Classification with BHI Dataset and VGG-style network

In this experiment you will set up a VGG-style network to classify histopathologic scans of breast tissue from the [BHI (https://www.kaggle.com/paultimothymooney/breast-histopathology-images)](https://www.kaggle.com/paultimothymooney/breast-histopathology-images) dataset.

In [1]:
```
import tensorflow.keras as keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten, MaxPooling2D
from tensorflow.keras.optimizers import SGD, Adam
from matplotlib import pyplot as plt
import numpy as np
```

```
WARNING:tensorflow:From C:\Users\Johan\anaconda3\envs\py311\Lib\site-packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Here we use a Keras utility function to load the dataset. I already organized the data into HDF5 files which are a good format for storing array data.

In [2]:
```
from tensorflow.keras.utils import get_file
x_train_path = get_file('idc_train.h5','https://storage.googleapis.com/data401-datasets/idc_train.h5')
x_test_path = get_file('idc_test.h5','https://storage.googleapis.com/data401-datasets/idc_test.h5')
```

We read the data from the HDF5 files into Numpy arrays.

I crop the images so they are all 48x48.

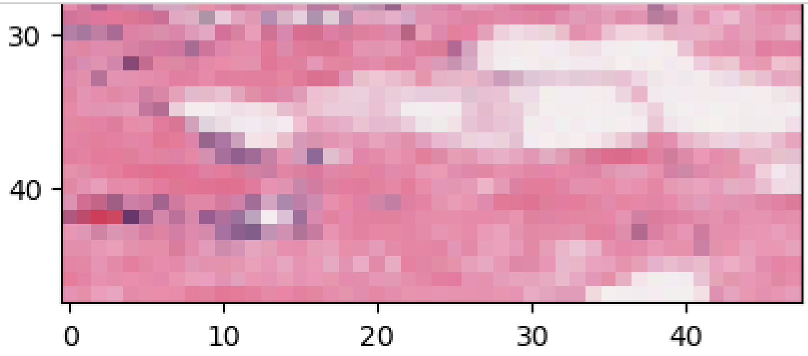
In [3]:
```
import h5py as h5
with h5.File(x_train_path,'r') as f:
  x_train = f['X'][::2,1:49,1:49] # Load half the data to avoid out-of-memory errors
  y_train = f['y'][::2]
with h5.File(x_test_path,'r') as f:
  x_test = f['X'][:,1:49,1:49]
  y_test = f['y'][:]
```

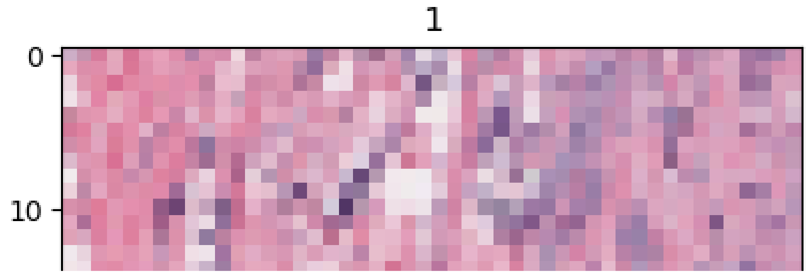
In [4]:
```
x_train.shape,y_train.shape,x_test.shape,y_test.shape
```

Out[4]: ((61925, 48, 48, 3), (61925,), (13761, 48, 48, 3), (13761,))

Showing a few images from the dataset.

In [5]:
```
for i in range(5):
  plt.imshow(np.squeeze(x_train[i]))
  plt.title(y_train[i])
  plt.show()
```





Data preprocessing

1. Convert the train and test images to floating point and divide by 255.
2. Compute the average value of the entire training image set.
3. Subtract the average value from the training and testing images.

In [6]:
```
x_test = x_test.astype("float32")/255
x_train = x_train.astype("float32")/255

x_train_avg = np.mean(x_train)

x_train = x_train-x_train_avg
x_test = x_test-x_train_avg
```

Build a VGG-style binary classifier model. For example, your network could contain the following:

1. 32 convolutional filters of size 3x3, zero padding, ReLU activation
2. 2x2 max pooling with stride 2
3. 64 filters
4. max pool
5. 128 filters
6. max pool
7. 256 filters
8. max pool
9. flatten
10. Fully-connected layer with 128 outputs
11. Final binary classification layer

In [59]:
```
model = Sequential([
        Input(x_train.shape[1:]),
        Conv2D(32,3,activation='relu',padding='same',name='conv1'),
        MaxPooling2D(2,2),
        Conv2D(64,3,activation='relu',padding='same',name='conv2'),
        MaxPooling2D(2,2),
        Conv2D(128,3,activation='relu',padding='same',name='conv3'),
        MaxPooling2D(2,2),
        Conv2D(256,3,activation='relu',padding='same',name='conv4'),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(128,activation='relu',name='dense1'),
        Dense(2,activation='softmax',name='z')
])
model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1 (Conv2D)              (None, 48, 48, 32)        896       
                                                                 
 max_pooling2d_26 (MaxPooli  (None, 24, 24, 32)        0         
 ng2D)                                                           
                                                                 
 conv2 (Conv2D)              (None, 24, 24, 64)        18496     
                                                                 
 max_pooling2d_27 (MaxPooli  (None, 12, 12, 64)        0         
 ng2D)                                                           
                                                                 
 conv3 (Conv2D)              (None, 12, 12, 128)       73856     
                                                                 
 max_pooling2d_28 (MaxPooli  (None, 6, 6, 128)         0         
 ng2D)                                                           
                                                                 
 conv4 (Conv2D)              (None, 6, 6, 256)         295168    
                                                                 
 max_pooling2d_29 (MaxPooli  (None, 3, 3, 256)         0         
 ng2D)                                                           
                                                                 
 flatten_6 (Flatten)         (None, 2304)              0         
                                                                 
 dense1 (Dense)              (None, 128)               295040    
                                                                 
 z (Dense)                   (None, 2)                 258       
                                                                 
=================================================================
Total params: 683714 (2.61 MB)
Trainable params: 683714 (2.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Set up the model to optimize the sparse categorical cross-entropy loss using Adam optimizer and learning rate of $.0003$. Calculate accuracy metrics during training.

In [60]:

```
learning_rate = 3e-4


opt = Adam(learning_rate=learning_rate)

model.compile(loss='sparse_categorical_crossentropy',optimizer=opt,metrics='accuracy')
```

Now `fit` the model to the data using a batch size of 32 and 10% validation split over 10 epochs.

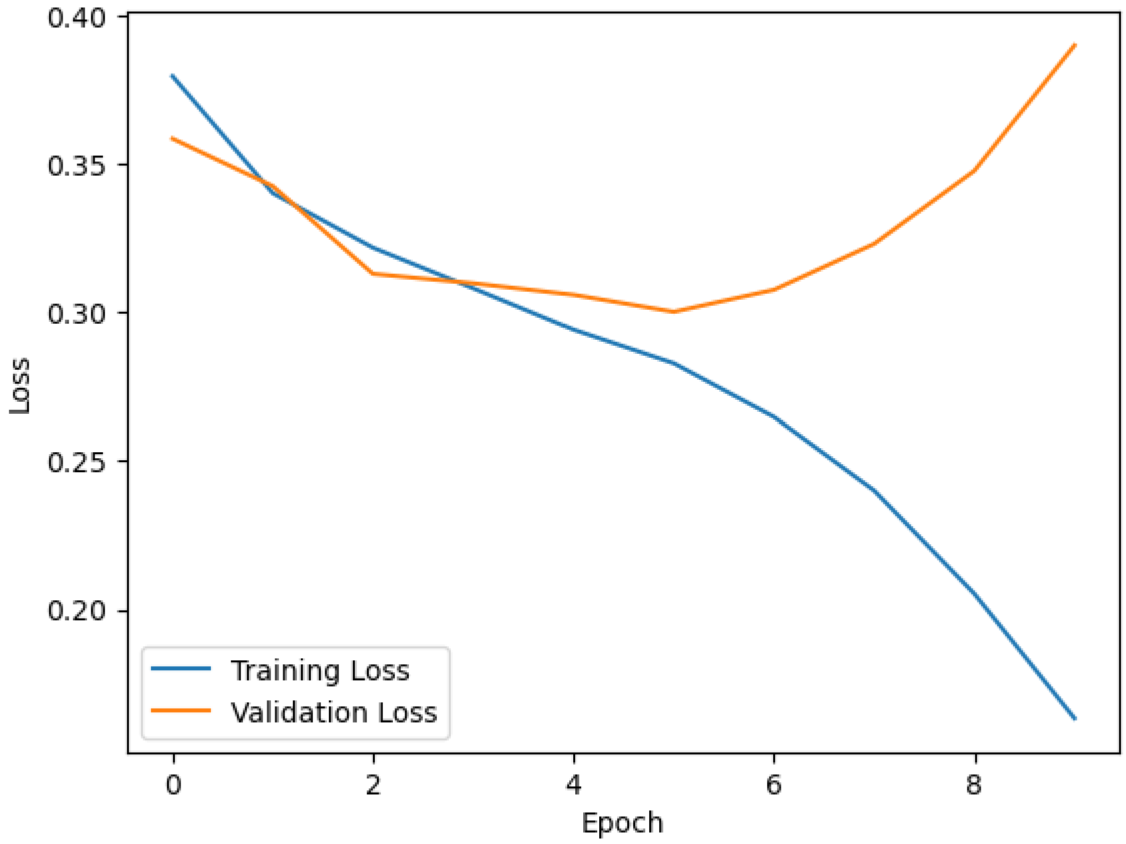
In [61]:
```
batch_size = 32
epochs = 10

history = model.fit(x_train,y_train,batch_size=batch_size,epochs=epochs,validation_split=0.1,verbose=True)
```

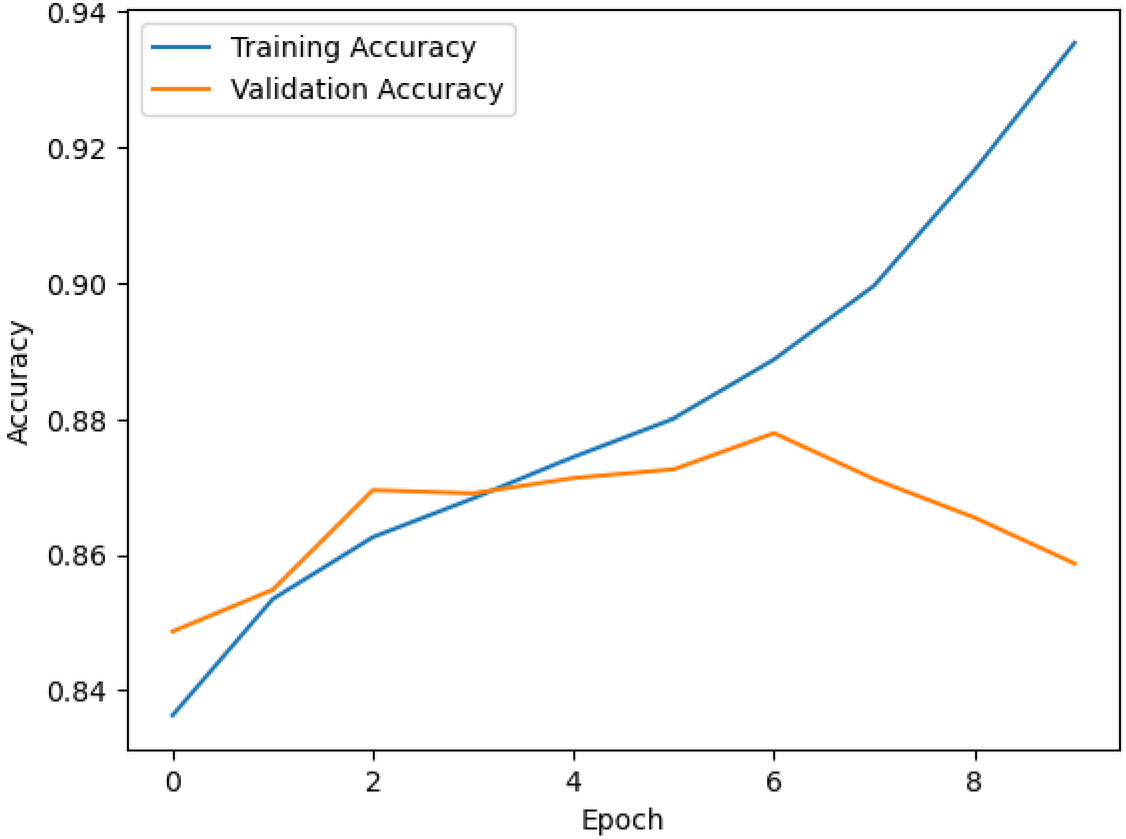
```
Epoch 1/10
1742/1742 [==============================] - 64s 35ms/step - loss: 0.3796 - accuracy: 0.8363 - val_loss: 0.3585 -
val_accuracy: 0.8487
Epoch 2/10
1742/1742 [==============================] - 52s 30ms/step - loss: 0.3402 - accuracy: 0.8535 - val_loss: 0.3425 -
val_accuracy: 0.8548
Epoch 3/10
1742/1742 [==============================] - 69s 39ms/step - loss: 0.3218 - accuracy: 0.8626 - val_loss: 0.3130 -
val_accuracy: 0.8695
Epoch 4/10
1742/1742 [==============================] - 55s 32ms/step - loss: 0.3081 - accuracy: 0.8683 - val_loss: 0.3098 -
val_accuracy: 0.8690
Epoch 5/10
1742/1742 [==============================] - 55s 32ms/step - loss: 0.2941 - accuracy: 0.8744 - val_loss: 0.3059 -
val_accuracy: 0.8713
Epoch 6/10
1742/1742 [==============================] - 53s 30ms/step - loss: 0.2828 - accuracy: 0.8801 - val_loss: 0.3001 -
val_accuracy: 0.8726
Epoch 7/10
1742/1742 [==============================] - 65s 37ms/step - loss: 0.2648 - accuracy: 0.8887 - val_loss: 0.3076 -
val_accuracy: 0.8779
Epoch 8/10
1742/1742 [==============================] - 54s 31ms/step - loss: 0.2401 - accuracy: 0.8996 - val_loss: 0.3231 -
val_accuracy: 0.8711
Epoch 9/10
1742/1742 [==============================] - 61s 35ms/step - loss: 0.2053 - accuracy: 0.9166 - val_loss: 0.3477 -
val_accuracy: 0.8655
Epoch 10/10
1742/1742 [==============================] - 53s 30ms/step - loss: 0.1633 - accuracy: 0.9354 - val_loss: 0.3900 -
val_accuracy: 0.8587
```

Plot loss and accuracy over the training run.

In [62]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.legend(['Training Loss','Validation Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



In [63]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['Training Accuracy','Validation Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```



Compute accuracy of the model on the training and testing sets.

In [64]:

```
# Evaluate the model on the test data
loss, accuracy = model.evaluate(x_test, y_test)

print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
431/431 [==============================] - 6s 13ms/step - loss: 0.3984 - accuracy: 0.8568
Test Loss: 0.3984
Test Accuracy: 85.68%
```

Try a different setting to see if you can improve the test set accuracy at all. Write about the results here.

Type *Markdown* and LaTeX: α^2