Intro to Health Analytics - Final Project

Objective:

- Create a neural network model which outputs a percentage likelihood the user has colorectal cancer

Import Libraries

In [1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os
import matplotlib.pyplot as plt
```

Import the Training & Testing Datasets

The original image sizes are (760, 576), and due to file size constraints with Tensorflow running off of my computer's CPU instead of a dedicated GPU, they are reduced to half the size. 20% of the data is used as validation data for the model, and the seed is set at 1337 for reproducibility

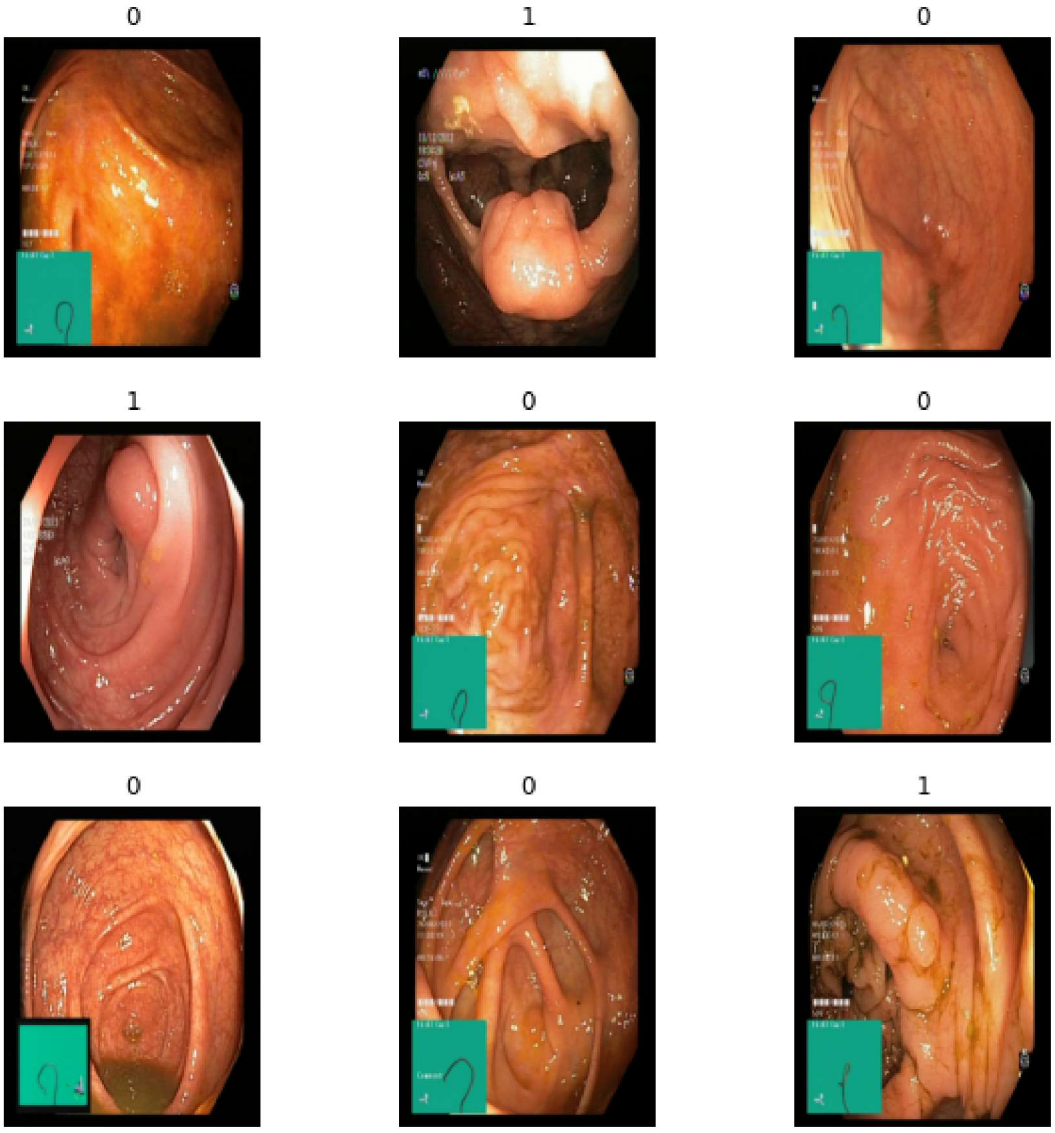
In [3]:
```
image_size = (360, 288)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "IHDA-Images",
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "IHDA-Images",
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=image_size,
    batch_size=batch_size,
)
```

```
Found 1000 files belonging to 2 classes.
Using 800 files for training.
Found 1000 files belonging to 2 classes.
Using 200 files for validation.
```

Printing Sample Images

The image labelsl are listed above each image, with a label of 1 indicating polyps are present, and 0 indicating a normal cecum scan

In [4]:
```
plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(int(labels[i]))
        plt.axis("off")
```



Creating Sample Data for the Dataset

The original image are processed using a Keras model layer to randomly flip the scans on the horizontal axis, or rotate the image by 10% randomly

In [6]:
```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.1),
    ]
)
```

Mapping the Data Augmentation Steps

The original training dataset is mapped to include horizontally flipped and randomly rotated versions

of the original image, to increase the amount of training data that is available, and also make the training data more diverse

In [7]:
```
augmented_train_ds = train_ds.map(
  lambda x, y: (data_augmentation(x, training=True), y))
```

In [8]:
```
train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)
```

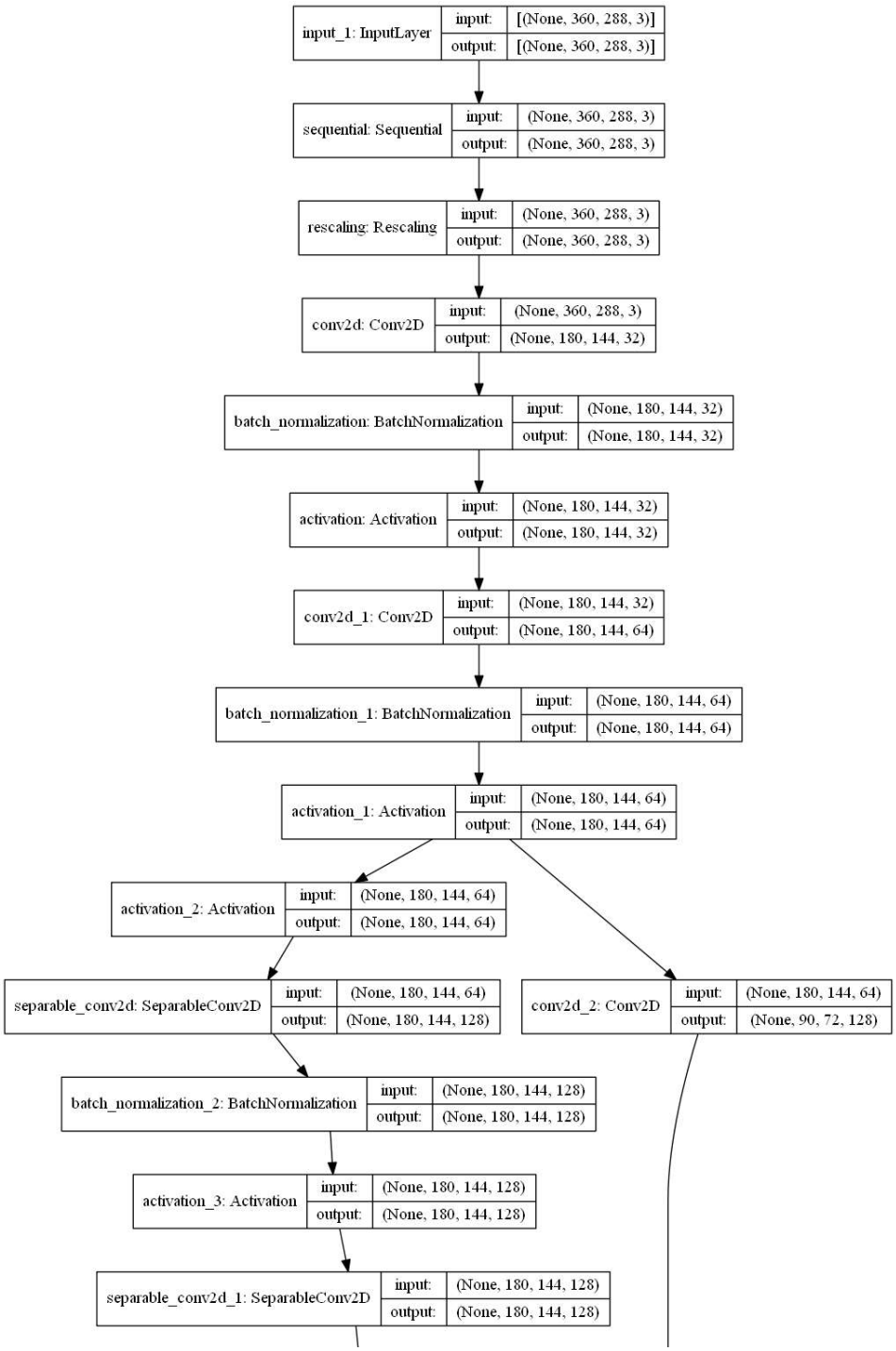
Creating the Neural Network

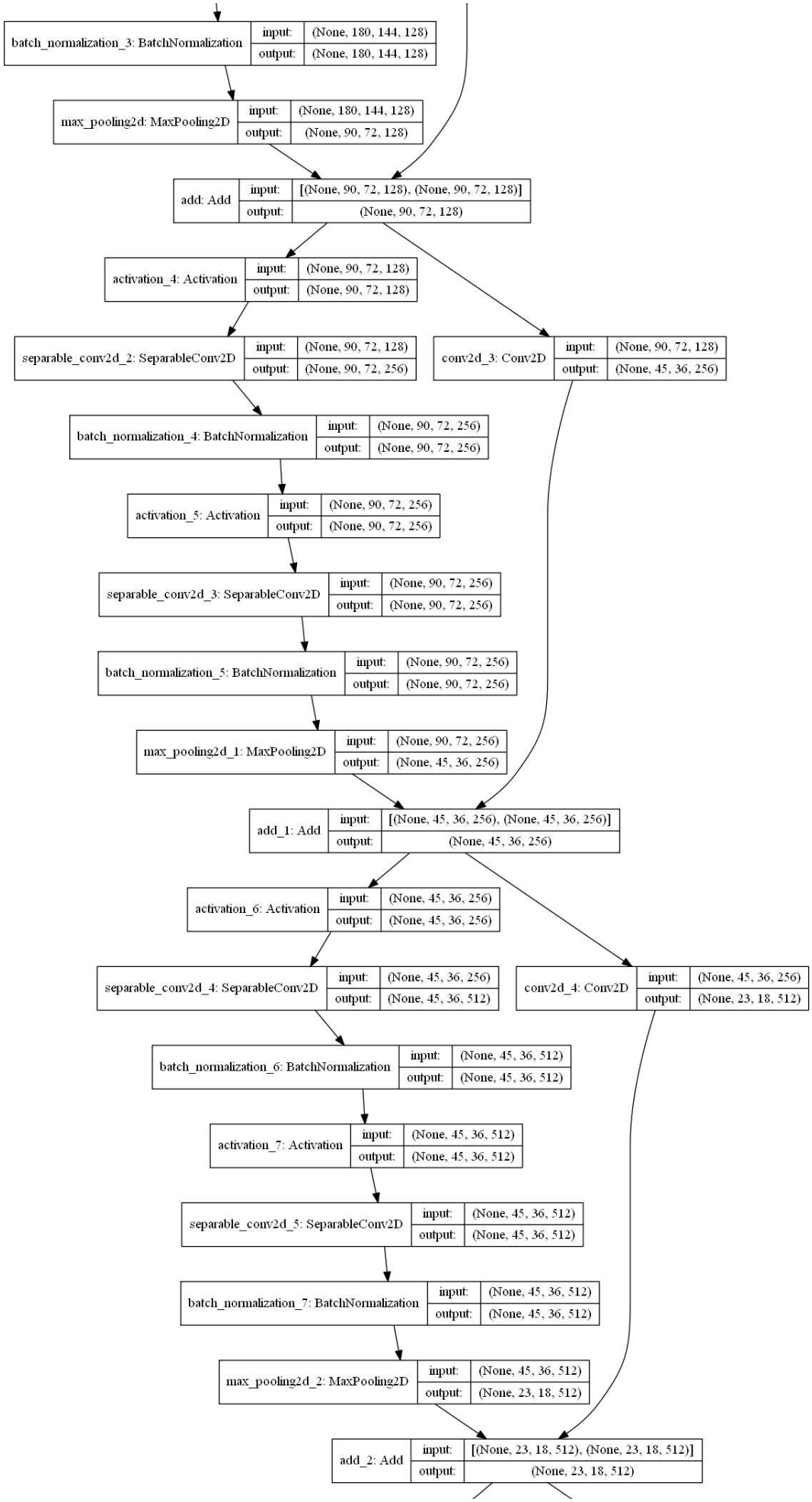
The original images here are rescaled, converged into a 2-dimensional convolution layer, normalized, and transformed via a rectified linear activation function (ReLU) for various image sizes, and then the average result for each of the image sizes is finally placed into a Softmax function, which takes an input and normalizes the input into a probability distribution to understand what the percent likelihood that a given result belongs to the class (i.e., the image contains polyps)

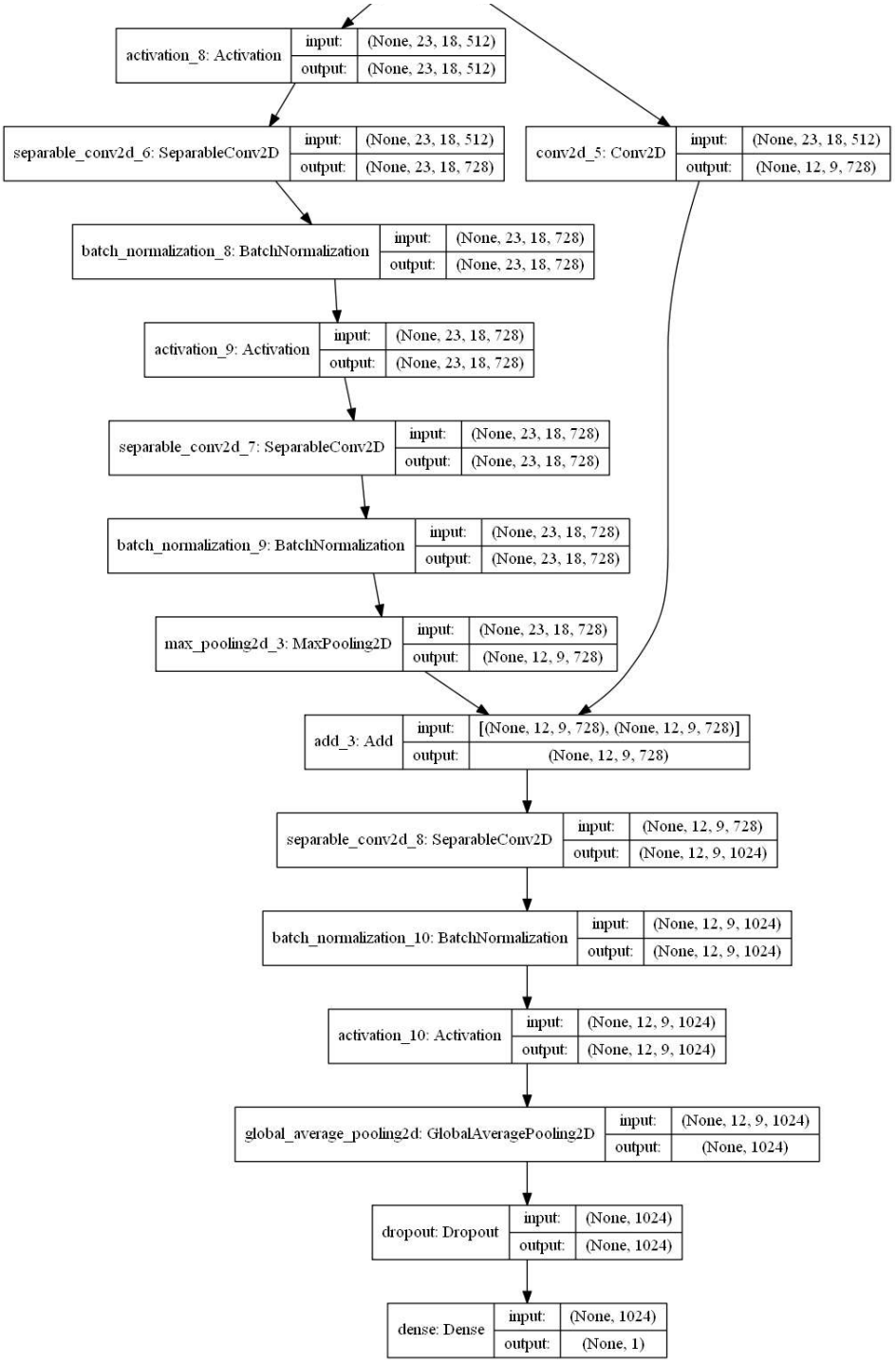
In [9]:
```
def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)

    # Entry block
    x = layers.experimental.preprocessing.Rescaling(1.0 / 255)(x)
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
```

```
    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size + (3,), num_classes=2)
keras.utils.plot_model(model, show_shapes=True)
```

Out[9]:




batch_normalization_3: BatchNormalization
input: (None, 180, 144, 128)
output: (None, 180, 144, 128)
max_pooling2d: MaxPooling2D
input: (None, 180, 144, 128)
output: (None, 90, 72, 128)
add: Add
input: [(None, 90, 72, 128), (None, 90, 72, 128)]
output: (None, 90, 72, 128)
activation_4: Activation
input: (None, 90, 72, 128)
output: (None, 90, 72, 128)
separable_conv2d_2: SeparableConv2D
input: (None, 90, 72, 128)
output: (None, 90, 72, 256)
conv2d_3: Conv2D
input: (None, 90, 72, 128)
output: (None, 45, 36, 256)
batch_normalization_4: BatchNormalization
input: (None, 90, 72, 256)
output: (None, 90, 72, 256)
activation_5: Activation
input: (None, 90, 72, 256)
output: (None, 90, 72, 256)
separable_conv2d_3: SeparableConv2D
input: (None, 90, 72, 256)
output: (None, 90, 72, 256)
batch_normalization_5: BatchNormalization
input: (None, 90, 72, 256)
output: (None, 90, 72, 256)
max_pooling2d_1: MaxPooling2D
input: (None, 90, 72, 256)
output: (None, 45, 36, 256)
add_1: Add
input: [(None, 45, 36, 256), (None, 45, 36, 256)]
output: (None, 45, 36, 256)
activation_6: Activation
input: (None, 45, 36, 256)
output: (None, 45, 36, 256)
separable_conv2d_4: SeparableConv2D
input: (None, 45, 36, 256)
output: (None, 45, 36, 512)
conv2d_4: Conv2D
input: (None, 45, 36, 256)
output: (None, 23, 18, 512)
batch_normalization_6: BatchNormalization
input: (None, 45, 36, 512)
output: (None, 45, 36, 512)
activation_7: Activation
input: (None, 45, 36, 512)
output: (None, 45, 36, 512)
separable_conv2d_5: SeparableConv2D
input: (None, 45, 36, 512)
output: (None, 45, 36, 512)
batch_normalization_7: BatchNormalization
input: (None, 45, 36, 512)
output: (None, 45, 36, 512)
max_pooling2d_2: MaxPooling2D
input: (None, 45, 36, 512)
output: (None, 23, 18, 512)
add_2: Add
input: [(None, 23, 18, 512), (None, 23, 18, 512)]
output: (None, 23, 18, 512)



Training the Neural Network

To train the neural network, 50 epochs were used to capture the loss, accuracy, validation loss, and validation accuracy of each training phase. Based on the results below, epoch 49 produced the best resulting model, and thus will be the .h5 TensorFlow model that will be saved and used in the application

In [10]:
```
epochs = 50

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
```

```
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

```
Epoch 1/50
25/25 [==============================] - 1045s 41s/step - loss: 0.5781 - accuracy: 0.719
2 - val_loss: 0.6910 - val_accuracy: 0.5350
Epoch 2/50
25/25 [==============================] - 934s 37s/step - loss: 0.4101 - accuracy: 0.8107
- val_loss: 0.6897 - val_accuracy: 0.5400
Epoch 3/50
25/25 [==============================] - 989s 39s/step - loss: 0.3464 - accuracy: 0.8486
- val_loss: 0.6943 - val_accuracy: 0.4650
Epoch 4/50
25/25 [==============================] - 1007s 40s/step - loss: 0.3333 - accuracy: 0.855
8 - val_loss: 0.7137 - val_accuracy: 0.4650
Epoch 5/50
25/25 [==============================] - 964s 38s/step - loss: 0.2704 - accuracy: 0.8920
- val_loss: 0.7755 - val_accuracy: 0.4650
Epoch 6/50
25/25 [==============================] - 1003s 39s/step - loss: 0.2804 - accuracy: 0.891
2 - val_loss: 0.7926 - val_accuracy: 0.4650
Epoch 7/50
25/25 [==============================] - 963s 38s/step - loss: 0.2454 - accuracy: 0.9093
- val_loss: 0.8401 - val_accuracy: 0.4650
Epoch 8/50
25/25 [==============================] - 1057s 42s/step - loss: 0.2063 - accuracy: 0.921
5 - val_loss: 0.9384 - val_accuracy: 0.4650
Epoch 9/50
25/25 [==============================] - 938s 37s/step - loss: 0.2529 - accuracy: 0.9179
- val_loss: 1.0331 - val_accuracy: 0.4650
Epoch 10/50
25/25 [==============================] - 930s 37s/step - loss: 0.1889 - accuracy: 0.9294
- val_loss: 0.9970 - val_accuracy: 0.4650
Epoch 11/50
25/25 [==============================] - 938s 37s/step - loss: 0.2013 - accuracy: 0.9219
- val_loss: 1.2329 - val_accuracy: 0.4650
Epoch 12/50
25/25 [==============================] - 972s 38s/step - loss: 0.1926 - accuracy: 0.9236
- val_loss: 1.7794 - val_accuracy: 0.4650
Epoch 13/50
25/25 [==============================] - 1006s 40s/step - loss: 0.1610 - accuracy: 0.931
5 - val_loss: 1.3094 - val_accuracy: 0.4650
Epoch 14/50
25/25 [==============================] - 951s 37s/step - loss: 0.1741 - accuracy: 0.9326
- val_loss: 1.8482 - val_accuracy: 0.4650
Epoch 15/50
25/25 [==============================] - 992s 39s/step - loss: 0.1854 - accuracy: 0.9295
- val_loss: 1.7151 - val_accuracy: 0.4650
Epoch 16/50
25/25 [==============================] - 939s 37s/step - loss: 0.1923 - accuracy: 0.9299
- val_loss: 1.0583 - val_accuracy: 0.4650
Epoch 17/50
25/25 [==============================] - 957s 38s/step - loss: 0.1620 - accuracy: 0.9472
- val_loss: 0.5027 - val_accuracy: 0.7550
Epoch 18/50
25/25 [==============================] - 937s 37s/step - loss: 0.1767 - accuracy: 0.9281
- val_loss: 1.5292 - val_accuracy: 0.4650
```

```
Epoch 19/50
25/25 [==============================] - 934s 37s/step - loss: 0.2109 - accuracy: 0.9309
- val_loss: 0.8678 - val_accuracy: 0.5600
Epoch 20/50
25/25 [==============================] - 911s 36s/step - loss: 0.1480 - accuracy: 0.9453
- val_loss: 0.3549 - val_accuracy: 0.8600
Epoch 21/50
25/25 [==============================] - 928s 36s/step - loss: 0.1778 - accuracy: 0.9329
- val_loss: 0.3107 - val_accuracy: 0.8900
Epoch 22/50
25/25 [==============================] - 1010s 40s/step - loss: 0.1768 - accuracy: 0.941
8 - val_loss: 0.2973 - val_accuracy: 0.8550
Epoch 23/50
25/25 [==============================] - 969s 38s/step - loss: 0.1617 - accuracy: 0.9330
- val_loss: 0.1569 - val_accuracy: 0.9450
Epoch 24/50
25/25 [==============================] - 950s 37s/step - loss: 0.1693 - accuracy: 0.9394
- val_loss: 2.3610 - val_accuracy: 0.7250
Epoch 25/50
25/25 [==============================] - 934s 37s/step - loss: 0.1557 - accuracy: 0.9452
- val_loss: 0.1878 - val_accuracy: 0.9250
Epoch 26/50
25/25 [==============================] - 942s 37s/step - loss: 0.1740 - accuracy: 0.9454
- val_loss: 0.9579 - val_accuracy: 0.6200
Epoch 27/50
25/25 [==============================] - 980s 38s/step - loss: 0.1336 - accuracy: 0.9569
- val_loss: 2.5607 - val_accuracy: 0.5350
Epoch 28/50
25/25 [==============================] - 953s 38s/step - loss: 0.1809 - accuracy: 0.9373
- val_loss: 0.9017 - val_accuracy: 0.6900
Epoch 29/50
25/25 [==============================] - 946s 37s/step - loss: 0.1565 - accuracy: 0.9328
- val_loss: 0.3233 - val_accuracy: 0.8550
Epoch 30/50
25/25 [==============================] - 952s 37s/step - loss: 0.1220 - accuracy: 0.9528
- val_loss: 1.3288 - val_accuracy: 0.6150
Epoch 31/50
25/25 [==============================] - 980s 38s/step - loss: 0.0915 - accuracy: 0.9726
- val_loss: 0.1855 - val_accuracy: 0.9500
Epoch 32/50
25/25 [==============================] - 932s 36s/step - loss: 0.1640 - accuracy: 0.9405
- val_loss: 0.5312 - val_accuracy: 0.8200
Epoch 33/50
25/25 [==============================] - 899s 35s/step - loss: 0.1140 - accuracy: 0.9635
- val_loss: 0.1796 - val_accuracy: 0.9350
Epoch 34/50
25/25 [==============================] - 941s 37s/step - loss: 0.1189 - accuracy: 0.9515
- val_loss: 0.5250 - val_accuracy: 0.8600
Epoch 35/50
25/25 [==============================] - 954s 38s/step - loss: 0.1383 - accuracy: 0.9413
- val_loss: 1.1515 - val_accuracy: 0.6400
Epoch 36/50
25/25 [==============================] - 934s 37s/step - loss: 0.1110 - accuracy: 0.9608
- val_loss: 2.9635 - val_accuracy: 0.5500
Epoch 37/50
25/25 [==============================] - 914s 36s/step - loss: 0.0902 - accuracy: 0.9559
- val_loss: 0.1434 - val_accuracy: 0.9300
Epoch 38/50
25/25 [==============================] - 1048s 41s/step - loss: 0.1022 - accuracy: 0.965
3 - val_loss: 3.8127 - val_accuracy: 0.7450
Epoch 39/50
25/25 [==============================] - 948s 37s/step - loss: 0.1277 - accuracy: 0.9477
- val_loss: 1.1155 - val_accuracy: 0.6800
Epoch 40/50
25/25 [==============================] - 905s 35s/step - loss: 0.1038 - accuracy: 0.9656
```

```
- val_loss: 1.4940 - val_accuracy: 0.6550
Epoch 41/50
25/25 [==============================] - 1010s 40s/step - loss: 0.1444 - accuracy: 0.947
9 - val_loss: 1.1634 - val_accuracy: 0.6400
Epoch 42/50
25/25 [==============================] - 947s 37s/step - loss: 0.1047 - accuracy: 0.9565
- val_loss: 0.4188 - val_accuracy: 0.9250
Epoch 43/50
25/25 [==============================] - 962s 38s/step - loss: 0.0902 - accuracy: 0.9699
- val_loss: 0.3454 - val_accuracy: 0.8650
Epoch 44/50
25/25 [==============================] - 939s 37s/step - loss: 0.1105 - accuracy: 0.9701
- val_loss: 0.3938 - val_accuracy: 0.9000
Epoch 45/50
25/25 [==============================] - 1024s 40s/step - loss: 0.1159 - accuracy: 0.955
8 - val_loss: 0.7379 - val_accuracy: 0.8000
Epoch 46/50
25/25 [==============================] - 971s 38s/step - loss: 0.1170 - accuracy: 0.9630
- val_loss: 0.2213 - val_accuracy: 0.9400
Epoch 47/50
25/25 [==============================] - 1025s 40s/step - loss: 0.1047 - accuracy: 0.964
8 - val_loss: 0.1469 - val_accuracy: 0.9500
Epoch 48/50
25/25 [==============================] - 907s 36s/step - loss: 0.1081 - accuracy: 0.9581
- val_loss: 0.8003 - val_accuracy: 0.7950
Epoch 49/50
25/25 [==============================] - 1011s 40s/step - loss: 0.1089 - accuracy: 0.960
0 - val_loss: 0.1950 - val_accuracy: 0.9450
Epoch 50/50
25/25 [==============================] - 910s 36s/step - loss: 0.0991 - accuracy: 0.9631
- val_loss: 3.2929 - val_accuracy: 0.5600
```

Out[10]: <tensorflow.python.keras.callbacks.History at 0x1e3d48cedc0>

Saving Keras Model

here we are saving the output of the Keras model that performed the best among all the Epochs. This will be imported into the Flask app

In [11]:
```
model.save('Keras Model - Predicting Polyps.h5')
```