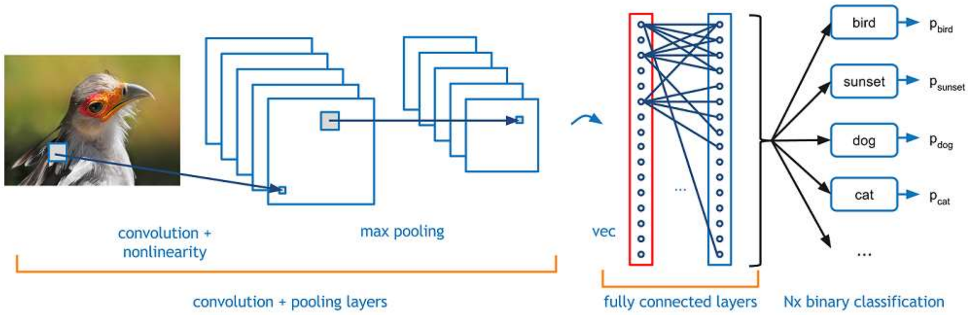
Type *Markdown* and LaTeX: α^2

CNN for lesion classification

1. Brief CNN theory

A Convolutional Neural Network (CNN, or ConvNet) is a type of **feed-forward** artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.



> source: https://flickrcode.files.wordpress.com/2014/10/conv-net2.png (https://flickrcode.files.wordpress.com/2014/10/conv-net2.png)

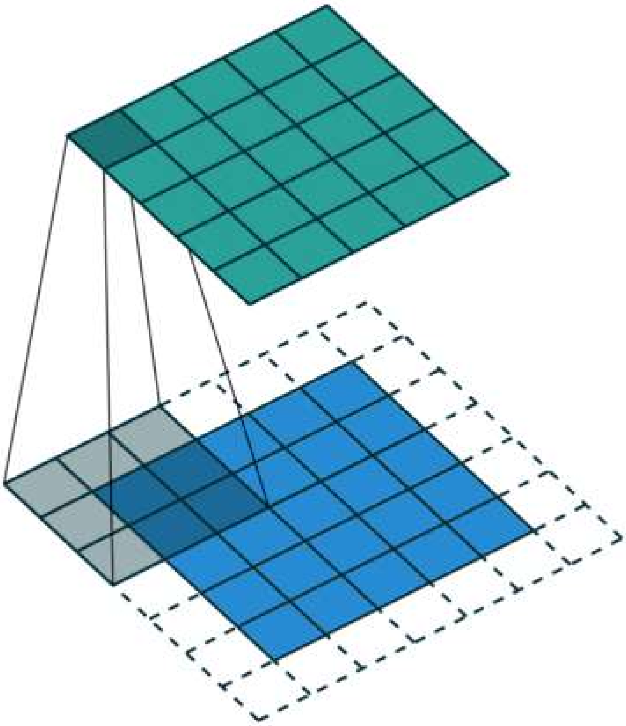
1.1 Structure of a CNN

> A more detailed overview of what CNNs do would be that you take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output. As we said earlier, the output can be a single class or a probability of classes that best describes the image.

source: [1]

Convolutional Layer

The first layer in a CNN is always a **Convolutional Layer**.



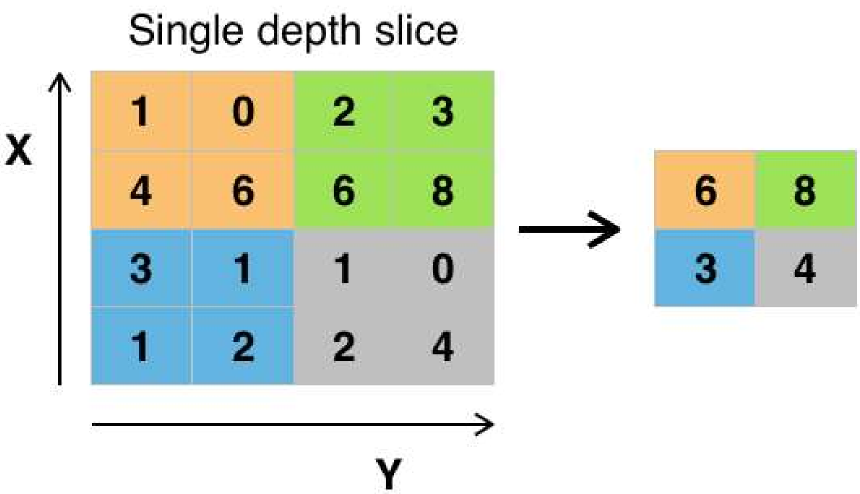
Typical CNN Structure

A traditional CNN architecture consists of other layers interspaced between convolution layers

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Pooling layer

After some ReLu layers, **pooling layer** is typically applied.



Example of Maxpool with a 2x2 filter and a stride of 2

Pooling reduces the amount of parameters (helping with computional efficiency) and controls overfitting

2. We build one using keras and tensorflow

2.1 Preparation

In [1]:

```
# setup code for this notebook
import numpy as np
import matplotlib.pyplot as plt
from functions import data, Timer
timer = Timer()

# This makes matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# Make the notebook reload external python modules;
# see http://stackoverflow.com/questions/1907993/autoreload-of-modules-in-ipython
%load_ext autoreload
%autoreload 2
```

The ultra sound scan data

- 163 scans total, clinically confirmed as having either bening or malignant (cancerous) lesions
- 100 scans for training, 63 for testing
- Training data was passed through 7 transformations to give us 800 training images total
- We balanced the training data to have half malignant and half benign
- Since the malignant cases were less than benign cases, we use only 528 images for training
- Emperically we have found this improves our overall performance
- Testing images not transformed
- Both training and testing images were resized to 224X224
- Raw pngs then converted to numpy arrays and saved

In [2]:

```
# Import keras libraries
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.pooling import MaxPooling2D
from keras import backend as K
```

```
Using TensorFlow backend.
```

In [3]:

```
img_rows, img_cols = 224, 224 # 224, 224 resized down from 360, 528
color_channels = 3

if K.image_data_format() == 'channels_first':
    input_shape = (color_channels, img_rows, img_cols)
else:
    input_shape = (img_rows, img_cols, color_channels)

print('Input shape', input_shape)
```

```
Input shape (224, 224, 3)
```

2.2 A CNN

In [4]:

```
# values for the convnet

# number of convolutional filters to use
filters = 32
# size of pooling area
pooling_area = 2
# conv kernel size
conv_kernel = 3
```

In [5]:

```
# We define the cnn model
def buildModelStructure():
    model = Sequential()

    model.add(Conv2D(filters, (conv_kernel, conv_kernel), padding='valid',
                     input_shape=input_shape))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(pooling_area, pooling_area)))

    model.add(Conv2D(filters, (conv_kernel, conv_kernel)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(pooling_area, pooling_area)))

    model.add(Conv2D(64, (conv_kernel, conv_kernel)))
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(pooling_area, pooling_area)))

    model.add(Flatten())
    model.add(Dense(64))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1))
    model.add(Activation('sigmoid'))

    return model
```

In [6]:

```
# Visualize
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

def visualize(model):
    model.summary()
    SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

In [7]:

```
# generator helpers
from keras.preprocessing.image import ImageDataGenerator
```

In [8]:

```
# data readers
base = "J:\\final year project\\code and models\\data\\augmented\\"
train_directory = base+'training'
validation_directory = base+'validation'

batch_size = 32

# normalization
train_generator = ImageDataGenerator(rescale=1./255)
validation_generator = ImageDataGenerator(rescale=1./255)

# this is a generator that will read scans found in
# the train directory, and indefinitely generate
# batches of image data
train_generator = train_generator.flow_from_directory(
        train_directory,
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='binary')

# A  similar generator, for validation data
validation_generator = validation_generator.flow_from_directory(
        validation_directory,
        target_size=(img_rows, img_cols),
        batch_size=batch_size,
        class_mode='binary')
```

```
Found 400 images belonging to 2 classes.
Found 128 images belonging to 2 classes.
```

In [9]:

```
def plot(network_history):
    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(network_history.history['loss'])
    plt.plot(network_history.history['val_loss'])
    plt.legend(['Training', 'Validation'])

    plt.figure()
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.plot(network_history.history['acc'])
    plt.plot(network_history.history['val_acc'])
    plt.legend(['Training', 'Validation'], loc='lower right')
```

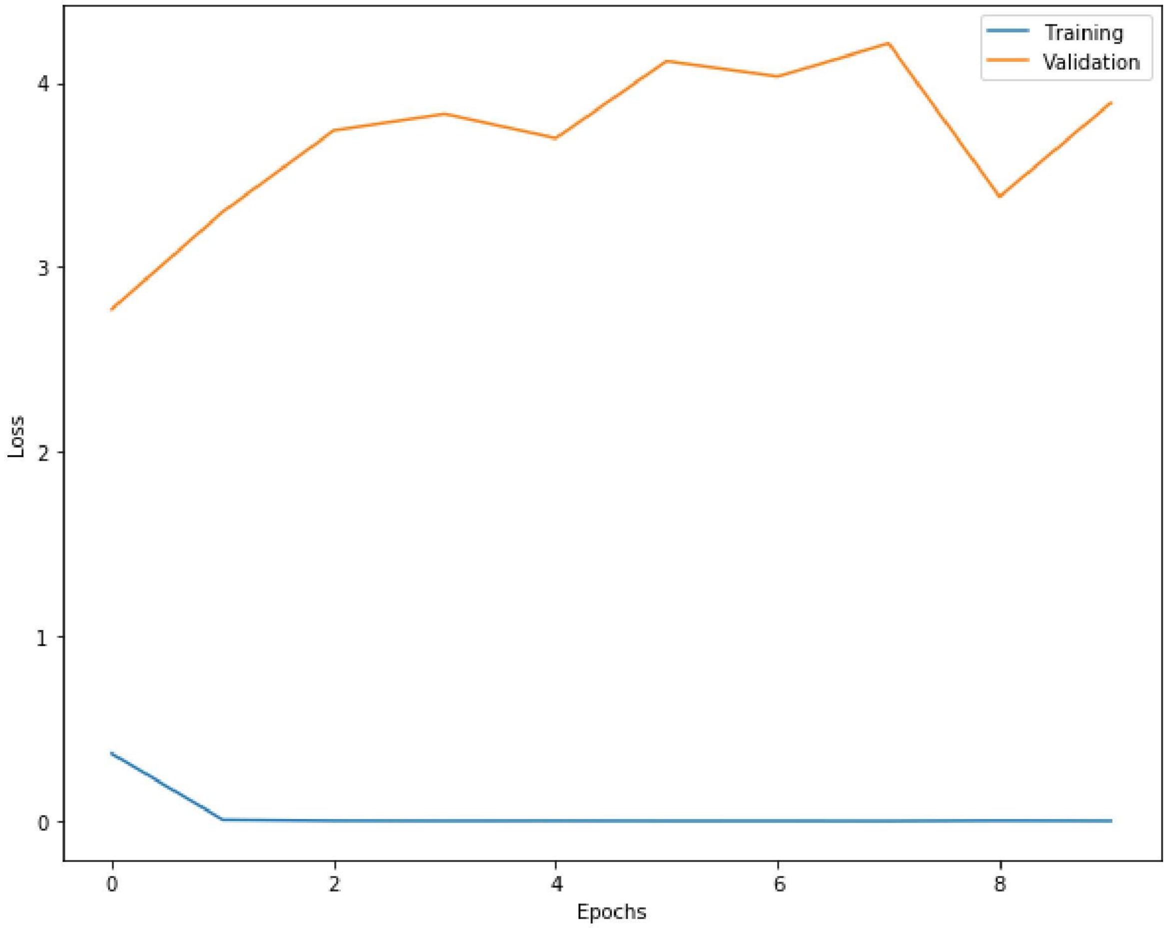
In [10]:

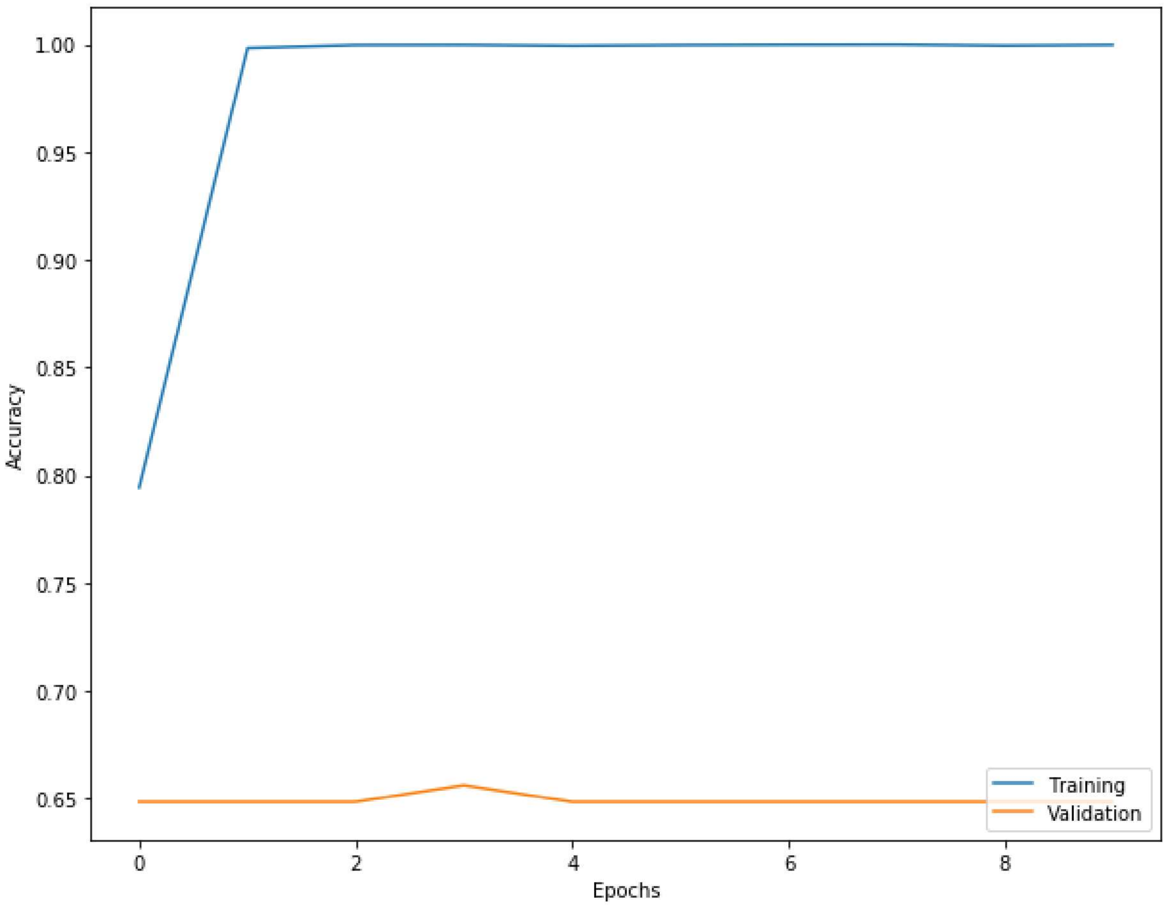
```
# helpers for checkpointing and early stopping
from keras.callbacks import ModelCheckpoint # , EarlyStopping

best_model_file = '4.2.h5'
# early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
best_model = ModelCheckpoint(best_model_file, verbose=True, save_best_only=True)

def testModel(optimizer='rmsprop', epochs=2, text="Training model"):
    # build model
    model = buildModelStructure()

    # Compile it
    model.compile(loss='binary_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
    timer.start()
    network_history = model.fit_generator(
            train_generator,
            steps_per_epoch=400,
            epochs=epochs,
            validation_data = validation_generator,
            validation_steps=128,
            verbose=True,
            callbacks=[best_model])
    timer.stop(text)
    plot(network_history)
```

In [11]:

```
#Adagrad
epochs = 10
testModel('adagrad', epochs, "training model with adagrad")
```

```
Epoch 1/10
400/400 [==============================] - 178s 446ms/step - loss: 0.3648 -
acc: 0.7941 - val_loss: 2.7733 - val_acc: 0.6484

Epoch 00001: val_loss improved from inf to 2.77332, saving model to 4.2.h5
Epoch 2/10
400/400 [==============================] - 149s 372ms/step - loss: 0.0069 -
acc: 0.9984 - val_loss: 3.3006 - val_acc: 0.6484

Epoch 00002: val_loss did not improve
Epoch 3/10
400/400 [==============================] - 148s 370ms/step - loss: 0.0017 -
acc: 0.9998 - val_loss: 3.7406 - val_acc: 0.6484

Epoch 00003: val_loss did not improve
Epoch 4/10
400/400 [==============================] - 149s 372ms/step - loss: 0.0011 -
acc: 0.9997 - val_loss: 3.8316 - val_acc: 0.6562

Epoch 00004: val_loss did not improve
Epoch 5/10
400/400 [==============================] - 150s 374ms/step - loss: 0.0014 -
acc: 0.9994 - val_loss: 3.6980 - val_acc: 0.6484

Epoch 00005: val_loss did not improve
Epoch 6/10
400/400 [==============================] - 150s 375ms/step - loss: 6.9086e-0
4 - acc: 0.9998 - val_loss: 4.1173 - val_acc: 0.6484

Epoch 00006: val_loss did not improve
Epoch 7/10
400/400 [==============================] - 153s 383ms/step - loss: 6.3775e-0
4 - acc: 0.9999 - val_loss: 4.0325 - val_acc: 0.6484

Epoch 00007: val_loss did not improve
Epoch 8/10
400/400 [==============================] - 148s 370ms/step - loss: 3.1524e-0
4 - acc: 1.0000 - val_loss: 4.2125 - val_acc: 0.6484

Epoch 00008: val_loss did not improve
Epoch 9/10
400/400 [==============================] - 147s 368ms/step - loss: 0.0023 -
acc: 0.9995 - val_loss: 3.3825 - val_acc: 0.6484

Epoch 00009: val_loss did not improve
Epoch 10/10
400/400 [==============================] - 150s 374ms/step - loss: 9.5867e-0
4 - acc: 0.9998 - val_loss: 3.8880 - val_acc: 0.6484

Epoch 00010: val_loss did not improve
Timing:: took 25 minutes training model with adagrad
-----------------------------------
```





2.3 Evaluating the CNNs performance

In [1]:

```
# Get test data
from functions import data
x_test, y_test = data.getTestData()
print('Test data shape: ', x_test.shape)
print('Test labels shape: ', y_test.shape)
```

```
Test data shape:  (63, 224, 224, 3)
Test labels shape:  (63, 1)
```

In [2]:

```
X_test = x_test/255
```

In [3]:

```
# Load and evaluate best model
from keras.models import load_model
best_model = load_model('4.2.h5')
best_model.summary()
```

```
Using TensorFlow backend.

_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 32)      896       
_________________________________________________________________
activation_1 (Activation)    (None, 222, 222, 32)      0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 111, 111, 32)      0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 109, 109, 32)      9248      
_________________________________________________________________
activation_2 (Activation)    (None, 109, 109, 32)      0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 54, 54, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 52, 52, 64)        18496     
_________________________________________________________________
activation_3 (Activation)    (None, 52, 52, 64)        0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 26, 26, 64)        0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 43264)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2768960   
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65        
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0         
=================================================================
Total params: 2,797,665
Trainable params: 2,797,665
Non-trainable params: 0
_________________________________________________________________
```

In [4]:

```
best_model.predict_classes(x_test)
```

Out[4]:

```
array([[1],
       [1],
       [1],
       [0],
       [1],
       [0],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [1],
       [1],
       [1],
       [0],
       [1],
       [0],
       [0],
       [1],
       [1],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [1],
       [1],
       [1],
       [1],
       [1],
       [0],
       [1],
       [0],
       [1],
       [1],
       [1],
       [1],
```

```
        [1],
        [1],
        [1],
        [1],
        [1],
        [0],
        [1],
        [1]])
```

In [9]:

```
def printMetrics(fn, tp):
    sensitivity = tp/(tp+fn)

    print("True Positive Rate (TPR) or Hit Rate or Recall or Sensitivity: ", sensitivity)
    print('\n')
```

tn is the true negative. These are scans classified correctly as not having malignant lesions.

fn is the false negative. These are scans classified incorrectly as having malignant lesions yet they are benign.

fn is the false negatives. These are images predicted as not having malignant lesions yet they do. This should be low.

tp is the true positive. These are correctly predicted to have malignant lesions.

Sensitivity is a function of false negatives and true positives two variables. Because we are dealing with cancer, false negatives should be very low and true positives high. Low number of false negatives and a high number of true positives gives a high sensitivity. A high sensitivity is desired in a good model

In [10]:

```
# Evaluate all
from sklearn.metrics import confusion_matrix
expected = y_test
prediction = best_model.predict_classes(x_test)
tn,fp,fn,tp = confusion_matrix(y_test, prediction).ravel()
print("false negatives: ", fn)
print("true positives: ", tp)
print("true negatives: ", tn)
print("false positives: ", fp)
printMetrics(fn, tp)
```

```
false negatives:  3
true positives:  18
true negatives:  17
false positives:  25
True Positive Rate (TPR) or Hit Rate or Recall or Sensitivity:  0.8571428571
43
```

```

```

This is our best performing network. Capable of recognising lesions with 0.85 sensitivty or 85% sensitivity.

References for images and some content:

[1] https://adeshpande3.github.io/adeshpande3.github.io/ ()
[2] "Neural Networks and Deep Learning" (http://neuralnetworksanddeeplearning.com/) by Michael Nielsen.
[3] Deep learning with TensorFlow and Keras by Valerio Maggio

In [ ]:

```

```