Capstone Project

Ewa Nowacka
Machine Learning Engineer
July 8th, 2018.

I. Definition

1.1 Project Overview

According to the information found in [2] (www.cancer.net) , lung cancer is the second most common cancer and the leading cause of cancer death for men and women. Lung cancer makes up 14% of all new cancer diagnoses and accounts for 1 in 4 cancer deaths. The American Cancer Society (see [1] for details) estimates that in 2018 there will be about 234,030 new cases of lung cancer (121,680 in men and 112,350 in women).The deaths from lung cancer in 2018 are expected to be about 154,050: 83,550 in men and 70,550 in women.

Lung cancer detected early can be successfully treated. However, usually the patients do not experience symptoms of the cancer unless they are already at an advanced, non-curable stage. These symptoms (cough, chest pain, wheezing) are often mistaken for respiratory infections or effects of long term smoking,delaying the correct diagnosis. Screening people with high risk of lung cancer, e.g. long-time smokers, people exposed to asbestos, mine, mill, textile.and plant workers, has been the most successful method in lung cancer detection. As per [3] (*Can Lung Cancer Be Found Early?*, American Cancer Society), chest x-rays aren't as effective as CT scans in detecting early signs of lung cancer. Using CT scans for lung cancer screening, has a much higher rate of successfully diagnosed lung cancer.

1.2 Project Statement

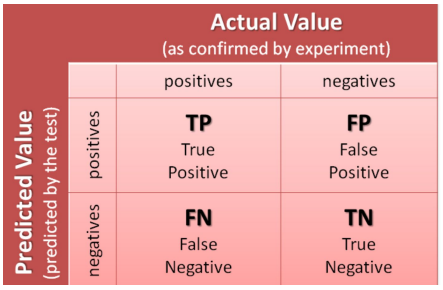
This project intends to design a model that will perform a classification of the potential cancer nodules found in the CT scans of two groups: negative and positive for cancer. Convolutional neural network framework has been chosen for this purpose. This selection is dictated by the research conducted in this field: deep neural network for skin cancer detection ([4]), deep neural network for breast cancer detection ([5]) and lung cancer detection with CT scans ([6]). The CNN performance will be benchmarked against SVM classifier which will be trained as a part of this project.

The project is inspired by research *Using a CNN to predict the presence of lung cancer* conducted by Adam Pollack, Chainatee Tanakulrungson and Nate Kaiser (see [10] for details) which defines a CNN classifier for the same dataset: lung CT scan data.

1.3 Metrics

1.3.1 Model Performance Metrics

Three evaluation metrics will be used to measure the model performance: accuracy, sensitivity and specificity. They are calculated basing on the model confusion matrix which shows a summary of how the model performs the classification task.



(from https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/60900/versions/13/screenshot.png)

Accuracy is defined as a percentage of the correctly classified images out of the total number of the images.

$$Accuracy = \frac{true\ negatives + true\ positives}{total}$$

Sensitivity and specificity are measures used mostly in medical applications. In this project, sensitivity will inform what percentage of the positive lung cancer was correctly classified as positive (true positive rate). On the other hand, specificity will provide a true negative rate - what percentage of the negative lung cancer was correctly classified as negative.

$$Sensitivity = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Specificity = \frac{True\ Negative}{True\ Negative + False\ Negative}$$

Since sensitivity and specificity formulas does not contain FP and TN/FN and TP (respectively), they may give biased results, especially for imbalanced classes. Therefore it is important to 1) make sure that all the samples are properly balanced and 2) use accuracy metrics which calculates correctly classified images over total of images and therefore is less prone to bias.

Ideally, we would like to have all these values as close to 100% as possible. Therefore, for benchmarking purposes, the better model will be the one with higher values of these metrics. The above metrics were calculated in the project using *confusion_matrix* function from *sklearn_metrics.*

1.3.2 Model Benchmark

As mentioned in Section 1, the project is inspired by research presented in [10], therefore I will use this CNN architecture as a benchmark mode. The obtained accuracy was 96.38%. Therefore, the main objective of this project will be to beat this result.

Another benchmark model will be a binary classifier based on the Support Vector Machine approach. A simple SVM classifier will be created as a part of this project. To decide which model performs better, accuracy, sensitivity and specificity measures, as described in Section 2.1. will be used.

II. Analysis

2.1 Data Exploration

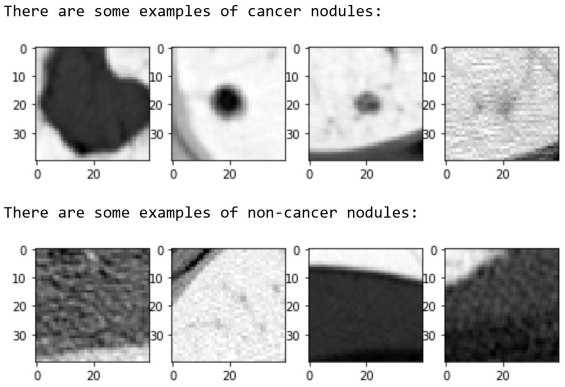
The dataset used for this project consists of 2948 nodule snippets extracted from LUNA16 dataset found on GitHub repository ([9]). Original LUNA16 can be found in [10] but it is available only upon participation in the challenge. LUNA16 is a collection of lung CT scans of 888 patients. They are originally in MetaImage format (mhd/raw) and they have size of 512x512 pixels. The snippets contain only potential cancer nodules and have size of 40x40 pixels, which makes them much more convenient for the modelling purposes. Apart from the CT scans, LUNA16 dataset contains also nodule coordinates used to extract these snippets, however this process is out of the scope of this project.

Each snippet file is a PNG file and has the following naming convention: seriesid (provided in the csv file with the nodules' coordinates), nodule id (nod0, nod1,etc.), slice id (slc1, slc2 , etc.) and label (pos for positive and neg for negative). The example of the file name is provided below:
1.3.6.1.4.1.14519.5.2.1.6279.6001.100621383016233746780170740405nod0sl c1pos.png.

No data abnormalities were observed in the analyzed dataset, however, it is very difficult to say whether there any mislabeled images. It is difficult even for a doctor to correctly label all the images and I have no way of verifying it. There always exists a risk that some of the images may not be labeled correctly.

2.2 Exploratory Visualization

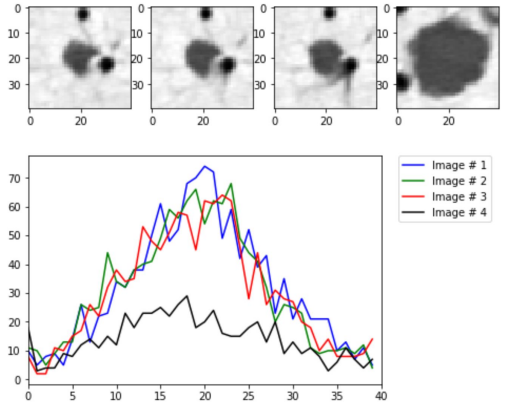
First step was to check if the dataset is balanced and is representative of both cases: positive and negative for cancer. A simple count showed that there are 1474 cancer nodules and 1474 non-cancer nodules, therefore the dataset can be deemed balanced (50%-50%) and we should maintain the same ratio of positive to negative examples for our training sample.



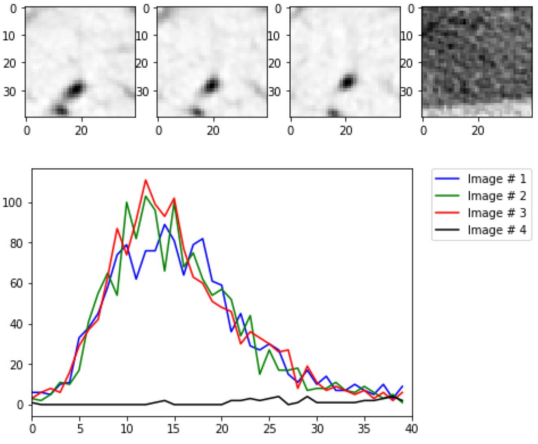
Dataset: example images

As mentioned in section 2.1, cancer classification is a very difficult task even for a medical professional and I have no way of telling whether there are any mislabeled pictures in the dataset. As it can be seen in the example images above, it is very hard to spot any pattern. This is also a reason why CNN is applied in medicine as it can find pattern than human is not able to.

I have also plot some intensity histograms for the example images shown above.



Cancer nodules: intensity histogram



Cancer nodules: intensity histogram

Intensity histograms represent the pixel intensity values. This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. As the pictures used in this project are grayscale only, there are 256 different possible intensities. When comparing the intensity of cancer and non-cancer images, no clear pattern can be observed.

2.3 Algorithms and Techniques

Convolutional Neural Networks (CNNs) are Multi-Layer Perceptions (MLP) for which not all the nodes in the network are connected. To understand better what MLP and CNN are, it is essentially to understand well how a smaller unit that they are build upon - the perceptron- works.

The perceptron is one of the simplest ANN (Artificial Neural Network) architectures. It was invented in 1957 by Frank Rosenblatt. The perceptron is a simple unit which takes several variables as an input and assigns them weights. Then, it uses an *activation function,* f(w,x) where *x* are inputs and *w* are the assigned weights. The value of the function is compared against a predefined *threshold.* If the value is greater or equal the threshold, the output is positive (1), otherwise it is 0. For this reason, the perceptron may be used to perform binary classification but it cannot be used for e.g. probability prediction.

This problem is solved by creating a network with multiple perceptrons, i.e. Multi-Layer Perceptron (MLP). MLP is composed of one *input layer, hidden layers* (LTUs) and one final layer of LTUs which is called the *output layer*. The weights are trained using *backpropagation* and an optimizer, e.g. *gradient descent.* Backpropagation works as follows:

1. For each training example the output of every neuron in each consecutive layer is calculated.
2. The network's output error is measured as a difference between the value predicted by the network and true value.
3. Calculate how much each neuron in the last hidden error contributed to this error, all the way back to the input layer.
4. Once all error contributions are calculated, optimizer is used to adjust the weights.

An important change that was made to MLP when comparing it with a single LPU is the activation function. The step function can be replaced by (e.g.):

- logistic function (sigmoid),
- hyperbolic tangent function
- RELU function

For my project I used ReLU function for hidden layers and sigmoid function for the final layer (to get the probabilities).
ReLU function is defined as:

$$RELU(z) = max(0, z)$$

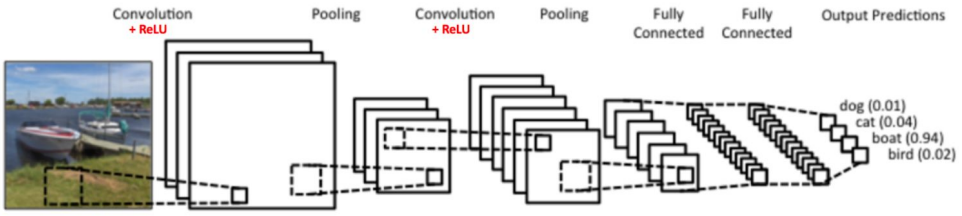
ReLU function is continuous but it is not differentiable at z=0.This issue is called *the vanishing gradient problem* (see [18] for more details).

I have chosen ReLU as it can be considered currently and *industry standard* and it usually gives good results. On top of that, it has a benefit of faster training and convergence and accounts for non-linear relationships. The output layer will use sigmoid function as its activation function since our CNN performs a classification task.

Convolutional Neural Networks are build on MLPs. However, they are a little more complicated and contain more elements. The base element is called *convolutional layer.* Convolutional layer is different from a fully connected layer (in MLPs) because it does not calculate weights for all neurons, meaning, not all neurons from one layer to another are connected. The advantage of this approach is a reduction in number of parameters to be trained.

Convolutional layer is accompanied by a *pooling layer.* This layer combines the outputs from the convolutional layer so that the condensed information can be passed to the next layer. Examples of pooling layers are: *max pooling* which takes maximum value from each of cluster of neurons and *average pooling layer* which uses the average value from each of a cluster of neurons.

Then, the last element in a CNN is a *fully connected layer* (there may be more than 1). They are the layers used in MLP, where all neurons are connected. They process all the information from the convolutional layers and provide the final output.
Therefore, the general CNN architecture looks like in the figure below:



CNN architecture Example (ConvNet) - (source: https://www.clarifai.com/technology)

There are very good blog posts describing the CNN architecture and how the algorithm works (see [19], [20] and [21]) so I will not go into details. I will only provide a high level description.

The image classification with CNN is a very interesting problem. First of all, the way the computer sees an image is very different from how human eye does; for a computer every image is just a numeric array like shown in the figure below:

What We See

What Computers See

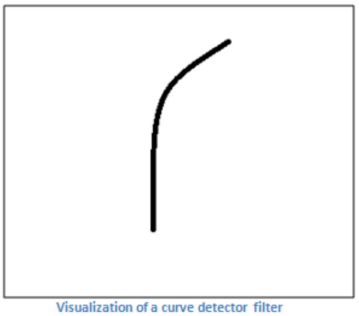
source (https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/)

So, every time a computer processes an image it converts it to an numeric array, 2 dimensional if the image is in gray scale or 3 dimensional if it is color image (there are 3 RGB channels: red, green and blue). In this form it gets fed to the CNN.

When the image get to the first layer, the most general patterns in the image are detected, such as shapes: curves, edges etc. They are called *features/filters.* A very good example is discussed in [20]. The author uses a curve detector as a filter as shown below. Every filter is represented by a numerical array, just like the input image.

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

source (https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/)

In order to detect a particular feature in an image, the filter slides over the input images. For every “slide step”, a matrix multiplication of the analyzed image area and the filter is performed. If the obtained results is big it means that the pattern has been detected.

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

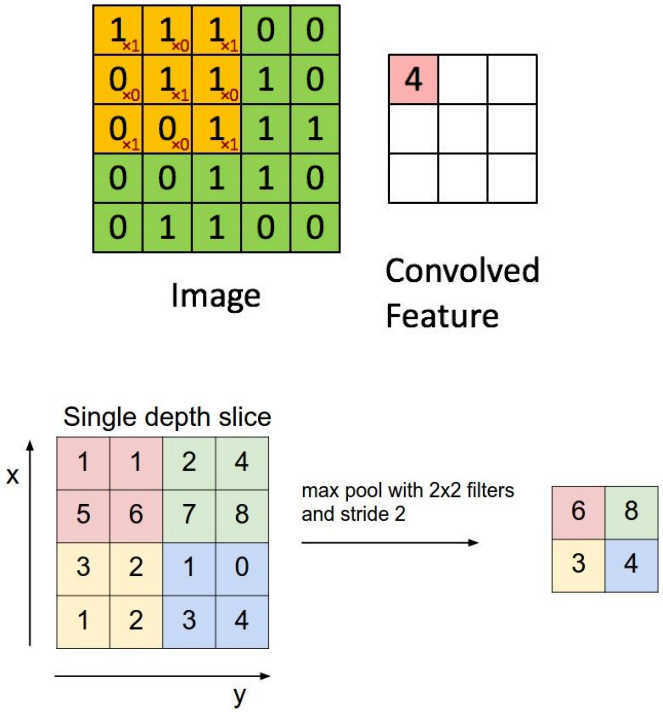
Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

source (https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/)

The filter has smaller dimensions than the image, therefore we can slide the filter over the image and define the “slide step” called a *stride.* For example, we can have a filter 2x2 pixels, image 40x40 pixels with stride 1 meaning that we always slide the filter by one pixel.

The above is a very high level description of how filters in convolutional layers work. The deeper in the CNN we are, the more complex filters are defined.



Convolutional Filters - source: [19]

The last block of the CNN are the fully connected layers which outputs the probabilities that an image belongs to every class in scope; that’s why softmax activation function is used here.

The last thing I will discuss is the optimizer used to train the CNN. I decided to use *adam* optimizer (see [22]). *Adam* optimizer is a popular choice for image classification problems due to its multiple benefits (see [23]):

- easy to implement,
- computationally efficient,
- requires little memory,
- well suited for problems that are large in terms of data and/or parameters.

Testing different optimizers is not in the scope of this project - this setting has been held constant throughout the CNN training.

2.4. Benchmark

As discussed in the previous sections, a simple SVM classifier was developed for CNN benchmarking. SVM can be an alternative to CNN in image classification tasks (see [8] and [9]). However, as this project focuses mostly on CNN approach, I will not get into details regarding the SVM methodology.

SVM model was defined by *GridSearchCV* option in Python package *sklearn* which outputs the best SVM model for a given data. *The best model* is a set of parameters which showed the best performance. The parameters are predefined by the user and *GridSearchCV* checks all possible combinations. The parameters to search over in this project were:
parameters = {'kernel':('linear','rbf','poly'), 'C':[1, 10]}
GridSearchCV returned a model with *rbf* kernel and *C* equal to 10.

The obtained results are compared in the table below. The CNN achieved better performance than the SVM benchmark.

Model	Accuracy	Sensitivity	Specificity
CNN (Model 8)	96.78%	98.98%	94.58%
SVM (*rbf, C=10*)	88.98%	84.41%	93.56%

III. Methodology

3.1 Data Preprocessing

In order to split the dataset into train, validation and test samples, *sklearn* function *train_test_split* was used twice. First, I split data to training and test (80% train, 20% test) and then the train sample was split again to train and validation samples (25% validation and 75% train). As result, I obtained train sample of size of 1768 CT scans and test and validation samples of size of 590 CT scans both. Also I used *stratify* option to make sure that all samples has equal number of positive and negative cases.

Another important step to data preprocessing was to convert data to the format required by the chosen model/package. Since I decided to use Keras to train our CNN, the required format is 4D tensor. For this purpose two functions were defined: *path_to_tensor* which takes a file path as an input and converts the image to 4D array, and function *paths_to_tensor* which simply calls *path_to_tensor* for every path in a list (input) and returns one Numpy array.

Similarly, for benchmark model, Support Vector Machine classifier, I needed to convert our images to 1D Numpy array. I defined a simple function *reshape_image* based on Numpy function *reshape.*

3.1.2 Data Augmentation

As part of this project, the following data augmentation techniques were explored:

- **rescaling**: it is performed in order to obtain vectors with values from 0 to 1 as they are easier to handle by the model (stability, overflow, optimization etc.).
- **rotation** (45 degrees): so the nodule is classified the same way even if the CT scan is rotated.
- **vertical and horizontal flips:** similar reasoning as for rotation.

Data augmentation techniques are proved to increase the performance of deep learning algorithms (see [13],[16]), especially for small datasets. Therefore, as a part of this project I wanted to check if using data augmentation will give us better results.

3.2 Implementation

In summary, the project consisted of the following steps:

1. **Data preprocessing:** data analysis, labels creation, splitting in training, validation and test sets, converting data to required formats.
2. **Data augmentation**.
3. **CNN training**: defining different CNN architectures and checking the model accuracy - model selection criteria. This step also involved training CNN with augmented data what did not yield satisfactory results.
4. **SVM benchmark:** definition of the SVM classifier to be used as a benchmark model.
5. **Performance assessment**: comparison of the accuracy, sensitivity and specificity achieved by CNN model and its SVM benchmark.

To start the training process, I have defined a simple CNN architecture (*Model 1)* with 3 convolutional layers and 2 fully connected layers:

```
model = Sequential()

model.add(Conv2D(32, (2, 2), activation='relu', input_shape=(40,40,3)))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.summary()
```

To prevent overfitting, I applied a dropout technique (see [15] for more details) with a dropout probability of 0.4. Dropout is an technique in which the neurons are randomly turned on and off during the training. In this way, the network is forced to explore new features and does not go all the time through the same neuron.

With *Model 1*, I have achieved an accuracy of 96.9492%. It is already above 96.38% (as mentioned in section 2.2), but I wanted to experiment different CNN architectures, so I continued the training.

3.3 Refinement

The results of the training are summarized in the table below. I have tested different CNN architectures using augmented data, however it decreased the accuracy and gave a very poor performance of 50%, meaning that the model was no better than a random chance. Therefore, I have used original data only with no augmentation.

The best performing model achieved accuracy of 96.9492% and it was the very first model I have tried. I have tried to make changes: add more layers, incorporate batch normalization, change number of filters but nothing helped to increase the accuracy achieved by model 1.

The second model with the highest accuracy was Model 6 with score of 96.4407%.

```
model6 = Sequential()

model6.add(Conv2D(32, (2, 2), activation='relu', input_shape=(40,40,3)))

model6.add(Conv2D(32, (3, 3), activation='relu'))
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(BatchNormalization())
model6.add(Dropout(0.4))

model6.add(Conv2D(32, (3, 3), activation='relu'))
model6.add(MaxPooling2D(pool_size=(2,2)))
model6.add(BatchNormalization())
model6.add(Dropout(0.4))

model6.add(Flatten())
model6.add(Dense(32, activation='relu'))
model6.add(Dense(2, activation='softmax'))

model6.summary()
```

Model 6 : Architecture

Since the accuracy of the both, Model 1 and Model 6 was pretty close and they are both above the benchmark of 96.38%, I analyzed their sensitivity, specificity and confusion matrix to chose to final model.

```
Confusion Matrix for model CNN Model 1 :
 [[289   6]
 [  7 288]]
Accuracy : 97.80%
Sensitivity 97.97%:
Specificity 97.63%:
```

Model 1: Performance metrics

```
Confusion Matrix for model CNN Model 6 :
 [[281  14]
 [  7 288]]
Accuracy : 96.44%
Sensitivity 95.25%:
Specificity 97.63%:
```

Model 6: Performance Metrics

I decided to chose Model 1 as accuracy and sensitivity are better for this model and the specificity is equal.

The main modifications I was testing during the training were:

- Greater number of layers,
- Greater number of filters,
- Global Average Pooling layer instead of Flatten layer (see [12] for motivation),
- Batch Normalization. It limits the amount to which the updating parameters in the earlier layers can affect the distribution of values in the consecutive layers. It let the layers to learn more independently from one another and it speeds up the learning process in the network (see [25]).

Model	Model Description	Data	Accuracy
Model 1	3 convolutional layers and 2 fully connected layers. (32,32,32,32,2) with relu and softmax (last FC layer) and dropout probability of 0.4.	Non-Augmented Data	97.7966%
Model 2	Model 1 with dropout layers, batch normalization and Global Average Pooling Layer.	Non-Augmented Data	92.2034%
Model 3	Model 2 but more filters (32,64,128,32,2) and higher dropout probability (0.5).	Non-Augmented Data	92.2034%
Model 4	Model 2 with 2 extra layers and additional dropout layer.	Non-Augmented Data	90.3390%
Model 5	Model 1 with Global Average Pooling layer.	Non-Augmented Data	83.3898%
Model 6	Model 1 with batch normalization	Non-Augmented Data	96.4407%
Model 7	Model 6 with extra layers.	Non-Augmented Data	95.5937%
Model 8	See figure above.	Non-Augmented Data	95.2542%
Model 1	3 convolutional layers and 2 fully connected layers.	Augmented Data	50.00%
Model 2	Model 1 with dropout layers, batch normalization and Global Average Pooling Layer.	Augmented Data	50.00%

Due to the time and resources limitations I have not continued the model training. I decided to stick with Model 1 and compare it against simple SVM classifier.

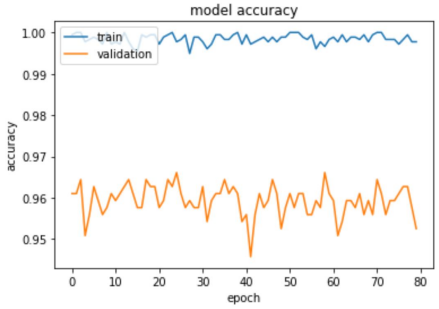
IV. Results

4.1 Model Evaluation and Validation

The architecture used in Model 1 is very simple. It has just two hidden layers with relu and dropout technique to reduce the risk of overfitting (dropout probability of 0.4). For hidden layer I also used MaxPooling. There are two fully connected layers at the end, one with ReLu and the last one that outputs the classification probabilities with softmax.

```
model = Sequential()

model.add(Conv2D(32, (2, 2), activation='relu', input_shape=(40,40,3)))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(2, activation='softmax'))

model.summary()
```

I believe that the results obtained from the model may be trusted. As discussed throughout the report, there were no signs of overfitting observed and the model was tested on an independent test set. The results are in line with the numbers achieved by SVM classifier.



Model 1: training history

4.2 Justification

The CNN model performs better than its SVM benchmark. When comparing all the evaluation metrics in scope (accuracy, sensitivity and sensibility), CNN achieves a higher score for each of them. In addition, the achieved accuracy was higher that the one reported in [10] which was one of the objectives of this project (see section 1.3.2 for more details).

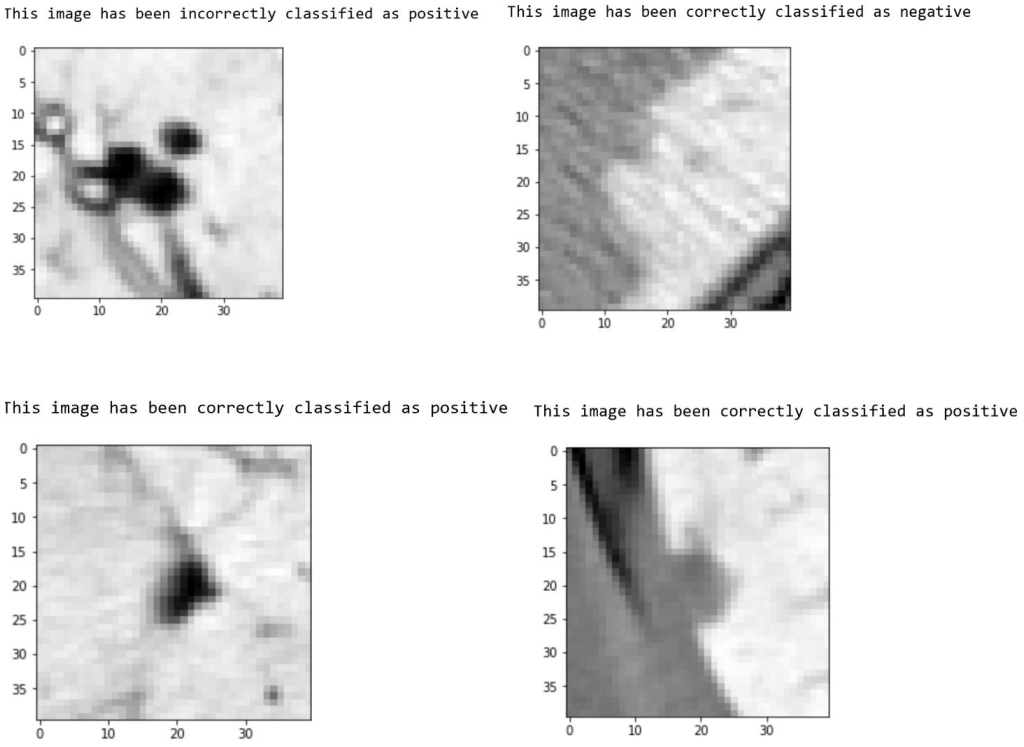
However, the difference between the benchmark model from [10] is very small. In my opinion, more training and testing of different architectures should be performed to find a better model. I think that a real improvement would be to find a model above 98%. However, the purpose of this project is purely educational, therefore the objective has been fulfilled.

Also, probably using transferred learning could elevate the model performance (discussed in section 5.3).

V. Conclusion

5.1 Free-Form Visualization

The below images shows an example output of the model with an information whether the classification was successful or not. There are 3 images which have been correctly classified and one that was incorrectly classified (false positive).



Also, another interesting visualization of the model in my opinion is the confusion matrix. I present the confusion matrices of both, CNN and SVM so that we can understand better why CNN outperformed its benchmark and got higher scores for all evaluation metrics.
As we can see from the data below, SVM has significantly higher number of false positives, 46 vs 6. While they are very similar in terms of false negatives (19 vs 7), CNN classified much more cancer CT scans correctly.

249	46
19	276

SVM:: Confusion Matrix

289	6
7	288

Model 1: Confusion Matrix

5.2 Reflection

Project overview

In summary, the project consisted of the following steps:

6. **Data preprocessing:** data analysis, labels creation, splitting in training, validation and test sets, converting data to required formats.
7. **Data augmentation**.
8. **CNN training**: defining different CNN architectures and checking the model accuracy - model selection criteria. This step also involved training CNN with augmented data what did not yield satisfactory results.
9. **SVM benchmark:** definition of the SVM classifier to be used as a benchmark model.
10. **Performance assessment**: comparison of the accuracy, sensitivity and specificity achieved by CNN model and its SVM benchmark.

Project takeaways

The most difficult part of the project was to train the CNN. I had to use external GPU provided by floydhub as otherwise it would take ages to do it on my personal computer. I had to configure everything accordingly to be able to run it in the cloud. Also, floydhub was not my first choice, I have tried Google Colab and Google Cloud before but there were not very intuitive so I switched to floydhub.

The most interesting part for me was to test the models and see the full potential of CNN methodology. It was exciting to see what accuracy was achieved by each of the CNN architectures and how changes in the CNN architecture impacted the final output.

The most interesting thing I learnt when doing this project is that data augmentation may not always work, especially when dealing with CT scans, X rays etc. Here the positioning of the image is very important and any deviation from that could lead to mislabeling for CNN and incorrect diagnosis in real world. It is very important to always think about the actual application of the model when defining the model.

5.3 Improvement

I think that the project could be improved by training on more data. After all, the important limitation of the chosen dataset was that it did not contained big number of observations, but it was enough to train the binary classifier. However, if more data is available, more labels could be used and the CT scan classifier could bring much bigger benefit if used in medicine. The CNN with binary classifier and two labels *positive* and *negative* is rather simplistic but it was used for education only and therefore it is fit for purpose.

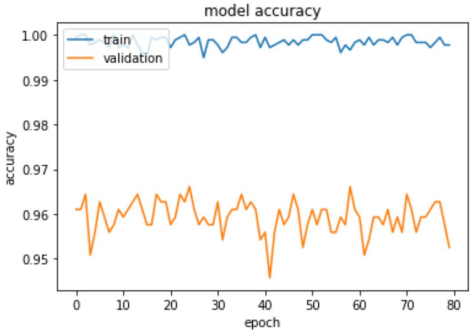
Another idea on how this project could be expanded is to try transferred learning. There are several CNN architectures that could be tested in this model and it could increase the accuracy, sensitivity and specificity of the model.

Model Robustness and Generalization

Model robustness has been tested by splitting the datasets to 3 different sets: train, validation and test. For training both sets, train and validation, were used and the test set was only used to check the network performance. The model is expected to perform well on train and validation sets as during the training the CNN weights were optimized so that the accuracy (not accuracy directly - by using loss function) on these tests is high. However, the test set is independent and therefore, it confirms the model robustness. It would be better if there was more data for both, testing and validation but this is a known limitation of the dataset used in this project. However, having this limitation in mind, the model achieved very good results.

Several techniques were applied in this project to ensure model generalization: balanced samples, dropout and batch normalization. Batch normalization generally plays a different role - it is used to speed up the learning process- however, it also adds some noise to the hidden units, and therefore it forces the downstream hidden units not to rely too much on other units - makes them more independent. Dropout, as mentioned in previous sections,turns neurons on and off and therefore, forces the network to discover new connections and not always follow the same path. And when it comes to balanced samples, they make sure that the model is not biased towards any of the labels and is trained, validated and tested equally for both -negative and positive - labels.

When analyzing the training history (figure below) it can be seen that there is no overfitting: the validation accuracy is always below the training accuracy what also shows that the model is robust.



Model 1: Training History

6.References

[1] *Cancer Facts and Figures 2018*, American Cancer Society.

[2] https://www.cancer.net/cancer-types/lung-cancer-non-small-cell/statistics

[3] *Can Lung Cancer Be Found Early?*, American Cancer Society.

[4] *Dermatologist-level classification of skin cancer with deep neural networks*, Andre Este, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau and Sebastian Thrun, 2017.

[5] *Convolutional Neural Networks for Breast Cancer Screening:Transfer Learning with Exponential Decay*, Hiba Chougrad, Hamid Zouaki and Omar Alheyane, 2017.

[6] *Deep Convolutional Neural Networks for Lung Cancer Diagnostics*, Mehdi Fatan Serj, Bahram Lavi, Gabriela Hoff and Domenec Puig Valls, 2018.

[7] *Going Deeper with Convolutions* , Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich, 2014.

[8] *Toward intelligent training of supervised image classifications: directing training data acquisition for SVM classification*, Giles M. Foody, Ajay Mathur, 2004.

[9] *Support vector machine for breast MRI image classification*, Chien-Shun Loa, Chuin-Mu Wang, 2012.

[10] https://apollack11.github.io/machine-learning.html

[11] https://luna16.grand-challenge.org/data/

[12] https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

[13]https://blog.keras.io/building-powerful-image-classification-models-using-very-littledata.html

[14]https://blog.insightdatascience.com/automating-breast-cancer-detection-with-deep-learning-d8b49da17950

[15] *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinon, 2014.

[16] *The Effectiveness of Data Augmentation in Image Classification using Deep Learning,* Jason Wang, Luis Perez.

[17] *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, Aurélien Géron, O'Reilly, 2017.

[18] https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257

[19] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[20]https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[21]https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8

[22] *Adam: A Method for Stochastic Optimization,* Diederik P. Kingma, Jimmy Ba, 2014.

[23] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[24] *Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift,* Sergey Ioffe, Christian Szegedy, 2015.