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) 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.

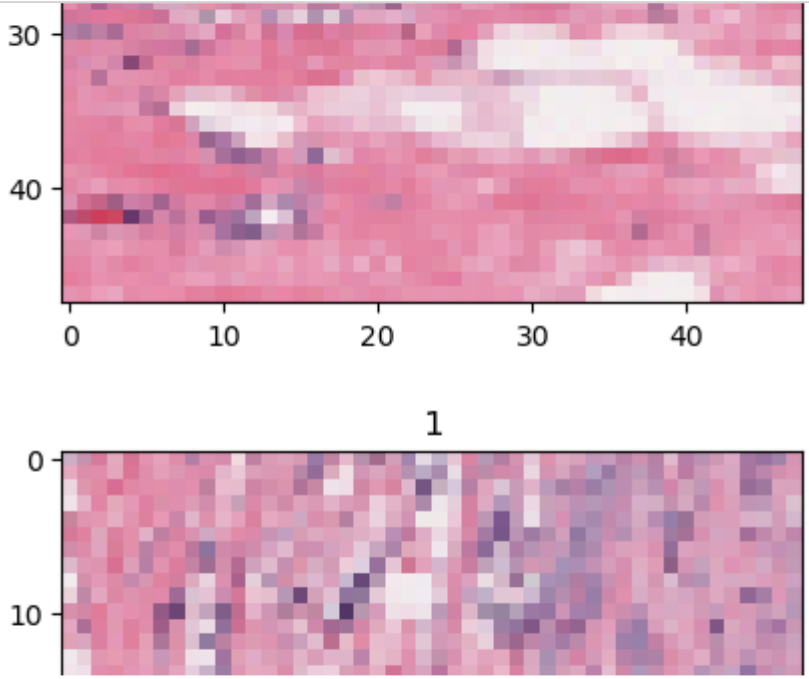
In [3]:
```
import h5py as h5
with h5.File(x_train_path,'r') as f:
  x_train = f['X'][::2,1:49,1:49] # load half the data to avoid out-of-memory errors
  y_train = f['y'][::2]
with h5.File(x_test_path,'r') as f:
  x_test = f['X'][:,1:49,1:49]
  y_test = f['y'][:]
```

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 [38]:

```
model = Sequential([
    Input(x_train.shape[1:]),
     Conv2D(64,3,activation='relu',padding='same',name='conv1'),
     MaxPooling2D(2,2),
     Conv2D(128,3,activation='relu',padding='same',name='conv2'),
     MaxPooling2D(2,2),
     Conv2D(256,3,activation='relu',padding='same',name='conv3'),
     MaxPooling2D(2,2),
     Conv2D(512,3,activation='relu',padding='same',name='conv4'),
     MaxPooling2D(2,2),
     Conv2D(512,3,activation='relu',padding='same',name='conv5'),
     MaxPooling2D(2,2),
     Flatten(),
     Dense(128,activation='relu',name='dense1'),
     Dense(2,activation='softmax',name='z')
])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1 (Conv2D)              (None, 48, 48, 64)        1792

 max_pooling2d_13 (MaxPooli  (None, 24, 24, 64)        0
 ng2D)

 conv2 (Conv2D)              (None, 24, 24, 128)       73856

 max_pooling2d_14 (MaxPooli  (None, 12, 12, 128)       0
 ng2D)

 conv3 (Conv2D)              (None, 12, 12, 256)       295168

 max_pooling2d_15 (MaxPooli  (None, 6, 6, 256)         0
 ng2D)

 conv4 (Conv2D)              (None, 6, 6, 512)         1180160

 max_pooling2d_16 (MaxPooli  (None, 3, 3, 512)         0
 ng2D)

 conv5 (Conv2D)              (None, 3, 3, 512)         2359808

 max_pooling2d_17 (MaxPooli  (None, 1, 1, 512)         0
 ng2D)

 flatten_3 (Flatten)         (None, 512)               0

 dense1 (Dense)              (None, 128)               65664

 z (Dense)                   (None, 2)                 258

=================================================================
Total params: 3976706 (15.17 MB)
Trainable params: 3976706 (15.17 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 [39]:

```
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.

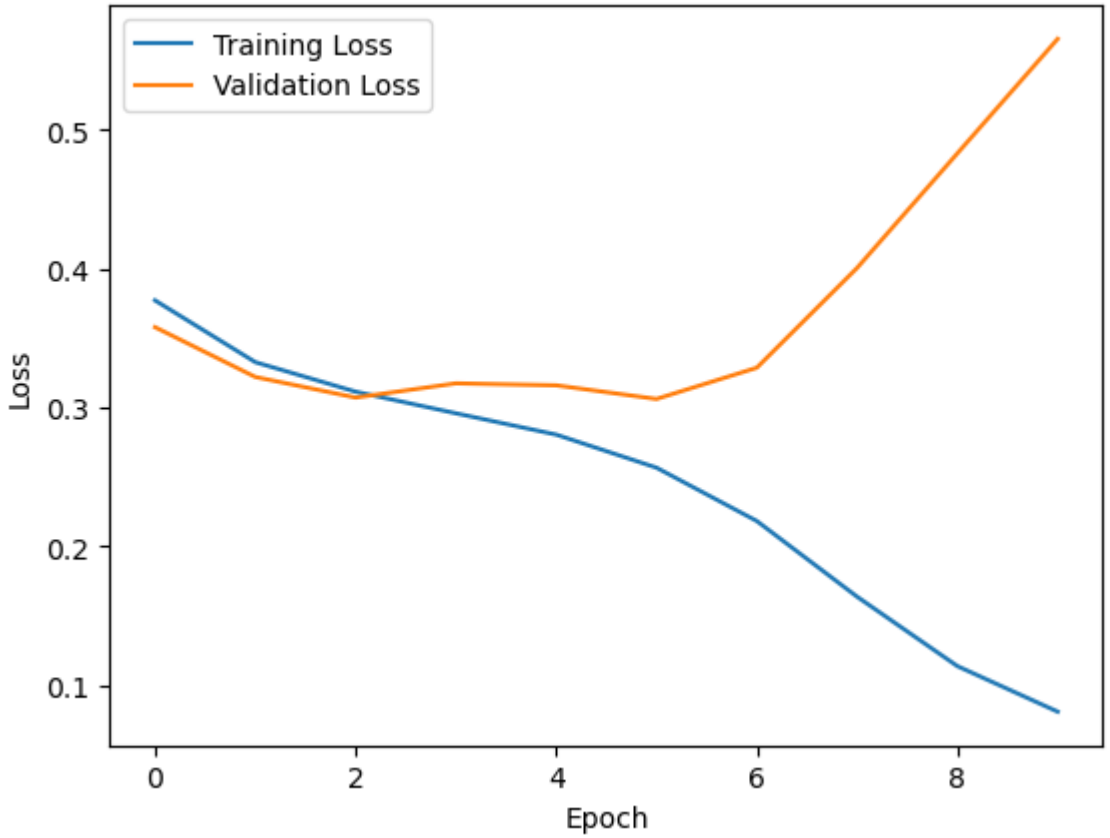
In [40]:
```
batch_size = 32
epochs = 10

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

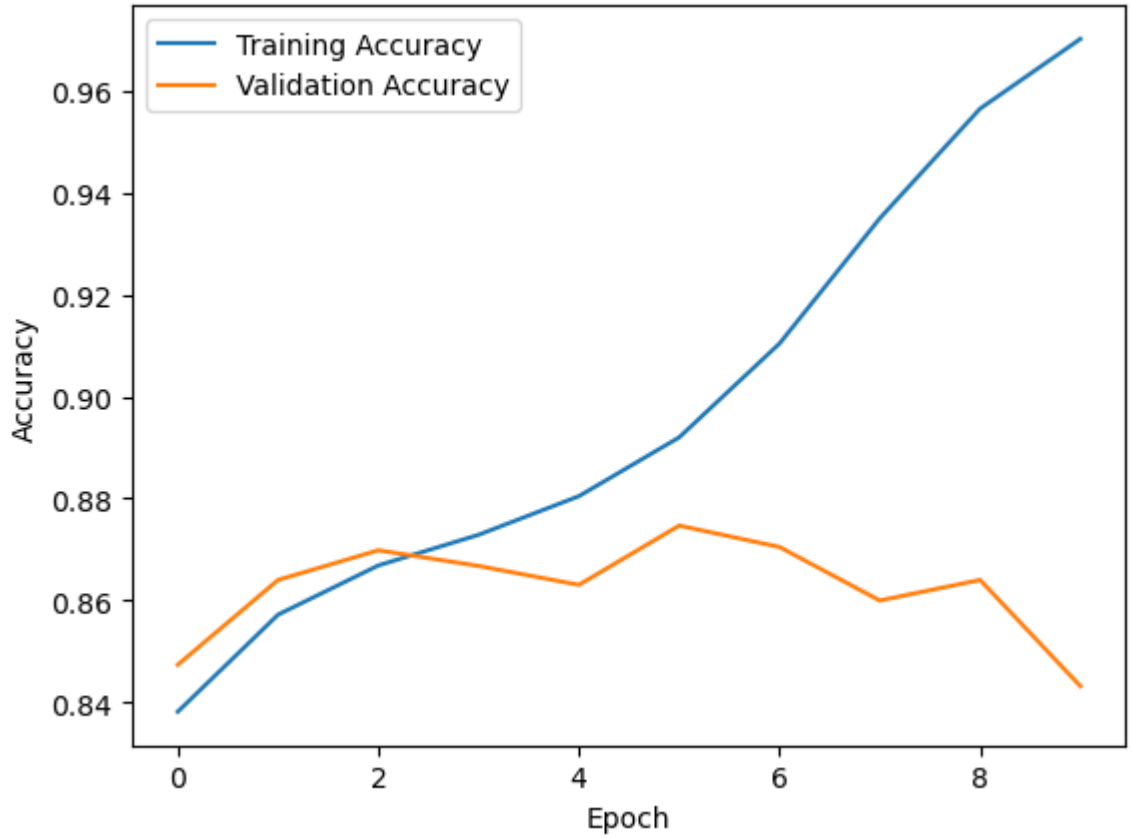
```
Epoch 1/10
1742/1742 [==============================] - 190s 108ms/step - loss: 0.3771 - accuracy: 0.8382 - val_loss: 0.3578
- val_accuracy: 0.8474
Epoch 2/10
1742/1742 [==============================] - 198s 114ms/step - loss: 0.3326 - accuracy: 0.8572 - val_loss: 0.3219
- val_accuracy: 0.8640
Epoch 3/10
1742/1742 [==============================] - 192s 110ms/step - loss: 0.3113 - accuracy: 0.8669 - val_loss: 0.3071
- val_accuracy: 0.8699
Epoch 4/10
1742/1742 [==============================] - 196s 113ms/step - loss: 0.2958 - accuracy: 0.8729 - val_loss: 0.3173
- val_accuracy: 0.8668
Epoch 5/10
1742/1742 [==============================] - 203s 117ms/step - loss: 0.2804 - accuracy: 0.8805 - val_loss: 0.3159
- val_accuracy: 0.8631
Epoch 6/10
1742/1742 [==============================] - 207s 119ms/step - loss: 0.2566 - accuracy: 0.8920 - val_loss: 0.3061
- val_accuracy: 0.8747
Epoch 7/10
1742/1742 [==============================] - 209s 120ms/step - loss: 0.2182 - accuracy: 0.9104 - val_loss: 0.3286
- val_accuracy: 0.8705
Epoch 8/10
1742/1742 [==============================] - 206s 118ms/step - loss: 0.1637 - accuracy: 0.9349 - val_loss: 0.4004
- val_accuracy: 0.8600
Epoch 9/10
1742/1742 [==============================] - 209s 120ms/step - loss: 0.1137 - accuracy: 0.9565 - val_loss: 0.4829
- val_accuracy: 0.8640
Epoch 10/10
1742/1742 [==============================] - 203s 116ms/step - loss: 0.0809 - accuracy: 0.9701 - val_loss: 0.5653
- val_accuracy: 0.8432
```

Plot loss and accuracy over the training run.

In [44]:
```
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 [45]:
```
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 [46]:
```
# 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 [==============================] - 13s 30ms/step - loss: 0.5627 - accuracy: 0.8435
Test Loss: 0.5627
Test Accuracy: 84.35%
```

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