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.

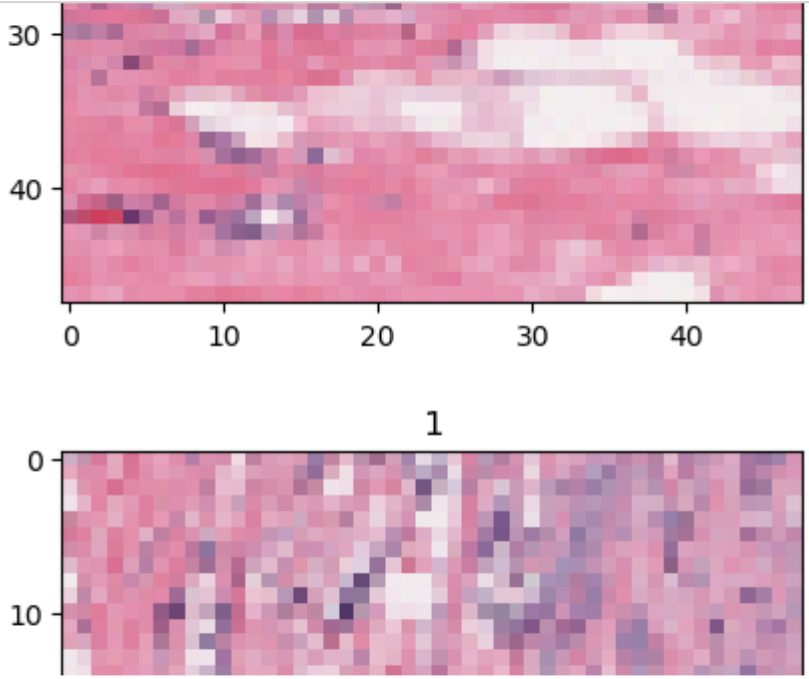
In [3]:
```
import h5py as h5
with h5.File(x_train_path,'r') as f:
  x_train = f['X'][::2,1:49,1:49] # load half the data to avoid out-of-memory errors
  y_train = f['y'][::2]
with h5.File(x_test_path,'r') as f:
  x_test = f['X'][:,1:49,1:49]
  y_test = f['y'][:]
```

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 [53]:

```
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_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1 (Conv2D)              (None, 48, 48, 16)        448

 max_pooling2d_22 (MaxPooli  (None, 24, 24, 16)        0
 ng2D)

 conv2 (Conv2D)              (None, 24, 24, 32)        4640

 max_pooling2d_23 (MaxPooli  (None, 12, 12, 32)        0
 ng2D)

 conv3 (Conv2D)              (None, 12, 12, 32)        9248

 max_pooling2d_24 (MaxPooli  (None, 6, 6, 32)          0
 ng2D)

 conv4 (Conv2D)              (None, 6, 6, 64)          18496

 max_pooling2d_25 (MaxPooli  (None, 3, 3, 64)          0
 ng2D)

 flatten_5 (Flatten)         (None, 576)               0

 dense1 (Dense)              (None, 128)               73856

 z (Dense)                   (None, 2)                 258

=================================================================
Total params: 106946 (417.76 KB)
Trainable params: 106946 (417.76 KB)
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 [54]:

```
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.

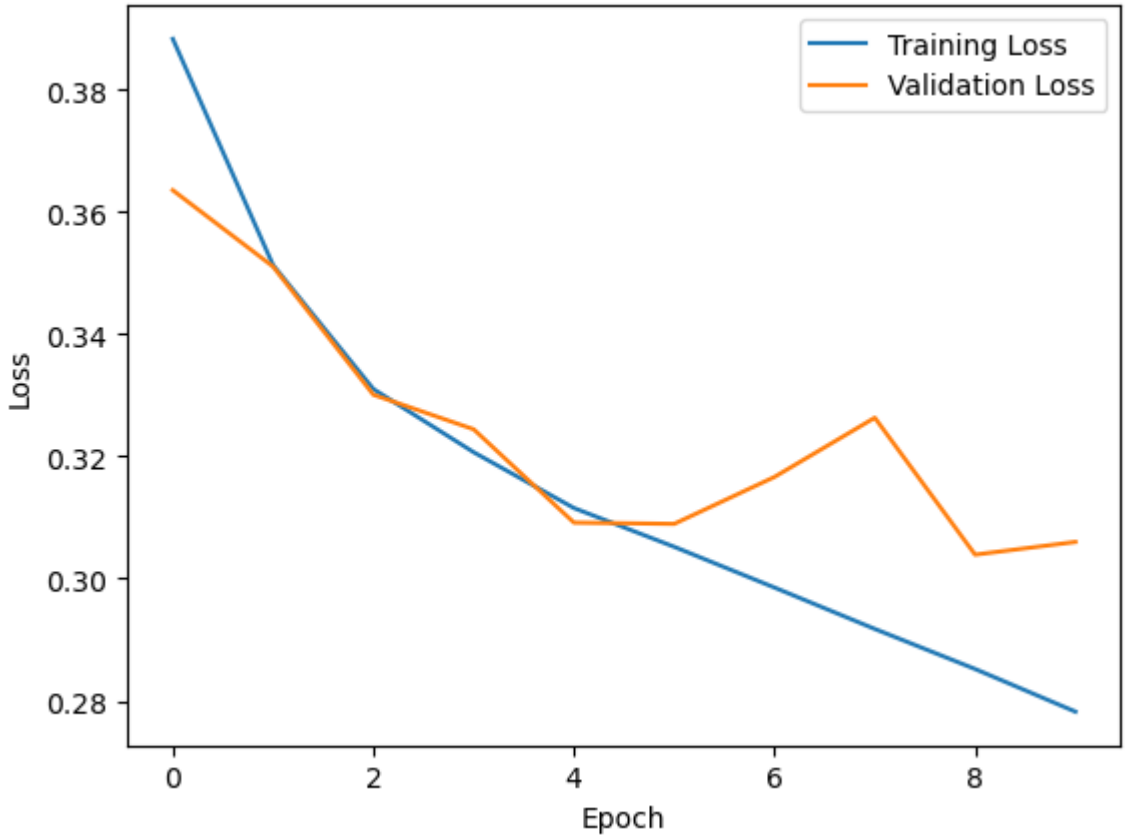
In [55]:
```
batch_size = 32
epochs = 10

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

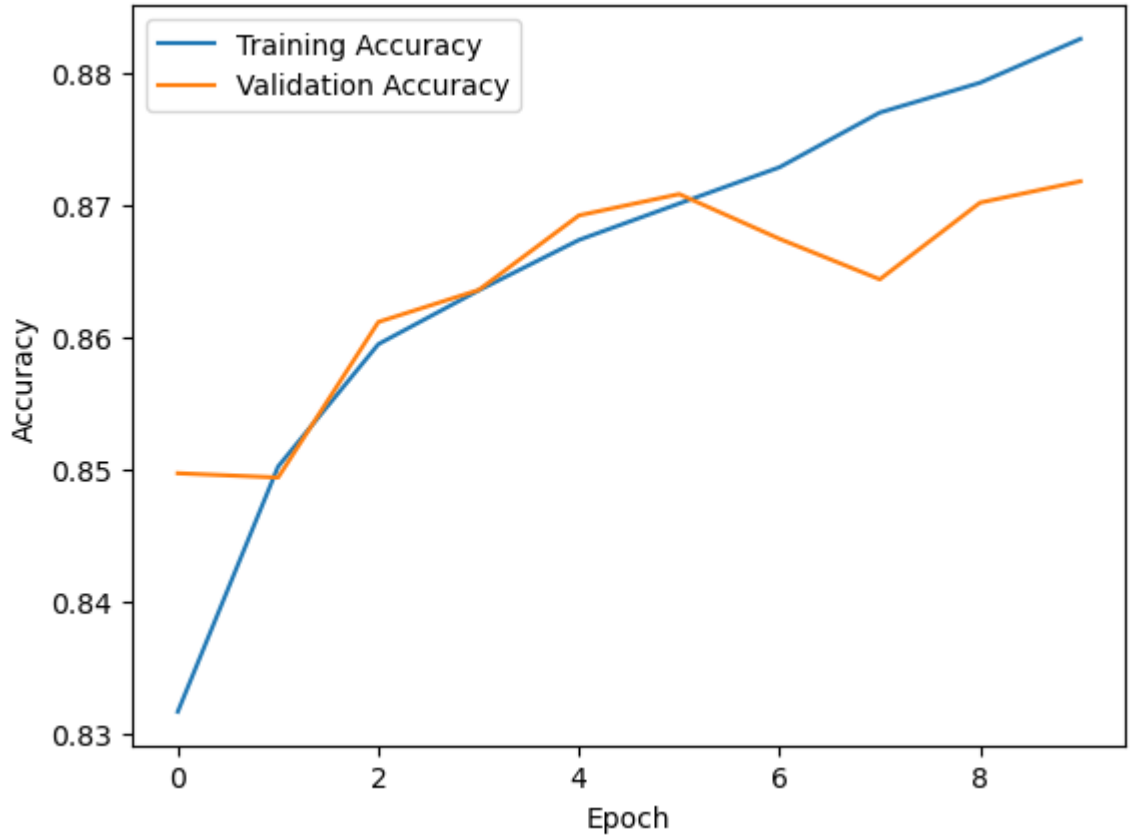
```
Epoch 1/10
1742/1742 [==============================] - 16s 9ms/step - loss: 0.3883 - accuracy: 0.8316 - val_loss: 0.3635 -
val_accuracy: 0.8497
Epoch 2/10
1742/1742 [==============================] - 15s 8ms/step - loss: 0.3513 - accuracy: 0.8502 - val_loss: 0.3510 -
val_accuracy: 0.8493
Epoch 3/10
1742/1742 [==============================] - 24s 14ms/step - loss: 0.3310 - accuracy: 0.8595 - val_loss: 0.3301 -
val_accuracy: 0.8611
Epoch 4/10
1742/1742 [==============================] - 21s 12ms/step - loss: 0.3207 - accuracy: 0.8635 - val_loss: 0.3244 -
val_accuracy: 0.8636
Epoch 5/10
1742/1742 [==============================] - 16s 9ms/step - loss: 0.3115 - accuracy: 0.8673 - val_loss: 0.3092 -
val_accuracy: 0.8692
Epoch 6/10
1742/1742 [==============================] - 14s 8ms/step - loss: 0.3052 - accuracy: 0.8701 - val_loss: 0.3090 -
val_accuracy: 0.8708
Epoch 7/10
1742/1742 [==============================] - 14s 8ms/step - loss: 0.2985 - accuracy: 0.8729 - val_loss: 0.3166 -
val_accuracy: 0.8674
Epoch 8/10
1742/1742 [==============================] - 15s 9ms/step - loss: 0.2918 - accuracy: 0.8770 - val_loss: 0.3263 -
val_accuracy: 0.8644
Epoch 9/10
1742/1742 [==============================] - 14s 8ms/step - loss: 0.2852 - accuracy: 0.8793 - val_loss: 0.3039 -
val_accuracy: 0.8702
Epoch 10/10
1742/1742 [==============================] - 14s 8ms/step - loss: 0.2782 - accuracy: 0.8826 - val_loss: 0.3060 -
val_accuracy: 0.8718
```

Plot loss and accuracy over the training run.

In [56]:
```
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 [57]:
```
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 [58]:
```
# 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 [==============================] - 2s 4ms/step - loss: 0.3120 - accuracy: 0.8645
Test Loss: 0.3120
Test Accuracy: 86.45%
```

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