SkinCancerFinal

May 5, 2023

[1]:
```
# imports
import pandas as pd
import IPython.display as display
import matplotlib.pyplot as plt
import numpy as np
import time
from google.colab import files
import os
import tensorflow as tf
import os
import zipfile
from tensorflow.keras import layers
from tensorflow.keras import Model
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from tensorflow.keras.optimizers import RMSprop
import random
from tensorflow.keras.preprocessing.image import img_to_array, load_img
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array,␣
 ↪load_img
```

[2]:
```
uploades = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving SkinCancer.zip to SkinCancer.zip
```

[3]:
```
local_zip = 'SkinCancer.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall()
zip_ref.close()
```

1 Setup

[4]:
```
base_dir = 'SkinCancer'
train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat pictures
train_malignant_dir = os.path.join(train_dir, 'malignant')

# Directory with our training dog pictures
train_benign_dir = os.path.join(train_dir, 'benign')

# Directory with our validation cat pictures
validation_malignant_dir = os.path.join(validation_dir, 'malignant')

# Directory with our validation dog pictures
validation_benign_dir = os.path.join(validation_dir, 'benign')
```

[5]:
```
train_malignant_fnames = os.listdir(train_malignant_dir)
train_malignant_fnames.sort()
print(train_malignant_fnames[:10])

train_benign_fnames = os.listdir(train_benign_dir)
train_benign_fnames.sort()
print(train_benign_fnames[:10])
```

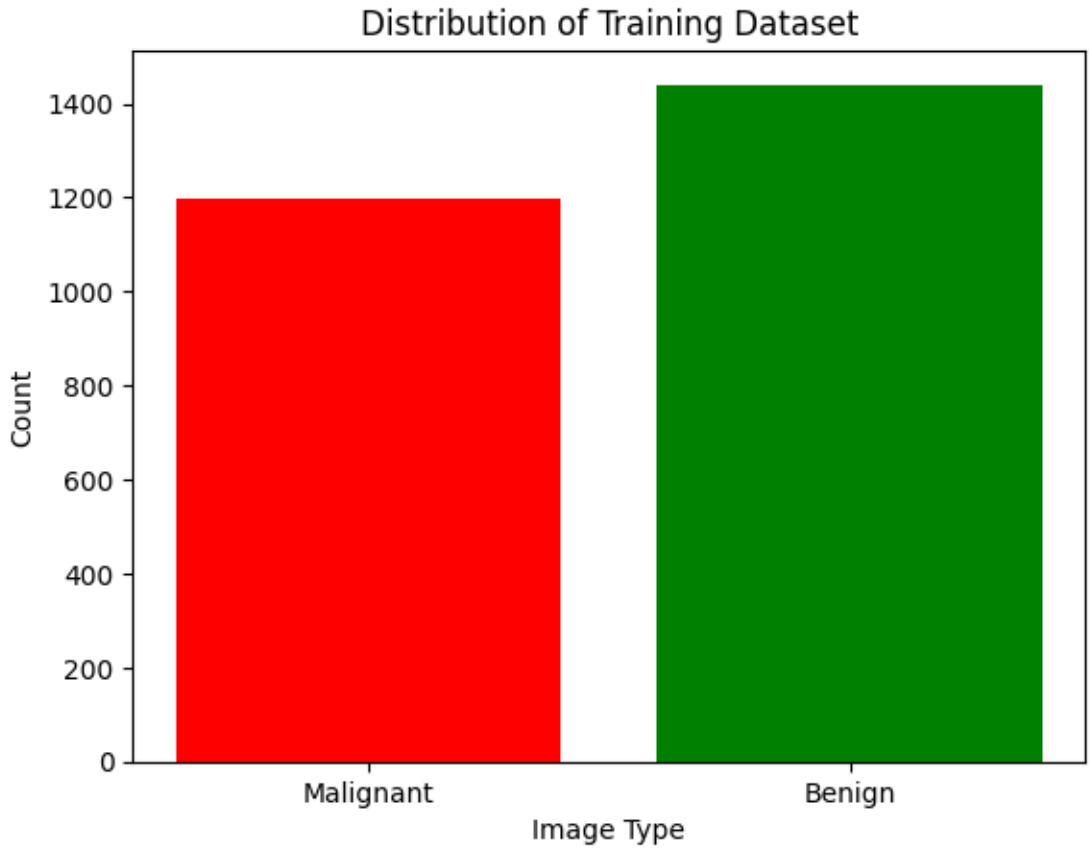
```
['10.jpg', '100.jpg', '1000.jpg', '1001.jpg', '1002.jpg', '1004.jpg',
'1006.jpg', '1008.jpg', '101.jpg', '1010.jpg']
['100.jpg', '1000.jpg', '1001.jpg', '1002.jpg', '1004.jpg', '1005.jpg',
'1007.jpg', '1008.jpg', '1009.jpg', '101.jpg']
```

[6]:
```
# 25% validation data, 75% training data
print('total training malignant images:', len(os.listdir(train_malignant_dir)))
print('total training benign images:', len(os.listdir(train_benign_dir)))
print('total validation malignant images:', len(os.
  ↪listdir(validation_malignant_dir)))
print('total validation benign images:', len(os.listdir(validation_benign_dir)))
```

```
total training malignant images: 1197
total training benign images: 1440
total validation malignant images: 300
total validation benign images: 360
```

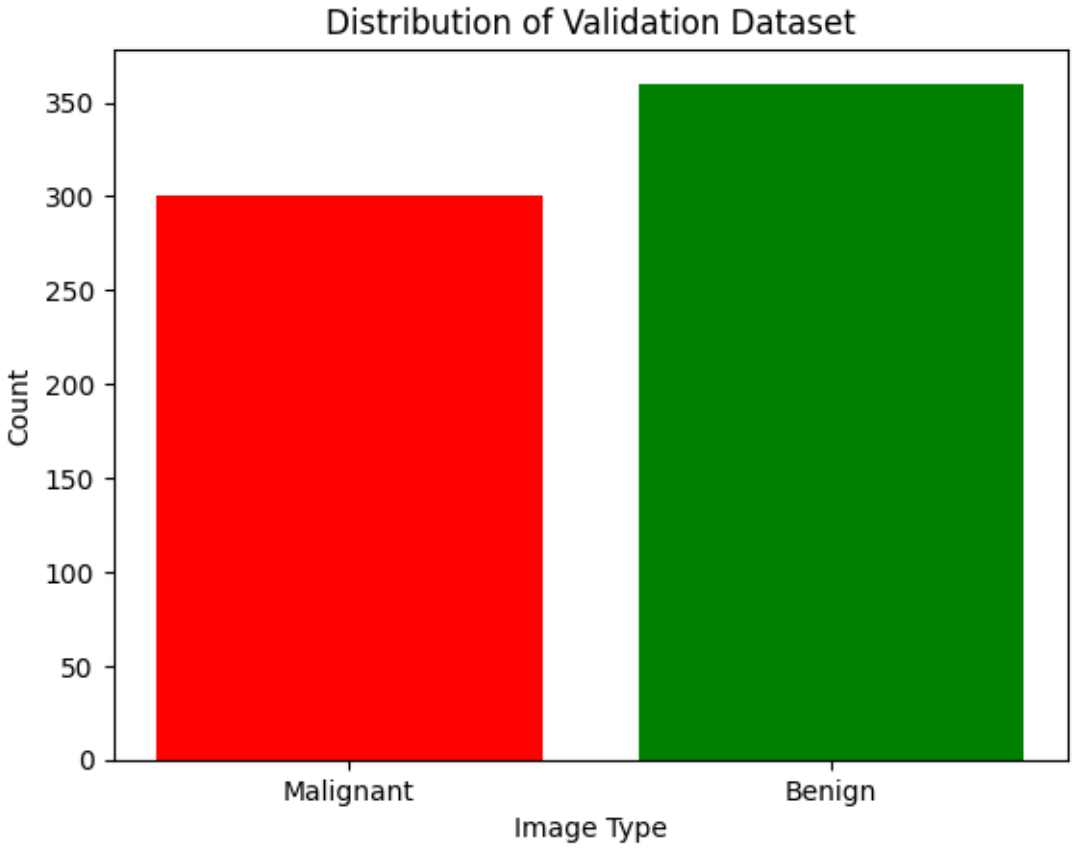
[ ]:
```
# visualizing training dataset

train_malignant_data = len(os.listdir(train_malignant_dir))
train_benign_data = len(os.listdir(train_benign_dir))
```

```
labels = ['Malignant', 'Benign']

values = [train_malignant_data, train_benign_data]

color = ['red', 'green']
plt.bar(labels, values, color = color)

plt.title('Distribution of Training Dataset')
plt.xlabel('Image Type')
plt.ylabel('Count')

plt.show()
```



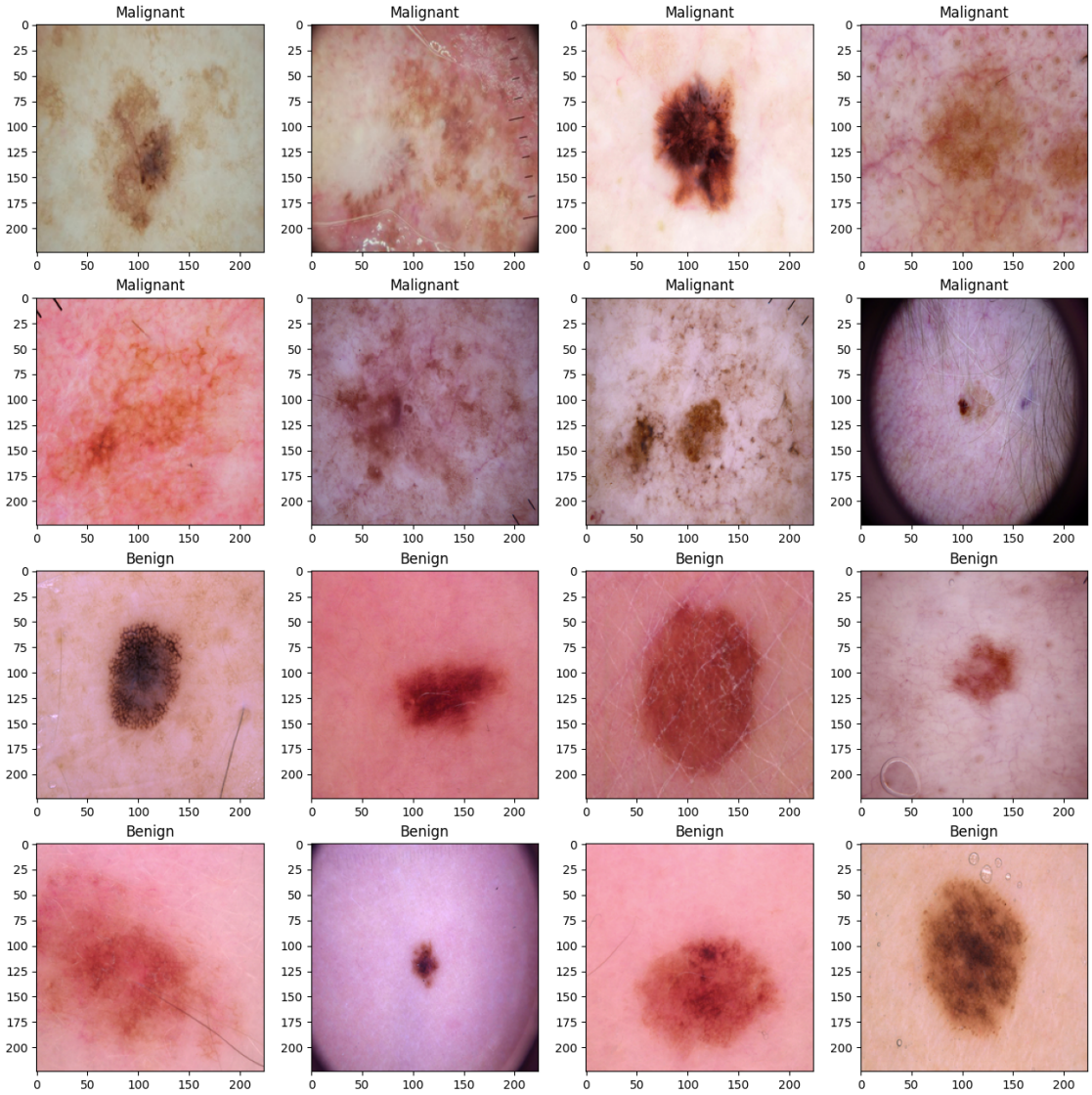
[ ]:
```
# visualizing validation dataset

validation_malignant_data = len(os.listdir(validation_malignant_dir))
validation_benign_data = len(os.listdir(validation_benign_dir))
labels = ['Malignant', 'Benign']

values = [validation_malignant_data, validation_benign_data]
```

```
color = ['red', 'green']
plt.bar(labels, values, color = color)

plt.title('Distribution of Validation Dataset')
plt.xlabel('Image Type')
plt.ylabel('Count')

plt.show()
```



[ ]:
```
%matplotlib inline

# Parameters for our graph; we'll output images in a 4x4 configuration
nrows = 4
ncols = 4

# Index for iterating over images
pic_index = 0
```

[ ]:
```
# Set up matplotlib fig, and size it to fit 4x4 pics
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)

pic_index += 8
next_malignant_pix = [os.path.join(train_malignant_dir, fname)
                for fname in train_malignant_fnames[pic_index-8:pic_index]]
next_benign_pix = [os.path.join(train_benign_dir, fname)
                for fname in train_benign_fnames[pic_index-8:pic_index]]

for i, img_path in enumerate(next_malignant_pix+next_benign_pix):
  # Set up subplot; subplot indices start at 1
  sp = plt.subplot(nrows, ncols, i + 1)
  sp.axis('On') # show axes (or gridlines)

  img = mpimg.imread(img_path)
  plt.imshow(img)

  # Set title for the subplot
  if i < len(next_malignant_pix):
    sp.set_title("Malignant")
  else:
    sp.set_title("Benign")

plt.show()

plt.show()
```



2 Building a Small Convnet from Scratch

The images that will go into our convnet are 150x150 color images (in the next section on Data Preprocessing, we'll add handling to resize all the images to 150x150 before feeding them into the neural network).

Let's code up the architecture. We will stack 3 {convolution + relu + maxpooling} modules. Our convolutions operate on 3x3 windows and our maxpooling layers operate on 2x2 windows. Our first convolution extracts 16 filters, the following one extracts 32 filters, and the last one extracts 64 filters.

[ ]:
```
# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for␣
↪the three color channels: R, G, and B
```

```
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
```

On top of it we stick two fully-connected layers. Because we are facing a two-class classification problem, i.e. a binary classification problem, we will end our network with a sigmoid activation, so that the output of our network will be a single scalar between 0 and 1, encoding the probability that the current image is class 1 (as opposed to class 0). In our problem, we are trying to predict if the image is Malignant or Benign.

[ ]:
```
# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units
x = layers.Dense(512, activation='relu')(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Create model:
# input = input feature map
# output = input feature map + stacked convolution/maxpooling layers + fully
# connected layer + sigmoid output layer
model = Model(img_input, output)
```

Summary of the model architecture:

[ ]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0
```

```
conv2d (Conv2D)              (None, 148, 148, 16)      448

max_pooling2d (MaxPooling2D  (None, 74, 74, 16)       0
)

conv2d_1 (Conv2D)            (None, 72, 72, 32)        4640

max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
2D)

conv2d_2 (Conv2D)            (None, 34, 34, 64)        18496

max_pooling2d_2 (MaxPooling  (None, 17, 17, 64)       0
2D)

flatten (Flatten)            (None, 18496)             0

dense (Dense)                (None, 512)               9470464

dense_1 (Dense)              (None, 1)                 513

=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

The "output shape" column shows how the size of your feature map evolves in each successive layer. The convolution layers reduce the size of the feature maps by a bit due to padding, and each pooling layer halves the feature map.

Next, we'll configure the specifications for model training. We will train our model with the binary_crossentropy loss, because it's a binary classification problem and our final activation is a sigmoid. We will use the rmsprop optimizer with a learning rate of 0.001. During training, we will want to monitor classification accuracy.

NOTE: In this case, using the RMSprop optimization algorithm is preferable to stochastic gradient descent (SGD), because RMSprop automates learning-rate tuning for us. (Other optimizers, such as Adam and Adagrad, also automatically adapt the learning rate during training, and would work equally well here.)

[ ]:
```
model.compile(loss='binary_crossentropy', optimizer=RMSprop(learning_rate=0.
↪001), metrics=['acc'])
```

3 Data Preprocessing

Let's set up data generators that will read pictures in our source folders, convert them to float32 tensors, and feed them (with their labels) to our network. We'll have one generator for the training

images and one for the validation images. Our generators will yield batches of 20 images of size 150x150 and their labels (binary).

As you may already know, data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network. (It is uncommon to feed raw pixels into a convnet.) In our case, we will preprocess our images by normalizing the pixel values to be in the [0, 1] range (originally all values are in the [0, 255] range).

In Keras this can be done via the keras.preprocessing.image.ImageDataGenerator class using the rescale parameter. This ImageDataGenerator class allows you to instantiate generators of augmented image batches (and their labels) via .flow(data, labels) or .flow_from_directory(directory). These generators can then be used with the Keras model methods that accept data generators as inputs: fit_generator, evaluate_generator, and predict_generator.

[ ]:
```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

Training on 2,637 images, validating on 660 images.

[ ]:
```
history = model.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 87s - loss: 0.6212 - acc: 0.6712 - val_loss: 0.4716 - val_acc: 0.7697
- 87s/epoch - 661ms/step
Epoch 2/15
```

```
132/132 - 92s - loss: 0.4971 - acc: 0.7497 - val_loss: 0.5176 - val_acc: 0.6197
- 92s/epoch - 697ms/step
Epoch 3/15
132/132 - 79s - loss: 0.4623 - acc: 0.7656 - val_loss: 0.4225 - val_acc: 0.7803
- 79s/epoch - 601ms/step
Epoch 4/15
132/132 - 80s - loss: 0.4352 - acc: 0.7857 - val_loss: 0.3929 - val_acc: 0.8076
- 80s/epoch - 604ms/step
Epoch 5/15
132/132 - 79s - loss: 0.4067 - acc: 0.7911 - val_loss: 0.3738 - val_acc: 0.8167
- 79s/epoch - 599ms/step
Epoch 6/15
132/132 - 83s - loss: 0.3878 - acc: 0.8111 - val_loss: 0.4086 - val_acc: 0.8000
- 83s/epoch - 629ms/step
Epoch 7/15
132/132 - 81s - loss: 0.3662 - acc: 0.8187 - val_loss: 0.4035 - val_acc: 0.7697
- 81s/epoch - 610ms/step
Epoch 8/15
132/132 - 80s - loss: 0.3780 - acc: 0.8221 - val_loss: 0.4100 - val_acc: 0.7924
- 80s/epoch - 604ms/step
Epoch 9/15
132/132 - 83s - loss: 0.3466 - acc: 0.8312 - val_loss: 0.3802 - val_acc: 0.8091
- 83s/epoch - 632ms/step
Epoch 10/15
132/132 - 84s - loss: 0.3191 - acc: 0.8464 - val_loss: 0.3524 - val_acc: 0.8227
- 84s/epoch - 635ms/step
Epoch 11/15
132/132 - 81s - loss: 0.3136 - acc: 0.8415 - val_loss: 0.3639 - val_acc: 0.8318
- 81s/epoch - 611ms/step
Epoch 12/15
132/132 - 84s - loss: 0.2891 - acc: 0.8680 - val_loss: 0.3788 - val_acc: 0.8242
- 84s/epoch - 636ms/step
Epoch 13/15
132/132 - 81s - loss: 0.2702 - acc: 0.8832 - val_loss: 0.4327 - val_acc: 0.8333
- 81s/epoch - 614ms/step
Epoch 14/15
132/132 - 80s - loss: 0.2549 - acc: 0.8889 - val_loss: 0.4580 - val_acc: 0.8106
- 80s/epoch - 608ms/step
Epoch 15/15
132/132 - 80s - loss: 0.2208 - acc: 0.9029 - val_loss: 0.4458 - val_acc: 0.8182
- 80s/epoch - 603ms/step
```

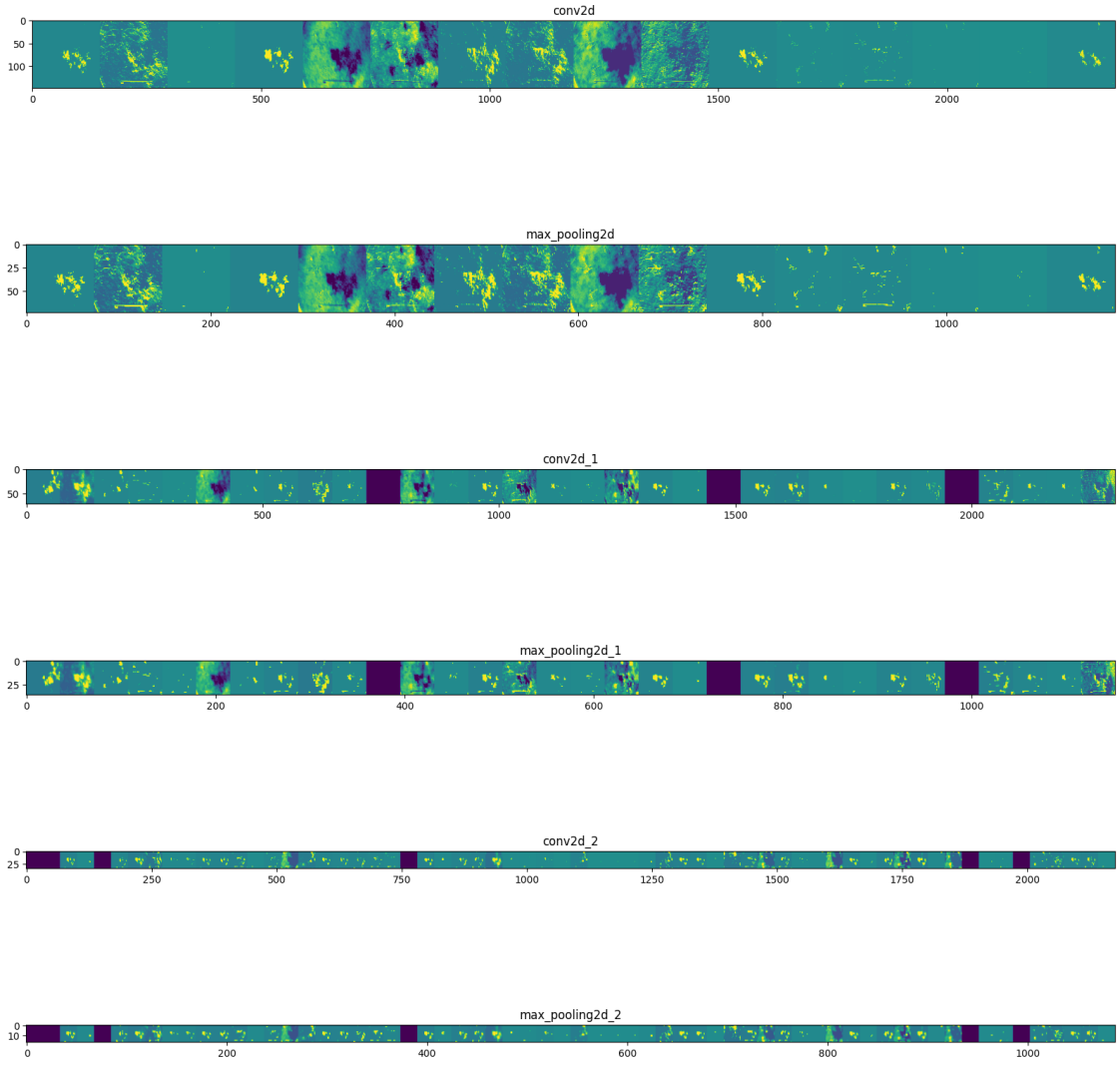
[ ]:
```
# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model.layers[1:]]
visualization_model = Model(img_input, successive_outputs)
```

```
# Let's prepare a random input image of a malignant or benign tumor from the␣
 ↪training set.
malignant_img_files = [os.path.join(train_malignant_dir, f) for f in␣
 ↪train_malignant_fnames]
benign_img_files = [os.path.join(train_benign_dir, f) for f in␣
 ↪train_benign_fnames]
img_path = random.choice(malignant_img_files + benign_img_files)

img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model.layers[1:]]

# Now let's display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected layers
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
      # We'll tile each filter into this big horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
```

```
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

```
1/1 [==============================] - 0s 158ms/step
```

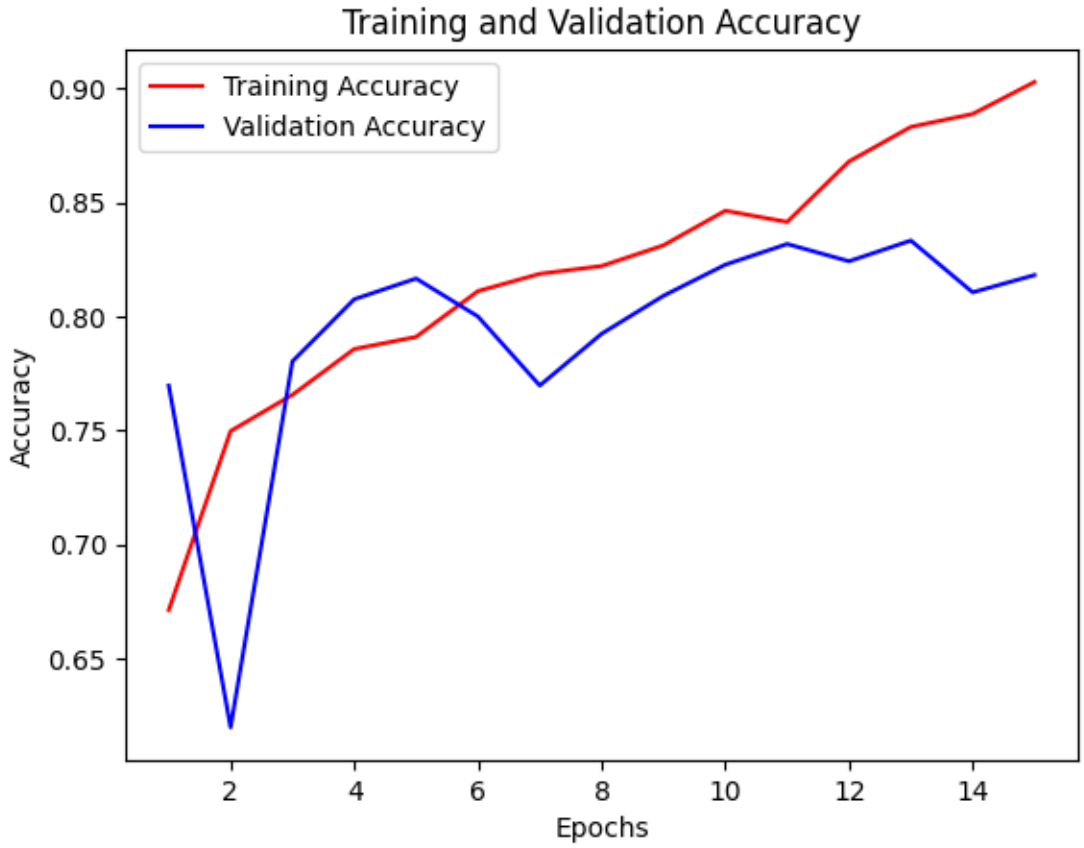
```
<ipython-input-18-24c723c52160>:39: RuntimeWarning: invalid value encountered in
true_divide
  x /= x.std()
```



[ ]:
```
# Accuracy
performance_dict = history.history
accuracy_values = performance_dict["acc"]
val_accuracy_values = performance_dict["val_acc"]
epochs = range(1, len(accuracy_values) + 1)
plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color = 'red')
plt.plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
```

```
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

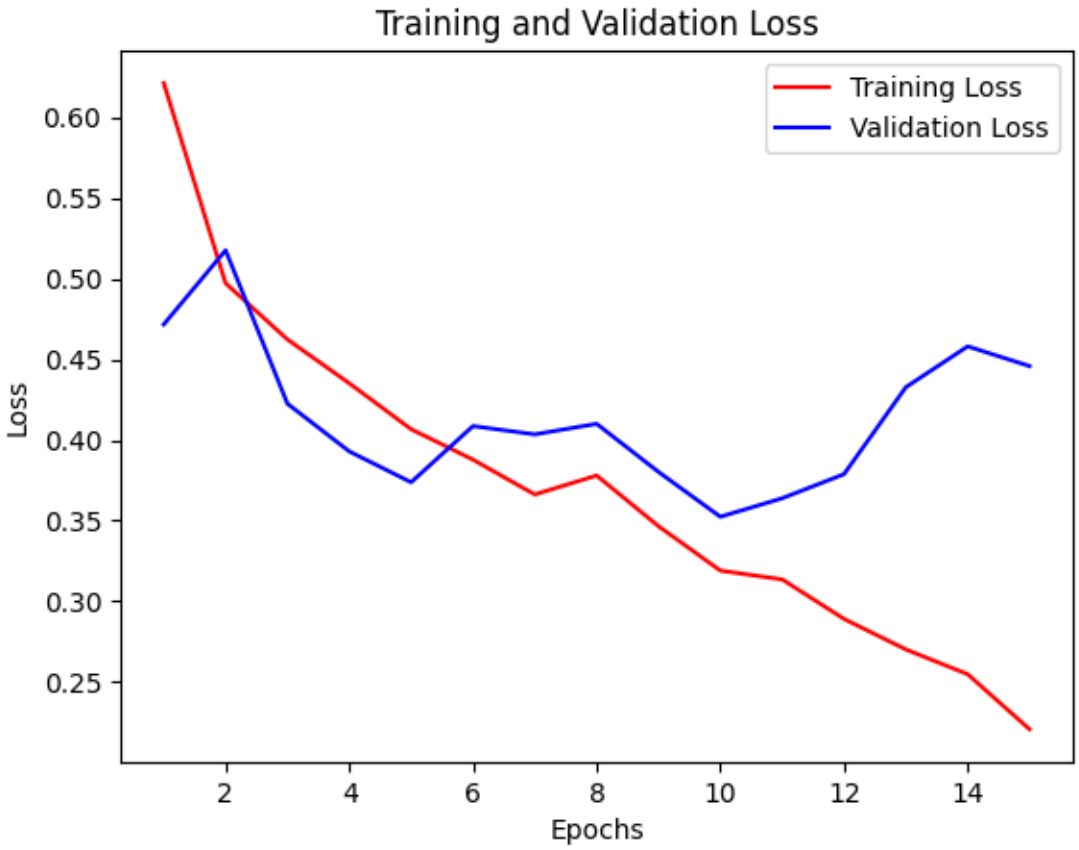
```
<ipython-input-19-7206d8438e07>:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color =
'red')
```



[ ]:
```
performance_dict = history.history
loss_values = performance_dict["loss"]
val_loss_values = performance_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
plt.plot(epochs, val_loss_values, "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
```

```
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
<ipython-input-20-91741870ffcc>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
```



4 Repeating Steps Above with Sigmoid Activation Function

[ ]:
```
# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for␣
 ↪the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='sigmoid')(img_input)
x = layers.MaxPooling2D(2)(x)
```

```
# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='sigmoid')(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(64, 3, activation='sigmoid')(x)
x = layers.MaxPooling2D(2)(x)
```

[ ]:
```
# Flatten feature map to a 1-dim tensor so we can add fully connected layers
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units
x = layers.Dense(512, activation='sigmoid')(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Create model:
# input = input feature map
# output = input feature map + stacked convolution/maxpooling layers + fully
# connected layer + sigmoid output layer
model_sigmoid = Model(img_input, output)
```

[ ]:
```
model_sigmoid.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 150, 150, 3)]     0

 conv2d_3 (Conv2D)           (None, 148, 148, 16)      448

 max_pooling2d_3 (MaxPooling  (None, 74, 74, 16)       0
 2D)

 conv2d_4 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 17, 17, 64)       0
```

```
2D)

flatten_1 (Flatten)          (None, 18496)             0

dense_2 (Dense)              (None, 512)               9470464

dense_3 (Dense)              (None, 1)                 513

=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

[ ]:
```
history_sigmoid = model.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 86s - loss: 0.1905 - acc: 0.9261 - val_loss: 0.7506 - val_acc: 0.7864
- 86s/epoch - 650ms/step
Epoch 2/15
```

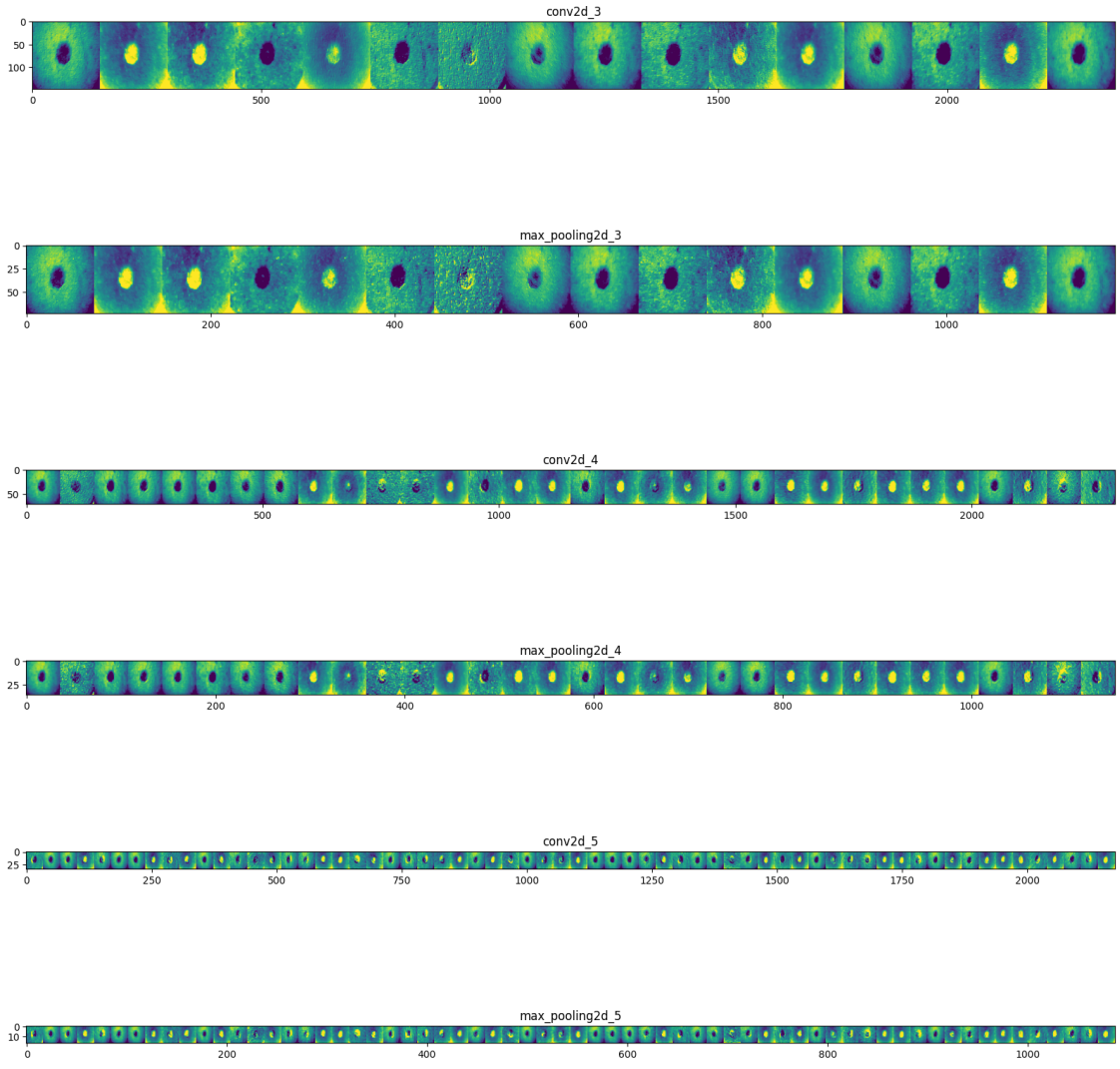
```
132/132 - 79s - loss: 0.1759 - acc: 0.9302 - val_loss: 0.4961 - val_acc: 0.8227
- 79s/epoch - 602ms/step
Epoch 3/15
132/132 - 87s - loss: 0.1671 - acc: 0.9355 - val_loss: 0.5720 - val_acc: 0.8530
- 87s/epoch - 655ms/step
Epoch 4/15
132/132 - 79s - loss: 0.1304 - acc: 0.9488 - val_loss: 0.5459 - val_acc: 0.8470
- 79s/epoch - 600ms/step
Epoch 5/15
132/132 - 86s - loss: 0.1199 - acc: 0.9568 - val_loss: 0.6596 - val_acc: 0.8106
- 86s/epoch - 650ms/step
Epoch 6/15
132/132 - 84s - loss: 0.1055 - acc: 0.9655 - val_loss: 0.6249 - val_acc: 0.8470
- 84s/epoch - 639ms/step
Epoch 7/15
132/132 - 80s - loss: 0.0837 - acc: 0.9689 - val_loss: 0.7182 - val_acc: 0.8364
- 80s/epoch - 607ms/step
Epoch 8/15
132/132 - 87s - loss: 0.0650 - acc: 0.9788 - val_loss: 0.7516 - val_acc: 0.8364
- 87s/epoch - 661ms/step
Epoch 9/15
132/132 - 86s - loss: 0.1321 - acc: 0.9689 - val_loss: 0.8175 - val_acc: 0.8424
- 86s/epoch - 655ms/step
Epoch 10/15
132/132 - 83s - loss: 0.0541 - acc: 0.9814 - val_loss: 0.9669 - val_acc: 0.8000
- 83s/epoch - 627ms/step
Epoch 11/15
132/132 - 87s - loss: 0.0600 - acc: 0.9799 - val_loss: 0.8534 - val_acc: 0.8394
- 87s/epoch - 661ms/step
Epoch 12/15
132/132 - 84s - loss: 0.0589 - acc: 0.9803 - val_loss: 1.0823 - val_acc: 0.8030
- 84s/epoch - 634ms/step
Epoch 13/15
132/132 - 87s - loss: 0.0575 - acc: 0.9837 - val_loss: 0.8899 - val_acc: 0.8348
- 87s/epoch - 663ms/step
Epoch 14/15
132/132 - 81s - loss: 0.0379 - acc: 0.9871 - val_loss: 1.0412 - val_acc: 0.8273
- 81s/epoch - 613ms/step
Epoch 15/15
132/132 - 82s - loss: 0.1050 - acc: 0.9852 - val_loss: 1.1635 - val_acc: 0.8242
- 82s/epoch - 621ms/step
```

[ ]:
```
# Let's define a new Model that will take an image as input, and will output
# intermediate representations for all layers in the previous model after
# the first.
successive_outputs = [layer.output for layer in model_sigmoid.layers[1:]]
visualization_model = Model(img_input, successive_outputs)
```

```
# Let's prepare a random input image of a malignant or benign tumor from the␣
 ↪training set.
malignant_img_files = [os.path.join(train_malignant_dir, f) for f in␣
 ↪train_malignant_fnames]
benign_img_files = [os.path.join(train_benign_dir, f) for f in␣
 ↪train_benign_fnames]
img_path = random.choice(malignant_img_files + benign_img_files)

img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# Rescale by 1/255
x /= 255

# Let's run our image through our network, thus obtaining all
# intermediate representations for this image.
successive_feature_maps = visualization_model.predict(x)

# These are the names of the layers, so can have them as part of our plot
layer_names = [layer.name for layer in model_sigmoid.layers[1:]]

# Now let's display our representations
for layer_name, feature_map in zip(layer_names, successive_feature_maps):
  if len(feature_map.shape) == 4:
    # Just do this for the conv / maxpool layers, not the fully-connected layers
    n_features = feature_map.shape[-1]  # number of features in feature map
    # The feature map has shape (1, size, size, n_features)
    size = feature_map.shape[1]
    # We will tile our images in this matrix
    display_grid = np.zeros((size, size * n_features))
    for i in range(n_features):
      # Postprocess the feature to make it visually palatable
      x = feature_map[0, :, :, i]
      x -= x.mean()
      x /= x.std()
      x *= 64
      x += 128
      x = np.clip(x, 0, 255).astype('uint8')
      # We'll tile each filter into this big horizontal grid
      display_grid[:, i * size : (i + 1) * size] = x
    # Display the grid
    scale = 20. / n_features
    plt.figure(figsize=(scale * n_features, scale))
    plt.title(layer_name)
    plt.grid(False)
```

```
    plt.imshow(display_grid, aspect='auto', cmap='viridis')
```

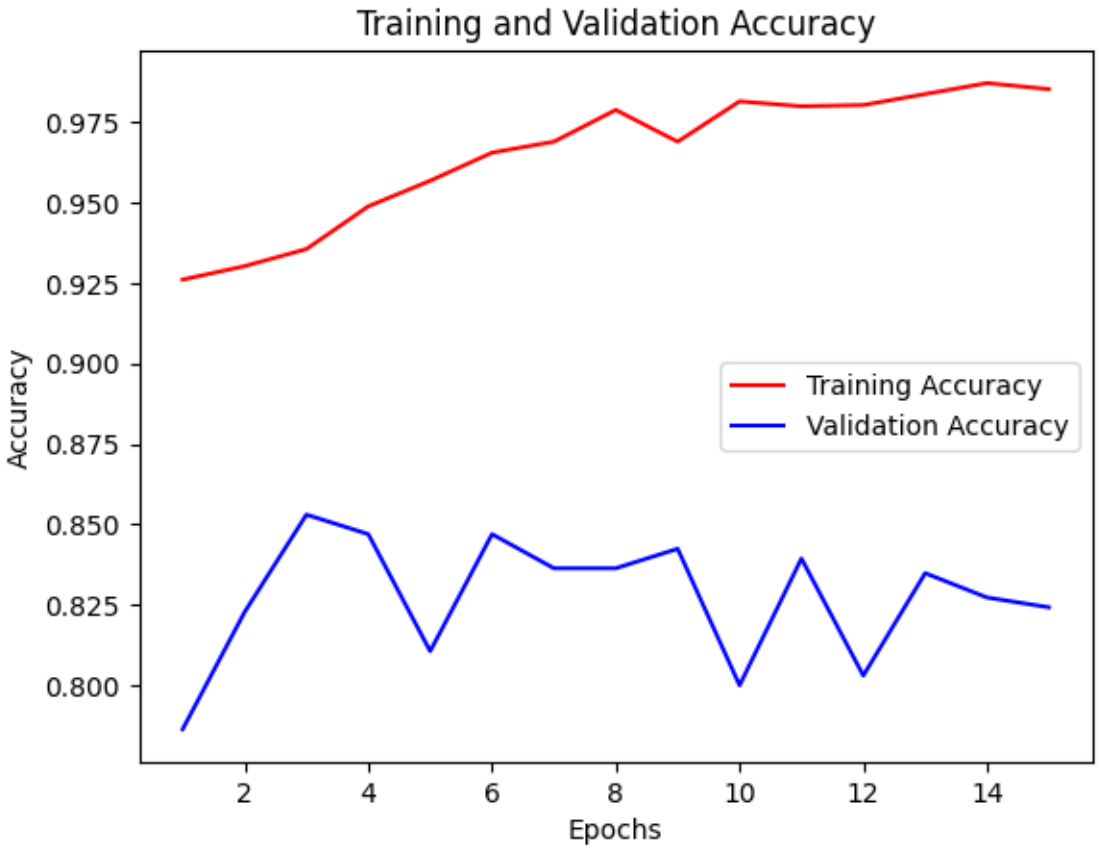
```
1/1 [==============================] - 0s 108ms/step
```



[ ]:
```
# Accuracy
performance_dict_sigmoid = history_sigmoid.history
accuracy_values_sigmoid = performance_dict_sigmoid["acc"]
val_accuracy_values_sigmoid = performance_dict_sigmoid["val_acc"]
epochs = range(1, len(accuracy_values_sigmoid) + 1)
plt.plot(epochs, accuracy_values_sigmoid, "b", label="Training Accuracy", color␣
  ↪= 'red')
plt.plot(epochs, val_accuracy_values_sigmoid, "b", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
```

```
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

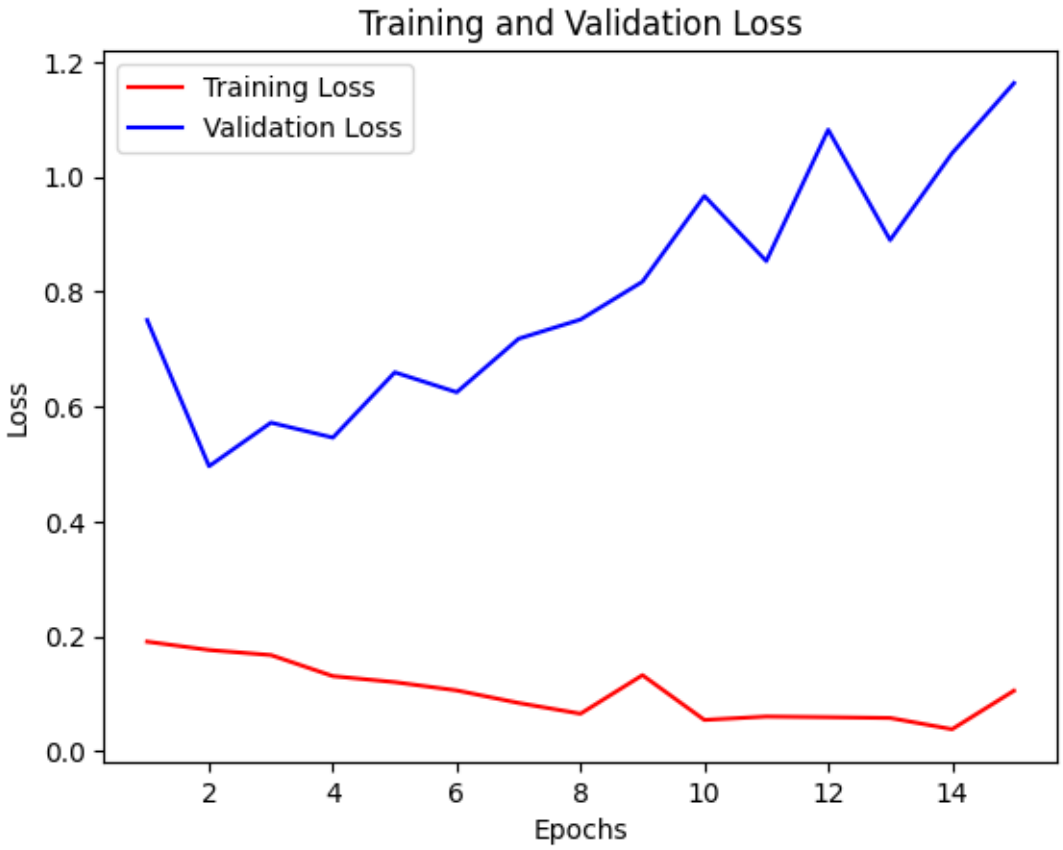
```
<ipython-input-27-e3bca0e6c8c8>:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, accuracy_values_sigmoid, "b", label="Training Accuracy",
color = 'red')
```



[ ]:
```
# Loss
performance_dict_sigmoid = history_sigmoid.history
loss_values_sigmoid = performance_dict_sigmoid["loss"]
val_loss_values_sigmoid = performance_dict_sigmoid["val_loss"]
epochs = range(1, len(loss_values_sigmoid) + 1)
plt.plot(epochs, loss_values_sigmoid, "b", label="Training Loss", color = 'red')
plt.plot(epochs, val_loss_values_sigmoid, "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
```

```
plt.legend()
plt.show()
```

```
<ipython-input-28-84fb18421a11>:6: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, loss_values_sigmoid, "b", label="Training Loss", color =
'red')
```



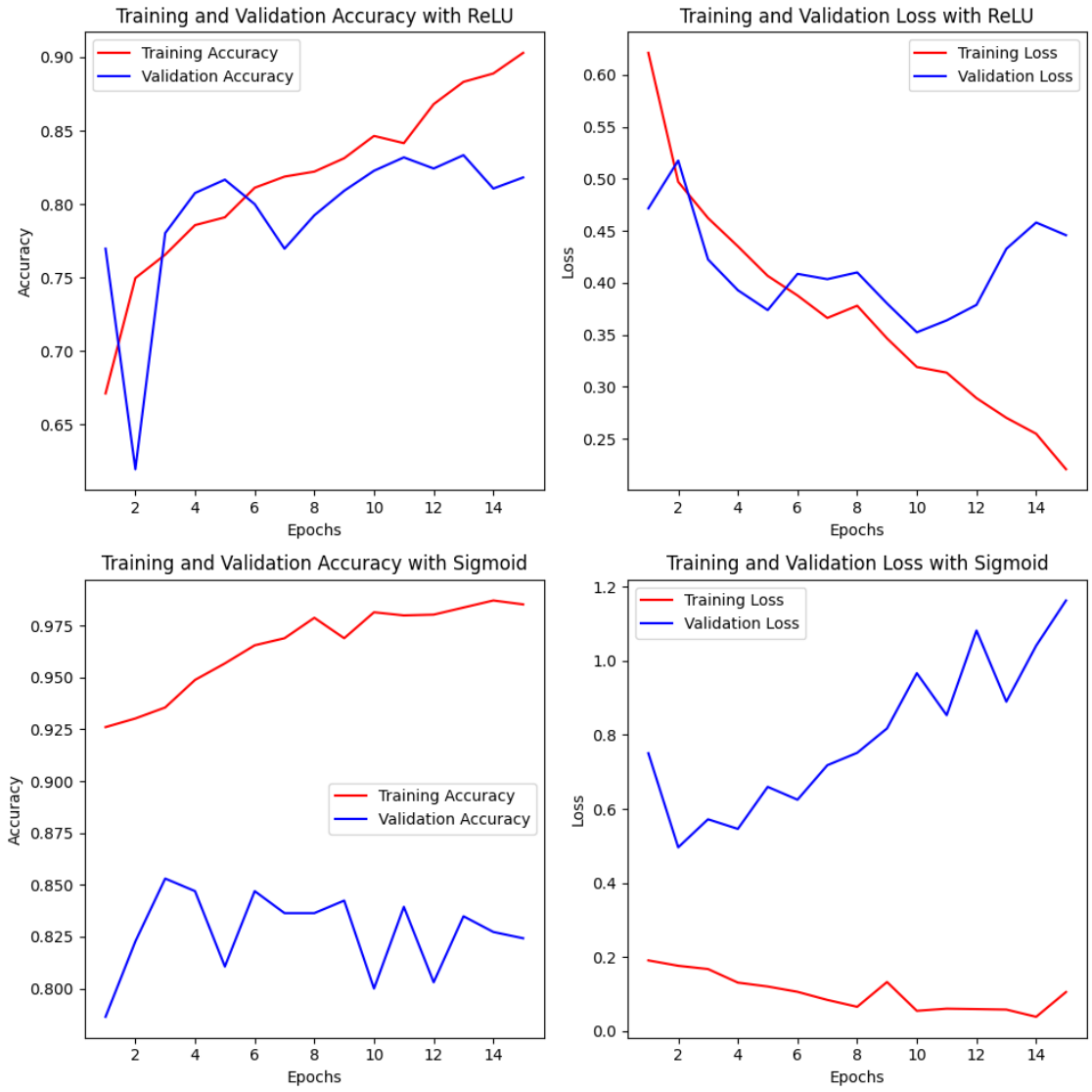
5 First Model Accuracy and Loss Results

[ ]:
```
# Plot for the first model
performance_dict = history.history
accuracy_values = performance_dict["acc"]
val_accuracy_values = performance_dict["val_acc"]
loss_values = performance_dict["loss"]
val_loss_values = performance_dict["val_loss"]
epochs = range(1, len(accuracy_values) + 1)
```

```
fig, axs = plt.subplots(2, 2, figsize=(10,10))

axs[0, 0].plot(epochs, accuracy_values, "b", label="Training Accuracy", color =␣
 ↪'red')
axs[0, 0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0, 0].set_title("Training and Validation Accuracy with ReLU")
axs[0, 0].set_xlabel("Epochs")
axs[0, 0].set_ylabel("Accuracy")
axs[0, 0].legend()

axs[0, 1].plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
axs[0, 1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[0, 1].set_title("Training and Validation Loss with ReLU")
axs[0, 1].set_xlabel("Epochs")
axs[0, 1].set_ylabel("Loss")
axs[0, 1].legend()


# Plot for the second model
performance_dict_sigmoid = history_sigmoid.history
accuracy_values_sigmoid = performance_dict_sigmoid["acc"]
val_accuracy_values_sigmoid = performance_dict_sigmoid["val_acc"]
loss_values_sigmoid = performance_dict_sigmoid["loss"]
val_loss_values_sigmoid = performance_dict_sigmoid["val_loss"]
epochs = range(1, len(accuracy_values_sigmoid) + 1)

axs[1, 0].plot(epochs, accuracy_values_sigmoid, "b", label="Training Accuracy",␣
 ↪color = 'red')
axs[1, 0].plot(epochs, val_accuracy_values_sigmoid, "b", label="Validation␣
 ↪Accuracy")
axs[1, 0].set_title("Training and Validation Accuracy with Sigmoid")
axs[1, 0].set_xlabel("Epochs")
axs[1, 0].set_ylabel("Accuracy")
axs[1, 0].legend()

axs[1, 1].plot(epochs, loss_values_sigmoid, "b", label="Training Loss", color =␣
 ↪'red')
axs[1, 1].plot(epochs, val_loss_values_sigmoid, "b", label="Validation Loss")
axs[1, 1].set_title("Training and Validation Loss with Sigmoid")
axs[1, 1].set_xlabel("Epochs")
axs[1, 1].set_ylabel("Loss")
axs[1, 1].legend()

plt.tight_layout()
plt.show()
```

```
<ipython-input-33-9525f1dacfdd>:11: UserWarning: color is redundantly defined by
```

```
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 0].plot(epochs, accuracy_values, "b", label="Training Accuracy", color
= 'red')
<ipython-input-33-9525f1dacfdd>:18: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 1].plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
<ipython-input-33-9525f1dacfdd>:34: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 0].plot(epochs, accuracy_values_sigmoid, "b", label="Training
Accuracy", color = 'red')
<ipython-input-33-9525f1dacfdd>:41: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 1].plot(epochs, loss_values_sigmoid, "b", label="Training Loss", color
= 'red')
```



6 Reducing Overfitting

We will try to improve accuracy by employing a couple strategies to reduce overfitting: data augmentation and dropout. We will follow these steps: Explore how data augmentation works by making random transformations to training images. Add data augmentation to our data preprocessing. Add dropout to the convnet. Retrain the model and evaluate loss and accuracy. Let's get started!

Exploring Data Augmentation: Let's get familiar with the concept of data augmentation, an essential way to fight overfitting for computer vision models.

In order to make the most of our few training examples, we will "augment" them via a number of random transformations, so that at training time, our model will never see the exact same picture

twice. This helps prevent overfitting and helps the model generalize better.

This can be done by configuring a number of random transformations to be performed on the images read by our ImageDataGenerator instance. Let's get started with an example:

7 Data Augmentation

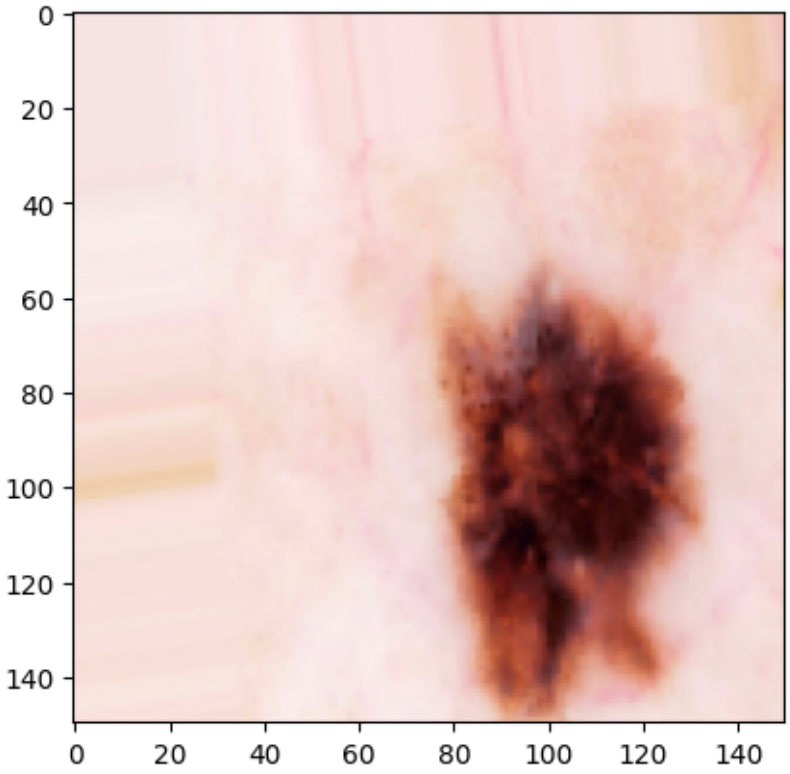
[ ]:
```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

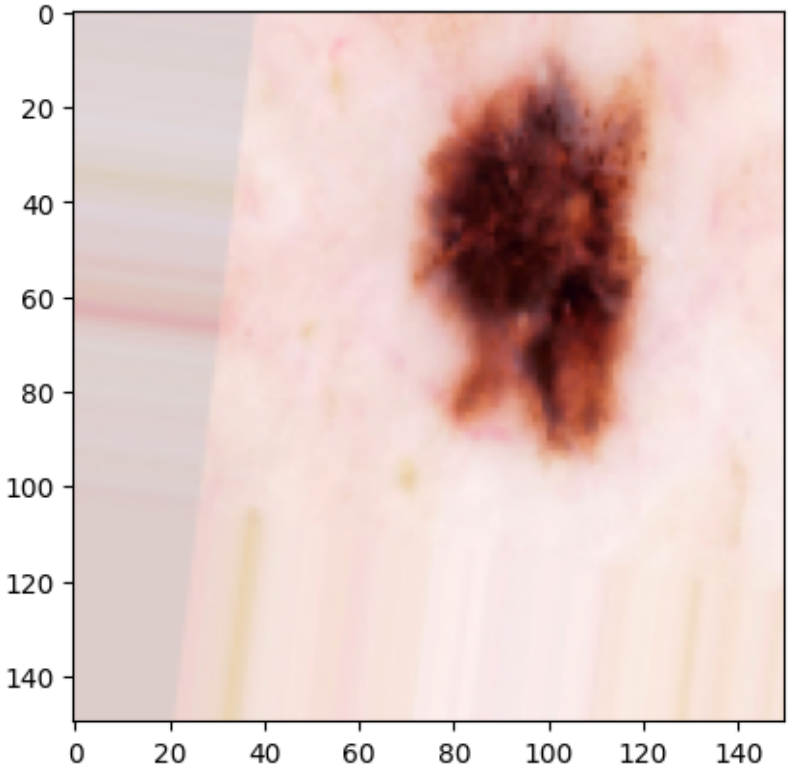
Let's take a look at our augmented images.

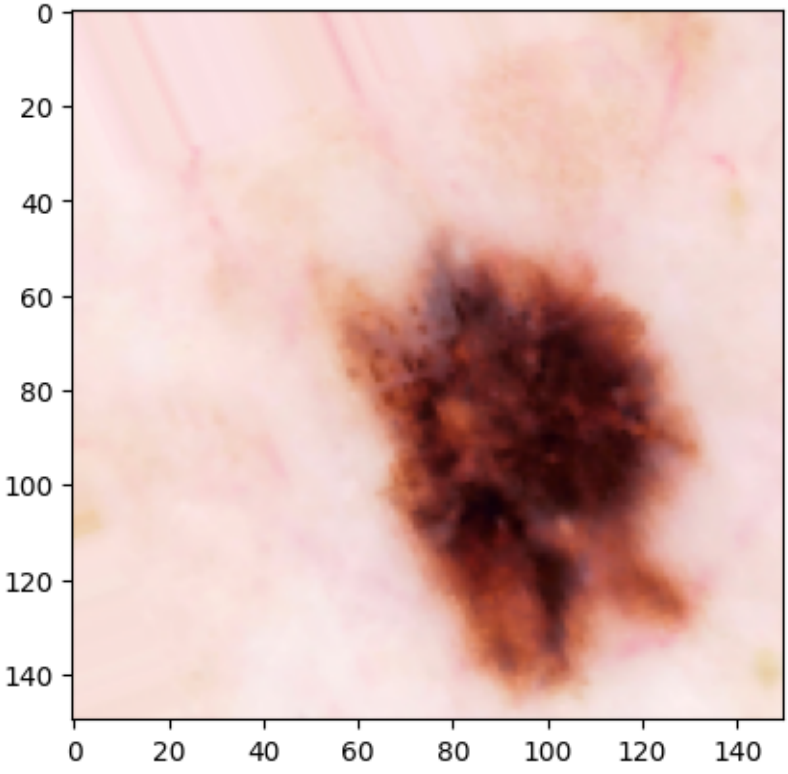
Let's apply the datagen transformations to a image from the training set to produce five random variants. Rerun the cell a few times to see fresh batches of random variants.

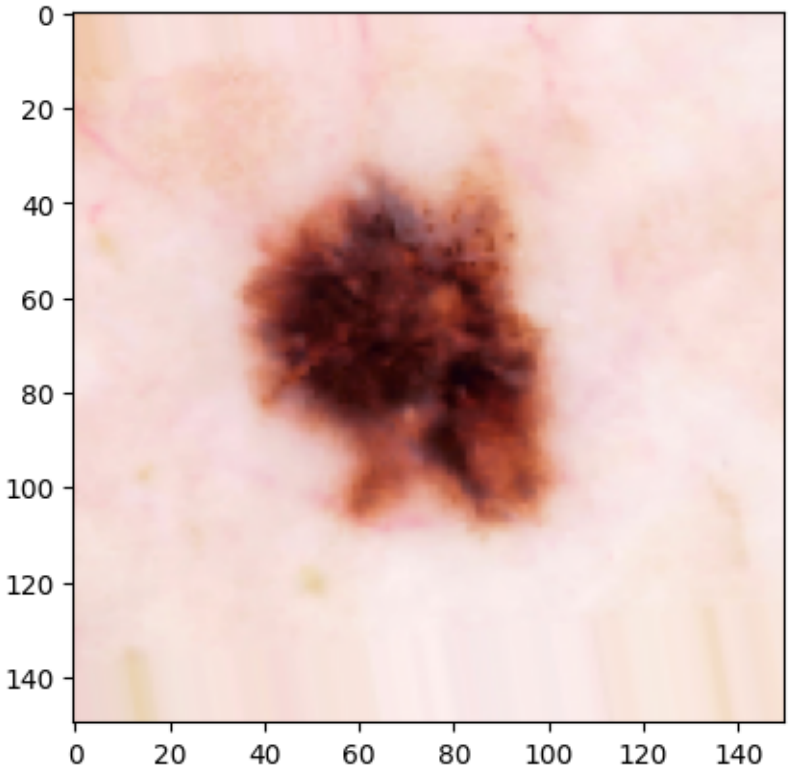
[ ]:
```
%matplotlib inline
img_path = os.path.join(train_malignant_dir, train_malignant_fnames[2])
img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# The .flow() command below generates batches of randomly transformed images
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
  plt.subplot(3,3,i+1)
  plt.axis("off")
  plt.suptitle("Augmented Malignant Images")
  imgplot = plt.imshow(array_to_img(batch[0]))
  i += 1
  if i % 9 == 0:
    break
```

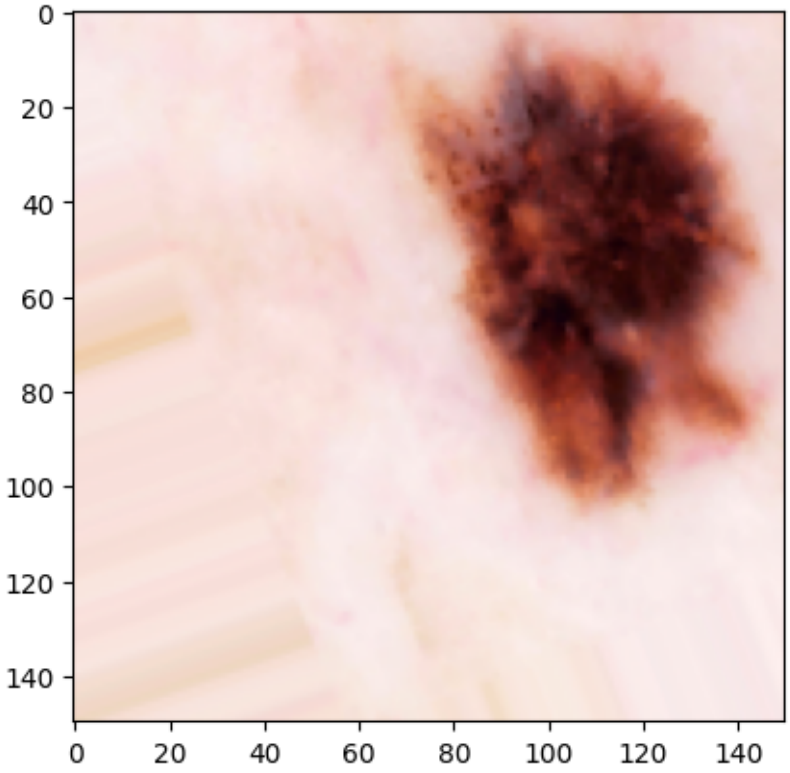
[ ]:
```
%matplotlib inline
img_path = os.path.join(train_malignant_dir, train_malignant_fnames[2])
img = load_img(img_path, target_size=(150, 150))  # this is a PIL image
x = img_to_array(img)  # Numpy array with shape (150, 150, 3)
x = x.reshape((1,) + x.shape)  # Numpy array with shape (1, 150, 150, 3)

# The .flow() command below generates batches of randomly transformed images
# It will loop indefinitely, so we need to `break` the loop at some point!
i = 0
for batch in datagen.flow(x, batch_size=1):
  plt.figure(i)
  imgplot = plt.imshow(array_to_img(batch[0]))
  i += 1
  if i % 5 == 0:
    break
```


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140


0
20
40
60
80
100
120
140
0
20
40
60
80
100
120
140

Add Data Augmentation to the Preprocessing Step Now let's add our data-augmentation transformations from Exploring Data Augmentation to our data preprocessing configuration:

[ ]:
```
# Adding rescale, rotation_range, width_shift_range, height_shift_range,
# shear_range, zoom_range, and horizontal flip to our ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

# Note that the validation data should not be augmented!
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 32 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

If we train a new network using this data augmentation configuration, our network will never see the same input twice. However the inputs that it sees are still heavily intercorrelated, so this might not be quite enough to completely get rid of overfitting.

8 Adding Dropout

Another popular strategy for fighting overfitting is to use dropout.

[ ]:
```
#ReLU
```

```
# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)

# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units -␣
 ↪added dropout
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model = Model(img_input, output)
model.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['acc'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

9 Retrain the Model

With data augmentation and dropout in place, let's retrain our convnet model. This time, let's train on all 2,637 images available, for 15 epochs, and validate on all 660 validation images.

[ ]:
```
history = model.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 106s - loss: 0.6368 - acc: 0.6390 - val_loss: 0.4815 - val_acc: 0.7591
- 106s/epoch - 802ms/step
Epoch 2/15
132/132 - 92s - loss: 0.5322 - acc: 0.7345 - val_loss: 0.4017 - val_acc: 0.7818
- 92s/epoch - 700ms/step
Epoch 3/15
132/132 - 95s - loss: 0.4906 - acc: 0.7573 - val_loss: 0.4838 - val_acc: 0.7379
- 95s/epoch - 721ms/step
Epoch 4/15
132/132 - 95s - loss: 0.4791 - acc: 0.7687 - val_loss: 0.3973 - val_acc: 0.7864
- 95s/epoch - 718ms/step
Epoch 5/15
132/132 - 92s - loss: 0.4502 - acc: 0.7880 - val_loss: 0.6544 - val_acc: 0.6091
- 92s/epoch - 700ms/step
Epoch 6/15
132/132 - 91s - loss: 0.4479 - acc: 0.7892 - val_loss: 0.4062 - val_acc: 0.7985
- 91s/epoch - 689ms/step
Epoch 7/15
132/132 - 92s - loss: 0.4374 - acc: 0.7816 - val_loss: 0.4599 - val_acc: 0.7955
- 92s/epoch - 700ms/step
Epoch 8/15
132/132 - 94s - loss: 0.4434 - acc: 0.7990 - val_loss: 0.3810 - val_acc: 0.8136
- 94s/epoch - 711ms/step
Epoch 9/15
132/132 - 92s - loss: 0.4303 - acc: 0.7960 - val_loss: 0.3827 - val_acc: 0.8197
- 92s/epoch - 697ms/step
Epoch 10/15
132/132 - 93s - loss: 0.4157 - acc: 0.8077 - val_loss: 0.3731 - val_acc: 0.8091
- 93s/epoch - 703ms/step
Epoch 11/15
132/132 - 94s - loss: 0.4175 - acc: 0.7990 - val_loss: 0.4121 - val_acc: 0.8091
- 94s/epoch - 709ms/step
Epoch 12/15
132/132 - 96s - loss: 0.4100 - acc: 0.8104 - val_loss: 0.3713 - val_acc: 0.8015
- 96s/epoch - 730ms/step
Epoch 13/15
132/132 - 96s - loss: 0.4120 - acc: 0.8104 - val_loss: 0.3701 - val_acc: 0.8061
- 96s/epoch - 727ms/step
```

```
Epoch 14/15
132/132 - 96s - loss: 0.4078 - acc: 0.8017 - val_loss: 0.3792 - val_acc: 0.8076
- 96s/epoch - 724ms/step
Epoch 15/15
132/132 - 92s - loss: 0.4191 - acc: 0.8138 - val_loss: 0.4089 - val_acc: 0.8045
- 92s/epoch - 698ms/step
```

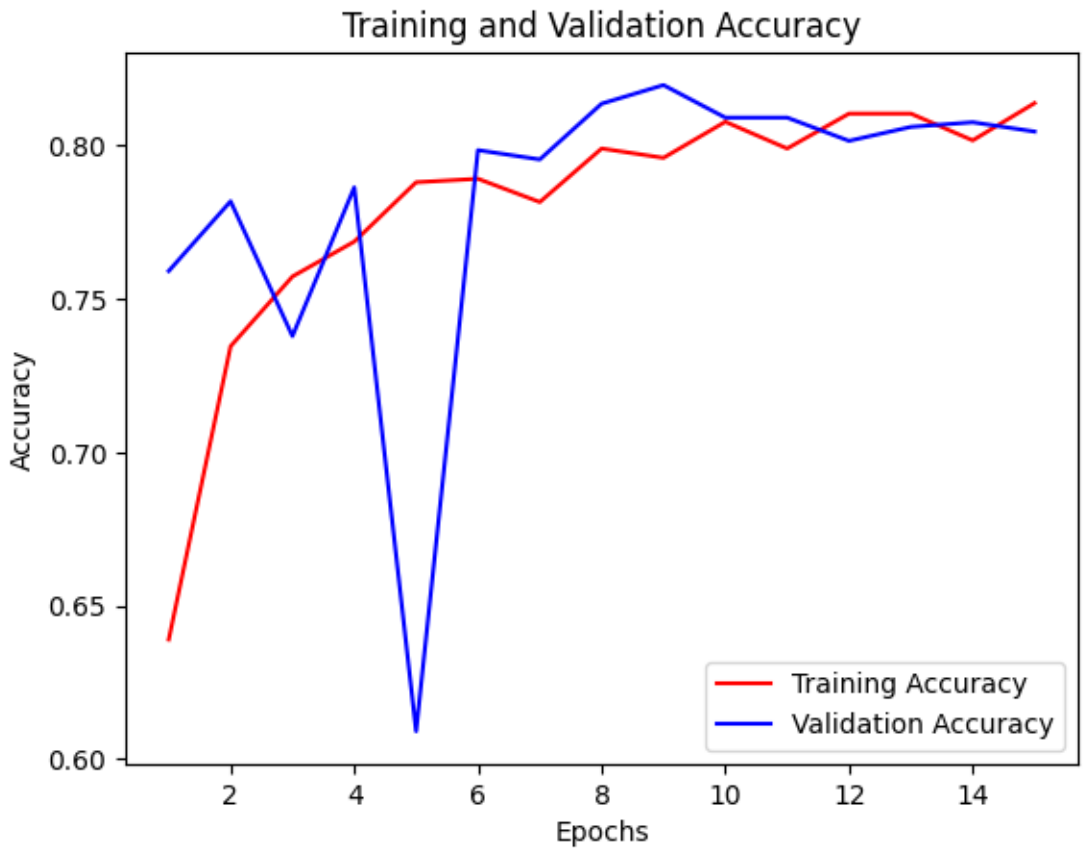
Note that with data augmentation in place, the 2,637 training images are randomly transformed each time a new training epoch runs, which means that the model will never see the same image twice during training.

10 Evaluate the Results

Let's evaluate the results of model training with data augmentation and dropout:

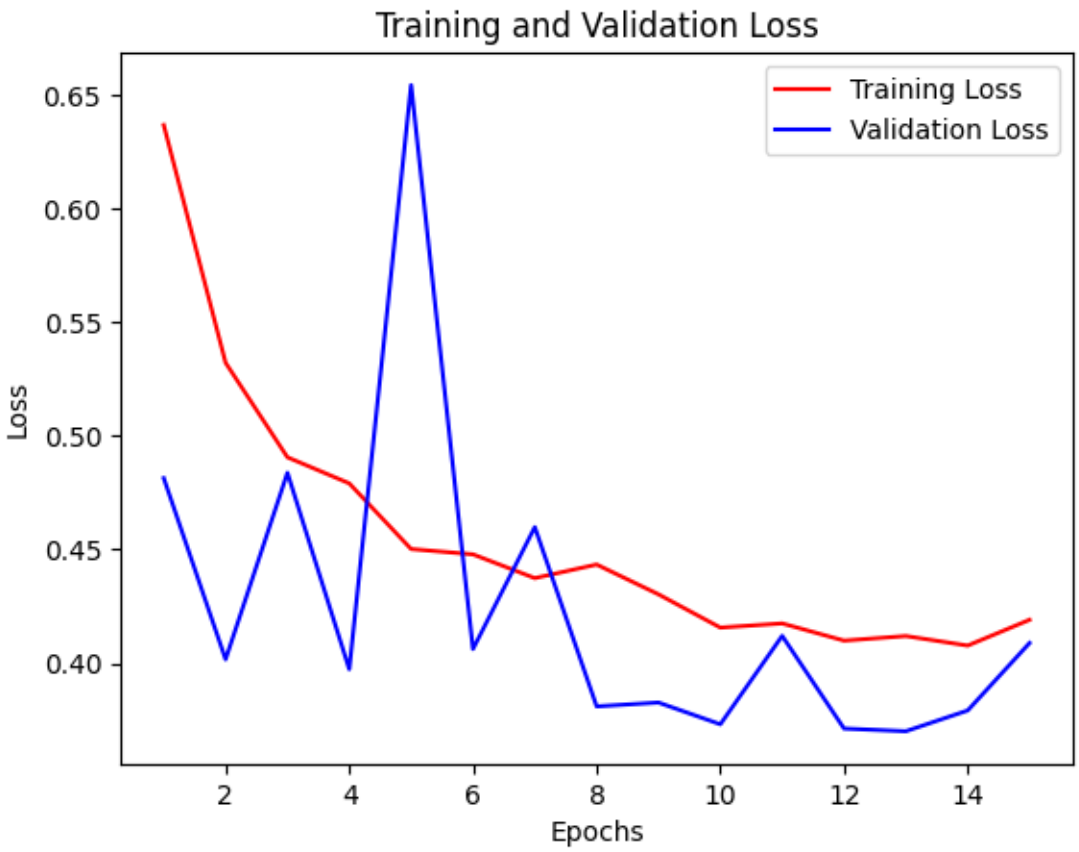
[ ]:
```
performance_dict = history.history
accuracy_values = performance_dict["acc"]
val_accuracy_values = performance_dict["val_acc"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color = 'red')
plt.plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
<ipython-input-40-ebf5d7f82cb7>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color =
'red')
```



[ ]:
```
performance_dict = history.history
loss_values = performance_dict["loss"]
val_loss_values = performance_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
plt.plot(epochs, val_loss_values, "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
<ipython-input-41-91741870ffcc>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
```



11 Repeating steps above with Sigmoid Activation Function

[ ]:
```
#Sigmoid

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='sigmoid')(img_input)
x = layers.MaxPooling2D(2)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='sigmoid')(x)
x = layers.MaxPooling2D(2)(x)

# Third convolution extracts 64 filters that are 3x3
```

```
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='sigmoid')(x)
x = layers.MaxPooling2D(2)(x)

# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units -␣
 ↪only adding dropout to this layer!
x = layers.Dense(512, activation='sigmoid')(x)

# Add a dropout rate of 0.5
x = layers.Dropout(0.5)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model_sigmoid_dropout = Model(img_input, output)
model_sigmoid_dropout.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

[ ]:
```
history_sigmoid_dropout = model_sigmoid_dropout.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

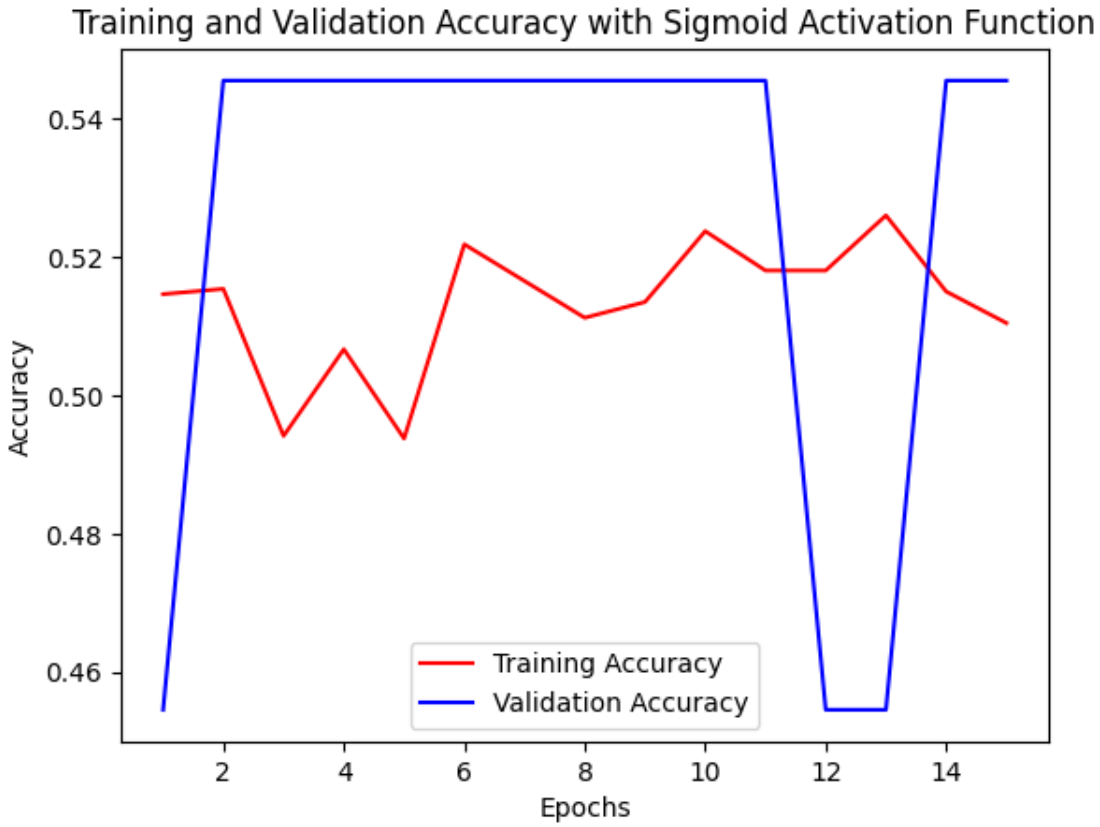
```
Epoch 1/15
132/132 - 101s - loss: 0.7987 - accuracy: 0.5146 - val_loss: 0.6982 -
val_accuracy: 0.4545 - 101s/epoch - 763ms/step
Epoch 2/15
132/132 - 98s - loss: 0.7675 - accuracy: 0.5154 - val_loss: 0.7199 -
val_accuracy: 0.5455 - 98s/epoch - 741ms/step
Epoch 3/15
132/132 - 99s - loss: 0.7652 - accuracy: 0.4941 - val_loss: 0.6985 -
val_accuracy: 0.5455 - 99s/epoch - 747ms/step
Epoch 4/15
132/132 - 98s - loss: 0.7340 - accuracy: 0.5066 - val_loss: 0.6895 -
val_accuracy: 0.5455 - 98s/epoch - 739ms/step
Epoch 5/15
```

```
132/132 - 101s - loss: 0.7363 - accuracy: 0.4937 - val_loss: 0.7026 -
val_accuracy: 0.5455 - 101s/epoch - 765ms/step
Epoch 6/15
132/132 - 105s - loss: 0.7156 - accuracy: 0.5218 - val_loss: 0.6901 -
val_accuracy: 0.5455 - 105s/epoch - 797ms/step
Epoch 7/15
132/132 - 103s - loss: 0.7133 - accuracy: 0.5165 - val_loss: 0.6934 -
val_accuracy: 0.5455 - 103s/epoch - 783ms/step
Epoch 8/15
132/132 - 101s - loss: 0.7084 - accuracy: 0.5112 - val_loss: 0.6913 -
val_accuracy: 0.5455 - 101s/epoch - 766ms/step
Epoch 9/15
132/132 - 101s - loss: 0.7094 - accuracy: 0.5135 - val_loss: 0.6899 -
val_accuracy: 0.5455 - 101s/epoch - 765ms/step
Epoch 10/15
132/132 - 105s - loss: 0.7083 - accuracy: 0.5237 - val_loss: 0.6921 -
val_accuracy: 0.5455 - 105s/epoch - 792ms/step
Epoch 11/15
132/132 - 99s - loss: 0.7001 - accuracy: 0.5180 - val_loss: 0.6894 -
val_accuracy: 0.5455 - 99s/epoch - 749ms/step
Epoch 12/15
132/132 - 100s - loss: 0.7021 - accuracy: 0.5180 - val_loss: 0.7003 -
val_accuracy: 0.4545 - 100s/epoch - 754ms/step
Epoch 13/15
132/132 - 103s - loss: 0.7031 - accuracy: 0.5260 - val_loss: 0.6976 -
val_accuracy: 0.4545 - 103s/epoch - 782ms/step
Epoch 14/15
132/132 - 101s - loss: 0.6989 - accuracy: 0.5150 - val_loss: 0.6931 -
val_accuracy: 0.5455 - 101s/epoch - 762ms/step
Epoch 15/15
132/132 - 97s - loss: 0.7018 - accuracy: 0.5104 - val_loss: 0.6890 -
val_accuracy: 0.5455 - 97s/epoch - 733ms/step
```

[ ]:
```
performance_dict = history_sigmoid_dropout.history
accuracy_values = performance_dict["accuracy"]
val_accuracy_values = performance_dict["val_accuracy"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color = 'red')
plt.plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
plt.title("Training and Validation Accuracy with Sigmoid Activation Function")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

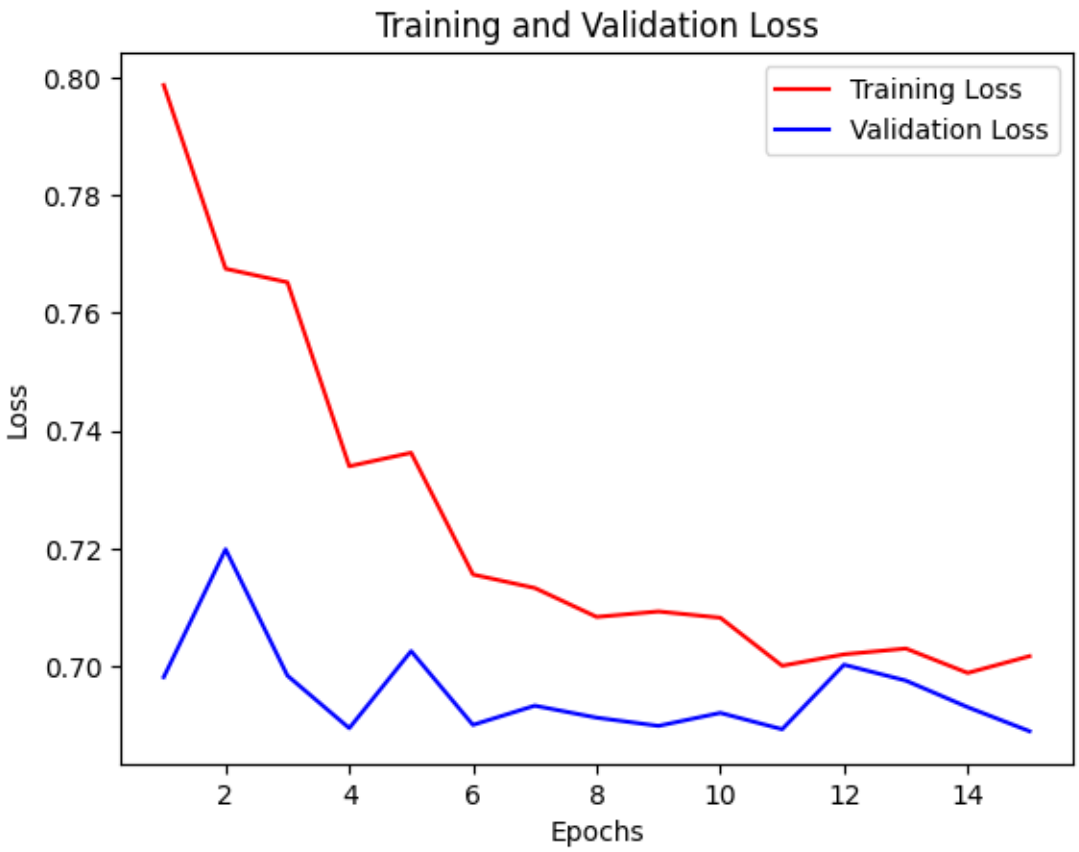
```
<ipython-input-44-18c0191afdb6>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
```

```
  plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color =
'red')
```



[ ]:
```
performance_dict = history_sigmoid_dropout.history
loss_values = performance_dict["loss"]
val_loss_values = performance_dict["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
plt.plot(epochs, val_loss_values, "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

```
<ipython-input-45-5486e1c78978>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
```



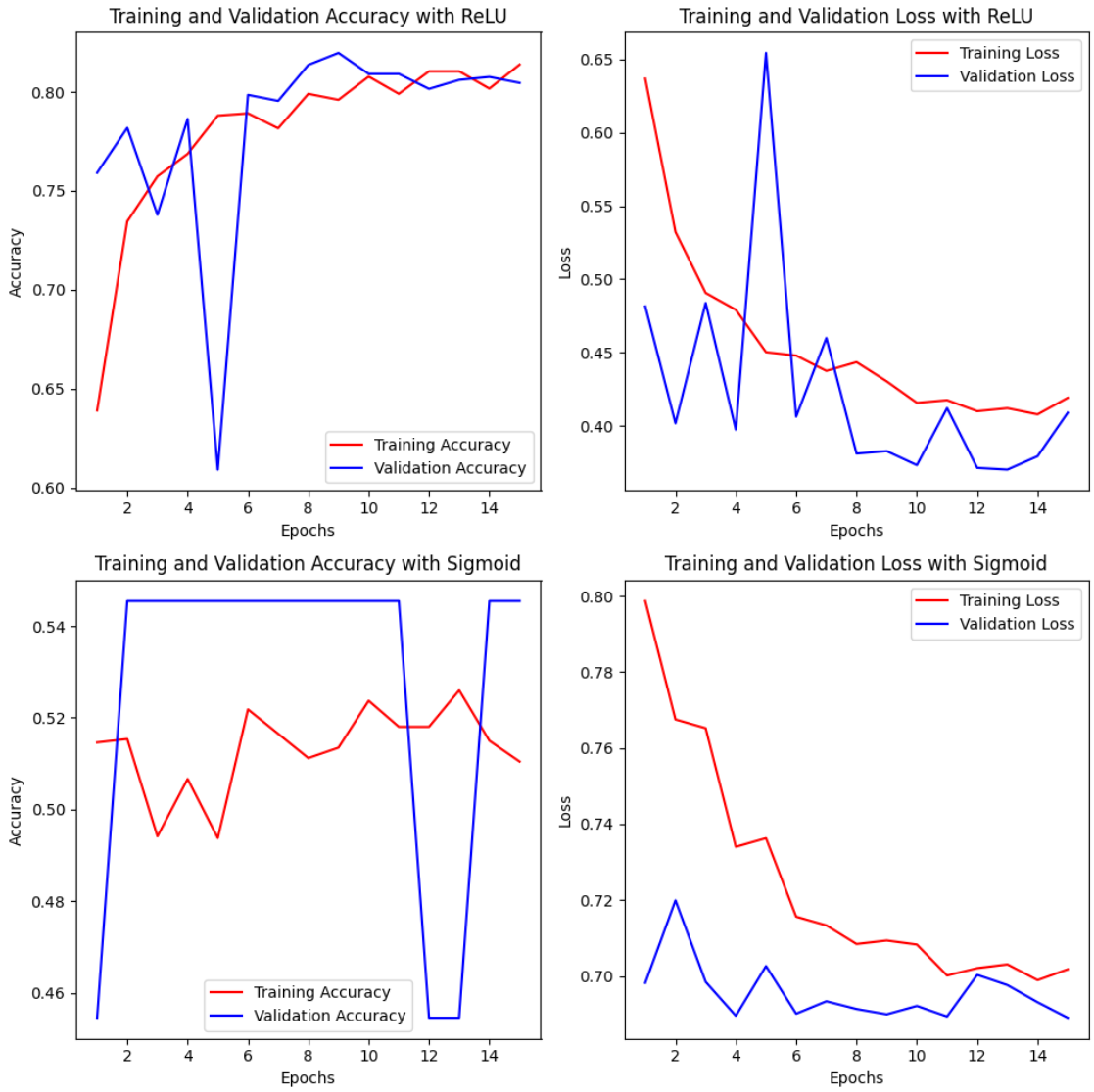
12 Second Model Accuracy and Loss Results

[ ]:
```
# Plot for the first model
performance_dict = history.history
accuracy_values = performance_dict["acc"]
val_accuracy_values = performance_dict["val_acc"]
loss_values = performance_dict["loss"]
val_loss_values = performance_dict["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

axs[0, 0].plot(epochs, accuracy_values, "b", label="Training Accuracy", color =␣
↪'red')
axs[0, 0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0, 0].set_title("Training and Validation Accuracy with ReLU")
axs[0, 0].set_xlabel("Epochs")
axs[0, 0].set_ylabel("Accuracy")
axs[0, 0].legend()
```

```
axs[0, 1].plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
axs[0, 1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[0, 1].set_title("Training and Validation Loss with ReLU")
axs[0, 1].set_xlabel("Epochs")
axs[0, 1].set_ylabel("Loss")
axs[0, 1].legend()


# Plot for the second model
performance_dict1 = history_sigmoid_dropout.history
accuracy_values1 = performance_dict1["accuracy"]
val_accuracy_values1 = performance_dict1["val_accuracy"]
loss_values1 = performance_dict1["loss"]
val_loss_values1 = performance_dict1["val_loss"]
epochs = range(1, len(loss_values1) + 1)

axs[1, 0].plot(epochs, accuracy_values1, "b", label="Training Accuracy", color␣
 ↪= 'red')
axs[1, 0].plot(epochs, val_accuracy_values1, "b", label="Validation Accuracy")
axs[1, 0].set_title("Training and Validation Accuracy with Sigmoid")
axs[1, 0].set_xlabel("Epochs")
axs[1, 0].set_ylabel("Accuracy")
axs[1, 0].legend()

axs[1, 1].plot(epochs, loss_values1, "b", label="Training Loss", color = 'red')
axs[1, 1].plot(epochs, val_loss_values1, "b", label="Validation Loss")
axs[1, 1].set_title("Training and Validation Loss with Sigmoid")
axs[1, 1].set_xlabel("Epochs")
axs[1, 1].set_ylabel("Loss")
axs[1, 1].legend()

plt.tight_layout()
plt.show()
```

```
<ipython-input-46-27d50f69120c>:11: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 0].plot(epochs, accuracy_values, "b", label="Training Accuracy", color
= 'red')
<ipython-input-46-27d50f69120c>:18: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 1].plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
<ipython-input-46-27d50f69120c>:34: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
```

```
  axs[1, 0].plot(epochs, accuracy_values1, "b", label="Training Accuracy", color
= 'red')
<ipython-input-46-27d50f69120c>:41: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 1].plot(epochs, loss_values1, "b", label="Training Loss", color =
'red')
```



13 Residual Block

[ ]:
```
# ReLU with Dropout of 0.2

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="relu", padding="same")(x)

    if pooling:
       x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
       residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
       residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x


x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)

x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)

x =residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)


x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(60, activation="relu")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model_Residual_Block_with_dropout_ReLU = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_with_dropout_ReLU.summary()

  # compile model
model_Residual_Block_with_dropout_ReLU.compile(loss= 'binary_crossentropy',␣
 ↪optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

```
Model: "model_6"
```

```
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_5 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 rescaling (Rescaling)          (None, 150, 150, 3)  0
['input_5[0][0]']

 batch_normalization (BatchNorm  (None, 150, 150, 3)  12
['rescaling[0][0]']
 alization)

 conv2d_12 (Conv2D)             (None, 150, 150, 64  1792
['batch_normalization[0][0]']
                                )

 max_pooling2d_12 (MaxPooling2D  (None, 75, 75, 64)  0
['conv2d_12[0][0]']
 )

 conv2d_13 (Conv2D)             (None, 75, 75, 64)   256
['rescaling[0][0]']

 add (Add)                      (None, 75, 75, 64)   0
['max_pooling2d_12[0][0]',
'conv2d_13[0][0]']

 dropout_2 (Dropout)            (None, 75, 75, 64)   0           ['add[0][0]']

 batch_normalization_1 (BatchNo  (None, 75, 75, 64)  256
['dropout_2[0][0]']
 rmalization)

 conv2d_14 (Conv2D)             (None, 75, 75, 64)   36928
['batch_normalization_1[0][0]']

 max_pooling2d_13 (MaxPooling2D  (None, 38, 38, 64)  0
['conv2d_14[0][0]']
 )

 conv2d_15 (Conv2D)             (None, 38, 38, 64)   4160
['dropout_2[0][0]']

 add_1 (Add)                    (None, 38, 38, 64)   0
['max_pooling2d_13[0][0]',
```

```
'conv2d_15[0][0]']

 dropout_3 (Dropout)            (None, 38, 38, 64)   0           ['add_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 38, 38, 64)  256
['dropout_3[0][0]']
 rmalization)

 conv2d_16 (Conv2D)             (None, 38, 38, 64)   36928
['batch_normalization_2[0][0]']

 max_pooling2d_14 (MaxPooling2D  (None, 19, 19, 64)  0
['conv2d_16[0][0]']
 )

 conv2d_17 (Conv2D)             (None, 19, 19, 64)   4160
['dropout_3[0][0]']

 add_2 (Add)                    (None, 19, 19, 64)   0
['max_pooling2d_14[0][0]',
'conv2d_17[0][0]']

 dropout_4 (Dropout)            (None, 19, 19, 64)   0           ['add_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 19, 19, 64)  256
['dropout_4[0][0]']
 rmalization)

 conv2d_18 (Conv2D)             (None, 19, 19, 16)   9232
['batch_normalization_3[0][0]']

 conv2d_19 (Conv2D)             (None, 19, 19, 16)   1040
['dropout_4[0][0]']

 add_3 (Add)                    (None, 19, 19, 16)   0
['conv2d_18[0][0]',
'conv2d_19[0][0]']

 flatten_4 (Flatten)            (None, 5776)         0           ['add_3[0][0]']

 dense_8 (Dense)                (None, 60)           346620
['flatten_4[0][0]']

 dense_9 (Dense)                (None, 1)            61
['dense_8[0][0]']

==================================================================================
==================
```

```
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
_________________________________________________________________________________
__________________
```

[ ]:
```
history_of_residualblock_withdropout_ReLU =␣
↪model_Residual_Block_with_dropout_ReLU.fit(
    train_generator,
    steps_per_epoch=None,
    epochs=15,
    validation_data=validation_generator,
    validation_steps=None,
    verbose=2)
```

```
Epoch 1/15
132/132 - 304s - loss: 0.8834 - accuracy: 0.7110 - val_loss: 0.7023 -
val_accuracy: 0.4545 - 304s/epoch - 2s/step
Epoch 2/15
132/132 - 292s - loss: 0.5320 - accuracy: 0.7455 - val_loss: 0.6945 -
val_accuracy: 0.4545 - 292s/epoch - 2s/step
Epoch 3/15
132/132 - 288s - loss: 0.4643 - accuracy: 0.7763 - val_loss: 0.6957 -
val_accuracy: 0.4545 - 288s/epoch - 2s/step
Epoch 4/15
132/132 - 283s - loss: 0.4633 - accuracy: 0.7797 - val_loss: 0.9385 -
val_accuracy: 0.4545 - 283s/epoch - 2s/step
Epoch 5/15
132/132 - 282s - loss: 0.4621 - accuracy: 0.7717 - val_loss: 0.6741 -
val_accuracy: 0.4742 - 282s/epoch - 2s/step
Epoch 6/15
132/132 - 277s - loss: 0.5037 - accuracy: 0.7766 - val_loss: 0.4621 -
val_accuracy: 0.7712 - 277s/epoch - 2s/step
Epoch 7/15
132/132 - 283s - loss: 0.4403 - accuracy: 0.7793 - val_loss: 0.5338 -
val_accuracy: 0.7576 - 283s/epoch - 2s/step
Epoch 8/15
132/132 - 280s - loss: 0.4286 - accuracy: 0.7808 - val_loss: 0.6601 -
val_accuracy: 0.4970 - 280s/epoch - 2s/step
Epoch 9/15
132/132 - 283s - loss: 0.4329 - accuracy: 0.7801 - val_loss: 0.4809 -
val_accuracy: 0.7682 - 283s/epoch - 2s/step
Epoch 10/15
132/132 - 279s - loss: 0.4273 - accuracy: 0.7823 - val_loss: 1.3832 -
val_accuracy: 0.4848 - 279s/epoch - 2s/step
Epoch 11/15
132/132 - 283s - loss: 0.4217 - accuracy: 0.7850 - val_loss: 0.4207 -
val_accuracy: 0.7636 - 283s/epoch - 2s/step
```

```
Epoch 12/15
132/132 - 278s - loss: 0.4477 - accuracy: 0.7884 - val_loss: 0.5916 -
val_accuracy: 0.6470 - 278s/epoch - 2s/step
Epoch 13/15
132/132 - 282s - loss: 0.4120 - accuracy: 0.7952 - val_loss: 0.6630 -
val_accuracy: 0.5212 - 282s/epoch - 2s/step
Epoch 14/15
132/132 - 280s - loss: 0.4276 - accuracy: 0.7967 - val_loss: 2.2313 -
val_accuracy: 0.5924 - 280s/epoch - 2s/step
Epoch 15/15
132/132 - 288s - loss: 0.4037 - accuracy: 0.7929 - val_loss: 0.5769 -
val_accuracy: 0.6394 - 288s/epoch - 2s/step
```

[ ]:
```
# Sigmoid with Dropout of 0.2

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="sigmoid",␣
 ↪padding="same")(x)

    if pooling:
       x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
       residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
       residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x


x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)

x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)

x =residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.2)(x)


x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
```

```
x = tf.keras.layers.Dense(60, activation="sigmoid")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model_Residual_Block_with_dropout_sigmoid = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_with_dropout_sigmoid.summary()

  # compile model
model_Residual_Block_with_dropout_sigmoid.compile(loss= 'binary_crossentropy',␣
 ↪optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

```
Model: "model_7"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_6 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 rescaling_1 (Rescaling)        (None, 150, 150, 3)  0
['input_6[0][0]']

 batch_normalization_4 (BatchNo  (None, 150, 150, 3)  12
['rescaling_1[0][0]']
 rmalization)

 conv2d_20 (Conv2D)             (None, 150, 150, 64  1792
['batch_normalization_4[0][0]']
                                )

 max_pooling2d_15 (MaxPooling2D  (None, 75, 75, 64)  0
['conv2d_20[0][0]']
 )

 conv2d_21 (Conv2D)             (None, 75, 75, 64)   256
['rescaling_1[0][0]']

 add_4 (Add)                    (None, 75, 75, 64)   0
['max_pooling2d_15[0][0]',
'conv2d_21[0][0]']

 dropout_5 (Dropout)            (None, 75, 75, 64)   0           ['add_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 75, 75, 64)  256
['dropout_5[0][0]']
```

```
 rmalization)

 conv2d_22 (Conv2D)             (None, 75, 75, 64)   36928
['batch_normalization_5[0][0]']

 max_pooling2d_16 (MaxPooling2D  (None, 38, 38, 64)  0
['conv2d_22[0][0]']
 )

 conv2d_23 (Conv2D)             (None, 38, 38, 64)   4160
['dropout_5[0][0]']

 add_5 (Add)                    (None, 38, 38, 64)   0
['max_pooling2d_16[0][0]',
'conv2d_23[0][0]']

 dropout_6 (Dropout)            (None, 38, 38, 64)   0           ['add_5[0][0]']

 batch_normalization_6 (BatchNo  (None, 38, 38, 64)  256
['dropout_6[0][0]']
 rmalization)

 conv2d_24 (Conv2D)             (None, 38, 38, 64)   36928
['batch_normalization_6[0][0]']

 max_pooling2d_17 (MaxPooling2D  (None, 19, 19, 64)  0
['conv2d_24[0][0]']
 )

 conv2d_25 (Conv2D)             (None, 19, 19, 64)   4160
['dropout_6[0][0]']

 add_6 (Add)                    (None, 19, 19, 64)   0
['max_pooling2d_17[0][0]',
'conv2d_25[0][0]']

 dropout_7 (Dropout)            (None, 19, 19, 64)   0           ['add_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 19, 19, 64)  256
['dropout_7[0][0]']
 rmalization)

 conv2d_26 (Conv2D)             (None, 19, 19, 16)   9232
['batch_normalization_7[0][0]']

 conv2d_27 (Conv2D)             (None, 19, 19, 16)   1040
['dropout_7[0][0]']
```

```
 add_7 (Add)                    (None, 19, 19, 16)   0
['conv2d_26[0][0]',
'conv2d_27[0][0]']

 flatten_5 (Flatten)            (None, 5776)         0           ['add_7[0][0]']

 dense_10 (Dense)               (None, 60)           346620
['flatten_5[0][0]']

 dense_11 (Dense)               (None, 1)            61
['dense_10[0][0]']

==================================================================================
==================
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
----------------------------------------------------------------------------------
------------------
```

[ ]:
```
history_residualblock_withdropout_sigmoid =␣
  ↪model_Residual_Block_with_dropout_sigmoid.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 301s - loss: 0.6999 - accuracy: 0.5419 - val_loss: 0.6904 -
val_accuracy: 0.5455 - 301s/epoch - 2s/step
Epoch 2/15
132/132 - 293s - loss: 0.6905 - accuracy: 0.5461 - val_loss: 0.6890 -
val_accuracy: 0.5455 - 293s/epoch - 2s/step
Epoch 3/15
132/132 - 297s - loss: 0.6892 - accuracy: 0.5438 - val_loss: 0.6914 -
val_accuracy: 0.5455 - 297s/epoch - 2s/step
Epoch 4/15
132/132 - 292s - loss: 0.6903 - accuracy: 0.5438 - val_loss: 0.6907 -
val_accuracy: 0.5455 - 292s/epoch - 2s/step
Epoch 5/15
132/132 - 299s - loss: 0.6893 - accuracy: 0.5415 - val_loss: 0.6937 -
val_accuracy: 0.5455 - 299s/epoch - 2s/step
Epoch 6/15
132/132 - 301s - loss: 0.6900 - accuracy: 0.5438 - val_loss: 0.6891 -
val_accuracy: 0.5455 - 301s/epoch - 2s/step
Epoch 7/15
```

```
132/132 - 302s - loss: 0.6899 - accuracy: 0.5438 - val_loss: 0.6905 -
val_accuracy: 0.5455 - 302s/epoch - 2s/step
Epoch 8/15
132/132 - 297s - loss: 0.6904 - accuracy: 0.5461 - val_loss: 0.6894 -
val_accuracy: 0.5455 - 297s/epoch - 2s/step
Epoch 9/15
132/132 - 298s - loss: 0.6902 - accuracy: 0.5461 - val_loss: 0.6890 -
val_accuracy: 0.5455 - 298s/epoch - 2s/step
Epoch 10/15
132/132 - 298s - loss: 0.6897 - accuracy: 0.5370 - val_loss: 0.6890 -
val_accuracy: 0.5455 - 298s/epoch - 2s/step
Epoch 11/15
132/132 - 297s - loss: 0.6899 - accuracy: 0.5461 - val_loss: 0.6899 -
val_accuracy: 0.5455 - 297s/epoch - 2s/step
Epoch 12/15
132/132 - 300s - loss: 0.6900 - accuracy: 0.5461 - val_loss: 0.6892 -
val_accuracy: 0.5455 - 300s/epoch - 2s/step
Epoch 13/15
132/132 - 294s - loss: 0.6904 - accuracy: 0.5461 - val_loss: 0.6892 -
val_accuracy: 0.5455 - 294s/epoch - 2s/step
Epoch 14/15
132/132 - 302s - loss: 0.6902 - accuracy: 0.5461 - val_loss: 0.6891 -
val_accuracy: 0.5455 - 302s/epoch - 2s/step
Epoch 15/15
132/132 - 303s - loss: 0.6903 - accuracy: 0.5461 - val_loss: 0.6901 -
val_accuracy: 0.5455 - 303s/epoch - 2s/step
```

[ ]:
```python
# Sigmoid

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="sigmoid",␣
 ↪padding="same")(x)

    if pooling:
        x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
        residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
        residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x
```

```
x= residual_block(x, filters=64, pooling=True)

x= residual_block(x, filters=64, pooling=True)

x =residual_block(x, filters=64, pooling=True)

x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(60, activation="sigmoid")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model_Residual_Block_without_dropout_sigmoid = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_without_dropout_sigmoid.summary()

  # compile model
model_Residual_Block_without_dropout_sigmoid.compile(loss=␣
 ↪'binary_crossentropy', optimizer=RMSprop(learning_rate=0.001),␣
 ↪metrics=['accuracy'])
```

```
Model: "model_8"
_________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================
==================
 input_7 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 rescaling_2 (Rescaling)        (None, 150, 150, 3)  0
['input_7[0][0]']

 batch_normalization_8 (BatchNo  (None, 150, 150, 3)  12
['rescaling_2[0][0]']
 rmalization)

 conv2d_28 (Conv2D)             (None, 150, 150, 64  1792
['batch_normalization_8[0][0]']
                                )

 max_pooling2d_18 (MaxPooling2D  (None, 75, 75, 64)  0
['conv2d_28[0][0]']
 )
```

```
 conv2d_29 (Conv2D)             (None, 75, 75, 64)   256
['rescaling_2[0][0]']

 add_8 (Add)                    (None, 75, 75, 64)   0
['max_pooling2d_18[0][0]',
'conv2d_29[0][0]']

 batch_normalization_9 (BatchNo  (None, 75, 75, 64)  256         ['add_8[0][0]']
 rmalization)

 conv2d_30 (Conv2D)             (None, 75, 75, 64)   36928
['batch_normalization_9[0][0]']

 max_pooling2d_19 (MaxPooling2D  (None, 38, 38, 64)  0
['conv2d_30[0][0]']
 )

 conv2d_31 (Conv2D)             (None, 38, 38, 64)   4160        ['add_8[0][0]']

 add_9 (Add)                    (None, 38, 38, 64)   0
['max_pooling2d_19[0][0]',
'conv2d_31[0][0]']

 batch_normalization_10 (BatchN  (None, 38, 38, 64)  256         ['add_9[0][0]']
 ormalization)

 conv2d_32 (Conv2D)             (None, 38, 38, 64)   36928
['batch_normalization_10[0][0]']

 max_pooling2d_20 (MaxPooling2D  (None, 19, 19, 64)  0
['conv2d_32[0][0]']
 )

 conv2d_33 (Conv2D)             (None, 19, 19, 64)   4160        ['add_9[0][0]']

 add_10 (Add)                   (None, 19, 19, 64)   0
['max_pooling2d_20[0][0]',
'conv2d_33[0][0]']

 batch_normalization_11 (BatchN  (None, 19, 19, 64)  256
['add_10[0][0]']
 ormalization)

 conv2d_34 (Conv2D)             (None, 19, 19, 16)   9232
['batch_normalization_11[0][0]']

 conv2d_35 (Conv2D)             (None, 19, 19, 16)   1040
['add_10[0][0]']
```

```
 add_11 (Add)                   (None, 19, 19, 16)   0
['conv2d_34[0][0]',
'conv2d_35[0][0]']

 flatten_6 (Flatten)            (None, 5776)         0
['add_11[0][0]']

 dense_12 (Dense)               (None, 60)           346620
['flatten_6[0][0]']

 dense_13 (Dense)               (None, 1)            61
['dense_12[0][0]']

==================================================================================
==================
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
----------------------------------------------------------------------------------
------------------
```

[ ]:
```
history_residualblock_withoutdropout_sigmoid =␣
  ↪model_Residual_Block_without_dropout_sigmoid.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 291s - loss: 0.6110 - accuracy: 0.7300 - val_loss: 0.8388 -
val_accuracy: 0.4545 - 291s/epoch - 2s/step
Epoch 2/15
132/132 - 277s - loss: 0.4832 - accuracy: 0.7717 - val_loss: 0.8784 -
val_accuracy: 0.4545 - 277s/epoch - 2s/step
Epoch 3/15
132/132 - 279s - loss: 0.4790 - accuracy: 0.7706 - val_loss: 1.1261 -
val_accuracy: 0.5455 - 279s/epoch - 2s/step
Epoch 4/15
132/132 - 282s - loss: 0.4641 - accuracy: 0.7766 - val_loss: 0.9211 -
val_accuracy: 0.5455 - 282s/epoch - 2s/step
Epoch 5/15
132/132 - 286s - loss: 0.4511 - accuracy: 0.7823 - val_loss: 0.8984 -
val_accuracy: 0.5500 - 286s/epoch - 2s/step
Epoch 6/15
132/132 - 282s - loss: 0.4587 - accuracy: 0.7782 - val_loss: 0.6975 -
```

```
val_accuracy: 0.6258 - 282s/epoch - 2s/step
Epoch 7/15
132/132 - 279s - loss: 0.4397 - accuracy: 0.7922 - val_loss: 1.5736 -
val_accuracy: 0.5455 - 279s/epoch - 2s/step
Epoch 8/15
132/132 - 272s - loss: 0.4507 - accuracy: 0.7804 - val_loss: 0.7293 -
val_accuracy: 0.6455 - 272s/epoch - 2s/step
Epoch 9/15
132/132 - 281s - loss: 0.4362 - accuracy: 0.7880 - val_loss: 1.3961 -
val_accuracy: 0.5455 - 281s/epoch - 2s/step
Epoch 10/15
132/132 - 284s - loss: 0.4303 - accuracy: 0.7911 - val_loss: 0.7190 -
val_accuracy: 0.5848 - 284s/epoch - 2s/step
Epoch 11/15
132/132 - 282s - loss: 0.4298 - accuracy: 0.7983 - val_loss: 0.7422 -
val_accuracy: 0.5955 - 282s/epoch - 2s/step
Epoch 12/15
132/132 - 284s - loss: 0.4238 - accuracy: 0.7986 - val_loss: 1.4261 -
val_accuracy: 0.5894 - 284s/epoch - 2s/step
Epoch 13/15
132/132 - 283s - loss: 0.4299 - accuracy: 0.7876 - val_loss: 1.3047 -
val_accuracy: 0.5591 - 283s/epoch - 2s/step
Epoch 14/15
132/132 - 280s - loss: 0.4271 - accuracy: 0.7861 - val_loss: 0.5250 -
val_accuracy: 0.7955 - 280s/epoch - 2s/step
Epoch 15/15
132/132 - 288s - loss: 0.4224 - accuracy: 0.7964 - val_loss: 0.7073 -
val_accuracy: 0.6803 - 288s/epoch - 2s/step
```

[ ]:
```python
# ReLU

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="relu", padding="same")(x)

    if pooling:
       x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
       residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
       residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x
```

```
x= residual_block(x, filters=64, pooling=True)

x= residual_block(x, filters=64, pooling=True)

x =residual_block(x, filters=64, pooling=True)

x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(60, activation="relu")(x)

outputs = tf.keras.layers.Dense(1, activation="relu")(x)

model_Residual_Block_without_dropout_ReLU = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_without_dropout_ReLU.summary()

  # compile model
model_Residual_Block_without_dropout_ReLU.compile(loss= 'binary_crossentropy',␣
 ↪optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

```
Model: "model_9"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_8 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 rescaling_3 (Rescaling)        (None, 150, 150, 3)  0
['input_8[0][0]']

 batch_normalization_12 (BatchN  (None, 150, 150, 3)  12
['rescaling_3[0][0]']
 ormalization)

 conv2d_36 (Conv2D)             (None, 150, 150, 64  1792
['batch_normalization_12[0][0]']
                                )

 max_pooling2d_21 (MaxPooling2D  (None, 75, 75, 64)  0
['conv2d_36[0][0]']
 )
```

```
 conv2d_37 (Conv2D)             (None, 75, 75, 64)   256
['rescaling_3[0][0]']

 add_12 (Add)                   (None, 75, 75, 64)   0
['max_pooling2d_21[0][0]',
'conv2d_37[0][0]']

 batch_normalization_13 (BatchN  (None, 75, 75, 64)  256
['add_12[0][0]']
 ormalization)

 conv2d_38 (Conv2D)             (None, 75, 75, 64)   36928
['batch_normalization_13[0][0]']

 max_pooling2d_22 (MaxPooling2D  (None, 38, 38, 64)  0
['conv2d_38[0][0]']
 )

 conv2d_39 (Conv2D)             (None, 38, 38, 64)   4160
['add_12[0][0]']

 add_13 (Add)                   (None, 38, 38, 64)   0
['max_pooling2d_22[0][0]',
'conv2d_39[0][0]']

 batch_normalization_14 (BatchN  (None, 38, 38, 64)  256
['add_13[0][0]']
 ormalization)

 conv2d_40 (Conv2D)             (None, 38, 38, 64)   36928
['batch_normalization_14[0][0]']

 max_pooling2d_23 (MaxPooling2D  (None, 19, 19, 64)  0
['conv2d_40[0][0]']
 )

 conv2d_41 (Conv2D)             (None, 19, 19, 64)   4160
['add_13[0][0]']

 add_14 (Add)                   (None, 19, 19, 64)   0
['max_pooling2d_23[0][0]',
'conv2d_41[0][0]']

 batch_normalization_15 (BatchN  (None, 19, 19, 64)  256
['add_14[0][0]']
 ormalization)

 conv2d_42 (Conv2D)             (None, 19, 19, 16)   9232
```

```
['batch_normalization_15[0][0]']

 conv2d_43 (Conv2D)             (None, 19, 19, 16)   1040
['add_14[0][0]']

 add_15 (Add)                   (None, 19, 19, 16)   0
['conv2d_42[0][0]',
'conv2d_43[0][0]']

 flatten_7 (Flatten)            (None, 5776)         0
['add_15[0][0]']

 dense_14 (Dense)               (None, 60)           346620
['flatten_7[0][0]']

 dense_15 (Dense)               (None, 1)            61
['dense_14[0][0]']

==================================================================================
==================
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
----------------------------------------------------------------------------------
------------------
```

[ ]:
```
history_residualblock_withoutdropout_ReLU =␣
 ↪model_Residual_Block_without_dropout_ReLU.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=15,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/15
132/132 - 278s - loss: 7.1767 - accuracy: 0.5324 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 278s/epoch - 2s/step
Epoch 2/15
132/132 - 266s - loss: 6.5837 - accuracy: 0.5715 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 266s/epoch - 2s/step
Epoch 3/15
132/132 - 266s - loss: 6.4857 - accuracy: 0.5791 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 266s/epoch - 2s/step
Epoch 4/15
132/132 - 267s - loss: 6.8850 - accuracy: 0.5533 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 267s/epoch - 2s/step
Epoch 5/15
```

```
132/132 - 268s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 268s/epoch - 2s/step
Epoch 6/15
132/132 - 272s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 272s/epoch - 2s/step
Epoch 7/15
132/132 - 269s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 269s/epoch - 2s/step
Epoch 8/15
132/132 - 267s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 267s/epoch - 2s/step
Epoch 9/15
132/132 - 264s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 264s/epoch - 2s/step
Epoch 10/15
132/132 - 260s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 260s/epoch - 2s/step
Epoch 11/15
132/132 - 268s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 268s/epoch - 2s/step
Epoch 12/15
132/132 - 267s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 267s/epoch - 2s/step
Epoch 13/15
132/132 - 264s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 264s/epoch - 2s/step
Epoch 14/15
132/132 - 267s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 267s/epoch - 2s/step
Epoch 15/15
132/132 - 265s - loss: 7.0018 - accuracy: 0.5461 - val_loss: 7.0113 -
val_accuracy: 0.5455 - 265s/epoch - 2s/step
```

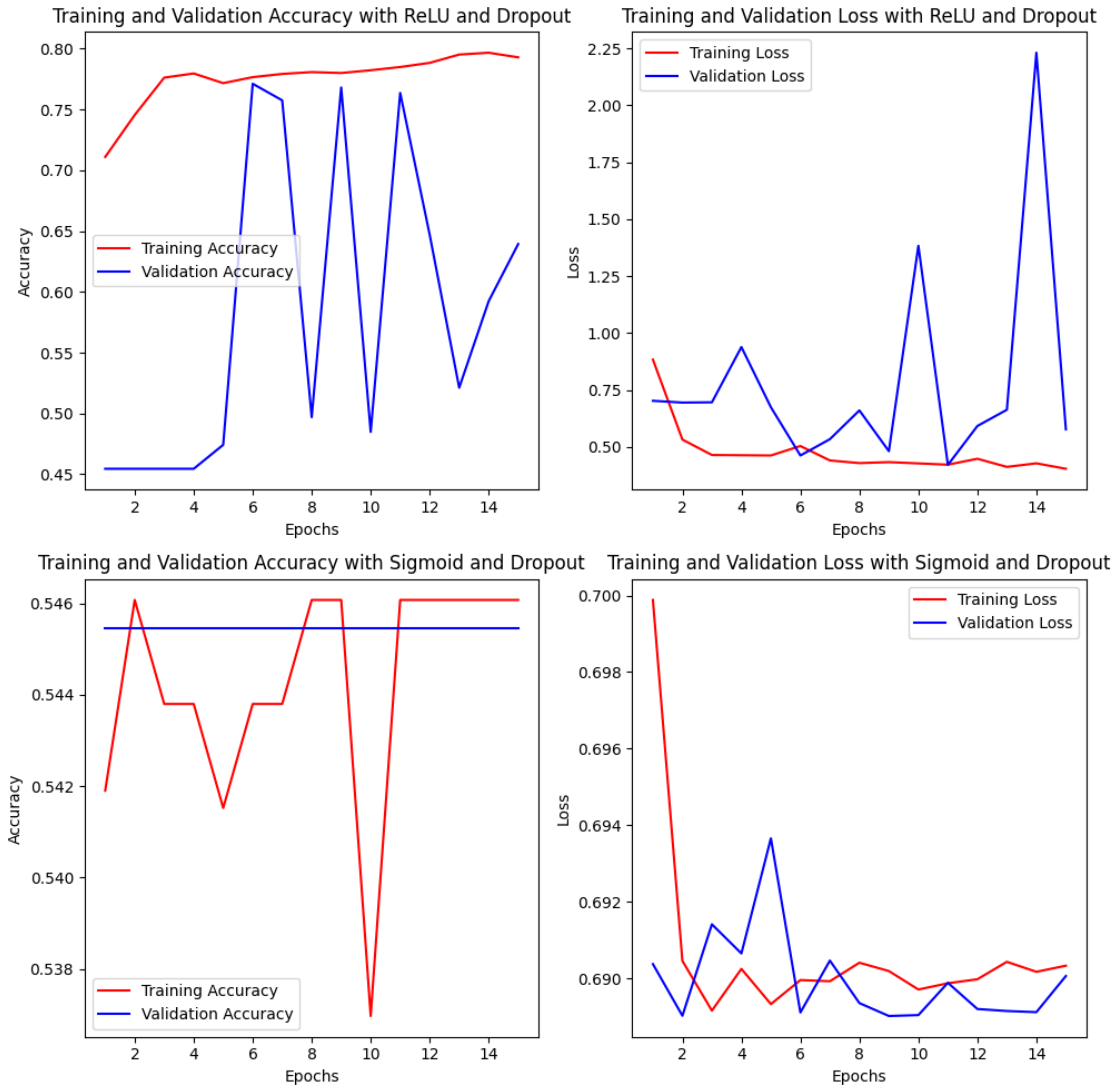
14 Third Model Accuracy and Loss Results

[ ]:
```
# ReLU and Sigmoid with Dropout
performance_dict_of_ReLU_residualblock_dropout =␣
  ↪history_of_residualblock_withdropout_ReLU.history
accuracy_values_ReLU_residualblock_dropout =␣
  ↪performance_dict_of_ReLU_residualblock_dropout["accuracy"]
val_accuracy_values_ReLU_residualblock_dropout =␣
  ↪performance_dict_of_ReLU_residualblock_dropout["val_accuracy"]
loss_values_ReLU_residualblock_dropout =␣
  ↪performance_dict_of_ReLU_residualblock_dropout["loss"]
val_loss_values_ReLU_residualblock_dropout =␣
  ↪performance_dict_of_ReLU_residualblock_dropout["val_loss"]
```

```
epochs = range(1, len(accuracy_values_ReLU_residualblock_dropout) + 1)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))

axs[0, 0].plot(epochs, accuracy_values_ReLU_residualblock_dropout, "b",␣
 ↪label="Training Accuracy", color = 'red')
axs[0, 0].plot(epochs, val_accuracy_values_ReLU_residualblock_dropout, "b",␣
 ↪label="Validation Accuracy")
axs[0, 0].set_title("Training and Validation Accuracy with ReLU and Dropout")
axs[0, 0].set_xlabel("Epochs")
axs[0, 0].set_ylabel("Accuracy")
axs[0, 0].legend()

axs[0, 1].plot(epochs, loss_values_ReLU_residualblock_dropout, "b",␣
 ↪label="Training Loss", color = 'red')
axs[0, 1].plot(epochs, val_loss_values_ReLU_residualblock_dropout, "b",␣
 ↪label="Validation Loss")
axs[0, 1].set_title("Training and Validation Loss with ReLU and Dropout")
axs[0, 1].set_xlabel("Epochs")
axs[0, 1].set_ylabel("Loss")
axs[0, 1].legend()


# Plot for the second model
performance_dict_residualblock_withdropout_sigmoid =␣
 ↪history_residualblock_withdropout_sigmoid.history
accuracy_values_performance_dict_residualblock_withdropout_sigmoid =␣
 ↪performance_dict_residualblock_withdropout_sigmoid["accuracy"]
val_accuracy_values_performance_dict_residualblock_withdropout_sigmoid =␣
 ↪performance_dict_residualblock_withdropout_sigmoid["val_accuracy"]
loss_values_performance_dict_residualblock_withdropout_sigmoid =␣
 ↪performance_dict_residualblock_withdropout_sigmoid["loss"]
val_loss_values_performance_dict_residualblock_withdropout_sigmoid =␣
 ↪performance_dict_residualblock_withdropout_sigmoid["val_loss"]
epochs = range(1, len(loss_values1) + 1)

axs[1, 0].plot(epochs,␣
 ↪accuracy_values_performance_dict_residualblock_withdropout_sigmoid, "b",␣
 ↪label="Training Accuracy", color = 'red')
axs[1, 0].plot(epochs,␣
 ↪val_accuracy_values_performance_dict_residualblock_withdropout_sigmoid, "b",␣
 ↪label="Validation Accuracy")
axs[1, 0].set_title("Training and Validation Accuracy with Sigmoid and Dropout")
axs[1, 0].set_xlabel("Epochs")
axs[1, 0].set_ylabel("Accuracy")
axs[1, 0].legend()
```

```
axs[1, 1].plot(epochs,␣
 ↪loss_values_performance_dict_residualblock_withdropout_sigmoid, "b",␣
 ↪label="Training Loss", color = 'red')
axs[1, 1].plot(epochs,␣
 ↪val_loss_values_performance_dict_residualblock_withdropout_sigmoid, "b",␣
 ↪label="Validation Loss")
axs[1, 1].set_title("Training and Validation Loss with Sigmoid and Dropout")
axs[1, 1].set_xlabel("Epochs")
axs[1, 1].set_ylabel("Loss")
axs[1, 1].legend()

plt.tight_layout()
plt.show()
```

```
<ipython-input-59-f1639bd9eb3b>:11: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 0].plot(epochs, accuracy_values_ReLU_residualblock_dropout, "b",
label="Training Accuracy", color = 'red')
<ipython-input-59-f1639bd9eb3b>:18: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 1].plot(epochs, loss_values_ReLU_residualblock_dropout, "b",
label="Training Loss", color = 'red')
<ipython-input-59-f1639bd9eb3b>:34: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 0].plot(epochs,
accuracy_values_performance_dict_residualblock_withdropout_sigmoid, "b",
label="Training Accuracy", color = 'red')
<ipython-input-59-f1639bd9eb3b>:41: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 1].plot(epochs,
loss_values_performance_dict_residualblock_withdropout_sigmoid, "b",
label="Training Loss", color = 'red')
```

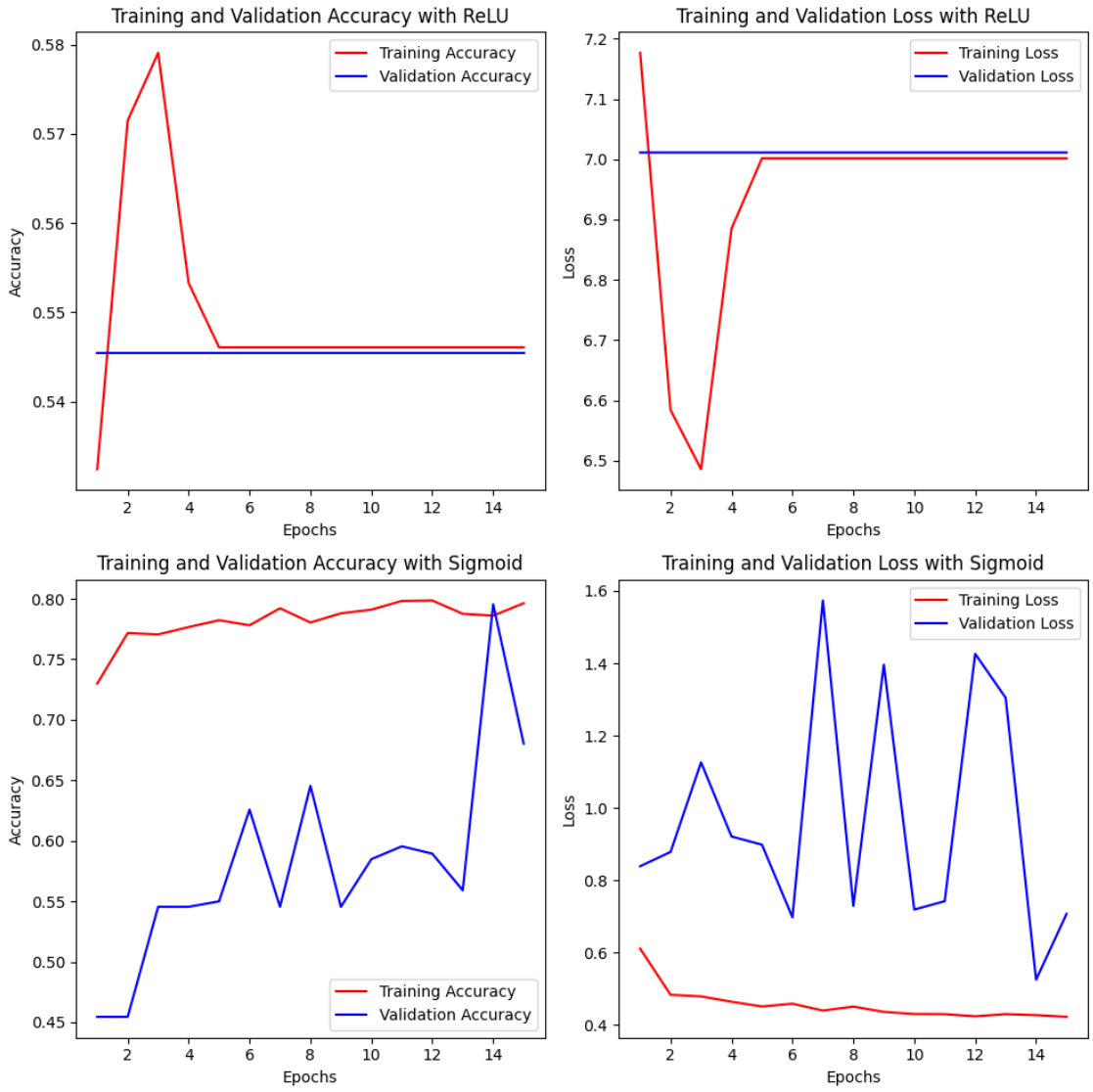


[ ]:
```
# ReLU and Sigmoid without Dropout
performance_dict_of_ReLU_residualblock =␣
 ↪history_residualblock_withoutdropout_ReLU.history
accuracy_values_ReLU_residualblock =␣
 ↪performance_dict_of_ReLU_residualblock["accuracy"]
val_accuracy_values_ReLU_residualblock =␣
 ↪performance_dict_of_ReLU_residualblock["val_accuracy"]
loss_values_ReLU_residualblock = performance_dict_of_ReLU_residualblock["loss"]
val_loss_values_ReLU_residualblock =␣
 ↪performance_dict_of_ReLU_residualblock["val_loss"]
epochs = range(1, len(accuracy_values_ReLU_residualblock) + 1)

fig, axs = plt.subplots(2, 2, figsize=(10, 10))
```

```
axs[0, 0].plot(epochs, accuracy_values_ReLU_residualblock, "b", label="Training␣
 ↪Accuracy", color = 'red')
axs[0, 0].plot(epochs, val_accuracy_values_ReLU_residualblock, "b",␣
 ↪label="Validation Accuracy")
axs[0, 0].set_title("Training and Validation Accuracy with ReLU")
axs[0, 0].set_xlabel("Epochs")
axs[0, 0].set_ylabel("Accuracy")
axs[0, 0].legend()

axs[0, 1].plot(epochs, loss_values_ReLU_residualblock, "b", label="Training␣
 ↪Loss", color = 'red')
axs[0, 1].plot(epochs, val_loss_values_ReLU_residualblock, "b",␣
 ↪label="Validation Loss")
axs[0, 1].set_title("Training and Validation Loss with ReLU")
axs[0, 1].set_xlabel("Epochs")
axs[0, 1].set_ylabel("Loss")
axs[0, 1].legend()


# Plot for the second model
performance_dict_residualblock_sigmoid =␣
 ↪history_residualblock_withoutdropout_sigmoid.history
accuracy_values_performance_dict_residualblock_sigmoid =␣
 ↪performance_dict_residualblock_sigmoid["accuracy"]
val_accuracy_values_performance_dict_residualblock_sigmoid =␣
 ↪performance_dict_residualblock_sigmoid["val_accuracy"]
loss_values_performance_dict_residualblock_sigmoid =␣
 ↪performance_dict_residualblock_sigmoid["loss"]
val_loss_values_performance_dict_residualblock_sigmoid =␣
 ↪performance_dict_residualblock_sigmoid["val_loss"]
epochs = range(1, len(loss_values1) + 1)

axs[1, 0].plot(epochs, accuracy_values_performance_dict_residualblock_sigmoid,␣
 ↪"b", label="Training Accuracy", color = 'red')
axs[1, 0].plot(epochs,␣
 ↪val_accuracy_values_performance_dict_residualblock_sigmoid, "b",␣
 ↪label="Validation Accuracy")
axs[1, 0].set_title("Training and Validation Accuracy with Sigmoid")
axs[1, 0].set_xlabel("Epochs")
axs[1, 0].set_ylabel("Accuracy")
axs[1, 0].legend()

axs[1, 1].plot(epochs, loss_values_performance_dict_residualblock_sigmoid, "b",␣
 ↪label="Training Loss", color = 'red')
```

```
axs[1, 1].plot(epochs, val_loss_values_performance_dict_residualblock_sigmoid,␣
 ↪"b", label="Validation Loss")
axs[1, 1].set_title("Training and Validation Loss with Sigmoid")
axs[1, 1].set_xlabel("Epochs")
axs[1, 1].set_ylabel("Loss")
axs[1, 1].legend()

plt.tight_layout()
plt.show()
```

```
<ipython-input-60-fd07f846d270>:11: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 0].plot(epochs, accuracy_values_ReLU_residualblock, "b",
label="Training Accuracy", color = 'red')
<ipython-input-60-fd07f846d270>:18: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0, 1].plot(epochs, loss_values_ReLU_residualblock, "b", label="Training
Loss", color = 'red')
<ipython-input-60-fd07f846d270>:34: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 0].plot(epochs, accuracy_values_performance_dict_residualblock_sigmoid,
"b", label="Training Accuracy", color = 'red')
<ipython-input-60-fd07f846d270>:41: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1, 1].plot(epochs, loss_values_performance_dict_residualblock_sigmoid,
"b", label="Training Loss", color = 'red')
```



15 Fine Tuning ReLU Models to Select Best Model

[ ]:
```
#ReLU

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)
```

```
x = layers.Dropout(0.05)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)


# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units ¬␣
 ↪added dropout
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model_self_selected_best = Model(img_input, output)
model_self_selected_best.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

[ ]:
```
model_self_selected_best.summary()
```

```
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_9 (InputLayer)        [(None, 150, 150, 3)]     0

 conv2d_44 (Conv2D)          (None, 148, 148, 16)      448
```

```
 max_pooling2d_24 (MaxPoolin  (None, 74, 74, 16)       0
 g2D)

 dropout_8 (Dropout)         (None, 74, 74, 16)        0

 conv2d_45 (Conv2D)          (None, 72, 72, 32)        4640

 max_pooling2d_25 (MaxPoolin  (None, 36, 36, 32)       0
 g2D)

 dropout_9 (Dropout)         (None, 36, 36, 32)        0

 conv2d_46 (Conv2D)          (None, 34, 34, 64)        18496

 max_pooling2d_26 (MaxPoolin  (None, 17, 17, 64)       0
 g2D)

 dropout_10 (Dropout)        (None, 17, 17, 64)        0

 flatten_8 (Flatten)         (None, 18496)             0

 dense_16 (Dense)            (None, 512)               9470464

 dropout_11 (Dropout)        (None, 512)               0

 dense_17 (Dense)            (None, 1)                 513

=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
history_self_selected_best_model = model_self_selected_best.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/20
132/132 - 148s - loss: 0.6006 - accuracy: 0.6936 - val_loss: 0.4888 -
val_accuracy: 0.7712 - 148s/epoch - 1s/step
Epoch 2/20
132/132 - 107s - loss: 0.4925 - accuracy: 0.7539 - val_loss: 0.4342 -
val_accuracy: 0.7955 - 107s/epoch - 814ms/step
Epoch 3/20
```

```
132/132 - 102s - loss: 0.4733 - accuracy: 0.7592 - val_loss: 0.4192 -
val_accuracy: 0.7955 - 102s/epoch - 771ms/step
Epoch 4/20
132/132 - 102s - loss: 0.4507 - accuracy: 0.7778 - val_loss: 0.4113 -
val_accuracy: 0.7909 - 102s/epoch - 771ms/step
Epoch 5/20
132/132 - 101s - loss: 0.4484 - accuracy: 0.7774 - val_loss: 0.4077 -
val_accuracy: 0.7879 - 101s/epoch - 766ms/step
Epoch 6/20
132/132 - 101s - loss: 0.4311 - accuracy: 0.7948 - val_loss: 0.3826 -
val_accuracy: 0.8030 - 101s/epoch - 763ms/step
Epoch 7/20
132/132 - 96s - loss: 0.4296 - accuracy: 0.7941 - val_loss: 0.4036 -
val_accuracy: 0.7955 - 96s/epoch - 728ms/step
Epoch 8/20
132/132 - 97s - loss: 0.4278 - accuracy: 0.7922 - val_loss: 0.3890 -
val_accuracy: 0.8076 - 97s/epoch - 737ms/step
Epoch 9/20
132/132 - 104s - loss: 0.4231 - accuracy: 0.7933 - val_loss: 0.3895 -
val_accuracy: 0.7894 - 104s/epoch - 789ms/step
Epoch 10/20
132/132 - 111s - loss: 0.4234 - accuracy: 0.7998 - val_loss: 0.3644 -
val_accuracy: 0.8212 - 111s/epoch - 843ms/step
Epoch 11/20
132/132 - 96s - loss: 0.4222 - accuracy: 0.8093 - val_loss: 0.3692 -
val_accuracy: 0.8091 - 96s/epoch - 727ms/step
Epoch 12/20
132/132 - 103s - loss: 0.4102 - accuracy: 0.8013 - val_loss: 0.4555 -
val_accuracy: 0.7545 - 103s/epoch - 781ms/step
Epoch 13/20
132/132 - 107s - loss: 0.4035 - accuracy: 0.8039 - val_loss: 0.3621 -
val_accuracy: 0.8273 - 107s/epoch - 811ms/step
Epoch 14/20
132/132 - 104s - loss: 0.4005 - accuracy: 0.8096 - val_loss: 0.3860 -
val_accuracy: 0.8152 - 104s/epoch - 784ms/step
Epoch 15/20
132/132 - 97s - loss: 0.3931 - accuracy: 0.8187 - val_loss: 0.3585 -
val_accuracy: 0.8136 - 97s/epoch - 735ms/step
Epoch 16/20
132/132 - 103s - loss: 0.4076 - accuracy: 0.8043 - val_loss: 0.3777 -
val_accuracy: 0.7894 - 103s/epoch - 784ms/step
Epoch 17/20
132/132 - 102s - loss: 0.4168 - accuracy: 0.8009 - val_loss: 0.3831 -
val_accuracy: 0.8288 - 102s/epoch - 773ms/step
Epoch 18/20
132/132 - 104s - loss: 0.3857 - accuracy: 0.8130 - val_loss: 0.3695 -
val_accuracy: 0.8258 - 104s/epoch - 784ms/step
Epoch 19/20
```

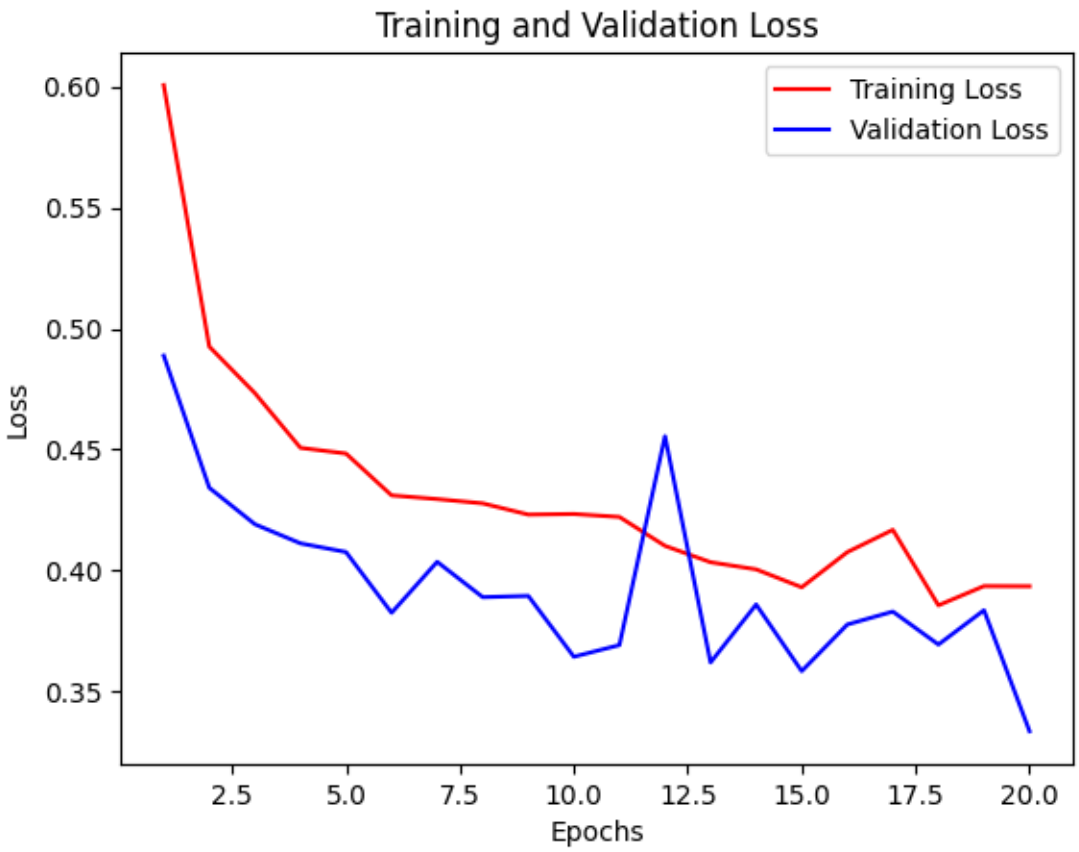
```
132/132 - 105s - loss: 0.3936 - accuracy: 0.8081 - val_loss: 0.3836 -
val_accuracy: 0.8000 - 105s/epoch - 794ms/step
Epoch 20/20
132/132 - 97s - loss: 0.3935 - accuracy: 0.8142 - val_loss: 0.3336 -
val_accuracy: 0.8258 - 97s/epoch - 735ms/step
```

[ ]:
```
performance_dict_self_selected_best_model = history_self_selected_best_model.
  ↪history
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
epochs = range(1, len(accuracy_values) + 1)
plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color = 'red')
plt.plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

```
<ipython-input-63-f233e3ce1278>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, accuracy_values, "b", label="Training Accuracy", color =
'red')
```

[ ]:
```
performance_dict_self_selected_best_model = history_self_selected_best_model.
  ↪history
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(loss_values) + 1)
plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
plt.plot(epochs, val_loss_values, "b", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

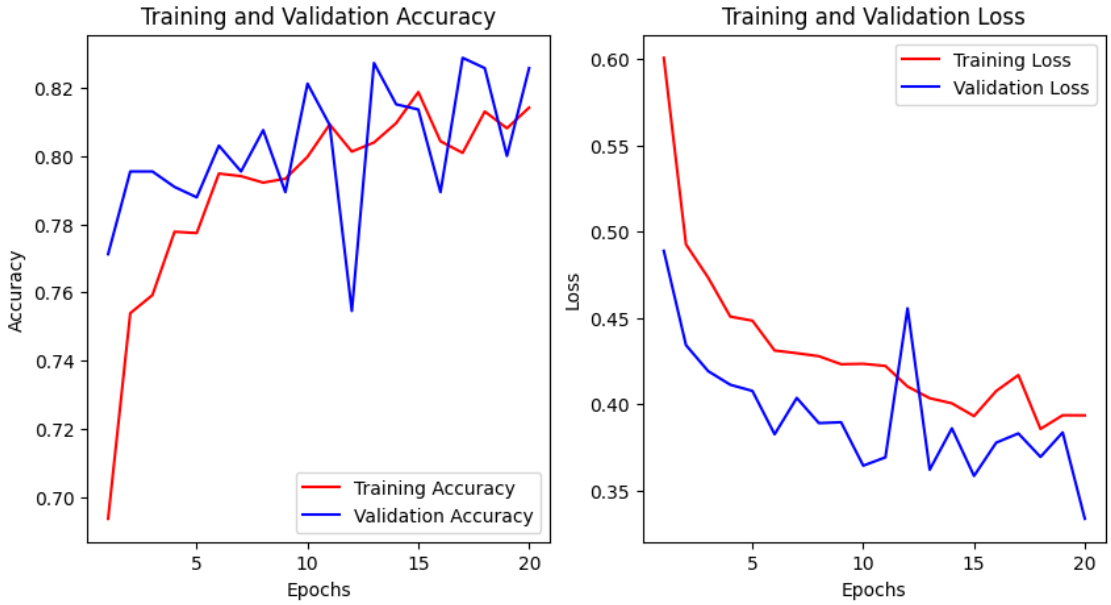
```
<ipython-input-65-f95944d2302b>:5: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  plt.plot(epochs, loss_values, "b", label="Training Loss", color = 'red')
```



[ ]:
```
performance_dict_self_selected_best_model = history_self_selected_best_model.
 ↪history
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
```

```
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-74-ec2623485220>:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-74-ec2623485220>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



[ ]:
```
# trial 2
#ReLU

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
```

```
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.05)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)


# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units -␣
 ↪added dropout
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.1
x = layers.Dropout(0.1)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model_self_selected_best2 = Model(img_input, output)
model_self_selected_best2.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

[ ]:
```
model_self_selected_best2.summary()
```

```
Model: "model_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_11 (InputLayer)       [(None, 150, 150, 3)]     0

 conv2d_50 (Conv2D)          (None, 148, 148, 16)      448
```

```
 max_pooling2d_30 (MaxPoolin  (None, 74, 74, 16)       0
 g2D)

 dropout_16 (Dropout)        (None, 74, 74, 16)        0

 conv2d_51 (Conv2D)          (None, 72, 72, 32)        4640

 max_pooling2d_31 (MaxPoolin  (None, 36, 36, 32)       0
 g2D)

 dropout_17 (Dropout)        (None, 36, 36, 32)        0

 conv2d_52 (Conv2D)          (None, 34, 34, 64)        18496

 max_pooling2d_32 (MaxPoolin  (None, 17, 17, 64)       0
 g2D)

 dropout_18 (Dropout)        (None, 17, 17, 64)        0

 flatten_10 (Flatten)        (None, 18496)             0

 dense_20 (Dense)            (None, 512)               9470464

 dropout_19 (Dropout)        (None, 512)               0

 dense_21 (Dense)            (None, 1)                 513

=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
history_self_selected_best_model = model_self_selected_best2.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

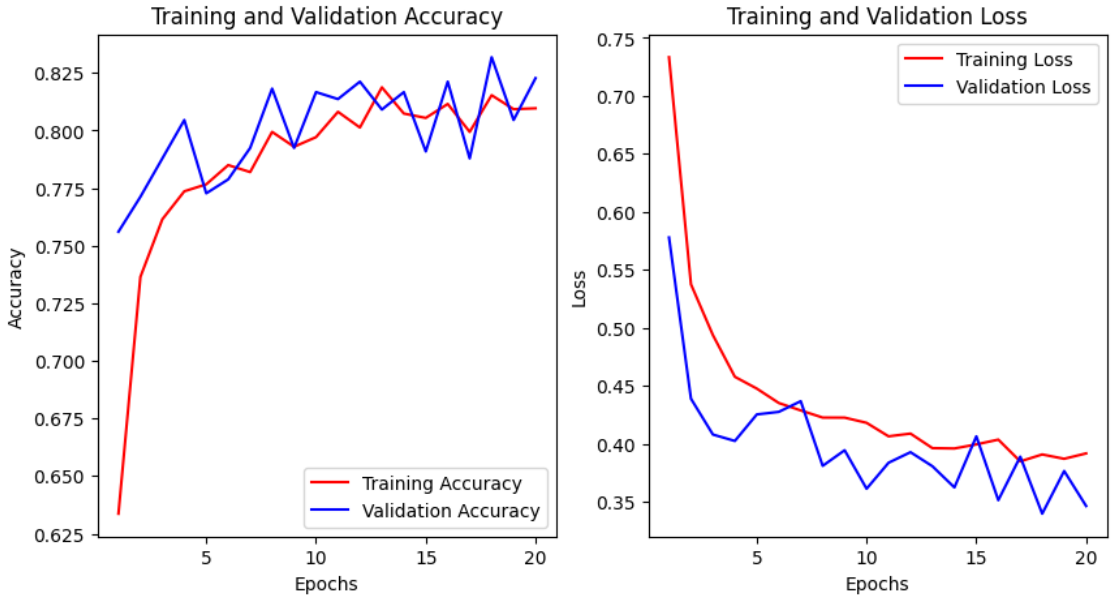
```
Epoch 1/20
132/132 - 131s - loss: 0.7333 - accuracy: 0.6337 - val_loss: 0.5779 -
val_accuracy: 0.7561 - 131s/epoch - 989ms/step
Epoch 2/20
132/132 - 109s - loss: 0.5376 - accuracy: 0.7364 - val_loss: 0.4390 -
val_accuracy: 0.7712 - 109s/epoch - 825ms/step
```

```
Epoch 3/20
132/132 - 102s - loss: 0.4938 - accuracy: 0.7615 - val_loss: 0.4081 -
val_accuracy: 0.7879 - 102s/epoch - 771ms/step
Epoch 4/20
132/132 - 101s - loss: 0.4578 - accuracy: 0.7736 - val_loss: 0.4025 -
val_accuracy: 0.8045 - 101s/epoch - 766ms/step
Epoch 5/20
132/132 - 101s - loss: 0.4477 - accuracy: 0.7766 - val_loss: 0.4254 -
val_accuracy: 0.7727 - 101s/epoch - 768ms/step
Epoch 6/20
132/132 - 109s - loss: 0.4352 - accuracy: 0.7850 - val_loss: 0.4276 -
val_accuracy: 0.7788 - 109s/epoch - 823ms/step
Epoch 7/20
132/132 - 108s - loss: 0.4288 - accuracy: 0.7819 - val_loss: 0.4368 -
val_accuracy: 0.7924 - 108s/epoch - 818ms/step
Epoch 8/20
132/132 - 101s - loss: 0.4227 - accuracy: 0.7994 - val_loss: 0.3811 -
val_accuracy: 0.8182 - 101s/epoch - 765ms/step
Epoch 9/20
132/132 - 103s - loss: 0.4226 - accuracy: 0.7929 - val_loss: 0.3945 -
val_accuracy: 0.7924 - 103s/epoch - 780ms/step
Epoch 10/20
132/132 - 100s - loss: 0.4182 - accuracy: 0.7971 - val_loss: 0.3613 -
val_accuracy: 0.8167 - 100s/epoch - 756ms/step
Epoch 11/20
132/132 - 108s - loss: 0.4065 - accuracy: 0.8081 - val_loss: 0.3837 -
val_accuracy: 0.8136 - 108s/epoch - 821ms/step
Epoch 12/20
132/132 - 102s - loss: 0.4089 - accuracy: 0.8013 - val_loss: 0.3929 -
val_accuracy: 0.8212 - 102s/epoch - 776ms/step
Epoch 13/20
132/132 - 105s - loss: 0.3963 - accuracy: 0.8187 - val_loss: 0.3807 -
val_accuracy: 0.8091 - 105s/epoch - 792ms/step
Epoch 14/20
132/132 - 108s - loss: 0.3960 - accuracy: 0.8074 - val_loss: 0.3625 -
val_accuracy: 0.8167 - 108s/epoch - 816ms/step
Epoch 15/20
132/132 - 106s - loss: 0.3996 - accuracy: 0.8055 - val_loss: 0.4065 -
val_accuracy: 0.7909 - 106s/epoch - 807ms/step
Epoch 16/20
132/132 - 100s - loss: 0.4037 - accuracy: 0.8115 - val_loss: 0.3515 -
val_accuracy: 0.8212 - 100s/epoch - 757ms/step
Epoch 17/20
132/132 - 100s - loss: 0.3849 - accuracy: 0.7994 - val_loss: 0.3890 -
val_accuracy: 0.7879 - 100s/epoch - 755ms/step
Epoch 18/20
132/132 - 101s - loss: 0.3909 - accuracy: 0.8153 - val_loss: 0.3398 -
val_accuracy: 0.8318 - 101s/epoch - 766ms/step
```

```
Epoch 19/20
132/132 - 107s - loss: 0.3872 - accuracy: 0.8093 - val_loss: 0.3766 -
val_accuracy: 0.8045 - 107s/epoch - 811ms/step
Epoch 20/20
132/132 - 106s - loss: 0.3918 - accuracy: 0.8096 - val_loss: 0.3465 -
val_accuracy: 0.8227 - 106s/epoch - 806ms/step
```

[ ]:
```
performance_dict_self_selected_best_model = history_self_selected_best_model.
 ↪history
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-81-8c5586dcec64>:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-81-8c5586dcec64>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



[ ]:
```
# Adding rescale, rotation_range, width_shift_range, height_shift_range,
# shear_range, zoom_range, and horizontal flip to our ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

# Note that the validation data should not be augmented!
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 32 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
```

```
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

[ ]:
```
# ReLU with Dropout of 0.1

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="relu", padding="same")(x)

    if pooling:
       x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
       residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
       residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x


x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)

x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)

x =residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)


x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(60, activation="relu")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model_Residual_Block_with_dropout_ReLU = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_with_dropout_ReLU.summary()

  # compile model
```

```
model_Residual_Block_with_dropout_ReLU.compile(loss= 'binary_crossentropy',␣
 ↪optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

```
Model: "model_1"
__________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
==================
 input_2 (InputLayer)           [(None, 150, 150, 3  0           []
                                )]

 rescaling_1 (Rescaling)        (None, 150, 150, 3)  0
['input_2[0][0]']

 batch_normalization_4 (BatchNo  (None, 150, 150, 3)  12
['rescaling_1[0][0]']
 rmalization)

 conv2d_8 (Conv2D)              (None, 150, 150, 64  1792
['batch_normalization_4[0][0]']
                                )

 max_pooling2d_3 (MaxPooling2D)  (None, 75, 75, 64)  0
['conv2d_8[0][0]']

 conv2d_9 (Conv2D)              (None, 75, 75, 64)   256
['rescaling_1[0][0]']

 add_4 (Add)                    (None, 75, 75, 64)   0
['max_pooling2d_3[0][0]',
'conv2d_9[0][0]']

 dropout_3 (Dropout)            (None, 75, 75, 64)   0           ['add_4[0][0]']

 batch_normalization_5 (BatchNo  (None, 75, 75, 64)  256
['dropout_3[0][0]']
 rmalization)

 conv2d_10 (Conv2D)             (None, 75, 75, 64)   36928
['batch_normalization_5[0][0]']

 max_pooling2d_4 (MaxPooling2D)  (None, 38, 38, 64)  0
['conv2d_10[0][0]']

 conv2d_11 (Conv2D)             (None, 38, 38, 64)   4160
['dropout_3[0][0]']
```

```
 add_5 (Add)                    (None, 38, 38, 64)   0
['max_pooling2d_4[0][0]',
'conv2d_11[0][0]']

 dropout_4 (Dropout)            (None, 38, 38, 64)   0           ['add_5[0][0]']

 batch_normalization_6 (BatchNo  (None, 38, 38, 64)  256
['dropout_4[0][0]']
 rmalization)

 conv2d_12 (Conv2D)             (None, 38, 38, 64)   36928
['batch_normalization_6[0][0]']

 max_pooling2d_5 (MaxPooling2D)  (None, 19, 19, 64)  0
['conv2d_12[0][0]']

 conv2d_13 (Conv2D)             (None, 19, 19, 64)   4160
['dropout_4[0][0]']

 add_6 (Add)                    (None, 19, 19, 64)   0
['max_pooling2d_5[0][0]',
'conv2d_13[0][0]']

 dropout_5 (Dropout)            (None, 19, 19, 64)   0           ['add_6[0][0]']

 batch_normalization_7 (BatchNo  (None, 19, 19, 64)  256
['dropout_5[0][0]']
 rmalization)

 conv2d_14 (Conv2D)             (None, 19, 19, 16)   9232
['batch_normalization_7[0][0]']

 conv2d_15 (Conv2D)             (None, 19, 19, 16)   1040
['dropout_5[0][0]']

 add_7 (Add)                    (None, 19, 19, 16)   0
['conv2d_14[0][0]',
'conv2d_15[0][0]']

 flatten_1 (Flatten)            (None, 5776)         0           ['add_7[0][0]']

 dense_2 (Dense)                (None, 60)           346620
['flatten_1[0][0]']

 dense_3 (Dense)                (None, 1)            61
['dense_2[0][0]']
```

```
==========================================================================
==================
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
--------------------------------------------------------------------------
------------------
```

[ ]:
```
history_of_residualblock_withdropout_ReLU2 =␣
↪model_Residual_Block_with_dropout_ReLU.fit(
    train_generator,
    steps_per_epoch=None,
    epochs=25,
    validation_data=validation_generator,
    validation_steps=None,
    verbose=2)
```

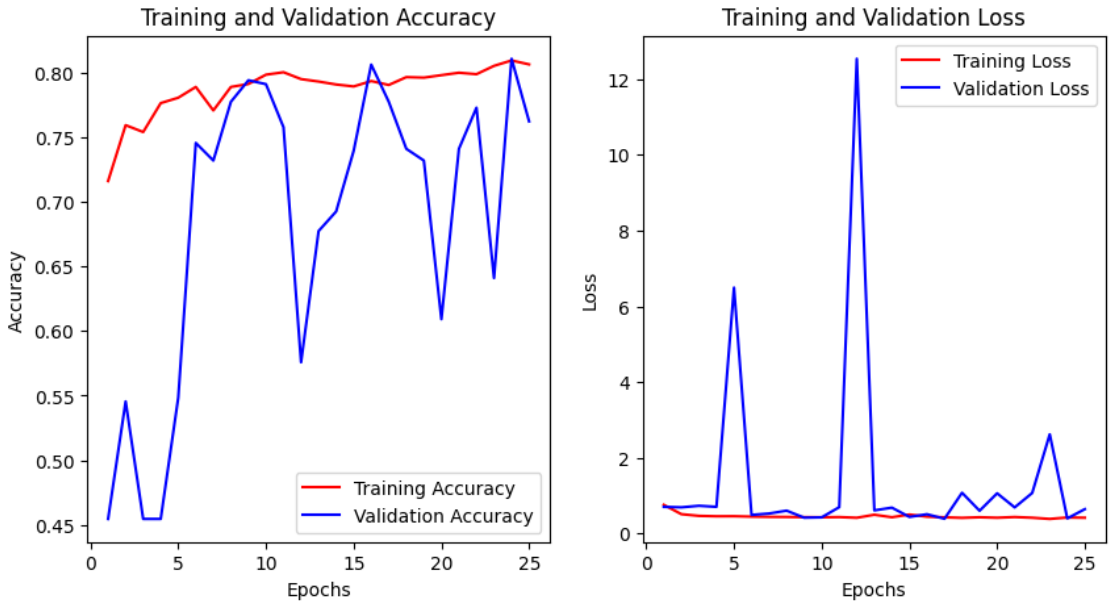
```
Epoch 1/25
132/132 - 276s - loss: 0.7553 - accuracy: 0.7160 - val_loss: 0.7018 -
val_accuracy: 0.4545 - 276s/epoch - 2s/step
Epoch 2/25
132/132 - 262s - loss: 0.5085 - accuracy: 0.7592 - val_loss: 0.6894 -
val_accuracy: 0.5455 - 262s/epoch - 2s/step
Epoch 3/25
132/132 - 261s - loss: 0.4636 - accuracy: 0.7539 - val_loss: 0.7303 -
val_accuracy: 0.4545 - 261s/epoch - 2s/step
Epoch 4/25
132/132 - 260s - loss: 0.4532 - accuracy: 0.7763 - val_loss: 0.7027 -
val_accuracy: 0.4545 - 260s/epoch - 2s/step
Epoch 5/25
132/132 - 260s - loss: 0.4546 - accuracy: 0.7804 - val_loss: 6.4930 -
val_accuracy: 0.5485 - 260s/epoch - 2s/step
Epoch 6/25
132/132 - 267s - loss: 0.4438 - accuracy: 0.7888 - val_loss: 0.4897 -
val_accuracy: 0.7455 - 267s/epoch - 2s/step
Epoch 7/25
132/132 - 261s - loss: 0.4377 - accuracy: 0.7706 - val_loss: 0.5258 -
val_accuracy: 0.7318 - 261s/epoch - 2s/step
Epoch 8/25
132/132 - 261s - loss: 0.4352 - accuracy: 0.7888 - val_loss: 0.6018 -
val_accuracy: 0.7773 - 261s/epoch - 2s/step
Epoch 9/25
132/132 - 264s - loss: 0.4294 - accuracy: 0.7911 - val_loss: 0.4160 -
val_accuracy: 0.7939 - 264s/epoch - 2s/step
Epoch 10/25
132/132 - 266s - loss: 0.4264 - accuracy: 0.7983 - val_loss: 0.4248 -
val_accuracy: 0.7909 - 266s/epoch - 2s/step
Epoch 11/25
```

```
132/132 - 261s - loss: 0.4316 - accuracy: 0.8002 - val_loss: 0.6937 -
val_accuracy: 0.7576 - 261s/epoch - 2s/step
Epoch 12/25
132/132 - 266s - loss: 0.4138 - accuracy: 0.7948 - val_loss: 12.5406 -
val_accuracy: 0.5758 - 266s/epoch - 2s/step
Epoch 13/25
132/132 - 259s - loss: 0.4936 - accuracy: 0.7929 - val_loss: 0.6089 -
val_accuracy: 0.6773 - 259s/epoch - 2s/step
Epoch 14/25
132/132 - 265s - loss: 0.4277 - accuracy: 0.7907 - val_loss: 0.6813 -
val_accuracy: 0.6924 - 265s/epoch - 2s/step
Epoch 15/25
132/132 - 260s - loss: 0.4990 - accuracy: 0.7892 - val_loss: 0.4358 -
val_accuracy: 0.7394 - 260s/epoch - 2s/step
Epoch 16/25
132/132 - 266s - loss: 0.4429 - accuracy: 0.7933 - val_loss: 0.5069 -
val_accuracy: 0.8061 - 266s/epoch - 2s/step
Epoch 17/25
132/132 - 261s - loss: 0.4257 - accuracy: 0.7903 - val_loss: 0.3868 -
val_accuracy: 0.7773 - 261s/epoch - 2s/step
Epoch 18/25
132/132 - 259s - loss: 0.4117 - accuracy: 0.7964 - val_loss: 1.0747 -
val_accuracy: 0.7409 - 259s/epoch - 2s/step
Epoch 19/25
132/132 - 265s - loss: 0.4272 - accuracy: 0.7960 - val_loss: 0.5989 -
val_accuracy: 0.7318 - 265s/epoch - 2s/step
Epoch 20/25
132/132 - 265s - loss: 0.4144 - accuracy: 0.7979 - val_loss: 1.0601 -
val_accuracy: 0.6091 - 265s/epoch - 2s/step
Epoch 21/25
132/132 - 265s - loss: 0.4330 - accuracy: 0.7998 - val_loss: 0.6869 -
val_accuracy: 0.7409 - 265s/epoch - 2s/step
Epoch 22/25
132/132 - 260s - loss: 0.4146 - accuracy: 0.7986 - val_loss: 1.0683 -
val_accuracy: 0.7727 - 260s/epoch - 2s/step
Epoch 23/25
132/132 - 281s - loss: 0.3828 - accuracy: 0.8051 - val_loss: 2.6186 -
val_accuracy: 0.6409 - 281s/epoch - 2s/step
Epoch 24/25
132/132 - 260s - loss: 0.4208 - accuracy: 0.8093 - val_loss: 0.3962 -
val_accuracy: 0.8106 - 260s/epoch - 2s/step
Epoch 25/25
132/132 - 267s - loss: 0.4146 - accuracy: 0.8062 - val_loss: 0.6414 -
val_accuracy: 0.7621 - 267s/epoch - 2s/step
```

[ ]:
```
performance_dict_self_selected_best_model =␣
  ↪history_of_residualblock_withdropout_ReLU2.history
```

```
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-16-4c3bfd25c679>:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-16-4c3bfd25c679>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



[ ]:
```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

[ ]:
```
# ReLU with Dropout of 0.1

inputs = tf.keras.Input(shape=(150, 150, 3))
x = tf.keras.layers.Rescaling(1/255)(inputs) # we rescale data
def residual_block(x, filters, pooling=False):
```

```
    residual = x
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.Conv2D(filters, 3, activation="relu", padding="same")(x)

    if pooling:
       x = tf.keras.layers.MaxPooling2D(2, padding="same")(x)
       residual = tf.keras.layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]:
       residual = tf.keras.layers.Conv2D(filters, 1)(residual)

    x = tf.keras.layers.add([x, residual])

    return x


x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)

x= residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)

x =residual_block(x, filters=64, pooling=True)
x = tf.keras.layers.Dropout(.1)(x)


x =residual_block(x, filters=16, pooling=False)

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dense(60, activation="relu")(x)

outputs = tf.keras.layers.Dense(1, activation="sigmoid")(x)

model_Residual_Block_with_dropout_ReLU = tf.keras.Model(inputs=inputs,␣
 ↪outputs=outputs)
model_Residual_Block_with_dropout_ReLU.summary()

  # compile model
model_Residual_Block_with_dropout_ReLU.compile(loss= 'binary_crossentropy',␣
 ↪optimizer=RMSprop(learning_rate=0.001), metrics=['accuracy'])
```

```
Model: "model"
_________________________________________________________________________________
__________________
 Layer (type)                    Output Shape         Param #     Connected to
=================================================================================
==================
 input_1 (InputLayer)            [(None, 150, 150, 3  0           []
```

```
                                )]

 rescaling (Rescaling)          (None, 150, 150, 3)  0
['input_1[0][0]']

 batch_normalization (BatchNorm  (None, 150, 150, 3)  12
['rescaling[0][0]']
 alization)

 conv2d (Conv2D)                (None, 150, 150, 64  1792
['batch_normalization[0][0]']
                                )

 max_pooling2d (MaxPooling2D)   (None, 75, 75, 64)   0
['conv2d[0][0]']

 conv2d_1 (Conv2D)              (None, 75, 75, 64)   256
['rescaling[0][0]']

 add (Add)                      (None, 75, 75, 64)   0
['max_pooling2d[0][0]',
'conv2d_1[0][0]']

 dropout (Dropout)              (None, 75, 75, 64)   0           ['add[0][0]']

 batch_normalization_1 (BatchNo  (None, 75, 75, 64)  256
['dropout[0][0]']
 rmalization)

 conv2d_2 (Conv2D)              (None, 75, 75, 64)   36928
['batch_normalization_1[0][0]']

 max_pooling2d_1 (MaxPooling2D)  (None, 38, 38, 64)  0
['conv2d_2[0][0]']

 conv2d_3 (Conv2D)              (None, 38, 38, 64)   4160
['dropout[0][0]']

 add_1 (Add)                    (None, 38, 38, 64)   0
['max_pooling2d_1[0][0]',
'conv2d_3[0][0]']

 dropout_1 (Dropout)            (None, 38, 38, 64)   0           ['add_1[0][0]']

 batch_normalization_2 (BatchNo  (None, 38, 38, 64)  256
['dropout_1[0][0]']
 rmalization)
```

```
 conv2d_4 (Conv2D)              (None, 38, 38, 64)   36928
['batch_normalization_2[0][0]']

 max_pooling2d_2 (MaxPooling2D)  (None, 19, 19, 64)  0
['conv2d_4[0][0]']

 conv2d_5 (Conv2D)              (None, 19, 19, 64)   4160
['dropout_1[0][0]']

 add_2 (Add)                    (None, 19, 19, 64)   0
['max_pooling2d_2[0][0]',
'conv2d_5[0][0]']

 dropout_2 (Dropout)            (None, 19, 19, 64)   0           ['add_2[0][0]']

 batch_normalization_3 (BatchNo  (None, 19, 19, 64)  256
['dropout_2[0][0]']
 rmalization)

 conv2d_6 (Conv2D)              (None, 19, 19, 16)   9232
['batch_normalization_3[0][0]']

 conv2d_7 (Conv2D)              (None, 19, 19, 16)   1040
['dropout_2[0][0]']

 add_3 (Add)                    (None, 19, 19, 16)   0
['conv2d_6[0][0]',
'conv2d_7[0][0]']

 flatten (Flatten)              (None, 5776)         0           ['add_3[0][0]']

 dense (Dense)                  (None, 60)           346620
['flatten[0][0]']

 dense_1 (Dense)                (None, 1)            61          ['dense[0][0]']

==================================================================================
==================
Total params: 441,957
Trainable params: 441,567
Non-trainable params: 390
----------------------------------------------------------------------------------
------------------
```

[ ]:
```
history_of_residualblock_withdropout_ReLU =␣
  ↪model_Residual_Block_with_dropout_ReLU.fit(
      train_generator,
```

```
      steps_per_epoch=None,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

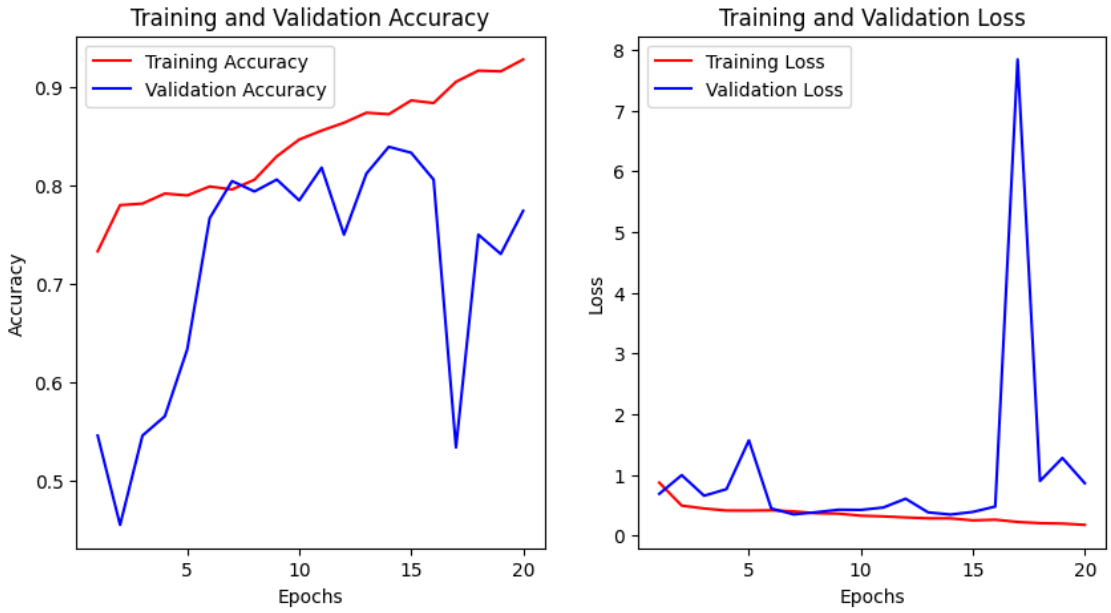
```
Epoch 1/20
132/132 - 253s - loss: 0.8733 - accuracy: 0.7330 - val_loss: 0.6896 -
val_accuracy: 0.5455 - 253s/epoch - 2s/step
Epoch 2/20
132/132 - 249s - loss: 0.4955 - accuracy: 0.7801 - val_loss: 0.9971 -
val_accuracy: 0.4545 - 249s/epoch - 2s/step
Epoch 3/20
132/132 - 253s - loss: 0.4479 - accuracy: 0.7816 - val_loss: 0.6580 -
val_accuracy: 0.5455 - 253s/epoch - 2s/step
Epoch 4/20
132/132 - 249s - loss: 0.4145 - accuracy: 0.7918 - val_loss: 0.7666 -
val_accuracy: 0.5652 - 249s/epoch - 2s/step
Epoch 5/20
132/132 - 249s - loss: 0.4135 - accuracy: 0.7899 - val_loss: 1.5696 -
val_accuracy: 0.6333 - 249s/epoch - 2s/step
Epoch 6/20
132/132 - 247s - loss: 0.4179 - accuracy: 0.7990 - val_loss: 0.4477 -
val_accuracy: 0.7667 - 247s/epoch - 2s/step
Epoch 7/20
132/132 - 254s - loss: 0.4009 - accuracy: 0.7960 - val_loss: 0.3511 -
val_accuracy: 0.8045 - 254s/epoch - 2s/step
Epoch 8/20
132/132 - 249s - loss: 0.3694 - accuracy: 0.8058 - val_loss: 0.3861 -
val_accuracy: 0.7939 - 249s/epoch - 2s/step
Epoch 9/20
132/132 - 254s - loss: 0.3639 - accuracy: 0.8297 - val_loss: 0.4281 -
val_accuracy: 0.8061 - 254s/epoch - 2s/step
Epoch 10/20
132/132 - 249s - loss: 0.3296 - accuracy: 0.8468 - val_loss: 0.4247 -
val_accuracy: 0.7848 - 249s/epoch - 2s/step
Epoch 11/20
132/132 - 255s - loss: 0.3178 - accuracy: 0.8559 - val_loss: 0.4639 -
val_accuracy: 0.8182 - 255s/epoch - 2s/step
Epoch 12/20
132/132 - 250s - loss: 0.3007 - accuracy: 0.8639 - val_loss: 0.6090 -
val_accuracy: 0.7500 - 250s/epoch - 2s/step
Epoch 13/20
132/132 - 254s - loss: 0.2861 - accuracy: 0.8741 - val_loss: 0.3847 -
val_accuracy: 0.8121 - 254s/epoch - 2s/step
Epoch 14/20
132/132 - 248s - loss: 0.2856 - accuracy: 0.8726 - val_loss: 0.3486 -
```

```
val_accuracy: 0.8394 - 248s/epoch - 2s/step
Epoch 15/20
132/132 - 280s - loss: 0.2525 - accuracy: 0.8866 - val_loss: 0.3910 -
val_accuracy: 0.8333 - 280s/epoch - 2s/step
Epoch 16/20
132/132 - 255s - loss: 0.2636 - accuracy: 0.8840 - val_loss: 0.4804 -
val_accuracy: 0.8061 - 255s/epoch - 2s/step
Epoch 17/20
132/132 - 258s - loss: 0.2265 - accuracy: 0.9056 - val_loss: 7.8389 -
val_accuracy: 0.5333 - 258s/epoch - 2s/step
Epoch 18/20
132/132 - 323s - loss: 0.2068 - accuracy: 0.9170 - val_loss: 0.9011 -
val_accuracy: 0.7500 - 323s/epoch - 2s/step
Epoch 19/20
132/132 - 251s - loss: 0.1997 - accuracy: 0.9162 - val_loss: 1.2794 -
val_accuracy: 0.7303 - 251s/epoch - 2s/step
Epoch 20/20
132/132 - 260s - loss: 0.1783 - accuracy: 0.9283 - val_loss: 0.8637 -
val_accuracy: 0.7742 - 260s/epoch - 2s/step
```

[ ]:
```
performance_dict_self_selected_best_model =␣
 ↪history_of_residualblock_withdropout_ReLU.history
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-11-3da76cfaa771>:10: UserWarning: color is redundantly defined by
```

```
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-11-3da76cfaa771>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



16 Fine Tuning Models Further with More Epochs

[ ]:
```
# All images will be rescaled by 1./255
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 20 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
```

```
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

[ ]:
```
#ReLU

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))

# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.05)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)


# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units -␣
 ↪added dropout
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.1
x = layers.Dropout(0.1)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model_self_selected_best2 = Model(img_input, output)
```

```
model_self_selected_best2.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

[ ]:
```
history_self_selected_best_model = model_self_selected_best2.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=40,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/40
132/132 - 95s - loss: 0.7163 - accuracy: 0.6303 - val_loss: 0.5536 -
val_accuracy: 0.6864 - 95s/epoch - 720ms/step
Epoch 2/40
132/132 - 97s - loss: 0.5177 - accuracy: 0.7414 - val_loss: 0.4357 -
val_accuracy: 0.7742 - 97s/epoch - 731ms/step
Epoch 3/40
132/132 - 97s - loss: 0.4748 - accuracy: 0.7706 - val_loss: 0.4299 -
val_accuracy: 0.8197 - 97s/epoch - 737ms/step
Epoch 4/40
132/132 - 98s - loss: 0.4538 - accuracy: 0.7861 - val_loss: 0.4235 -
val_accuracy: 0.7697 - 98s/epoch - 742ms/step
Epoch 5/40
132/132 - 98s - loss: 0.4302 - accuracy: 0.7876 - val_loss: 0.3887 -
val_accuracy: 0.8182 - 98s/epoch - 746ms/step
Epoch 6/40
132/132 - 100s - loss: 0.4092 - accuracy: 0.8077 - val_loss: 0.4178 -
val_accuracy: 0.8015 - 100s/epoch - 757ms/step
Epoch 7/40
132/132 - 94s - loss: 0.3867 - accuracy: 0.8093 - val_loss: 0.3591 -
val_accuracy: 0.8288 - 94s/epoch - 714ms/step
Epoch 8/40
132/132 - 98s - loss: 0.3852 - accuracy: 0.8149 - val_loss: 0.3758 -
val_accuracy: 0.7909 - 98s/epoch - 744ms/step
Epoch 9/40
132/132 - 93s - loss: 0.3740 - accuracy: 0.8111 - val_loss: 0.4482 -
val_accuracy: 0.7848 - 93s/epoch - 707ms/step
Epoch 10/40
132/132 - 101s - loss: 0.3574 - accuracy: 0.8267 - val_loss: 0.3396 -
val_accuracy: 0.8303 - 101s/epoch - 765ms/step
Epoch 11/40
132/132 - 99s - loss: 0.3349 - accuracy: 0.8369 - val_loss: 0.3467 -
val_accuracy: 0.8212 - 99s/epoch - 749ms/step
```

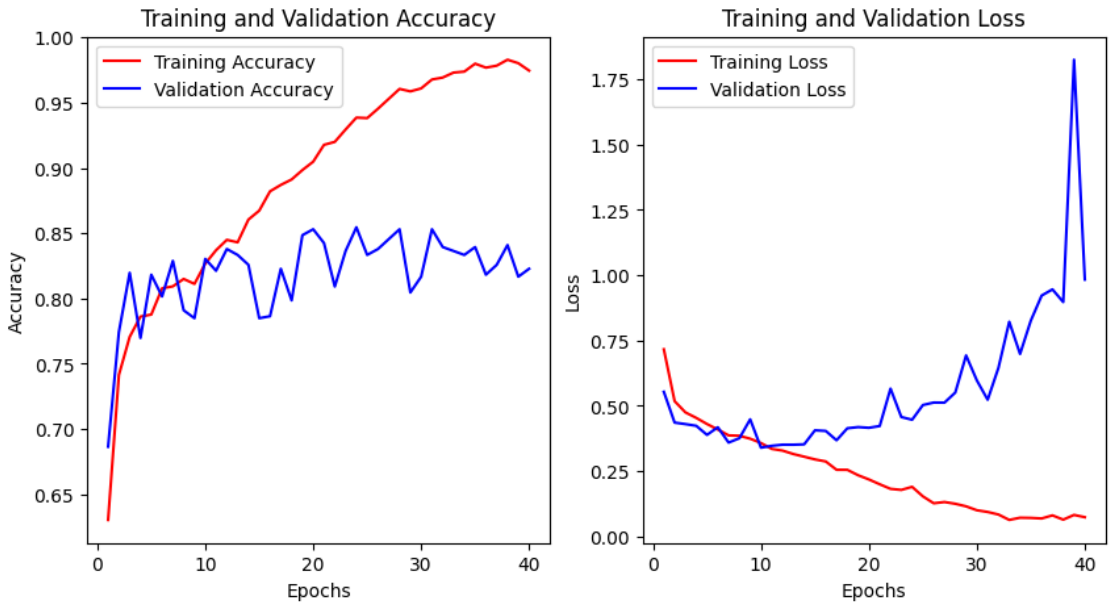
```
Epoch 12/40
132/132 - 96s - loss: 0.3284 - accuracy: 0.8449 - val_loss: 0.3511 -
val_accuracy: 0.8379 - 96s/epoch - 727ms/step
Epoch 13/40
132/132 - 99s - loss: 0.3153 - accuracy: 0.8430 - val_loss: 0.3511 -
val_accuracy: 0.8333 - 99s/epoch - 748ms/step
Epoch 14/40
132/132 - 100s - loss: 0.3051 - accuracy: 0.8604 - val_loss: 0.3524 -
val_accuracy: 0.8258 - 100s/epoch - 755ms/step
Epoch 15/40
132/132 - 98s - loss: 0.2948 - accuracy: 0.8673 - val_loss: 0.4061 -
val_accuracy: 0.7848 - 98s/epoch - 741ms/step
Epoch 16/40
132/132 - 105s - loss: 0.2869 - accuracy: 0.8821 - val_loss: 0.4037 -
val_accuracy: 0.7864 - 105s/epoch - 795ms/step
Epoch 17/40
132/132 - 97s - loss: 0.2550 - accuracy: 0.8870 - val_loss: 0.3680 -
val_accuracy: 0.8227 - 97s/epoch - 738ms/step
Epoch 18/40
132/132 - 98s - loss: 0.2548 - accuracy: 0.8912 - val_loss: 0.4140 -
val_accuracy: 0.7985 - 98s/epoch - 744ms/step
Epoch 19/40
132/132 - 93s - loss: 0.2341 - accuracy: 0.8984 - val_loss: 0.4185 -
val_accuracy: 0.8485 - 93s/epoch - 706ms/step
Epoch 20/40
132/132 - 93s - loss: 0.2180 - accuracy: 0.9048 - val_loss: 0.4158 -
val_accuracy: 0.8530 - 93s/epoch - 708ms/step
Epoch 21/40
132/132 - 103s - loss: 0.1996 - accuracy: 0.9177 - val_loss: 0.4225 -
val_accuracy: 0.8424 - 103s/epoch - 784ms/step
Epoch 22/40
132/132 - 98s - loss: 0.1821 - accuracy: 0.9200 - val_loss: 0.5657 -
val_accuracy: 0.8091 - 98s/epoch - 744ms/step
Epoch 23/40
132/132 - 94s - loss: 0.1780 - accuracy: 0.9295 - val_loss: 0.4571 -
val_accuracy: 0.8364 - 94s/epoch - 713ms/step
Epoch 24/40
132/132 - 94s - loss: 0.1897 - accuracy: 0.9386 - val_loss: 0.4466 -
val_accuracy: 0.8545 - 94s/epoch - 710ms/step
Epoch 25/40
132/132 - 94s - loss: 0.1532 - accuracy: 0.9382 - val_loss: 0.5030 -
val_accuracy: 0.8333 - 94s/epoch - 713ms/step
Epoch 26/40
132/132 - 97s - loss: 0.1268 - accuracy: 0.9454 - val_loss: 0.5124 -
val_accuracy: 0.8379 - 97s/epoch - 732ms/step
Epoch 27/40
132/132 - 99s - loss: 0.1316 - accuracy: 0.9530 - val_loss: 0.5121 -
val_accuracy: 0.8455 - 99s/epoch - 747ms/step
```

```
Epoch 28/40
132/132 - 94s - loss: 0.1249 - accuracy: 0.9606 - val_loss: 0.5508 -
val_accuracy: 0.8530 - 94s/epoch - 711ms/step
Epoch 29/40
132/132 - 132s - loss: 0.1151 - accuracy: 0.9587 - val_loss: 0.6928 -
val_accuracy: 0.8045 - 132s/epoch - 1s/step
Epoch 30/40
132/132 - 95s - loss: 0.1001 - accuracy: 0.9609 - val_loss: 0.5970 -
val_accuracy: 0.8167 - 95s/epoch - 722ms/step
Epoch 31/40
132/132 - 96s - loss: 0.0935 - accuracy: 0.9678 - val_loss: 0.5232 -
val_accuracy: 0.8530 - 96s/epoch - 729ms/step
Epoch 32/40
132/132 - 98s - loss: 0.0837 - accuracy: 0.9693 - val_loss: 0.6456 -
val_accuracy: 0.8394 - 98s/epoch - 745ms/step
Epoch 33/40
132/132 - 100s - loss: 0.0630 - accuracy: 0.9731 - val_loss: 0.8218 -
val_accuracy: 0.8364 - 100s/epoch - 754ms/step
Epoch 34/40
132/132 - 95s - loss: 0.0715 - accuracy: 0.9738 - val_loss: 0.6985 -
val_accuracy: 0.8333 - 95s/epoch - 716ms/step
Epoch 35/40
132/132 - 99s - loss: 0.0708 - accuracy: 0.9799 - val_loss: 0.8264 -
val_accuracy: 0.8394 - 99s/epoch - 752ms/step
Epoch 36/40
132/132 - 96s - loss: 0.0686 - accuracy: 0.9769 - val_loss: 0.9221 -
val_accuracy: 0.8182 - 96s/epoch - 725ms/step
Epoch 37/40
132/132 - 98s - loss: 0.0804 - accuracy: 0.9784 - val_loss: 0.9458 -
val_accuracy: 0.8258 - 98s/epoch - 742ms/step
Epoch 38/40
132/132 - 99s - loss: 0.0641 - accuracy: 0.9829 - val_loss: 0.8972 -
val_accuracy: 0.8409 - 99s/epoch - 753ms/step
Epoch 39/40
132/132 - 94s - loss: 0.0822 - accuracy: 0.9803 - val_loss: 1.8250 -
val_accuracy: 0.8167 - 94s/epoch - 714ms/step
Epoch 40/40
132/132 - 95s - loss: 0.0734 - accuracy: 0.9746 - val_loss: 0.9834 -
val_accuracy: 0.8227 - 95s/epoch - 720ms/step
```

[ ]:
```
performance_dict_self_selected_best_model = history_self_selected_best_model.
  ↪history
accuracy_values = performance_dict_self_selected_best_model["accuracy"]
val_accuracy_values = performance_dict_self_selected_best_model["val_accuracy"]
loss_values = performance_dict_self_selected_best_model["loss"]
val_loss_values = performance_dict_self_selected_best_model["val_loss"]
epochs = range(1, len(accuracy_values) + 1)
```

```
fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-10-8c5586dcec64>:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-10-8c5586dcec64>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



Very good model with high training accuracy, however, validation accuracy is 10% lower. Still might be overfitting data as we are using the original training images, not the augmented ones. Let's repeat this but use augmented training data!

17 Final Model

[7]:
```
# Adding rescale, rotation_range, width_shift_range, height_shift_range,
# shear_range, zoom_range, and horizontal flip to our ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,)

# Note that the validation data should not be augmented!
val_datagen = ImageDataGenerator(rescale=1./255)

# Flow training images in batches of 32 using train_datagen generator
train_generator = train_datagen.flow_from_directory(
        train_dir,  # This is the source directory for training images
        target_size=(150, 150),  # All images will be resized to 150x150
        batch_size=20,
        # Since we use binary_crossentropy loss, we need binary labels
        class_mode='binary')

# Flow validation images in batches of 32 using val_datagen generator
validation_generator = val_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

```
Found 2637 images belonging to 2 classes.
Found 660 images belonging to 2 classes.
```

[8]:
```
#ReLU

# Our input feature map is 150x150x3: 150x150 for the image pixels, and 3 for
# the three color channels: R, G, and B
img_input = layers.Input(shape=(150, 150, 3))
```

```
# First convolution extracts 16 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(16, 3, activation='relu')(img_input)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.05)(x)

# Second convolution extracts 32 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Conv2D(32, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)

# Third convolution extracts 64 filters that are 3x3
# Convolution is followed by max-pooling layer with a 2x2 window
x = layers.Convolution2D(64, 3, activation='relu')(x)
x = layers.MaxPooling2D(2)(x)
x = layers.Dropout(0.1)(x)


# Flatten feature map to a 1-dim tensor
x = layers.Flatten()(x)

# Create a fully connected layer with ReLU activation and 512 hidden units -␣
 ↪added dropout
x = layers.Dense(512, activation='relu')(x)

# Add a dropout rate of 0.1
x = layers.Dropout(0.1)(x)

# Create output layer with a single node and sigmoid activation
output = layers.Dense(1, activation='sigmoid')(x)

# Configure and compile the model
model_self_selected_best23 = Model(img_input, output)
model_self_selected_best23.compile(loss='binary_crossentropy',
              optimizer=RMSprop(lr=0.001),
              metrics=['accuracy'])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate`
or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

[9]:
```
model_self_selected_best23.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2D  (None, 74, 74, 16)       0
 )

 dropout (Dropout)           (None, 74, 74, 16)        0

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPooling  (None, 36, 36, 32)       0
 2D)

 dropout_1 (Dropout)         (None, 36, 36, 32)        0

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 17, 17, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 17, 17, 64)        0

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 512)               9470464

 dropout_3 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```

[10]:
```
history_self_selected_best_model23 = model_self_selected_best23.fit(
      train_generator,
      steps_per_epoch=None,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=None,
      verbose=2)
```

```
Epoch 1/100
132/132 - 117s - loss: 0.6959 - accuracy: 0.6022 - val_loss: 0.6034 -
val_accuracy: 0.7636 - 117s/epoch - 889ms/step
```

```
Epoch 2/100
132/132 - 112s - loss: 0.5571 - accuracy: 0.7228 - val_loss: 0.6693 -
val_accuracy: 0.5985 - 112s/epoch - 847ms/step
Epoch 3/100
132/132 - 110s - loss: 0.5115 - accuracy: 0.7584 - val_loss: 0.4212 -
val_accuracy: 0.7909 - 110s/epoch - 835ms/step
Epoch 4/100
132/132 - 111s - loss: 0.4761 - accuracy: 0.7683 - val_loss: 0.4106 -
val_accuracy: 0.7894 - 111s/epoch - 844ms/step
Epoch 5/100
132/132 - 112s - loss: 0.4645 - accuracy: 0.7763 - val_loss: 0.4942 -
val_accuracy: 0.7364 - 112s/epoch - 845ms/step
Epoch 6/100
132/132 - 107s - loss: 0.4559 - accuracy: 0.7884 - val_loss: 0.4757 -
val_accuracy: 0.7485 - 107s/epoch - 813ms/step
Epoch 7/100
132/132 - 112s - loss: 0.4539 - accuracy: 0.7861 - val_loss: 0.4125 -
val_accuracy: 0.7894 - 112s/epoch - 847ms/step
Epoch 8/100
132/132 - 111s - loss: 0.4333 - accuracy: 0.8002 - val_loss: 0.3857 -
val_accuracy: 0.8061 - 111s/epoch - 839ms/step
Epoch 9/100
132/132 - 111s - loss: 0.4299 - accuracy: 0.7937 - val_loss: 0.3755 -
val_accuracy: 0.8061 - 111s/epoch - 839ms/step
Epoch 10/100
132/132 - 107s - loss: 0.4327 - accuracy: 0.7941 - val_loss: 0.3897 -
val_accuracy: 0.7985 - 107s/epoch - 810ms/step
Epoch 11/100
132/132 - 107s - loss: 0.4224 - accuracy: 0.7914 - val_loss: 0.3893 -
val_accuracy: 0.8152 - 107s/epoch - 811ms/step
Epoch 12/100
132/132 - 107s - loss: 0.4129 - accuracy: 0.7994 - val_loss: 0.4073 -
val_accuracy: 0.7970 - 107s/epoch - 810ms/step
Epoch 13/100
132/132 - 112s - loss: 0.4170 - accuracy: 0.7971 - val_loss: 0.3725 -
val_accuracy: 0.8167 - 112s/epoch - 847ms/step
Epoch 14/100
132/132 - 110s - loss: 0.4070 - accuracy: 0.8013 - val_loss: 0.3768 -
val_accuracy: 0.8152 - 110s/epoch - 834ms/step
Epoch 15/100
132/132 - 107s - loss: 0.3964 - accuracy: 0.8085 - val_loss: 0.3779 -
val_accuracy: 0.8197 - 107s/epoch - 808ms/step
Epoch 16/100
132/132 - 110s - loss: 0.3848 - accuracy: 0.8233 - val_loss: 0.3557 -
val_accuracy: 0.8242 - 110s/epoch - 834ms/step
Epoch 17/100
132/132 - 110s - loss: 0.3957 - accuracy: 0.8058 - val_loss: 0.3962 -
val_accuracy: 0.7985 - 110s/epoch - 837ms/step
```

```
Epoch 18/100
132/132 - 110s - loss: 0.3934 - accuracy: 0.8225 - val_loss: 0.3404 -
val_accuracy: 0.8303 - 110s/epoch - 835ms/step
Epoch 19/100
132/132 - 110s - loss: 0.3843 - accuracy: 0.8195 - val_loss: 0.3465 -
val_accuracy: 0.8197 - 110s/epoch - 836ms/step
Epoch 20/100
132/132 - 111s - loss: 0.3703 - accuracy: 0.8206 - val_loss: 0.3361 -
val_accuracy: 0.8273 - 111s/epoch - 840ms/step
Epoch 21/100
132/132 - 112s - loss: 0.3795 - accuracy: 0.8142 - val_loss: 0.3479 -
val_accuracy: 0.8333 - 112s/epoch - 846ms/step
Epoch 22/100
132/132 - 112s - loss: 0.3734 - accuracy: 0.8199 - val_loss: 0.3580 -
val_accuracy: 0.8242 - 112s/epoch - 845ms/step
Epoch 23/100
132/132 - 107s - loss: 0.3776 - accuracy: 0.8294 - val_loss: 0.3419 -
val_accuracy: 0.8364 - 107s/epoch - 811ms/step
Epoch 24/100
132/132 - 112s - loss: 0.3682 - accuracy: 0.8237 - val_loss: 0.3553 -
val_accuracy: 0.8258 - 112s/epoch - 847ms/step
Epoch 25/100
132/132 - 112s - loss: 0.3663 - accuracy: 0.8221 - val_loss: 0.3570 -
val_accuracy: 0.8333 - 112s/epoch - 848ms/step
Epoch 26/100
132/132 - 112s - loss: 0.3720 - accuracy: 0.8312 - val_loss: 0.3535 -
val_accuracy: 0.8106 - 112s/epoch - 845ms/step
Epoch 27/100
132/132 - 110s - loss: 0.3689 - accuracy: 0.8294 - val_loss: 0.4549 -
val_accuracy: 0.7606 - 110s/epoch - 835ms/step
Epoch 28/100
132/132 - 107s - loss: 0.3613 - accuracy: 0.8335 - val_loss: 0.4771 -
val_accuracy: 0.7985 - 107s/epoch - 809ms/step
Epoch 29/100
132/132 - 107s - loss: 0.3701 - accuracy: 0.8309 - val_loss: 0.3447 -
val_accuracy: 0.8167 - 107s/epoch - 808ms/step
Epoch 30/100
132/132 - 107s - loss: 0.3537 - accuracy: 0.8369 - val_loss: 0.3321 -
val_accuracy: 0.8409 - 107s/epoch - 808ms/step
Epoch 31/100
132/132 - 111s - loss: 0.3544 - accuracy: 0.8312 - val_loss: 0.3190 -
val_accuracy: 0.8500 - 111s/epoch - 843ms/step
Epoch 32/100
132/132 - 107s - loss: 0.3559 - accuracy: 0.8449 - val_loss: 0.3447 -
val_accuracy: 0.8136 - 107s/epoch - 809ms/step
Epoch 33/100
132/132 - 107s - loss: 0.3527 - accuracy: 0.8347 - val_loss: 0.4082 -
val_accuracy: 0.8167 - 107s/epoch - 809ms/step
```

```
Epoch 34/100
132/132 - 111s - loss: 0.3496 - accuracy: 0.8377 - val_loss: 0.3887 -
val_accuracy: 0.8303 - 111s/epoch - 838ms/step
Epoch 35/100
132/132 - 107s - loss: 0.3712 - accuracy: 0.8392 - val_loss: 0.3400 -
val_accuracy: 0.8379 - 107s/epoch - 807ms/step
Epoch 36/100
132/132 - 112s - loss: 0.3781 - accuracy: 0.8259 - val_loss: 0.3280 -
val_accuracy: 0.8258 - 112s/epoch - 846ms/step
Epoch 37/100
132/132 - 112s - loss: 0.3566 - accuracy: 0.8335 - val_loss: 0.3508 -
val_accuracy: 0.8136 - 112s/epoch - 846ms/step
Epoch 38/100
132/132 - 112s - loss: 0.3480 - accuracy: 0.8362 - val_loss: 0.3369 -
val_accuracy: 0.8394 - 112s/epoch - 848ms/step
Epoch 39/100
132/132 - 107s - loss: 0.3479 - accuracy: 0.8426 - val_loss: 0.3797 -
val_accuracy: 0.8076 - 107s/epoch - 811ms/step
Epoch 40/100
132/132 - 107s - loss: 0.3570 - accuracy: 0.8278 - val_loss: 0.3510 -
val_accuracy: 0.8182 - 107s/epoch - 811ms/step
Epoch 41/100
132/132 - 107s - loss: 0.3484 - accuracy: 0.8385 - val_loss: 0.3155 -
val_accuracy: 0.8348 - 107s/epoch - 810ms/step
Epoch 42/100
132/132 - 112s - loss: 0.3456 - accuracy: 0.8392 - val_loss: 0.3414 -
val_accuracy: 0.8182 - 112s/epoch - 846ms/step
Epoch 43/100
132/132 - 107s - loss: 0.3524 - accuracy: 0.8369 - val_loss: 0.3629 -
val_accuracy: 0.8303 - 107s/epoch - 811ms/step
Epoch 44/100
132/132 - 107s - loss: 0.3734 - accuracy: 0.8396 - val_loss: 0.3142 -
val_accuracy: 0.8439 - 107s/epoch - 811ms/step
Epoch 45/100
132/132 - 107s - loss: 0.3479 - accuracy: 0.8411 - val_loss: 0.5330 -
val_accuracy: 0.7909 - 107s/epoch - 810ms/step
Epoch 46/100
132/132 - 107s - loss: 0.3596 - accuracy: 0.8347 - val_loss: 0.3082 -
val_accuracy: 0.8576 - 107s/epoch - 810ms/step
Epoch 47/100
132/132 - 110s - loss: 0.3641 - accuracy: 0.8320 - val_loss: 0.3101 -
val_accuracy: 0.8576 - 110s/epoch - 835ms/step
Epoch 48/100
132/132 - 107s - loss: 0.3704 - accuracy: 0.8343 - val_loss: 0.3034 -
val_accuracy: 0.8621 - 107s/epoch - 811ms/step
Epoch 49/100
132/132 - 107s - loss: 0.3448 - accuracy: 0.8453 - val_loss: 0.3148 -
val_accuracy: 0.8424 - 107s/epoch - 811ms/step
```

```
Epoch 50/100
132/132 - 107s - loss: 0.3402 - accuracy: 0.8400 - val_loss: 0.3323 -
val_accuracy: 0.8212 - 107s/epoch - 807ms/step
Epoch 51/100
132/132 - 112s - loss: 0.3340 - accuracy: 0.8422 - val_loss: 0.3663 -
val_accuracy: 0.8318 - 112s/epoch - 847ms/step
Epoch 52/100
132/132 - 107s - loss: 0.3297 - accuracy: 0.8495 - val_loss: 0.3040 -
val_accuracy: 0.8409 - 107s/epoch - 812ms/step
Epoch 53/100
132/132 - 107s - loss: 0.3369 - accuracy: 0.8506 - val_loss: 0.3480 -
val_accuracy: 0.8348 - 107s/epoch - 811ms/step
Epoch 54/100
132/132 - 111s - loss: 0.3488 - accuracy: 0.8335 - val_loss: 0.3265 -
val_accuracy: 0.8348 - 111s/epoch - 839ms/step
Epoch 55/100
132/132 - 107s - loss: 0.3384 - accuracy: 0.8510 - val_loss: 0.3228 -
val_accuracy: 0.8439 - 107s/epoch - 811ms/step
Epoch 56/100
132/132 - 111s - loss: 0.3289 - accuracy: 0.8479 - val_loss: 0.3407 -
val_accuracy: 0.8318 - 111s/epoch - 839ms/step
Epoch 57/100
132/132 - 111s - loss: 0.3475 - accuracy: 0.8400 - val_loss: 0.3080 -
val_accuracy: 0.8636 - 111s/epoch - 838ms/step
Epoch 58/100
132/132 - 111s - loss: 0.3435 - accuracy: 0.8441 - val_loss: 0.3776 -
val_accuracy: 0.8303 - 111s/epoch - 840ms/step
Epoch 59/100
132/132 - 110s - loss: 0.3323 - accuracy: 0.8373 - val_loss: 0.3187 -
val_accuracy: 0.8652 - 110s/epoch - 834ms/step
Epoch 60/100
132/132 - 107s - loss: 0.3393 - accuracy: 0.8449 - val_loss: 0.3337 -
val_accuracy: 0.8303 - 107s/epoch - 809ms/step
Epoch 61/100
132/132 - 110s - loss: 0.3416 - accuracy: 0.8536 - val_loss: 0.3176 -
val_accuracy: 0.8333 - 110s/epoch - 834ms/step
Epoch 62/100
132/132 - 110s - loss: 0.3392 - accuracy: 0.8312 - val_loss: 0.3372 -
val_accuracy: 0.8318 - 110s/epoch - 835ms/step
Epoch 63/100
132/132 - 107s - loss: 0.3353 - accuracy: 0.8601 - val_loss: 0.3465 -
val_accuracy: 0.8227 - 107s/epoch - 808ms/step
Epoch 64/100
132/132 - 107s - loss: 0.3243 - accuracy: 0.8479 - val_loss: 0.3944 -
val_accuracy: 0.8167 - 107s/epoch - 807ms/step
Epoch 65/100
132/132 - 112s - loss: 0.3272 - accuracy: 0.8366 - val_loss: 0.3163 -
val_accuracy: 0.8439 - 112s/epoch - 846ms/step
```

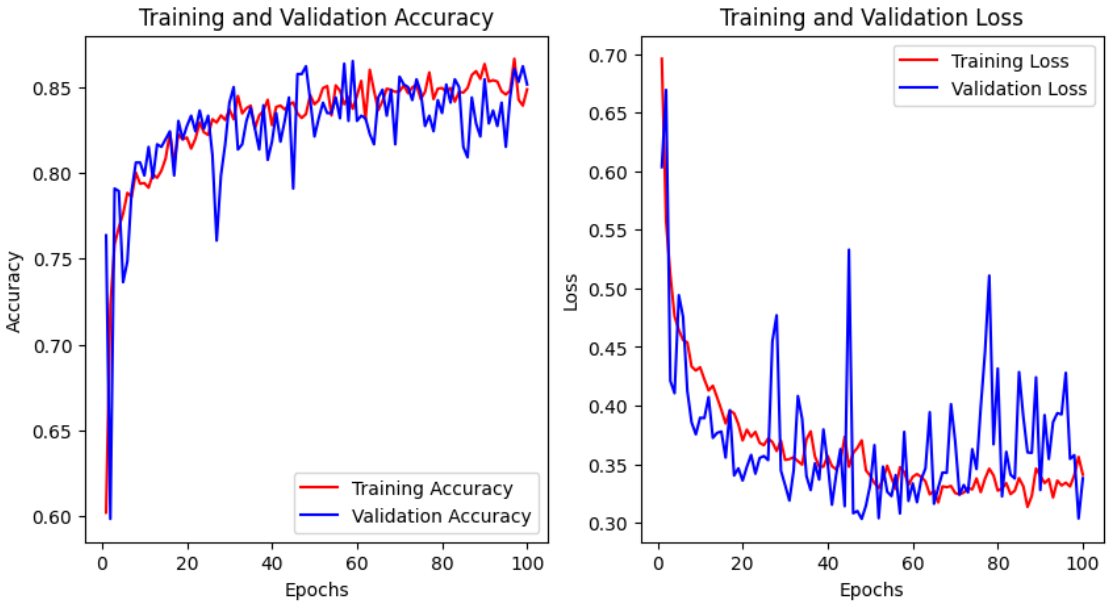
```
Epoch 66/100
132/132 - 107s - loss: 0.3174 - accuracy: 0.8419 - val_loss: 0.3300 -
val_accuracy: 0.8485 - 107s/epoch - 810ms/step
Epoch 67/100
132/132 - 111s - loss: 0.3311 - accuracy: 0.8491 - val_loss: 0.3430 -
val_accuracy: 0.8333 - 111s/epoch - 838ms/step
Epoch 68/100
132/132 - 107s - loss: 0.3304 - accuracy: 0.8483 - val_loss: 0.3427 -
val_accuracy: 0.8485 - 107s/epoch - 808ms/step
Epoch 69/100
132/132 - 111s - loss: 0.3315 - accuracy: 0.8472 - val_loss: 0.4012 -
val_accuracy: 0.8167 - 111s/epoch - 838ms/step
Epoch 70/100
132/132 - 107s - loss: 0.3254 - accuracy: 0.8476 - val_loss: 0.3692 -
val_accuracy: 0.8561 - 107s/epoch - 807ms/step
Epoch 71/100
132/132 - 111s - loss: 0.3244 - accuracy: 0.8510 - val_loss: 0.3240 -
val_accuracy: 0.8515 - 111s/epoch - 838ms/step
Epoch 72/100
132/132 - 110s - loss: 0.3259 - accuracy: 0.8464 - val_loss: 0.3323 -
val_accuracy: 0.8500 - 110s/epoch - 835ms/step
Epoch 73/100
132/132 - 110s - loss: 0.3296 - accuracy: 0.8498 - val_loss: 0.3259 -
val_accuracy: 0.8424 - 110s/epoch - 837ms/step
Epoch 74/100
132/132 - 110s - loss: 0.3285 - accuracy: 0.8513 - val_loss: 0.3628 -
val_accuracy: 0.8545 - 110s/epoch - 834ms/step
Epoch 75/100
132/132 - 110s - loss: 0.3379 - accuracy: 0.8430 - val_loss: 0.3459 -
val_accuracy: 0.8470 - 110s/epoch - 834ms/step
Epoch 76/100
132/132 - 110s - loss: 0.3262 - accuracy: 0.8472 - val_loss: 0.3988 -
val_accuracy: 0.8273 - 110s/epoch - 836ms/step
Epoch 77/100
132/132 - 110s - loss: 0.3377 - accuracy: 0.8586 - val_loss: 0.4457 -
val_accuracy: 0.8333 - 110s/epoch - 834ms/step
Epoch 78/100
132/132 - 112s - loss: 0.3462 - accuracy: 0.8430 - val_loss: 0.5109 -
val_accuracy: 0.8242 - 112s/epoch - 845ms/step
Epoch 79/100
132/132 - 107s - loss: 0.3405 - accuracy: 0.8491 - val_loss: 0.3670 -
val_accuracy: 0.8424 - 107s/epoch - 809ms/step
Epoch 80/100
132/132 - 107s - loss: 0.3274 - accuracy: 0.8495 - val_loss: 0.4317 -
val_accuracy: 0.8348 - 107s/epoch - 809ms/step
Epoch 81/100
132/132 - 107s - loss: 0.3295 - accuracy: 0.8472 - val_loss: 0.3226 -
val_accuracy: 0.8515 - 107s/epoch - 810ms/step
```

```
Epoch 82/100
132/132 - 107s - loss: 0.3338 - accuracy: 0.8495 - val_loss: 0.3606 -
val_accuracy: 0.8409 - 107s/epoch - 809ms/step
Epoch 83/100
132/132 - 111s - loss: 0.3245 - accuracy: 0.8415 - val_loss: 0.3411 -
val_accuracy: 0.8545 - 111s/epoch - 845ms/step
Epoch 84/100
132/132 - 110s - loss: 0.3275 - accuracy: 0.8472 - val_loss: 0.3379 -
val_accuracy: 0.8500 - 110s/epoch - 835ms/step
Epoch 85/100
132/132 - 107s - loss: 0.3381 - accuracy: 0.8468 - val_loss: 0.4287 -
val_accuracy: 0.8152 - 107s/epoch - 810ms/step
Epoch 86/100
132/132 - 110s - loss: 0.3312 - accuracy: 0.8495 - val_loss: 0.3906 -
val_accuracy: 0.8091 - 110s/epoch - 834ms/step
Epoch 87/100
132/132 - 110s - loss: 0.3136 - accuracy: 0.8570 - val_loss: 0.3601 -
val_accuracy: 0.8439 - 110s/epoch - 836ms/step
Epoch 88/100
132/132 - 107s - loss: 0.3231 - accuracy: 0.8593 - val_loss: 0.3598 -
val_accuracy: 0.8288 - 107s/epoch - 809ms/step
Epoch 89/100
132/132 - 110s - loss: 0.3464 - accuracy: 0.8548 - val_loss: 0.4242 -
val_accuracy: 0.8212 - 110s/epoch - 835ms/step
Epoch 90/100
132/132 - 107s - loss: 0.3403 - accuracy: 0.8635 - val_loss: 0.3280 -
val_accuracy: 0.8545 - 107s/epoch - 808ms/step
Epoch 91/100
132/132 - 110s - loss: 0.3339 - accuracy: 0.8532 - val_loss: 0.3920 -
val_accuracy: 0.8288 - 110s/epoch - 832ms/step
Epoch 92/100
132/132 - 107s - loss: 0.3373 - accuracy: 0.8540 - val_loss: 0.3545 -
val_accuracy: 0.8364 - 107s/epoch - 811ms/step
Epoch 93/100
132/132 - 107s - loss: 0.3217 - accuracy: 0.8532 - val_loss: 0.3859 -
val_accuracy: 0.8273 - 107s/epoch - 810ms/step
Epoch 94/100
132/132 - 110s - loss: 0.3361 - accuracy: 0.8476 - val_loss: 0.3936 -
val_accuracy: 0.8409 - 110s/epoch - 835ms/step
Epoch 95/100
132/132 - 107s - loss: 0.3319 - accuracy: 0.8457 - val_loss: 0.3925 -
val_accuracy: 0.8152 - 107s/epoch - 809ms/step
Epoch 96/100
132/132 - 110s - loss: 0.3343 - accuracy: 0.8479 - val_loss: 0.4280 -
val_accuracy: 0.8439 - 110s/epoch - 835ms/step
Epoch 97/100
132/132 - 107s - loss: 0.3310 - accuracy: 0.8665 - val_loss: 0.3546 -
val_accuracy: 0.8606 - 107s/epoch - 810ms/step
```

```
Epoch 98/100
132/132 - 107s - loss: 0.3407 - accuracy: 0.8426 - val_loss: 0.3578 -
val_accuracy: 0.8530 - 107s/epoch - 810ms/step
Epoch 99/100
132/132 - 111s - loss: 0.3563 - accuracy: 0.8392 - val_loss: 0.3037 -
val_accuracy: 0.8621 - 111s/epoch - 844ms/step
Epoch 100/100
132/132 - 115s - loss: 0.3411 - accuracy: 0.8487 - val_loss: 0.3381 -
val_accuracy: 0.8515 - 115s/epoch - 871ms/step
```

[11]:
```
performance_dict_self_selected_best_model12 =␣
 ↪history_self_selected_best_model23.history
accuracy_values = performance_dict_self_selected_best_model12["accuracy"]
val_accuracy_values =␣
 ↪performance_dict_self_selected_best_model12["val_accuracy"]
loss_values = performance_dict_self_selected_best_model12["loss"]
val_loss_values = performance_dict_self_selected_best_model12["val_loss"]
epochs = range(1, len(accuracy_values) + 1)

fig, axs = plt.subplots(1, 2, figsize=(10, 5))

axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",␣
 ↪color='red')
axs[0].plot(epochs, val_accuracy_values, "b", label="Validation Accuracy")
axs[0].set_title("Training and Validation Accuracy")
axs[0].set_xlabel("Epochs")
axs[0].set_ylabel("Accuracy")
axs[0].legend()

axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
axs[1].plot(epochs, val_loss_values, "b", label="Validation Loss")
axs[1].set_title("Training and Validation Loss")
axs[1].set_xlabel("Epochs")
axs[1].set_ylabel("Loss")
axs[1].legend()

plt.show()
```

```
<ipython-input-11-b3ff324a21e2>:10: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[0].plot(epochs, accuracy_values, "b", label="Training Accuracy",
color='red')
<ipython-input-11-b3ff324a21e2>:17: UserWarning: color is redundantly defined by
the 'color' keyword argument and the fmt string "b" (-> color=(0.0, 0.0, 1.0,
1)). The keyword argument will take precedence.
  axs[1].plot(epochs, loss_values, "b", label="Training Loss", color='red')
```



This model will be our final model as we no longer see overfitting as we are training on augmented images, and testing on non-augmented images. We also have implemented dropout on numerous layers. Our training accuracy high is 86.65% and validation accuracy high is 86.52%.