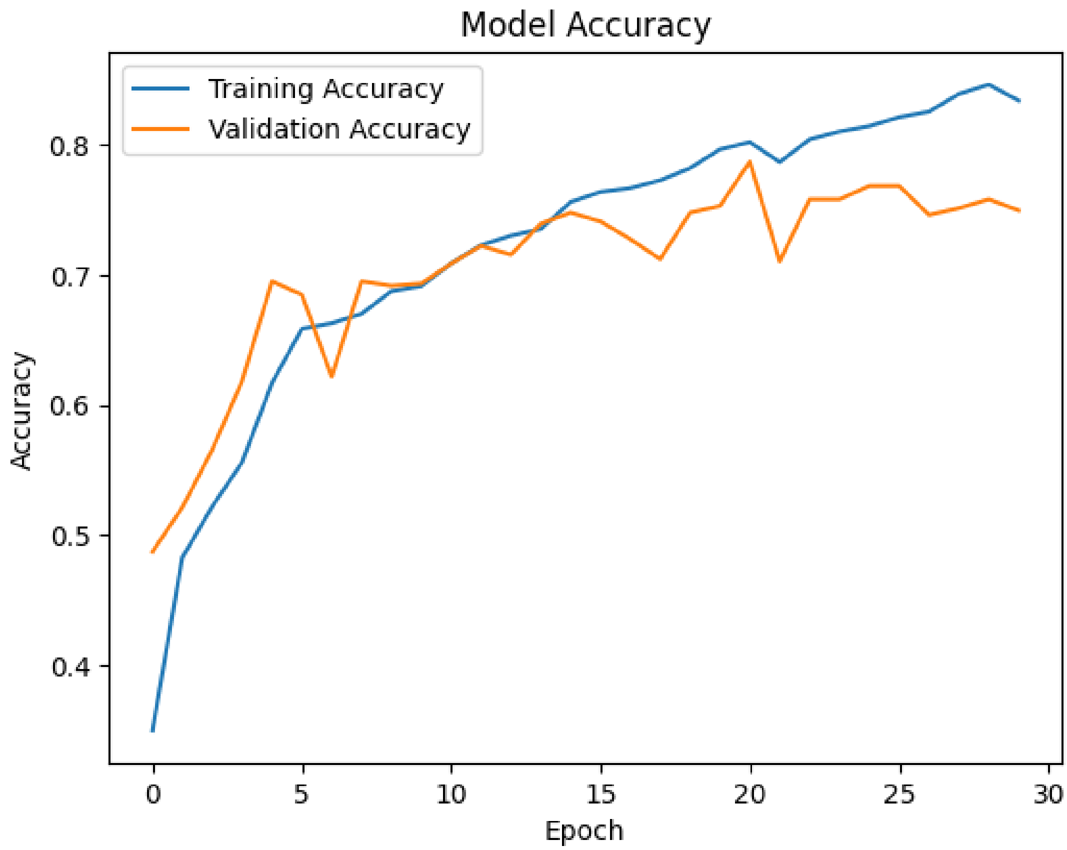
```
Epoch 1/30
37/37                 54s 1s/step -
accuracy: 0.2803 - loss: 1.6424 - val_accuracy: 0.4872 - val_loss: 1.2606
Epoch 2/30
37/37                 54s 1s/step -
accuracy: 0.4812 - loss: 1.1890 - val_accuracy: 0.5213 - val_loss: 1.1640
Epoch 3/30
37/37                 55s 1s/step -
accuracy: 0.5234 - loss: 1.1279 - val_accuracy: 0.5656 - val_loss: 1.1114
Epoch 4/30
37/37                 54s 1s/step -
accuracy: 0.5453 - loss: 1.1017 - val_accuracy: 0.6184 - val_loss: 1.0712
Epoch 5/30
37/37                 54s 1s/step -
accuracy: 0.6150 - loss: 0.9963 - val_accuracy: 0.6951 - val_loss: 0.9050
Epoch 6/30
37/37                 55s 1s/step -
accuracy: 0.6527 - loss: 0.9035 - val_accuracy: 0.6848 - val_loss: 0.8932
Epoch 7/30
```

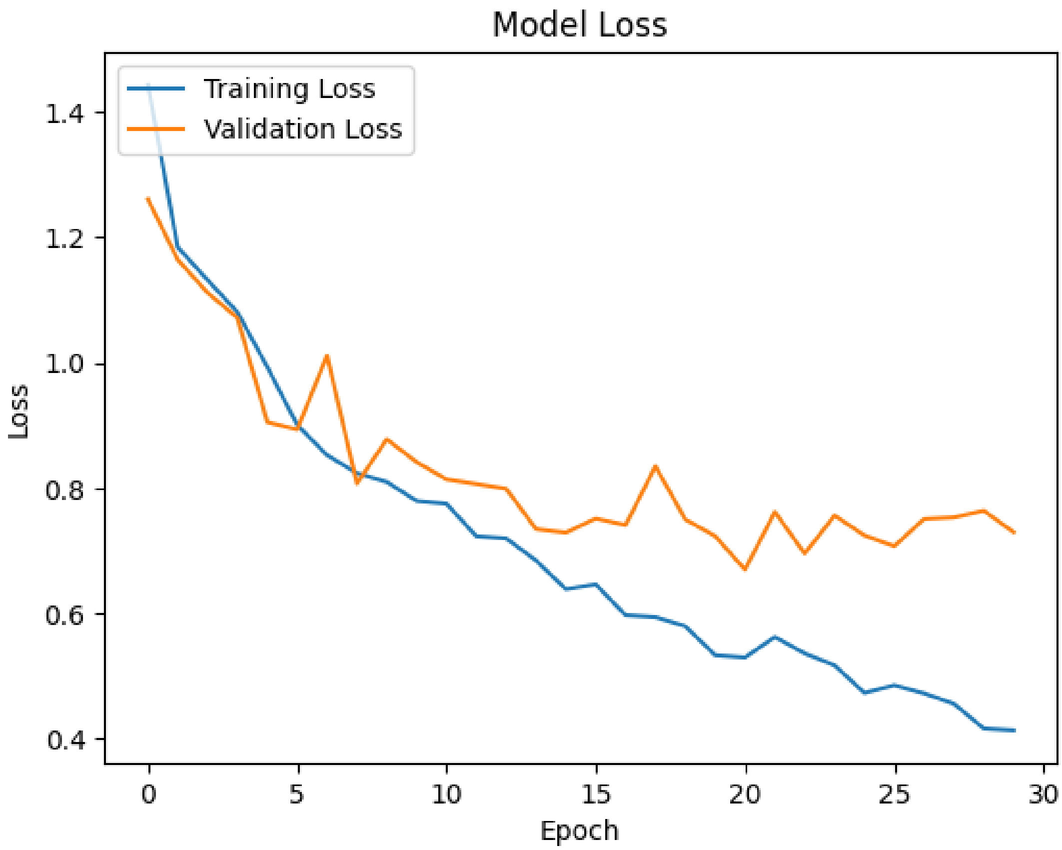
```
37/37              54s 1s/step -
accuracy: 0.6695 - loss: 0.8574 - val_accuracy: 0.6218 - val_loss: 1.0111
Epoch 8/30
37/37              54s 1s/step -
accuracy: 0.6582 - loss: 0.8303 - val_accuracy: 0.6951 - val_loss: 0.8069
Epoch 9/30
37/37              53s 1s/step -
accuracy: 0.6789 - loss: 0.8209 - val_accuracy: 0.6917 - val_loss: 0.8776
Epoch 10/30
37/37              54s 1s/step -
accuracy: 0.6897 - loss: 0.7704 - val_accuracy: 0.6934 - val_loss: 0.8412
Epoch 11/30
37/37              54s 1s/step -
accuracy: 0.7045 - loss: 0.7826 - val_accuracy: 0.7087 - val_loss: 0.8138
Epoch 12/30
37/37              54s 1s/step -
accuracy: 0.7169 - loss: 0.7155 - val_accuracy: 0.7223 - val_loss: 0.8063
Epoch 13/30
37/37              54s 1s/step -
accuracy: 0.7264 - loss: 0.7209 - val_accuracy: 0.7155 - val_loss: 0.7987
Epoch 14/30
37/37              54s 1s/step -
accuracy: 0.7353 - loss: 0.6765 - val_accuracy: 0.7394 - val_loss: 0.7344
Epoch 15/30
37/37              54s 1s/step -
accuracy: 0.7623 - loss: 0.6393 - val_accuracy: 0.7479 - val_loss: 0.7288
Epoch 16/30
37/37              54s 1s/step -
accuracy: 0.7677 - loss: 0.6321 - val_accuracy: 0.7411 - val_loss: 0.7511
Epoch 17/30
37/37              55s 1s/step -
accuracy: 0.7790 - loss: 0.5797 - val_accuracy: 0.7274 - val_loss: 0.7406
Epoch 18/30
37/37              54s 1s/step -
accuracy: 0.7742 - loss: 0.5808 - val_accuracy: 0.7121 - val_loss: 0.8346
Epoch 19/30
37/37              54s 1s/step -
accuracy: 0.7836 - loss: 0.5811 - val_accuracy: 0.7479 - val_loss: 0.7497
Epoch 20/30
37/37              59s 2s/step -
accuracy: 0.7974 - loss: 0.5369 - val_accuracy: 0.7530 - val_loss: 0.7232
Epoch 21/30
37/37              61s 2s/step -
accuracy: 0.8172 - loss: 0.5154 - val_accuracy: 0.7871 - val_loss: 0.6698
Epoch 22/30
37/37              58s 2s/step -
accuracy: 0.7972 - loss: 0.5437 - val_accuracy: 0.7104 - val_loss: 0.7617
Epoch 23/30
```

```
37/37                62s 2s/step -
accuracy: 0.8027 - loss: 0.5298 - val_accuracy: 0.7581 - val_loss: 0.6953
Epoch 24/30
37/37                60s 2s/step -
accuracy: 0.8145 - loss: 0.5076 - val_accuracy: 0.7581 - val_loss: 0.7560
Epoch 25/30
37/37                68s 2s/step -
accuracy: 0.8006 - loss: 0.4878 - val_accuracy: 0.7683 - val_loss: 0.7243
Epoch 26/30
37/37                55s 1s/step -
accuracy: 0.8320 - loss: 0.4440 - val_accuracy: 0.7683 - val_loss: 0.7072
Epoch 27/30
37/37                56s 1s/step -
accuracy: 0.8297 - loss: 0.4706 - val_accuracy: 0.7462 - val_loss: 0.7505
Epoch 28/30
37/37                54s 1s/step -
accuracy: 0.8454 - loss: 0.4321 - val_accuracy: 0.7513 - val_loss: 0.7530
Epoch 29/30
37/37                58s 2s/step -
accuracy: 0.8495 - loss: 0.4205 - val_accuracy: 0.7581 - val_loss: 0.7637
Epoch 30/30
37/37                55s 1s/step -
accuracy: 0.8355 - loss: 0.3898 - val_accuracy: 0.7496 - val_loss: 0.7293
```

[43]:

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_6.history['accuracy'], label='Training Accuracy')
plt.plot(history_6.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_6.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_6.history['loss'], label='Training Loss')
plt.plot(history_6.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_6.png')
plt.show()
```


Model Accuracy
Training Accuracy
Validation Accuracy
0.8
0.7
0.6
0.5
0.4
Accuracy
0
5
10
15
20
25
30
Epoch



7 Model 7

7.1 Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, batch size 64 and epochs 30

[44]:
```
model_7 = Sequential()

# First Convolutional and pooling layer block
model_7.add(Conv2D(filters = 32, kernel_size = (5, 5), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_7.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_7.add(Conv2D(filters = 64, kernel_size = (5, 5), activation = 'relu'))
model_7.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_7.add(Conv2D(filters = 128, kernel_size = (5, 5), activation = 'relu'))
model_7.add(MaxPooling2D(pool_size = (2, 2)))
```

```
# Adding Flatten Layer
model_7.add(Flatten())
# Dense layer with 256 neurons
model_7.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 40%
model_7.add(Dropout(0.4))
# Output dense layer with 5 neuron and softmax act function
model_7.add(Dense(5, activation = 'softmax'))

model_7.summary()
```

```
Model: "sequential_9"


 Layer (type)                         Output Shape                              ␣
↪Param #

 conv2d_27 (Conv2D)                   (None, 124, 124, 32)                        ␣
↪2,432

 max_pooling2d_27 (MaxPooling2D)      (None, 62, 62, 32)                            ␣
↪   0

 conv2d_28 (Conv2D)                   (None, 58, 58, 64)                          ␣
↪51,264

 max_pooling2d_28 (MaxPooling2D)      (None, 29, 29, 64)                            ␣
↪   0

 conv2d_29 (Conv2D)                   (None, 25, 25, 128)                       ␣
↪204,928

 max_pooling2d_29 (MaxPooling2D)      (None, 12, 12, 128)                           ␣
↪   0

 flatten_9 (Flatten)                  (None, 18432)                                 ␣
↪   0

 dense_18 (Dense)                     (None, 256)                             ␣
↪4,718,848

 dropout_9 (Dropout)                  (None, 256)                                   ␣
↪   0
```

```
 dense_19 (Dense)                    (None, 5)                              ␣
↪1,285
```

Total params: 4,978,757 (18.99 MB)

Trainable params: 4,978,757 (18.99 MB)

Non-trainable params: 0 (0.00 B)

[45]:
```
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
↪0.01

model_7.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
↪metrics=['accuracy'])

batch_size = 64 # Batch size set to 64
history_7 = model_7.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 30, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation")) # Epochs set to␣
↪30
```

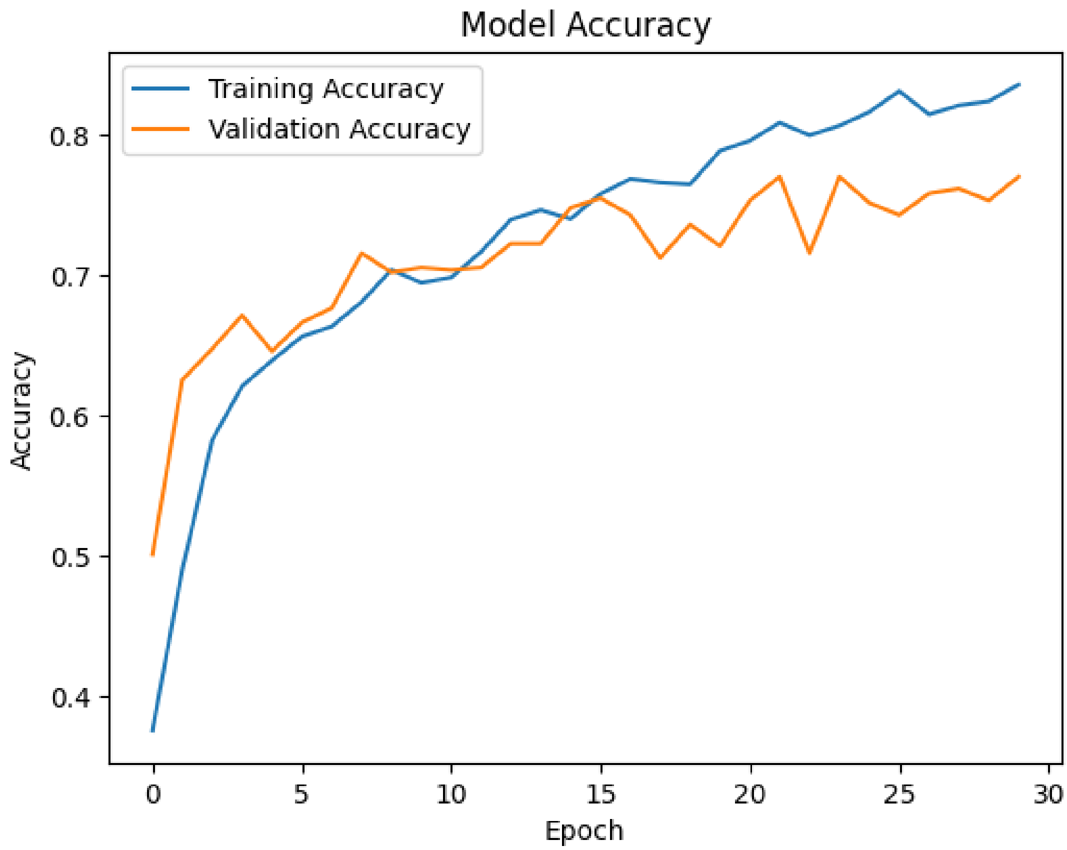
```
Epoch 1/30
37/37                 55s 1s/step -
accuracy: 0.3301 - loss: 1.5119 - val_accuracy: 0.5009 - val_loss: 1.2973
Epoch 2/30
37/37                 54s 1s/step -
accuracy: 0.4506 - loss: 1.2267 - val_accuracy: 0.6252 - val_loss: 1.0368
Epoch 3/30
37/37                 55s 1s/step -
accuracy: 0.5914 - loss: 1.0193 - val_accuracy: 0.6474 - val_loss: 0.9502
Epoch 4/30
37/37                 73s 2s/step -
accuracy: 0.6112 - loss: 1.0038 - val_accuracy: 0.6712 - val_loss: 0.9101
Epoch 5/30
37/37                 88s 2s/step -
accuracy: 0.6422 - loss: 0.9316 - val_accuracy: 0.6457 - val_loss: 0.9362
Epoch 6/30
37/37                 88s 2s/step -
accuracy: 0.6445 - loss: 0.9050 - val_accuracy: 0.6661 - val_loss: 0.8788
Epoch 7/30
```

```
37/37            87s 2s/step -
accuracy: 0.6625 - loss: 0.8604 - val_accuracy: 0.6763 - val_loss: 0.8287
Epoch 8/30
37/37            84s 2s/step -
accuracy: 0.6799 - loss: 0.8262 - val_accuracy: 0.7155 - val_loss: 0.7934
Epoch 9/30
37/37            88s 2s/step -
accuracy: 0.7062 - loss: 0.7854 - val_accuracy: 0.7019 - val_loss: 0.7743
Epoch 10/30
37/37            91s 2s/step -
accuracy: 0.7069 - loss: 0.7502 - val_accuracy: 0.7053 - val_loss: 0.7929
Epoch 11/30
37/37            86s 2s/step -
accuracy: 0.6937 - loss: 0.7756 - val_accuracy: 0.7036 - val_loss: 0.7821
Epoch 12/30
37/37            87s 2s/step -
accuracy: 0.7158 - loss: 0.7373 - val_accuracy: 0.7053 - val_loss: 0.8408
Epoch 13/30
37/37            87s 2s/step -
accuracy: 0.7335 - loss: 0.7031 - val_accuracy: 0.7223 - val_loss: 0.7251
Epoch 14/30
37/37            87s 2s/step -
accuracy: 0.7422 - loss: 0.6635 - val_accuracy: 0.7223 - val_loss: 0.8146
Epoch 15/30
37/37            87s 2s/step -
accuracy: 0.7377 - loss: 0.6365 - val_accuracy: 0.7479 - val_loss: 0.7204
Epoch 16/30
37/37            90s 2s/step -
accuracy: 0.7656 - loss: 0.5979 - val_accuracy: 0.7547 - val_loss: 0.7119
Epoch 17/30
37/37            86s 2s/step -
accuracy: 0.7800 - loss: 0.5731 - val_accuracy: 0.7428 - val_loss: 0.7157
Epoch 18/30
37/37            89s 2s/step -
accuracy: 0.7576 - loss: 0.6288 - val_accuracy: 0.7121 - val_loss: 0.7763
Epoch 19/30
37/37            89s 2s/step -
accuracy: 0.7702 - loss: 0.5718 - val_accuracy: 0.7359 - val_loss: 0.8314
Epoch 20/30
37/37            85s 2s/step -
accuracy: 0.7920 - loss: 0.5629 - val_accuracy: 0.7206 - val_loss: 0.8118
Epoch 21/30
37/37            87s 2s/step -
accuracy: 0.8012 - loss: 0.5151 - val_accuracy: 0.7530 - val_loss: 0.7317
Epoch 22/30
37/37            88s 2s/step -
accuracy: 0.8101 - loss: 0.5217 - val_accuracy: 0.7700 - val_loss: 0.7217
Epoch 23/30
```

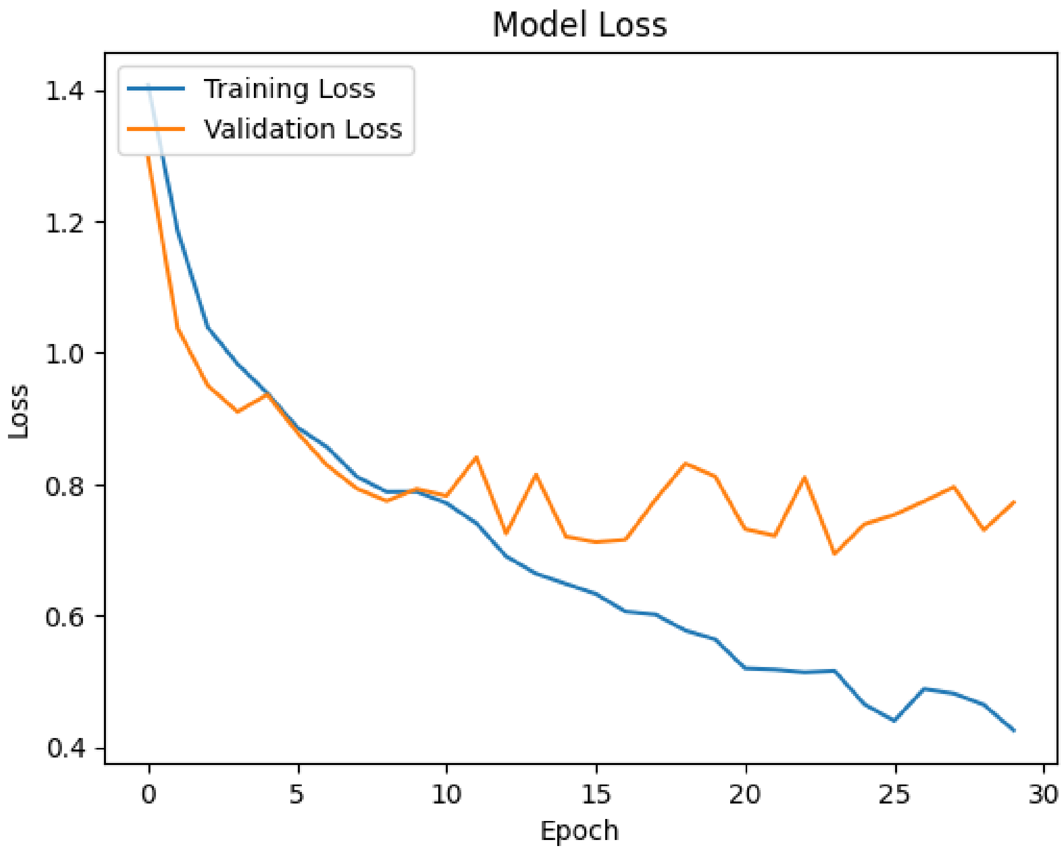
```
37/37             87s 2s/step -
accuracy: 0.7939 - loss: 0.5159 - val_accuracy: 0.7155 - val_loss: 0.8106
Epoch 24/30
37/37             82s 2s/step -
accuracy: 0.7986 - loss: 0.5285 - val_accuracy: 0.7700 - val_loss: 0.6939
Epoch 25/30
37/37             88s 2s/step -
accuracy: 0.8125 - loss: 0.4619 - val_accuracy: 0.7513 - val_loss: 0.7392
Epoch 26/30
37/37             87s 2s/step -
accuracy: 0.8295 - loss: 0.4532 - val_accuracy: 0.7428 - val_loss: 0.7535
Epoch 27/30
37/37             85s 2s/step -
accuracy: 0.8092 - loss: 0.4999 - val_accuracy: 0.7581 - val_loss: 0.7740
Epoch 28/30
37/37             84s 2s/step -
accuracy: 0.8183 - loss: 0.4912 - val_accuracy: 0.7615 - val_loss: 0.7958
Epoch 29/30
37/37             86s 2s/step -
accuracy: 0.8243 - loss: 0.4542 - val_accuracy: 0.7530 - val_loss: 0.7303
Epoch 30/30
37/37             86s 2s/step -
accuracy: 0.8379 - loss: 0.4383 - val_accuracy: 0.7700 - val_loss: 0.7722
```

[46]:

```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_7.history['accuracy'], label='Training Accuracy')
plt.plot(history_7.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_7.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_7.history['loss'], label='Training Loss')
plt.plot(history_7.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_7.png')
plt.show()
```



Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30
Epoch



8 Model 8

8.1 Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, batch size 128 and epochs 30

[47]:
```
model_8 = Sequential()

# First Convolutional and pooling layer block
model_8.add(Conv2D(filters = 32, kernel_size = (5, 5), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_8.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_8.add(Conv2D(filters = 64, kernel_size = (5, 5), activation = 'relu'))
model_8.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_8.add(Conv2D(filters = 128, kernel_size = (5, 5), activation = 'relu'))
model_8.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding Flatten Layer
model_8.add(Flatten())
# Dense layer with 256 neurons
model_8.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 40%
model_8.add(Dropout(0.4))
# Output dense layer with 5 neuron and softmax act function
model_8.add(Dense(5, activation = 'softmax'))

model_8.summary()
```

```
Model: "sequential_10"


 Layer (type)                          Output Shape                          ␣
↪Param #

 conv2d_30 (Conv2D)                    (None, 124, 124, 32)                   ␣
↪2,432

 max_pooling2d_30 (MaxPooling2D)       (None, 62, 62, 32)                       ␣
↪   0

 conv2d_31 (Conv2D)                    (None, 58, 58, 64)                     ␣
↪51,264

 max_pooling2d_31 (MaxPooling2D)       (None, 29, 29, 64)                       ␣
↪   0

 conv2d_32 (Conv2D)                    (None, 25, 25, 128)                   ␣
↪204,928

 max_pooling2d_32 (MaxPooling2D)       (None, 12, 12, 128)                      ␣
↪   0

 flatten_10 (Flatten)                  (None, 18432)                            ␣
↪   0

 dense_20 (Dense)                      (None, 256)                          ␣
↪4,718,848

 dropout_10 (Dropout)                  (None, 256)                              ␣
↪   0
```

```
 dense_21 (Dense)                          (None, 5)                        ␣
 ↪1,285
```

```
Total params: 4,978,757 (18.99 MB)

Trainable params: 4,978,757 (18.99 MB)

Non-trainable params: 0 (0.00 B)
```

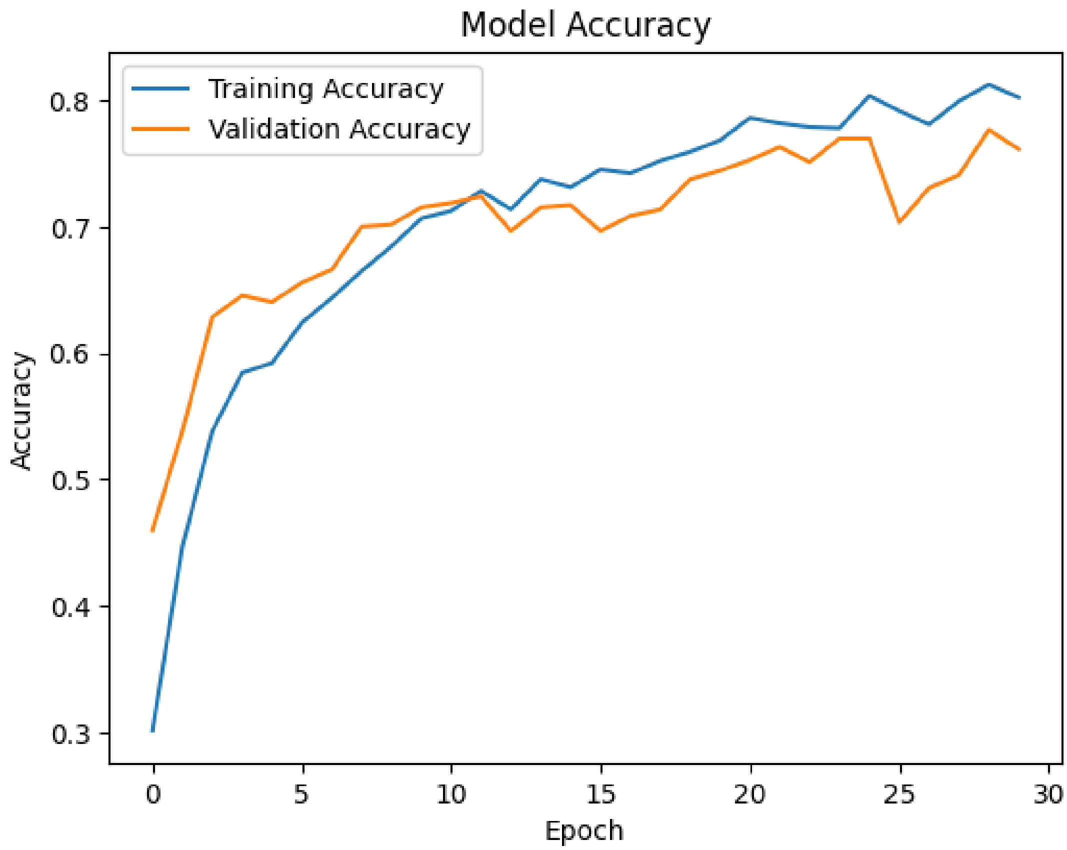
[48]:
```python
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
 ↪0.01

model_8.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

batch_size = 128 # Batch size set to 128
history_8 = model_8.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 30, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation")) # Epochs set to␣
 ↪30
```

```
Epoch 1/30
19/19                 65s 3s/step -
accuracy: 0.2526 - loss: 1.6539 - val_accuracy: 0.4600 - val_loss: 1.2815
Epoch 2/30
19/19                 54s 3s/step -
accuracy: 0.4175 - loss: 1.2859 - val_accuracy: 0.5383 - val_loss: 1.1667
Epoch 3/30
19/19                 53s 3s/step -
accuracy: 0.5300 - loss: 1.1383 - val_accuracy: 0.6286 - val_loss: 1.0313
Epoch 4/30
19/19                 54s 3s/step -
accuracy: 0.5784 - loss: 1.0606 - val_accuracy: 0.6457 - val_loss: 1.0439
Epoch 5/30
19/19                 53s 3s/step -
accuracy: 0.6138 - loss: 1.0179 - val_accuracy: 0.6405 - val_loss: 1.0103
Epoch 6/30
19/19                 55s 3s/step -
accuracy: 0.6340 - loss: 0.9923 - val_accuracy: 0.6559 - val_loss: 0.9722
Epoch 7/30
```

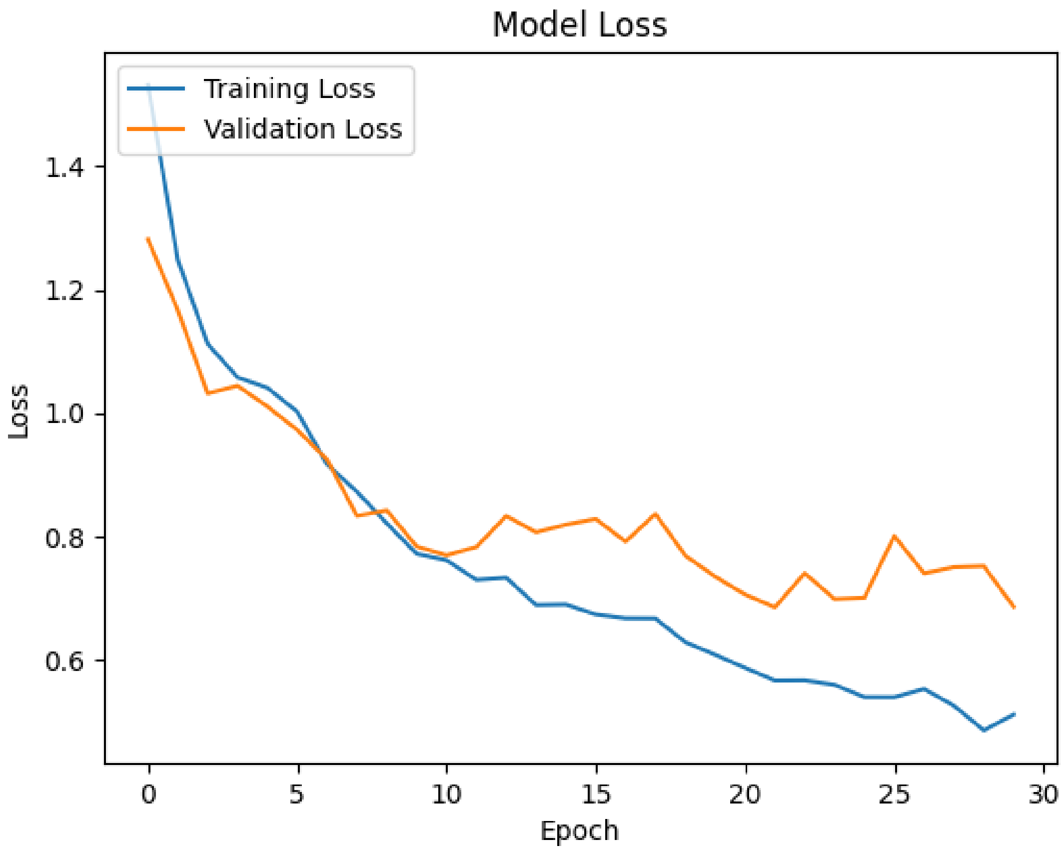
```
19/19              53s 3s/step -
accuracy: 0.6459 - loss: 0.9337 - val_accuracy: 0.6661 - val_loss: 0.9241
Epoch 8/30
19/19              54s 3s/step -
accuracy: 0.6710 - loss: 0.8689 - val_accuracy: 0.7002 - val_loss: 0.8327
Epoch 9/30
19/19              53s 3s/step -
accuracy: 0.6930 - loss: 0.8154 - val_accuracy: 0.7019 - val_loss: 0.8417
Epoch 10/30
19/19              54s 3s/step -
accuracy: 0.7080 - loss: 0.7674 - val_accuracy: 0.7155 - val_loss: 0.7831
Epoch 11/30
19/19              53s 3s/step -
accuracy: 0.7228 - loss: 0.7537 - val_accuracy: 0.7189 - val_loss: 0.7688
Epoch 12/30
19/19              53s 3s/step -
accuracy: 0.7320 - loss: 0.7184 - val_accuracy: 0.7240 - val_loss: 0.7823
Epoch 13/30
19/19              54s 3s/step -
accuracy: 0.7137 - loss: 0.7387 - val_accuracy: 0.6968 - val_loss: 0.8325
Epoch 14/30
19/19              53s 3s/step -
accuracy: 0.7387 - loss: 0.6864 - val_accuracy: 0.7155 - val_loss: 0.8063
Epoch 15/30
19/19              54s 3s/step -
accuracy: 0.7168 - loss: 0.7279 - val_accuracy: 0.7172 - val_loss: 0.8184
Epoch 16/30
19/19              53s 3s/step -
accuracy: 0.7510 - loss: 0.6656 - val_accuracy: 0.6968 - val_loss: 0.8276
Epoch 17/30
19/19              54s 3s/step -
accuracy: 0.7422 - loss: 0.6623 - val_accuracy: 0.7087 - val_loss: 0.7911
Epoch 18/30
19/19              53s 3s/step -
accuracy: 0.7620 - loss: 0.6226 - val_accuracy: 0.7138 - val_loss: 0.8358
Epoch 19/30
19/19              54s 3s/step -
accuracy: 0.7707 - loss: 0.6182 - val_accuracy: 0.7376 - val_loss: 0.7683
Epoch 20/30
19/19              53s 3s/step -
accuracy: 0.7789 - loss: 0.5734 - val_accuracy: 0.7445 - val_loss: 0.7342
Epoch 21/30
19/19              53s 3s/step -
accuracy: 0.7994 - loss: 0.5733 - val_accuracy: 0.7530 - val_loss: 0.7052
Epoch 22/30
19/19              54s 3s/step -
accuracy: 0.7847 - loss: 0.5617 - val_accuracy: 0.7632 - val_loss: 0.6845
Epoch 23/30
```

```
19/19                54s 3s/step -
accuracy: 0.7792 - loss: 0.5759 - val_accuracy: 0.7513 - val_loss: 0.7398
Epoch 24/30
19/19                55s 3s/step -
accuracy: 0.7820 - loss: 0.5371 - val_accuracy: 0.7700 - val_loss: 0.6978
Epoch 25/30
19/19                53s 3s/step -
accuracy: 0.8080 - loss: 0.5350 - val_accuracy: 0.7700 - val_loss: 0.7000
Epoch 26/30
19/19                54s 3s/step -
accuracy: 0.8204 - loss: 0.4827 - val_accuracy: 0.7036 - val_loss: 0.8002
Epoch 27/30
19/19                53s 3s/step -
accuracy: 0.7831 - loss: 0.5553 - val_accuracy: 0.7308 - val_loss: 0.7395
Epoch 28/30
19/19                54s 3s/step -
accuracy: 0.7889 - loss: 0.5084 - val_accuracy: 0.7411 - val_loss: 0.7497
Epoch 29/30
19/19                53s 3s/step -
accuracy: 0.8080 - loss: 0.5072 - val_accuracy: 0.7768 - val_loss: 0.7512
Epoch 30/30
19/19                53s 3s/step -
accuracy: 0.8136 - loss: 0.4712 - val_accuracy: 0.7615 - val_loss: 0.6852
```

[49]:
```
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_8.history['accuracy'], label='Training Accuracy')
plt.plot(history_8.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_8.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_8.history['loss'], label='Training Loss')
plt.plot(history_8.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_8.png')
plt.show()
```



Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30



9 Model 9

9.1 Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, batch size 128 and epochs 30

[50]:
```
model_9 = Sequential()

# First Convolutional and pooling layer block
model_9.add(Conv2D(filters = 32, kernel_size = (3, 3), input_shape = (128, 128,␣
↪3), activation = 'relu'))
model_9.add(MaxPooling2D(pool_size = (2, 2)))

# Second Convolutional and pooling layer block
model_9.add(Conv2D(filters = 64, kernel_size = (3, 3), activation = 'relu'))
model_9.add(MaxPooling2D(pool_size = (2, 2)))

# Third Convolutional and pooling layer block
model_9.add(Conv2D(filters = 128, kernel_size = (3, 3), activation = 'relu'))
model_9.add(MaxPooling2D(pool_size = (2, 2)))
```

```python
# Adding Flatten Layer
model_9.add(Flatten())
# Dense layer with 256 neurons
model_9.add(Dense(256, activation = 'relu'))
# Dropout rate on Dense layer of 40%
model_9.add(Dropout(0.2))
# Output dense layer with 5 neuron and softmax act function
model_9.add(Dense(5, activation = 'softmax'))

model_9.summary()
```

```
Model: "sequential_11"


 Layer (type)                           Output Shape                              ␣
 ↪Param #

 conv2d_33 (Conv2D)                     (None, 126, 126, 32)                        ␣
 ↪896

 max_pooling2d_33 (MaxPooling2D)        (None, 63, 63, 32)                          ␣
 ↪   0

 conv2d_34 (Conv2D)                     (None, 61, 61, 64)                        ␣
 ↪18,496

 max_pooling2d_34 (MaxPooling2D)        (None, 30, 30, 64)                          ␣
 ↪   0

 conv2d_35 (Conv2D)                     (None, 28, 28, 128)                       ␣
 ↪73,856

 max_pooling2d_35 (MaxPooling2D)        (None, 14, 14, 128)                         ␣
 ↪   0

 flatten_11 (Flatten)                   (None, 25088)                               ␣
 ↪   0

 dense_22 (Dense)                       (None, 256)                             ␣
 ↪6,422,784

 dropout_11 (Dropout)                   (None, 256)                                 ␣
 ↪   0
```

```
 dense_23 (Dense)                      (None, 5)                          ␣
↪1,285
```

Total params: 6,517,317 (24.86 MB)

Trainable params: 6,517,317 (24.86 MB)

Non-trainable params: 0 (0.00 B)

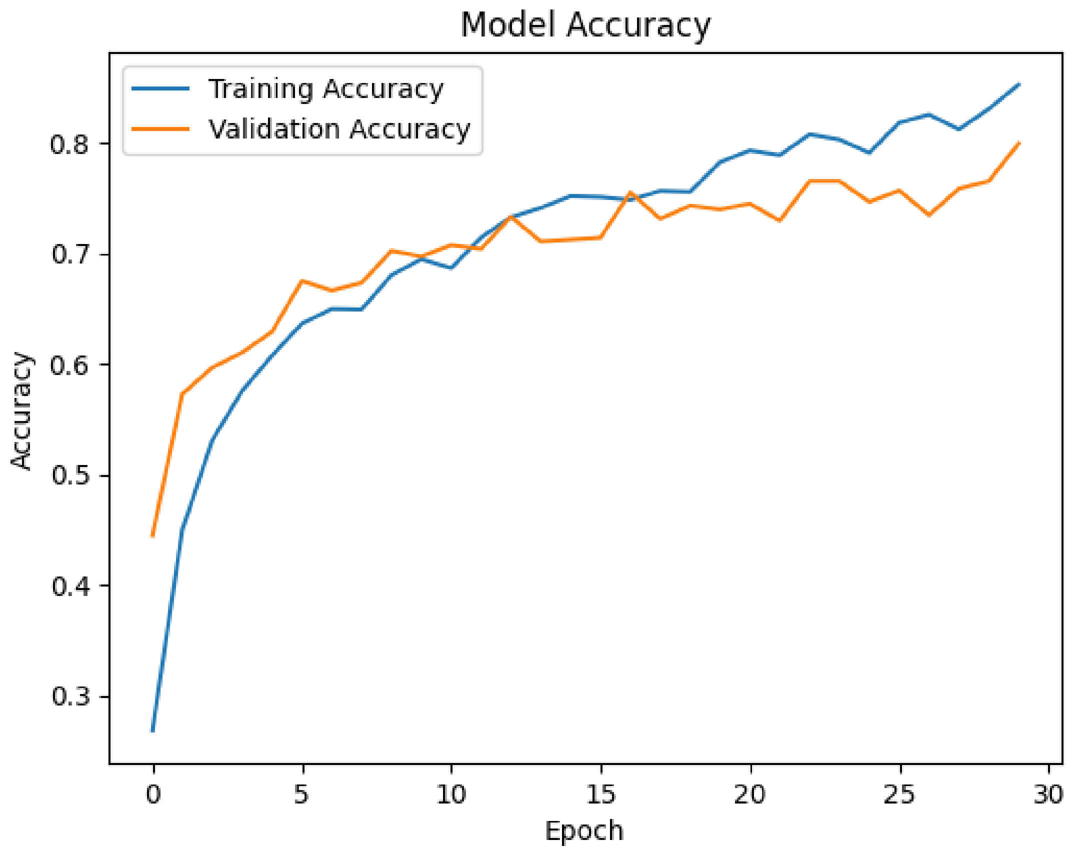
[51]:
```
adam_optimizer = Adam(learning_rate = 0.001) # Learning rate set to default of␣
 ↪0.01

model_9.compile(optimizer=adam_optimizer, loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

batch_size = 128 # Batch size set to 128
history_9 = model_9.fit(train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "training"),
                    epochs = 30, validation_data =
                    train_datagen.flow(x_train, y_train_cat,
                                       batch_size = batch_size,
                                       subset = "validation")) # Epochs set to␣
 ↪30
```

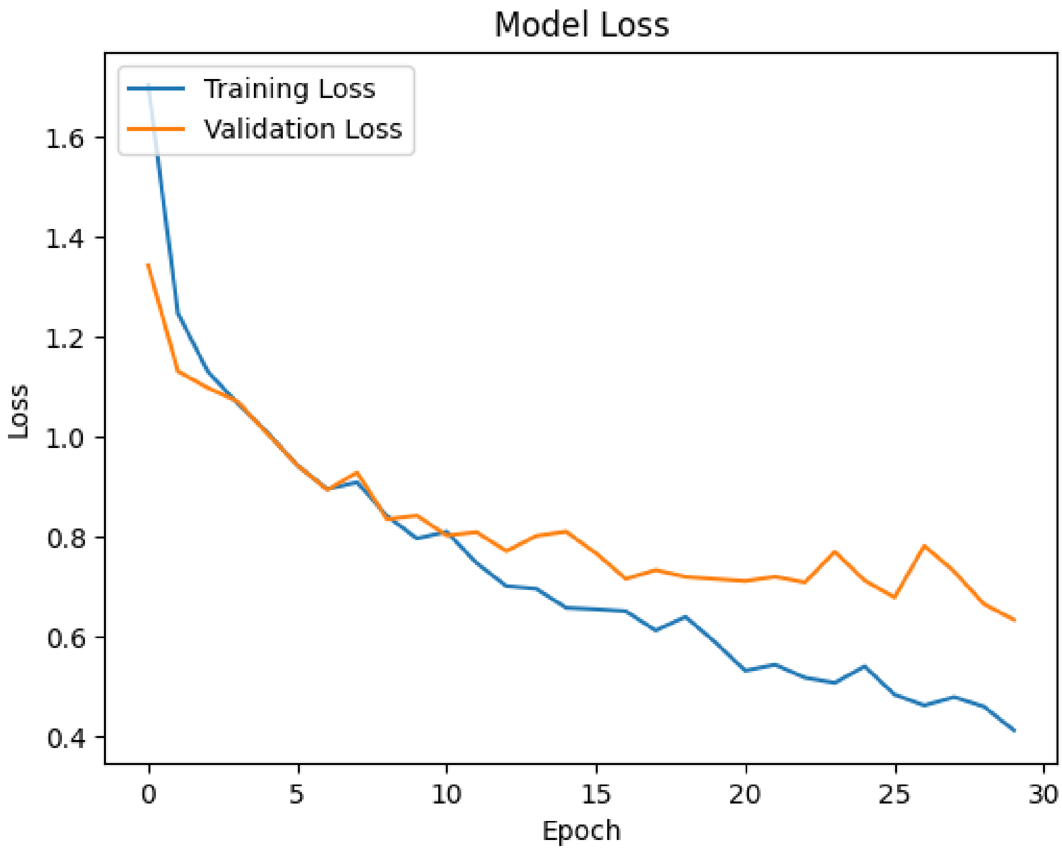
```
Epoch 1/30
19/19                 101s 3s/step -
accuracy: 0.2502 - loss: 1.9154 - val_accuracy: 0.4446 - val_loss: 1.3441
Epoch 2/30
19/19                 57s 3s/step -
accuracy: 0.4224 - loss: 1.3033 - val_accuracy: 0.5724 - val_loss: 1.1315
Epoch 3/30
19/19                 53s 2s/step -
accuracy: 0.5144 - loss: 1.1454 - val_accuracy: 0.5963 - val_loss: 1.0988
Epoch 4/30
19/19                 56s 3s/step -
accuracy: 0.5608 - loss: 1.0760 - val_accuracy: 0.6099 - val_loss: 1.0715
Epoch 5/30
19/19                 55s 2s/step -
accuracy: 0.6065 - loss: 1.0086 - val_accuracy: 0.6286 - val_loss: 1.0064
Epoch 6/30
19/19                 54s 2s/step -
accuracy: 0.6250 - loss: 0.9625 - val_accuracy: 0.6746 - val_loss: 0.9438
Epoch 7/30
```

```
19/19              54s 2s/step -
accuracy: 0.6470 - loss: 0.9010 - val_accuracy: 0.6661 - val_loss: 0.8946
Epoch 8/30
19/19              54s 2s/step -
accuracy: 0.6362 - loss: 0.9297 - val_accuracy: 0.6729 - val_loss: 0.9294
Epoch 9/30
19/19              51s 2s/step -
accuracy: 0.6651 - loss: 0.8680 - val_accuracy: 0.7019 - val_loss: 0.8356
Epoch 10/30
19/19              33s 2s/step -
accuracy: 0.7035 - loss: 0.7740 - val_accuracy: 0.6968 - val_loss: 0.8430
Epoch 11/30
19/19              34s 2s/step -
accuracy: 0.6906 - loss: 0.7975 - val_accuracy: 0.7070 - val_loss: 0.8033
Epoch 12/30
19/19              33s 2s/step -
accuracy: 0.7059 - loss: 0.7811 - val_accuracy: 0.7036 - val_loss: 0.8099
Epoch 13/30
19/19              34s 2s/step -
accuracy: 0.7309 - loss: 0.7042 - val_accuracy: 0.7325 - val_loss: 0.7721
Epoch 14/30
19/19              34s 2s/step -
accuracy: 0.7449 - loss: 0.6750 - val_accuracy: 0.7104 - val_loss: 0.8024
Epoch 15/30
19/19              34s 2s/step -
accuracy: 0.7552 - loss: 0.6504 - val_accuracy: 0.7121 - val_loss: 0.8109
Epoch 16/30
19/19              34s 2s/step -
accuracy: 0.7521 - loss: 0.6258 - val_accuracy: 0.7138 - val_loss: 0.7678
Epoch 17/30
19/19              35s 2s/step -
accuracy: 0.7446 - loss: 0.6551 - val_accuracy: 0.7547 - val_loss: 0.7164
Epoch 18/30
19/19              34s 2s/step -
accuracy: 0.7674 - loss: 0.6022 - val_accuracy: 0.7308 - val_loss: 0.7339
Epoch 19/30
19/19              34s 2s/step -
accuracy: 0.7495 - loss: 0.6568 - val_accuracy: 0.7428 - val_loss: 0.7205
Epoch 20/30
19/19              34s 2s/step -
accuracy: 0.7770 - loss: 0.5982 - val_accuracy: 0.7394 - val_loss: 0.7162
Epoch 21/30
19/19              34s 2s/step -
accuracy: 0.7858 - loss: 0.5387 - val_accuracy: 0.7445 - val_loss: 0.7121
Epoch 22/30
19/19              34s 2s/step -
accuracy: 0.7973 - loss: 0.5199 - val_accuracy: 0.7291 - val_loss: 0.7207
Epoch 23/30
```

```
19/19              34s 2s/step -
accuracy: 0.8091 - loss: 0.5067 - val_accuracy: 0.7649 - val_loss: 0.7093
Epoch 24/30
19/19              34s 2s/step -
accuracy: 0.8047 - loss: 0.5040 - val_accuracy: 0.7649 - val_loss: 0.7706
Epoch 25/30
19/19              34s 2s/step -
accuracy: 0.7924 - loss: 0.5472 - val_accuracy: 0.7462 - val_loss: 0.7131
Epoch 26/30
19/19              34s 2s/step -
accuracy: 0.8256 - loss: 0.4745 - val_accuracy: 0.7564 - val_loss: 0.6794
Epoch 27/30
19/19              34s 2s/step -
accuracy: 0.8222 - loss: 0.4617 - val_accuracy: 0.7342 - val_loss: 0.7826
Epoch 28/30
19/19              33s 2s/step -
accuracy: 0.8082 - loss: 0.4815 - val_accuracy: 0.7581 - val_loss: 0.7311
Epoch 29/30
19/19              34s 2s/step -
accuracy: 0.8456 - loss: 0.4420 - val_accuracy: 0.7649 - val_loss: 0.6658
Epoch 30/30
19/19              34s 2s/step -
accuracy: 0.8494 - loss: 0.4135 - val_accuracy: 0.7990 - val_loss: 0.6346
```

[52]:
```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.plot(history_9.history['accuracy'], label='Training Accuracy')
plt.plot(history_9.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Accuracy_model_9.png')
plt.show()

# Plot training & validation loss values
plt.plot(history_9.history['loss'], label='Training Loss')
plt.plot(history_9.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.savefig('Loss_model_9.png')
plt.show()
```

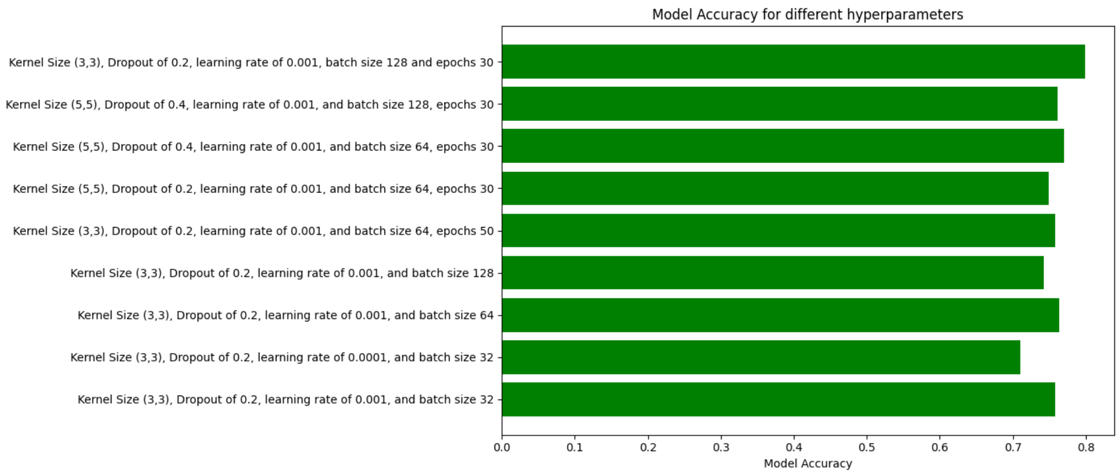

Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epoch
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30



[56]:
```
import pandas as pd
import matplotlib.pyplot as plt

results = {
    'Model': ["Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and␣
↪batch size 32",
"Kernel Size (3,3), Dropout of 0.2, learning rate of 0.0001, and batch size 32",
"Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 64",
"Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 128",
"Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 64,␣
↪epochs 50",
"Kernel Size (5,5), Dropout of 0.2, learning rate of 0.001, and batch size 64,␣
↪epochs 30",
"Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, and batch size 64,␣
↪epochs 30",
"Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, and batch size 128,␣
↪epochs 30",
"Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, batch size 128 and␣
↪epochs 30"
```

```
            ],
    'Model_Accuracy': [0.7581,0.7104,0.7632,0.7428,0.7581,0.7496,0.7700,0.
 ↪7615,0.7990],
}

# Create a DataFrame to display the results
results_df = pd.DataFrame(results)

# Display the DataFrame
print("Model Performance Comparison:")
print(results_df)

# Plot the results
plt.figure(figsize=(14, 6))

plt.barh(results_df['Model'], results_df['Model_Accuracy'], color='green')
plt.xlabel('Model Accuracy')
plt.title('Model Accuracy for different hyperparameters')

plt.tight_layout()

plt.savefig('Model_accuracy_and_loss.png')
plt.show()
```

```
Model Performance Comparison:
                                               Model  Model_Accuracy
0  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7581
1  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7104
2  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7632
3  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7428
4  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7581
5  Kernel Size (5,5), Dropout of 0.2, learning ra…          0.7496
6  Kernel Size (5,5), Dropout of 0.4, learning ra…          0.7700
7  Kernel Size (5,5), Dropout of 0.4, learning ra…          0.7615
8  Kernel Size (3,3), Dropout of 0.2, learning ra…          0.7990
```



Model Accuracy for different hyperparameters
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, batch size 128 and epochs 30
Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, and batch size 128, epochs 30
Kernel Size (5,5), Dropout of 0.4, learning rate of 0.001, and batch size 64, epochs 30
Kernel Size (5,5), Dropout of 0.2, learning rate of 0.001, and batch size 64, epochs 30
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 64, epochs 50
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 128
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 64
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.0001, and batch size 32
Kernel Size (3,3), Dropout of 0.2, learning rate of 0.001, and batch size 32
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
Model Accuracy

[ ]: