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]:

```python
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]:

```python
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
```

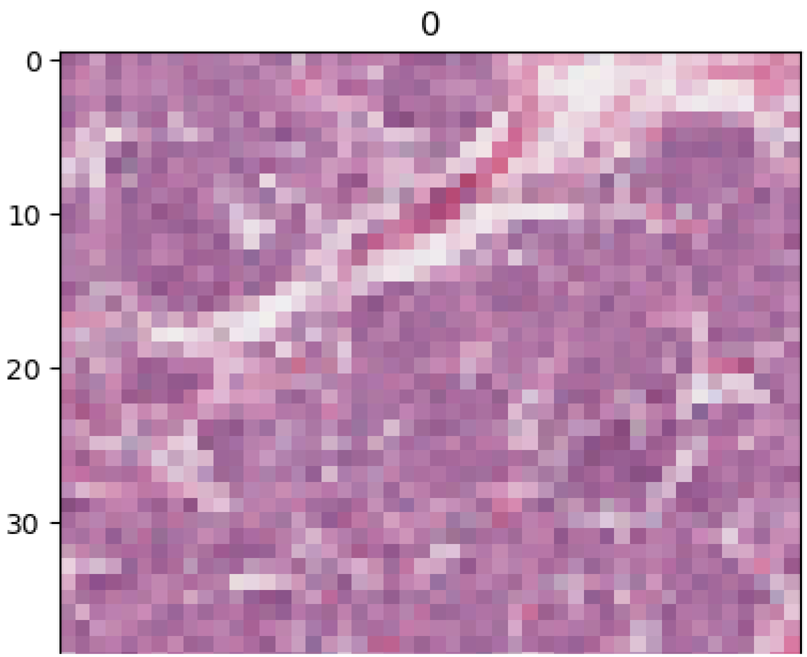
Out[4]:

```
((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 [7]:

```
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()
```

```
WARNING:tensorflow:From C:\Users\Johan\anaconda3\envs\py311\Lib\site-packages\keras\src\backend.py:1398: The name
tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions
instead.

WARNING:tensorflow:From C:\Users\Johan\anaconda3\envs\py311\Lib\site-packages\keras\src\layers\pooling\max_poolin
g2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d instead.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv1 (Conv2D)              (None, 48, 48, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2  (None, 24, 24, 32)        0         
 D)                                                              
                                                                 
 conv2 (Conv2D)              (None, 24, 24, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 12, 12, 64)        0         
 g2D)                                                            
                                                                 
 conv3 (Conv2D)              (None, 12, 12, 128)       73856     
                                                                 
 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 128)         0         
 g2D)                                                            
                                                                 
 conv4 (Conv2D)              (None, 6, 6, 256)         295168    
                                                                 
 max_pooling2d_3 (MaxPoolin  (None, 3, 3, 256)         0         
 g2D)                                                            
                                                                 
 flatten (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 [8]:

```
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 [9]:

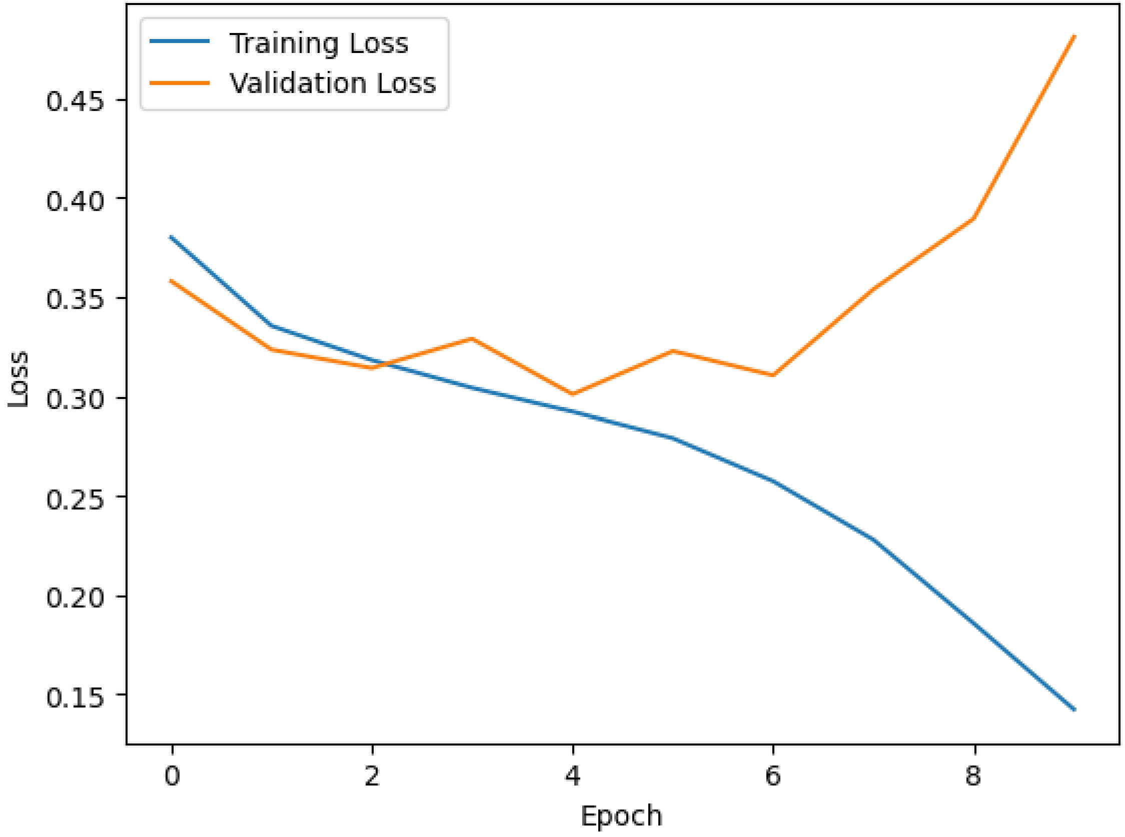
```
batch_size = 32
epochs = 10

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

```
Epoch 1/10
WARNING:tensorflow:From C:\Users\Johan\anaconda3\envs\py311\Lib\site-packages\keras\src\utils\tf_utils.py:492: Th
e name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\Johan\anaconda3\envs\py311\Lib\site-packages\keras\src\engine\base_layer_utils.p
y:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_o
utside_functions instead.

1742/1742 [==============================] - 44s 24ms/step - loss: 0.3801 - accuracy: 0.8357 - val_loss: 0.3581 -
val_accuracy: 0.8510
Epoch 2/10
1742/1742 [==============================] - 42s 24ms/step - loss: 0.3356 - accuracy: 0.8564 - val_loss: 0.3235 -
val_accuracy: 0.8660
Epoch 3/10
1742/1742 [==============================] - 42s 24ms/step - loss: 0.3183 - accuracy: 0.8637 - val_loss: 0.3143 -
val_accuracy: 0.8699
Epoch 4/10
1742/1742 [==============================] - 43s 24ms/step - loss: 0.3043 - accuracy: 0.8715 - val_loss: 0.3290 -
val_accuracy: 0.8629
Epoch 5/10
1742/1742 [==============================] - 56s 32ms/step - loss: 0.2924 - accuracy: 0.8754 - val_loss: 0.3011 -
val_accuracy: 0.8750
Epoch 6/10
1742/1742 [==============================] - 59s 34ms/step - loss: 0.2791 - accuracy: 0.8827 - val_loss: 0.3228 -
val_accuracy: 0.8652
Epoch 7/10
1742/1742 [==============================] - 67s 38ms/step - loss: 0.2575 - accuracy: 0.8926 - val_loss: 0.3106 -
val_accuracy: 0.8715
Epoch 8/10
1742/1742 [==============================] - 62s 35ms/step - loss: 0.2280 - accuracy: 0.9065 - val_loss: 0.3540 -
val_accuracy: 0.8653
Epoch 9/10
1742/1742 [==============================] - 57s 33ms/step - loss: 0.1858 - accuracy: 0.9251 - val_loss: 0.3895 -
val_accuracy: 0.8510
Epoch 10/10
1742/1742 [==============================] - 54s 31ms/step - loss: 0.1425 - accuracy: 0.9451 - val_loss: 0.4812 -
val_accuracy: 0.8595
```

Plot loss and accuracy over the training run.

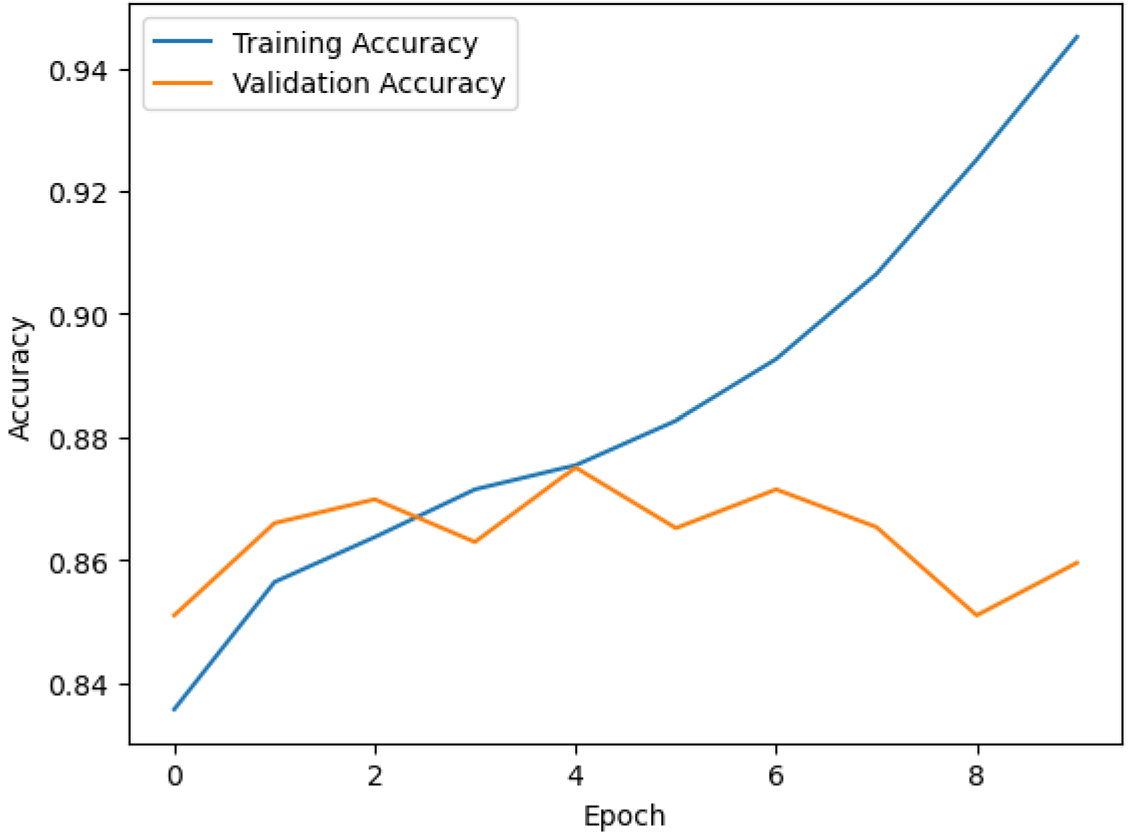
In [10]:

```
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 [11]:

```
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 [12]:

```python
# 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 [==============================] - 4s 9ms/step - loss: 0.4732 - accuracy: 0.8609
Test Loss: 0.4732
Test Accuracy: 86.09%
```

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

```
fiResults:
    - I played with the VGG blueprint a bit.
    - Increasing the conv sizes and the amount of conv runs actually did not improve the outcome at all.
      It actually showed the phenomena of overfitting discussed in class more.
      Because the model got too complex it fit irrelevant  data
    -
```