UNIVERSITY OF COPENHAGEN

DRAFT: 21/06/2018, 10:33

# Segmenting tubular structures in the pancreas using deep learning

Møllgaard, Christian Edsberg
dpj482@alumni.ku.dk

June 2018

# Contents

# 1 Abstract

This is an abstract

## 2 Introduction

There exists a hypothesis that the structure of the lumen network in pancreas, dictates which cells turn into beta cells. To test this hypothesis, annotated data of the pancreas is required. To annotate this data by hand would be a huge project, so we explore if it's possible to automate.

Deep learning using neural networks has become one of the main tools for computer vision. With enough training data it is possible to use convolutional deep learning for almost every traditional computer vision task.

We study the possibility of segmenting the lumen structure of the pancreas using deeplearning. It is tested if unlabeled data can help the prediction process, test if the prediction can benefit from preprocessing.

# Contract

## 2.1 description

There is a biological hypotheses that during pancreatic differentiation, the structure of the tubular lumen network (in which the cells are situated) dictates which sorrounding cells are turning into beta cells. Finding these structures so they can be studied, is the first step in proving this hypothesis. For this analysis, five 3d films of the pancreas development have been recorded on mice, and have been annotated with some labels in preperation for this study.

The purpose of this MSc project is to segment the tubular lumen network from the film using the deep learning library tensorflow.

## Project objectives and timeline

- Explore the data, and fix errors in the dataset
  Deadline 15.3
- Segment the images using a CNN with the given set of annotated labels.
  Deadline 30.4
- Try different preprocessing methods (Normalization, filtering) and record the impact.
  Deadline 30.6
- Try different semisupervised learning techniques to include unlabeled data in analasys.
  Deadline 15.5

## Learning goals

- Implement a CNN classifier for image segmentation using tensorflow
- Implement Augmented CNN in a semisupervised fashion using the unlabeled data
- Test improvements of segmentation when preprocessing

## Risk

The data have already been shown to include some errors, so these have to be fixed or removed before segmentation. There is also a possibility, that more images needs annotations.

# 3 Data

The data provided for this project consists of five three dimensional films of the pancreas of mice, while the pancreas develops over time. Including time and colour channels, means that the data consists of five dimensions. The colour channels does not represent the typical RGB, but is the product of different lasers interacting with tissue, and a fluorescent material injected into the mice. The fluorescent material mainly interacts with the wall of the tubular structure. In table ?? a overview of the data from a prestudy is shown.

Antager at der er vist et billede med dataen tidligere

Only the "green" colour channel interacts with the fluorescent material, so the other channels were excluded, reducing the dataset to four dimensions. The last 20 timesteps of the dataset 3, 4 and 5 the pancreas cells died, and showed no further development. It was thus discarded.

| Dataset | Width | Height | Depth | Timesteps | Colours | mouse id |
|---|---|---|---|---|---|---|
| 1 | 1024 | 1024 | 40 | 159 | 1(3) | 1 |
| 2 | 1024 | 1024 | 40 | 159 | 1(3) | 1 |
| 3 | 1024 | 1024 | 27 | 111(131) | 1(3) | 2 |
| 4 | 1024 | 1024 | 27 | 111(131) | 1(3) | 2 |
| 5 | 1024 | 1024 | 27 | 111(131) | 1(3) | 2 |

Table 1: Overview of datasets. Original size in the parenthesis, while used size without parenthesis.

Only a small part of the data came with labels. The labels were annotated in a way, so images were only partly labeled, and labels were concentrated around edges of foreground structure and big swathes of background outside of the general structure. No annotation had been done in the inner of any tubular structure, and only very little foreground was present.

## 3.1 Data incoherence

In the initial data study, the sums of all images in the datasets where ploted to check for inconsistensies . This generated the plot shown in Figure ??. It can be seen here that dataset 1 and 2 looks very different from dataset 3, 4 and 5.

Ikke stavet rigtigt?

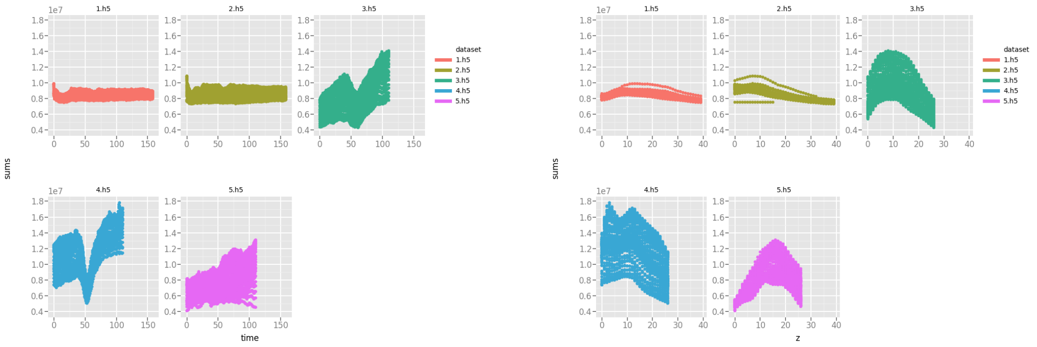

Figure 1: Sum over intensities over axis t and axis z

After discussing this with the biology supervisor, we decided to ignore dataset 1 and 2, as they looked preprocessed, and we did not have any idea how and when in the process they had been preprocessed.

## 3.2 Bad label mappings

It was discovered that the annotated labels for dataset 3, 4 and 5, did not directly match the image it was paired with. Because of this and the fact that we wanted to predict the whole structure it was decided to scrap those images and focus on creating new annotations instead of remapping the old labels to the correct image.

## 3.3 New labels

As a result of the bad labels as mentioned in Section **??** new labels are required to train a classifier. These new labels were done by me, and the goal was not to make perfect labels, but rather to make labels good enough to show that the method works.

A previous study of the data had succesfully tried to used clustering of patches, and it was decided that to ease the annotation process, we would preprocess the data with kmeans clustering. The centroids were retrained on each slice of the image, but the centroid were initialised to be the same as from the previous image. This way the centroid were almost the same for each image, but would rearrange based on contents of te current image.

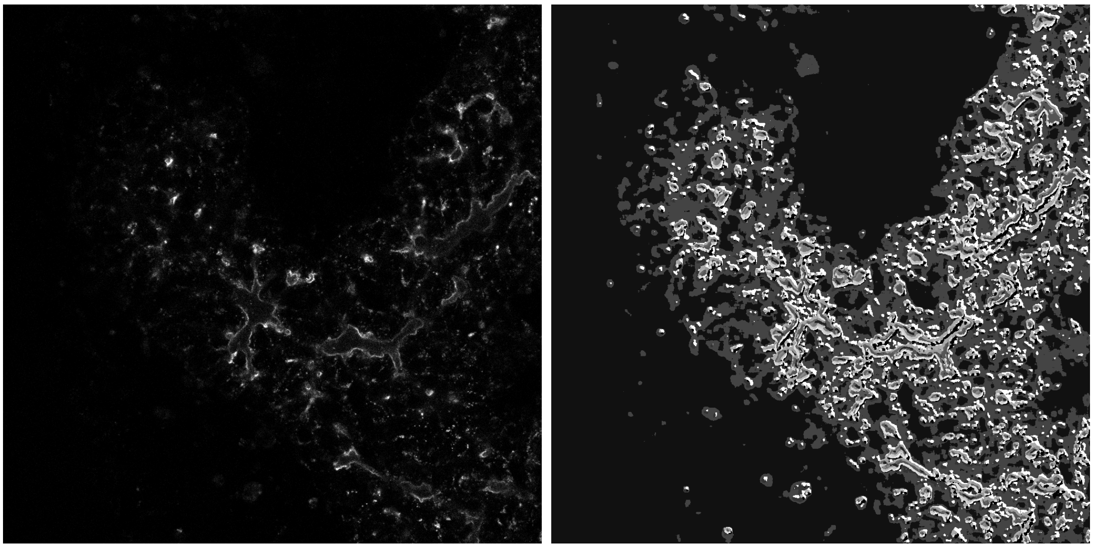

Figure 2: An example of the clusters

### 3.3.1 labeling difficulties

Even though kmeans helped the annotation process greatly, it was still very hard for someone with little knowledge about the workings of the pancreas, to make great annotations. As a result of this the most important and difficult areas around the edges of the structure, is not annotated when there is the slightest difficulty doing it.

# 4 Methods

## 4.1 Neural networks

Neural networks is a versatile machine learning model, where data signals are passed through several layers of neurons before a final prediction is made. An example of such a model can be seen in Figure ??. The intuition behind this model is, that as the signal is transfered from layer to layer a new representation is created. The initial layer creates a basic representation, like edges in an image. The representation are then becoming more and more abstract untill the last layer make a prediction.

Could easily explain more about everything, but it's kinda out of scope?

A single neuron takes as input the output from the previous layer and a bias value. These values are multiplied by a trainable weight for each input before being passed to an activation function.

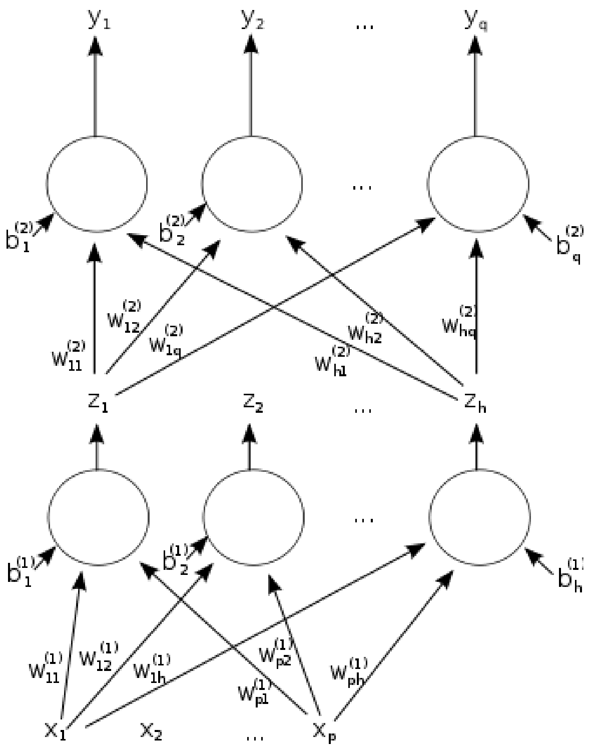

Figure 3: Example neural network with two layers

With neural networks it is often easier to regularize a network instead of tuning the network architecture.

### 4.1.1 Convolution

When working with structured data such as images, translation invariance is a very desirable property to have. To incoorperate this into neural networks, convolutional neural networks were introduced. Instead of a layer consisting of all the values from the previous layers being passed trough a single neuron, patches of data is passed through different neurons that are sharing the same weights. This way the network becomes more resistant to moving structures around the data without having to actually include data with structures moved around.

Convolutional neural networks often include pooling layers on the output of the convolution layers. The pooling layers outputs a statistical summary of the input of a input patch, similar to the concolution patch, defined for the pooling layer. This is partly used to reduce the output size, but also to select convolution patches with high signal values.

explain maxpooling better

### 4.1.2 Mini batching

To avoid overfitting the training data, it is desireable to not get the true gradiant over all the data. One method to avoid this, is to use the training data in very small batches, and update the weights using this as an aproximation of the true gradient.

This has the additional benefit, that it speeds up computation dramaticly, as we can get many weight updates from a single pass through the training data.

### 4.1.3 Dropout

A regularization technique to avoid overfitting is called dropout. It works as a layer in between the neural network layers. The intuition is that with a given probability a input value is set to zero, or otherwise just passed on as it is. This method helps the network not depend too much on any value from the previous layer, which helps the network be more robust to small changes in the input, as it cannot expect every value to exist.

### 4.1.4 RELU

A activation function is almost always part of the setup of a layer. This function

Droppet leaky relu så det bliver simplere. Skal lige genskrives works as a method to introduce nonlinearities to the network. RELU is a simple activation that is defined as:

$$RELU(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

One of the problems with this method is, that the gradiant is 0 when below 0. This can lead to inactive neurons if the output is always negative, if they are initialized badly. To combat this, the leaky relu function can be used. It is defined as

$$leakyRELU(x, \alpha) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha * x, & \text{otherwise} \end{cases}$$

where alpha is a hyper parameter. If x is less than 0, it will still learn a little, and can thus escape being always negative.

A positive thing about this function is that the gradiant is also very easy to compute, even though the function is not continues. It has a gradiant of 1 if above 0 or 0 below. For leaky relu its gradient is $\alpha$ below 0.

### 4.1.5 Softmax

Softmax is another activation function with a very desireable property. It normalizes the sum of all output to 1. Before the normalization, the data is used in an exponential function, which helps select high input more than low input. This makes it a great activation function for the last layer of the network, when working with classification.

### 4.1.6 Autoencoder

Autoencoders are an unsupervised neural network structure. The goal of an autoencoder is to reconstruct it self. The reconstruction is done after having been pushed through a small layer, so only the most important features are preserved. The intuition is, that the network consists of a encoder and a decoder. The encoder tries to encode the data to a selected size, and the decoder then tries to reconstruct the original data from the encoding.

This can be used to generate new data, but this can also be used as a dimensionality reduction technique. To use it for dimensionality reduction, the encoding is simply the desired output. The encoder is then simply used as the first layers of the prediction model.

### 4.1.7 Mean squared error

Mean squared error is a loss function. It simply squares the error, before computing the mean error of all predictions. This has the benefit that small errors are almost not punished, while big errors are punished more severly.

### 4.1.8 Categorical cross entropy

Categorical cross entropy is a loss function for classification. It optimizes the probability that the correct answer is selected. This is great because this is indirectly the same as optimizing the accuracy, but with a continues function. An additional benefit is, that it also improves on uncertain predictions even thouhg they are correct. A prediction of 51% on the correct category can still be improved alot.

### 4.1.9 ADAM optimizer

One of the most widely used optimizer for neural network is some kind of gradient descent. The idea is to follow the negative gradient untill it flattens out. With big neural networks, the gradient tends to fluctuate alot while going in a general direction that we want. To counter this fluctuations and stabalize the optimizer, momentum was introduced to keep the general direction while minimizing small fluctuations. Adaptive Moment Estimation (Adam) incoorperate momentum together with an adaptive learning rate which has made this a widely used optimizer.

### 4.1.10 Early stopping

To combat overfitting on training data, it can be beneficial to have an external validation training set that is used to validate that the model have not started to overfit. With early stopping a validation set is scored after each epoch, to validate that the the validation score has not gotten worse. If the model does not improve validation score after a few epochs, the training is stopped to avoid further overfitting.

### 4.1.11 Learning rate decay

Having a to high learning rate, it can be hard for the optimizer to find optimal solution, but having a two low learning rate, it will take a long time for it actually reach a solution. Using learning rate decay, the learning rate will gradually be decreased. This way the learning rate may be to high initially, but it as the learning rate decreases, it will eventually start to really learn.

## 4.2 Preprocessing

Often it is a good idea to help prediction by preprocessing the data, to incoorperate domain knowledge into the system, so the model do not have to learn what you already know, or to prepare the data using features not present in the data.

### 4.2.1 Normalization

Often it can be hard to compare data from different places in the data space. To counter this, the data can be normalized with a mean at 0 with standard deviation 1, so data looks the same.

### 4.2.2 Median time filtering

When using time series it can help to reduce noise, using the time axis. There are several ways to do it, and simply taking the median of the data of the time axis can help remove fast moving noise, while still keeping the image sharp.

The idea is that fast moving objects and random noise in the data will decline, while slow moving objects will remain in the image. This will happen in a way such that the noisy outliers will be removed, but the image should still be sharp.

# 5 Implementation

Everything was implemented using Python?? and packages for it. Manipulating images were done using scikit image ??. General handling of data after being imported from images were done using numpy ??. Some data preprocessing and dataset splits were done using scikit-learn??. The neural networks were done using keras??. To extract data from original formats to a single format, the bioformats?? was used, and the data was saved in the h5 format??

Syntes kun det skal siges at de ligger i hdf5 format? Den anden del var en del af forrige projekt

## 5.1 Data pipeline

The data pipeline is a Python class that handles all data loading and preprocessing, so the output can be fed directly into whatever model you want to use, as long as it predicts it's output pixel by pixel. It loads a the needed slices as simple two dimensional images. At this point it does time median filtering and slice normalization if needed. slices are then concatenates into the desired three dimensional blocks. The features are then turned into the patches that are fed into other models.

It has a variety om other features, such as sampling, loading several images blocks etc. These are implemented to make sure that model creation is as simple as possible, as all data handling is done using this module.

As a standard it samples data so annotations are balanced. In addition to this, it can prioritize background that is close to the tubular structures. The reason for this feature can be seen in the *Labeling difficulties* section??. This is done by making a binary dilation using a binary mask of the annotated foreground. This is logicly anded together with a binary mask of annotated background. This mask of close background is then used to fill up to fifty percent of the background labels.

Data augmentation is also done by this data model in the form of rotations. Due to technical difficulties, this is done in a really naive way. It is only able to provide ninety degree rotations.

Per default it drops patches, where the label is zero as these are the unlabeled pixels.

## 5.2 Neural network Models

All neural network models are regularized with early stopping. It has a patience of 5 epochs to decrease validation loss, before being stopped prematurely.

All models are also run with learning rate decay. There can be benefits to doing so, but here it is merely so the learning rate can be set too high in the begining, and then have a high decay, which means it after af few epochs will have a decent learning rate.

They are all trained using the adam optimizer. All hidden layers use the RELU activation function.

### 5.2.1 Simple network

The simple network is very limited by a limited amount of training data. It has two convolutional layers, followed by two dense layers. The first one has 32 filters, and the second has 64 filters, and a kernel size of three times three. After each convolution, there is a maxpool layer with pool size two times two. The first dense layer has 128 neurons, and the second have two neurons. The last layer has the softmax activation function.

Due to the lack of training data, the input patch size is rather small. it only uses a 9x9x5 patch size. Categorical crossentropy is used as the loss function.

Trainable parameter report, and maybe visuals?

### 5.2.2 Semi supervised network

The semi supervised consists of two parts. The encoder and the decoder parts are almost the inverse of eachother, so I will just explain the encoder. Instead of max pooling there is upsampling.

The encoder have three convolutional layers. The first one has 32 filters and a patch size of 5x5. The second has 64 filters and a patch size of 5x5. The third has 128 filters and a patch size of 3x3. Then the data is flattened and followed by two dense layers. First with 1024 neurons, and then with 128 neurons.

The autoencoder is trained using mean squared error, as we can both get unnormalized data and normalized data.

The classification part takes the output from the 128 neuron layer as input. It then appends three additional dense layers, with 64 neurons, 32 neurons and 2 neurons. The last layer has the softmax activation function.

Trainable parameter report, and maybe visuals?

# 6 Experiments

Lav en intro?

## 6.1 Better sampling

After early neural network structure testing, it was discovered that while loss looked great, the visual result did not look as good as the numbers showed. It was decided to see if we could sample better than random, to improve the visual results of the model.

The areas that were hardest to improve was the difficult area around the tubular structure. The experiment consists of visiually comparing images before and after sampling data, prioritizing data around the tubular structure. As this data will be harder to predict, it is expected that the accuracy and loss will increase.

## 6.2 Simple convolution

This experiment is just to show that it is possible to achieve a result close to what is expected. It will not have to generalize to all the data, but merely work as a start for more elaborate experiments to come.

## 6.3 Semi supervised convolution

Because of all the trouble with the data labels, an experiment to test the gain from utilizing the unlabeled data. It is expected that the model will generalize to unseen data, and to improve performance overall.

The idea is to pretrain a big part of the neural network using an autoencoder, and then only train the last few layers using the labels provided.

Both a visual result and a quantitive result is expected.

## 6.4 Preprocessing

The data originally have both a 3d image structure and a time axis. The time axis will not be directly used, so an experiment utilizing the time axis for preprocessing purposes will be needed. In addition an experiment to check if normalization will benefit prediction, is also desired. The results are to be compared quantitively.

# A test1

# B test2