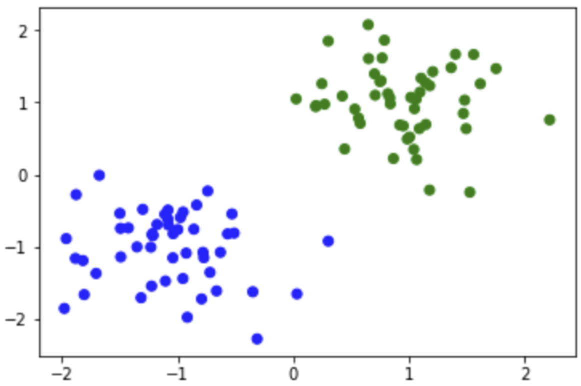
Neural Networks - Advanced Image Analysis

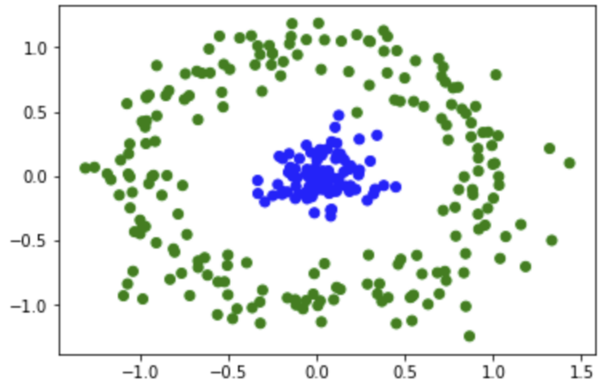
Jacob Jon Hansen (s134097) Phillip Brinck Vetter (s144097)

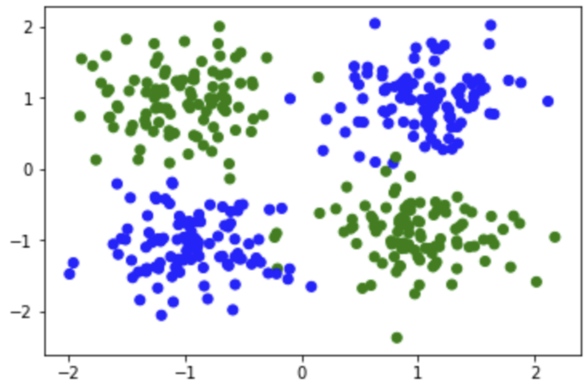
Exercise 1

Using python and sklearn we construct our 3 training sets (using our own written functions:

```
#Load data and plot the three cases
plt.figure()
X, y = gen_data("2blop")
plot_data(X,y)

plt.figure()
X, y = gen_data("blop_circle")
plot_data(X,y)

plt.figure()
X, y = gen_data("4blop")
plot_data(X,y)

plt.show()
```







The following code are our high level hyper parameters for all the three networks:

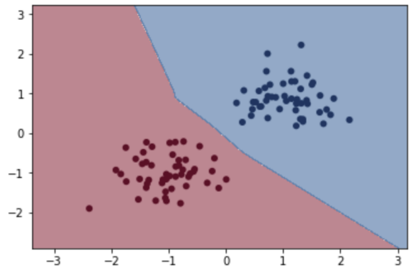
```
#Parameters for NN
num_classes = 2
opt = "adam"
loss_f = 'categorical_crossentropy'
batch_size = 64
nb_epochs = 100
```

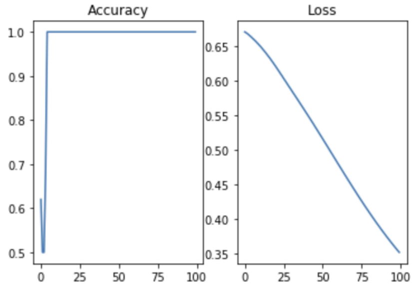
Training is run with following code:

```
#Load dataset and fit
X, y = gen_data("2blop")
y_cat = np_utils.to_categorical(y, num_classes)

model = Sequential()
model.add(Dense(3, activation="relu", input_dim=2))
model.add(Dense(5, activation="relu"))
model.add(Dense(2, activation="softmax"))
model.summary()

#Compile the model
model.compile(optimizer = opt, loss = loss_f, metrics=["accuracy"])


hist = model.fit(X, y_cat, batch_size = batch_size,
                 epochs=nb_epochs, verbose=0)

#Plot decision boundary
plot_decision_boundary(X, y, model, cmap='RdBu')

#Plot how the training went
plt.figure()
plot_training(hist)

plt.show()
```

2 blops

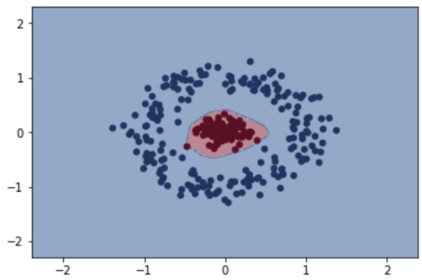
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_60 (Dense)             (None, 3)                 9         
_________________________________________________________________
dense_61 (Dense)             (None, 5)                 20        
_________________________________________________________________
dense_62 (Dense)             (None, 2)                 12        
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

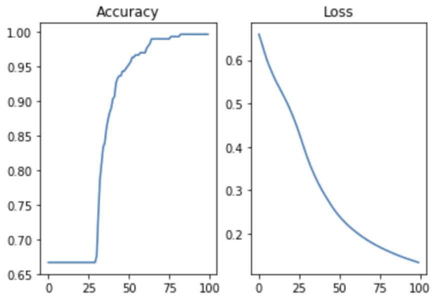




Blop and circle

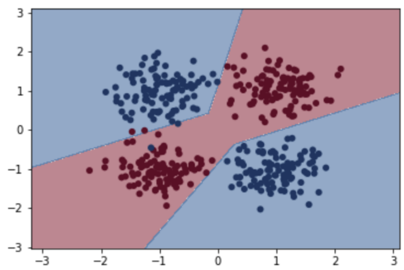
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_63 (Dense)             (None, 5)                 15
_________________________________________________________________
dense_64 (Dense)             (None, 15)                90
_________________________________________________________________
dense_65 (Dense)             (None, 10)                160
_________________________________________________________________
dense_66 (Dense)             (None, 2)                 22
=================================================================
Total params: 287
Trainable params: 287
Non-trainable params: 0
_________________________________________________________________
```

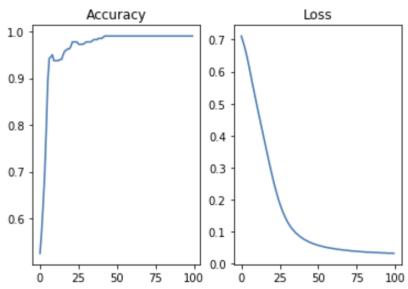




4 blops

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_67 (Dense)             (None, 5)                 15
_________________________________________________________________
dense_68 (Dense)             (None, 15)                90
_________________________________________________________________
dense_69 (Dense)             (None, 10)                160
_________________________________________________________________
dense_70 (Dense)             (None, 2)                 22
=================================================================
Total params: 287
Trainable params: 287
Non-trainable params: 0
_________________________________________________________________
```





Exercise 2: MNIST

For this we used Keras since we have extensive knowledge of DL and therefore it seemed redundant to sit and program everything from scratch. This is thus not a "valid" competition attempt, but it shows the power of leveraging a good framework to achieve an accuracy of **99.4%**

In order to have a network which generalizes better we use a datagenerator which takes the images from the training set and arguments them with random rotations, zoom and translations:

```
datagen = ImageDataGenerator(rotation_range = 20,
                             width_shift_range = 0.1,
                             height_shift_range = 0.1,
                             zoom_range=0.2)
datagen.fit(X_train)
```

The following model is defined:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
batch_normalization_7 (Batch (None, 26, 26, 32)        128
_________________________________________________________________
activation_7 (Activation)    (None, 26, 26, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 24, 24, 32)        9248
_________________________________________________________________
batch_normalization_8 (Batch (None, 24, 24, 32)        128
_________________________________________________________________
activation_8 (Activation)    (None, 24, 24, 32)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
batch_normalization_9 (Batch (None, 10, 10, 64)        256
_________________________________________________________________
activation_9 (Activation)    (None, 10, 10, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 8, 8, 64)          36928
_________________________________________________________________
batch_normalization_10 (Batc (None, 8, 8, 64)          256
_________________________________________________________________
activation_10 (Activation)   (None, 8, 8, 64)          0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_4 (Dense)              (None, 512)               524800
_________________________________________________________________
batch_normalization_11 (Batc (None, 512)               2048
_________________________________________________________________
activation_11 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_12 (Batc (None, 256)               1024
_________________________________________________________________
activation_12 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2570
=================================================================
Total params: 727,530
Trainable params: 725,610
Non-trainable params: 1,920
```

It uses and CNN, with tricks such as batch normalization (improves training speed) and dropout to increase the ability to generate to new data.

The network is trained with the following parameters:

```
#Parameters
num_classes = 10
opt = "adam"
loss_f = 'categorical_crossentropy'
batch_size = 64
nb_epochs = 15
```

Training is performed on a big server with a 8 GB GPU, each epoch took 12 seconds (400 on my laptop running a gtx 1050). During training the best model (highest acc on test set) is saved so ensure that we will have the optimal model even if it starts to overfit the training set. This can be done with the following code:

```
#Save best model under training
from keras.callbacks import ModelCheckpoint
filepath="weights_CNN.best.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc',
                             verbose=1, save_best_only=True, mode='max')
callbacks_list = [checkpoint]


#Fit the model
model.fit_generator(train_generator,
                    steps_per_epoch = len(X_train) / (2*batch_size),
                    epochs = nb_epochs,
                    validation_data = test_generator,
                    validation_steps = len(y_test)/ batch_size,
                    callbacks = callbacks_list)
```

After training we load the best weights and test:

```
#Evaluate performance of model
model.load_weights("weights_CNN.best.hdf5")
#Compile the model
model.compile(loss=loss_f,
              optimizer=opt,
              metrics=['accuracy'])

score = model.evaluate(X_test, y_test, verbose=0)
print(score[1])
```

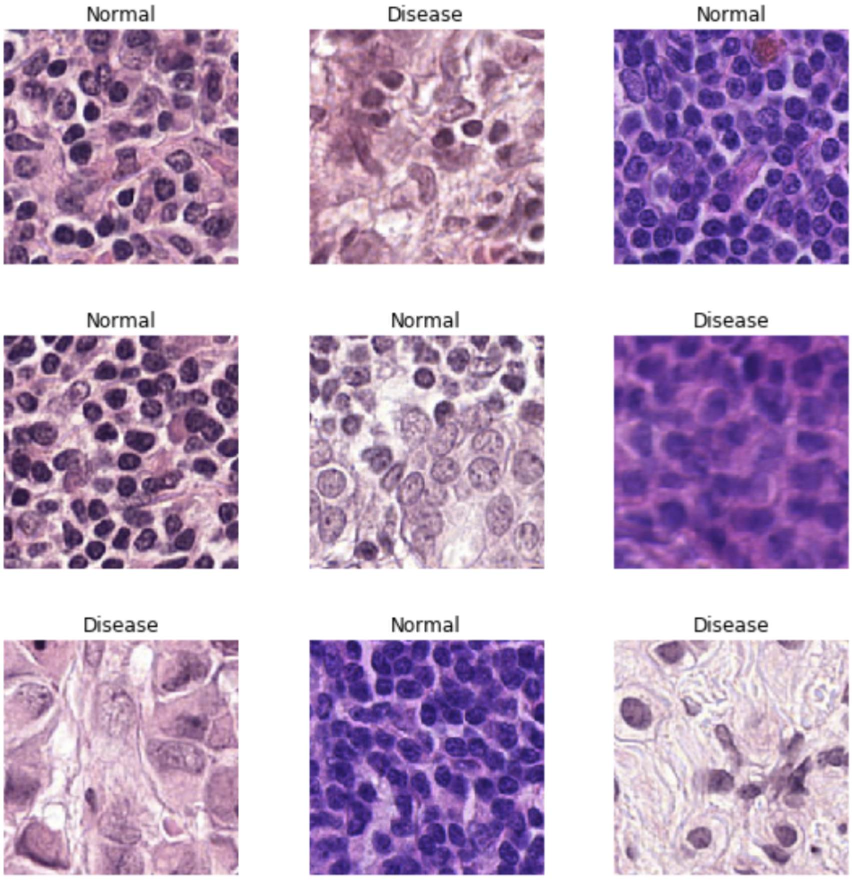
```
0.9941
```

Exercise 3: Using VGG19 on breast cancer images.

VGG19 is a powerful architecture that can be downloaded with pretrained weights. The layers in the network are training on the ImageNet challenge, so they will produce a lot of image features. The early layers will give simple features such as lines, circles etc. The last layers will give more abstract features such as eyes, fur, ears etc used for the imagenet dataset. We have used the first fully connected layer at the end and used this to generate features for the images in the breast cancer dataset. A KNN is fitted to these features and we see a performance of **88%** with 11 neighbours. Another method would be to do retraining.

This can be done by freezing all the CNN layers and defining our own output layer and training this with our data to do transfer learning.

Dataset:



Model:

Code to load the model (note that we say include_top=True, this ensure that the model is also using the last FCC layers. We can define a new model from this base model, specifying which layer should be our output.

```
from keras import applications
from keras.models import Sequential, Model

img_width, img_height = (X.shape[0], X.shape[1])

#Define model
model = applications.VGG19(weights = "imagenet", include_top=True,
                           input_shape = (img_width, img_height, 3))
model = Model(inputs=model.input, outputs=model.get_layer('fc1').output)
```

The model has the following architecture, notice that we have 122M parameters so it would take forever to train ourselves:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
=================================================================
Total params: 122,788,928
Trainable params: 122,788,928
Non-trainable params: 0
_________________________________________________________________
```

Now we just predict the features, combine these into a big array and feed them into a KNN.

```
n = X.shape[3]
features_all = np.zeros((n, features.flatten().shape[0]))

#Generate features
for i in range(n):
    x = image.img_to_array(X[:,:,:,i])
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)

    features = model.predict(x)
    features_all[i,:] = features.flatten()

from sklearn import neighbors, datasets
from sklearn.metrics import accuracy_score

from sklearn.preprocessing import normalize
normed = normalize(features_all)

knn = neighbors.KNeighborsClassifier(n_neighbors = 11)


knn.fit(normed, y.ravel()-1)
```

```
pred = knn.predict(features_all)
print(accuracy_score(y.ravel()-1, pred)) #-> 0.880
```