Efficient nuclei detection and classification with deep learning

Peter Grønborg, s124262

Kongens Lyngby 2018

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

The goal of the thesis is to

Summary (Danish)

Målet for dette eksamensprojekt er

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring a M.Sc. in Engineering. The work was carried out from January 22nd to June 22nd 2018 under supervision of Professor Anders Bjorholm Dahl. The thesis represents a workload of 30 ECTS points.

Lyngby, 22-June-2018

Peter Grønborg, s124262

Acknowledgements

I would like to thank my....

Contents

Chapter 1

Introduction

1.1 Motivation

1.2 State-of-the-art

1.3 Aim and outline

Chapter 2

Deep learning

2.1 Deep learning in general

2.2 Faster R-CNN

Faster R-CNN is a unified network consisting of two modules[1], a Region Proposal Network(RPN) and a Fast Region based CNN(Fast R-CNN).

The core idea for Faster R-CNN is based in Region based CNN(R-CNN)[2]. The concept for R-CNN was to take an image make an external algorithm, like selective search[3], to generate proposals. Each of these proposals was extracted as a region and send through a CNN which could classify them. This method performed very well compared to, at that time, good methods. One huge problem for this method was time. Here each proposal/region had to run through a CNN which easily could take a couple of milliseconds. The amount of proposals suggested in the original paper[2] was 2000. This could take around 50 seconds to run through 1 image[4].

One of the authors of the original paper, Ross Girshick, decided to further develop on R-CNN and handle the time problem and thereby make a Fast R-CNN[5]. He suggested to take the entire image, run it through a CNN, take each proposal and make roi pooling after a feature map. From that each roi pool was sent through the remaining network and classified. This method made sure that the CNN should only run 1 time for each image and thereby lowering the runtime drastically. This made the runtime go from 50 seconds per image to around 2 seconds per image[4].

Even though Fast R-CNN made a huge increase in the speed it had a huge draw back. It still relied on proposals from an external method. This was then replaced with a new network, Region Proposal Network(RPN), in the paper Faster R-CNN once again with Ross Girshick as one of the authors[1]. This unified network, which consists of the RPN and Fast R-CNN, took the lead and became the state-of-the-art network. Once again the runtime went down. This time at around 0.2 second per image[4]. This unified network can be seen in figure 2.7.

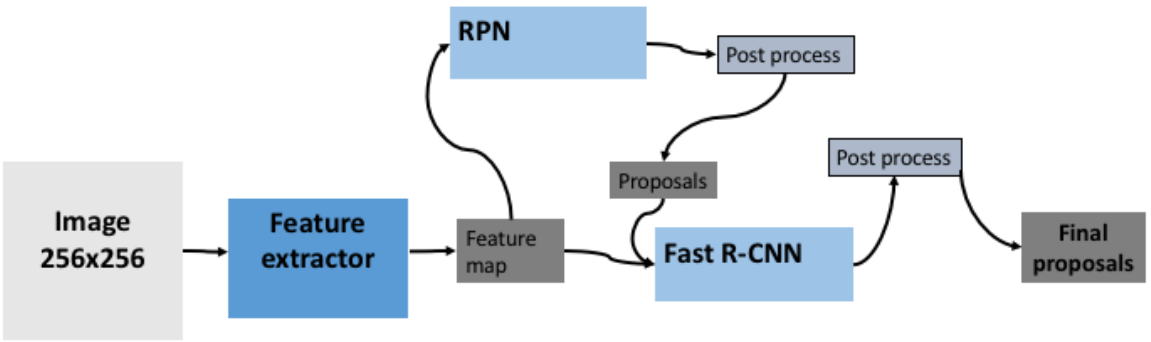


Figure 2.1: Illustration of Faster R-CNN

Even though the structure in figure 2.7 seems straight forward it is worth noting that the complexity in

this network is high.

In this model the object proposals happens in the RPN while the classifications of these proposals happens in the Fast R-CNN. The feature extractor network creates the foundation for both proposal layer and classification layer. Both post processes creates ways to clean the proposals from bad proposals and overlapping proposals.

The following sections take base in the paper *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*[1]. A lot of the structure in both RPN and Fast R-CNN isn't locked by this model and can be changed/modified. Also both methods can be standalone methods and applied in other models[6].

2.2.1 Feature extractor

The job for the feature extractor network is to take an image extract different filters and give out a feature map of the image. This can be seen in figure 2.2.

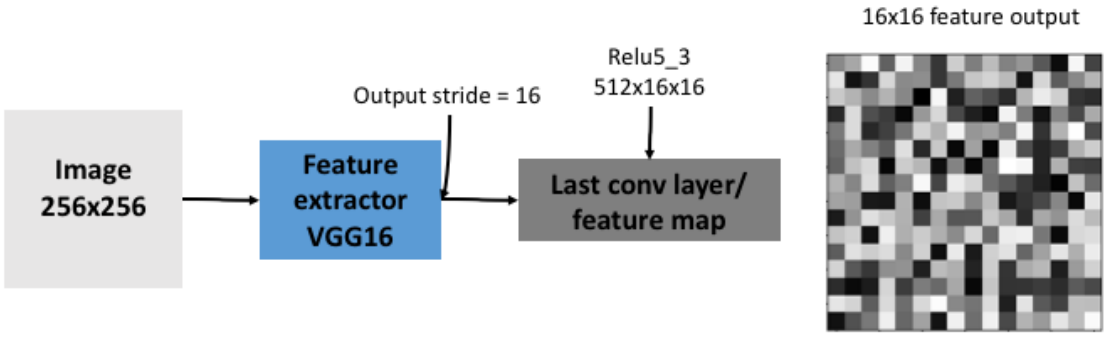


Figure 2.2: Feature extractor in Faster R-CNN

The feature extractor network for Faster R-CNN is usually a pre-trained image network like AlexNet, ResNet101 or VGG16. For the illustration a VGG16 have been chosen in figure 2.3. A reason to choose a pre-trained network is that it already performs all the operations needed for classifying an image based on the features extracted from the image[7].

For the object detection part, the features extracted at a certain point can be used to determine if there is an object(s) in the image or not. This part can also detect where these obects is placed in the image.

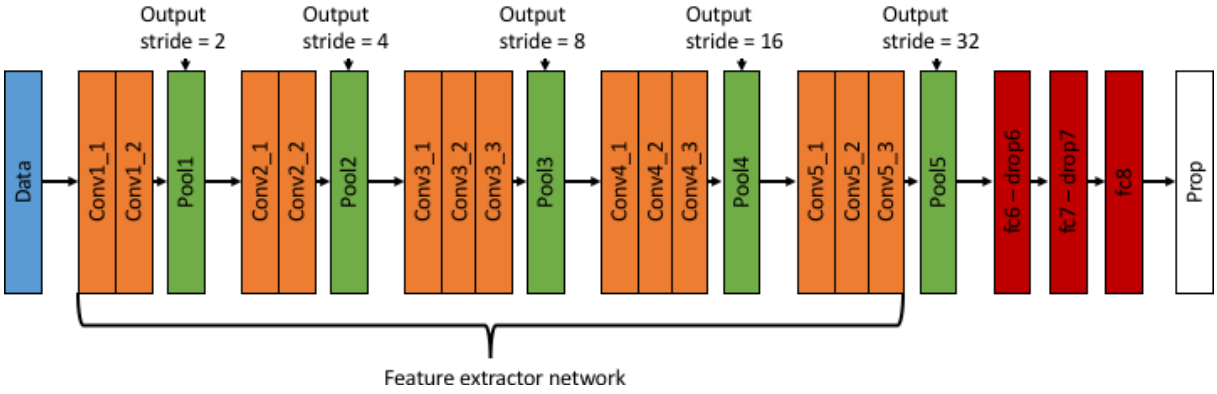


Figure 2.3: VGG16 layers

The VGG16 network consists of multiple convolutional layers, their corresponding relu layers, maxpooling layers and a couple of fully connected layers at the end[7]. Each convolutional layer takes an input convolute the input with a 3x3 kernel with padding and stride 1 and output the new convoluted map[8]. This map is further send to a relu layer which activates certain neurons. This process is done a couple of times within each convolutional group. After each convolutional group the maxpooling is made with a 2x2 kernel and stride 2. The convolutional layers increase the number of feature maps while the maxpooling reduces the size of each feature map. As seen in figure 2.2 after the fifth convolutional layer the feature map is 512x16x16 for an image of size 256x256x3.

In each convolutional group neurons is activated based on the input it get. The output from these convolutional groups is activation/feature maps which represent a certain feature in each map. In other words, these convolutional groups function as filters to get features[9]. This also becomes clear when seeing the actually activation maps[10].
In each convolutional group the features extracted is represented in a hierarchy. After the first convolutional group, `conv_1`, the features represented is lines, squares, circles etc. These features is considered low hierarchy because of their simplicity but also because they make the base for more complex shapes[10]. After the fifth layer the features activated is much more complex like letters, numbers, faces etc. The complexity within these objects is much higher and will therefore be considered as high hierarchy features[10].

In this way an image can then be transformed to a feature map which proposals can be generated on.

2.2.2 Region Proposal Network - RPN

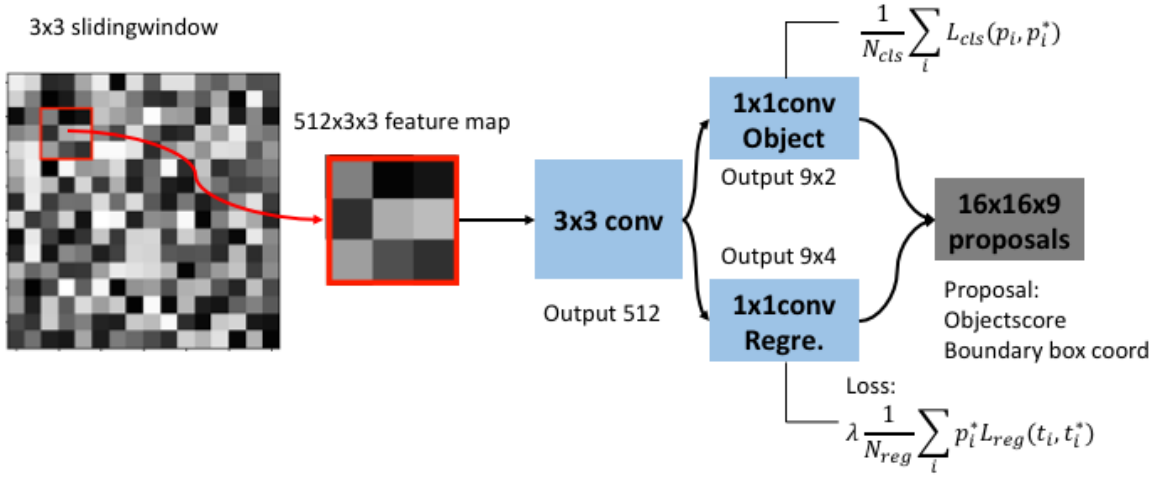


Figure 2.4: Region proposal network

2.2.3 Post processing

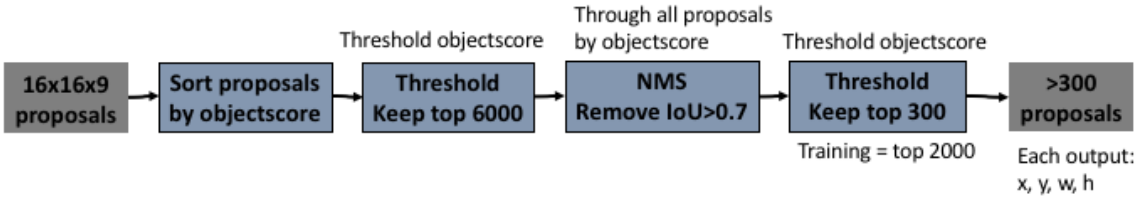


Figure 2.5: Post processing after RPN

2.2.4 Fast R-CNN

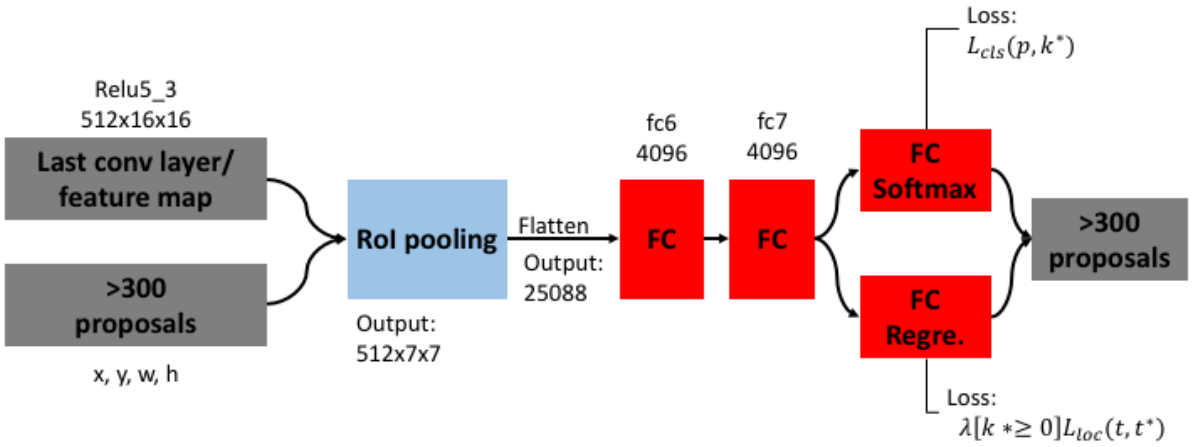


Figure 2.6: Illustration of Faster R-CNN

RoI-pooling

2.2.5 Post processing

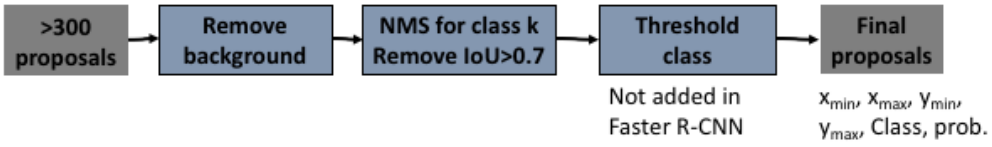


Figure 2.7: Post processing after Fast R-CNN

2.2.6 Training

Chapter 3

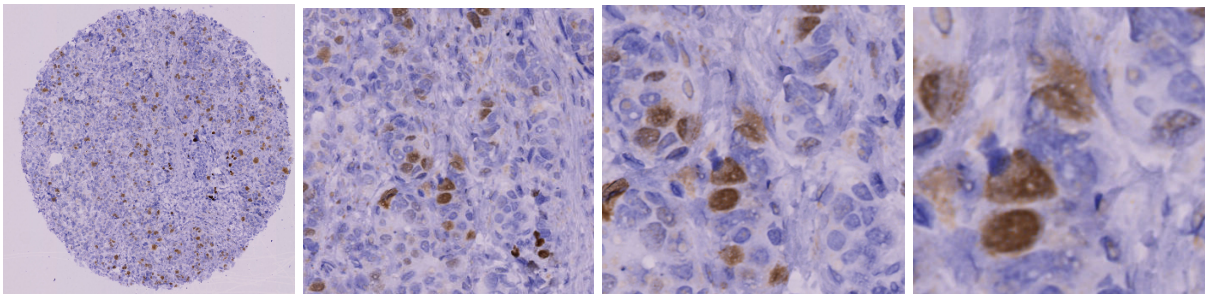
Data

The data delivered from Visiopharm consist of 7 scan scope images with up to 18 whole slices in each scan scope image. Each of these whole slices can be extracted as an image with resolution around 9200x9200 pixels.

3.1 Tumour to images

3.2 Raw data

The raw data is in quite high resolution in its raw format. To make the image more manageable it's necessary to magnify the image and take out a pad of the image. A standard format for magnification in Vis is x2.5, x5, x10, x20, x40. As i can be seen in figure 3.1 the entire Whole slice can be seen at x2.5. Here it's hard to distinguish each nucleus. From the figure it's also clear that the higher the magnification the easier it is to distinguish each cell. Here there will be a trade off between magnification and number of pads to process.



(a) x2.5 **(b)** x10 **(c)** x20 **(d)** x40

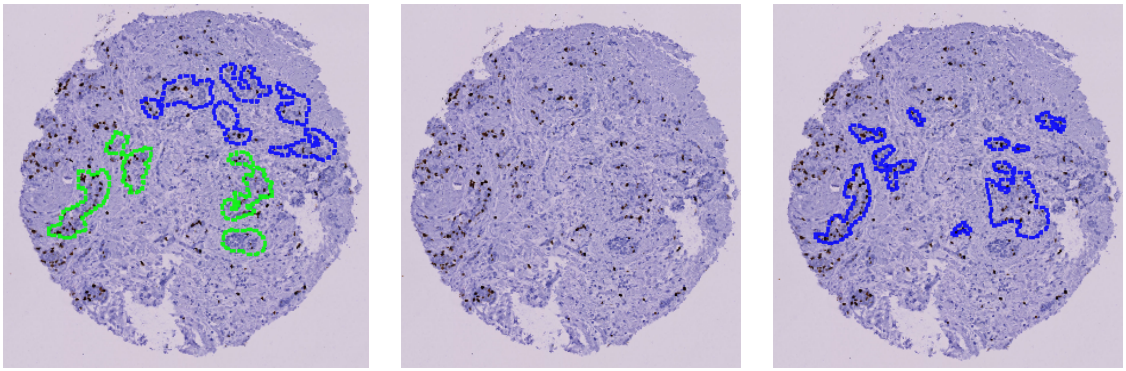
Figure 3.1: Whole slice image of different magnification

In figure 3.1c and 3.1d it's easy to see where some of the nuclei is but sharply distinguish background and nuclei isn't quite straight forward. It can be observed that how the nuclei is shaped and its appearance differs quite some. It's also worth to mention that in some areas of a whole slice the amount of nuclei is quite low leaving a lot of background and weird looking artefacts.

How the two pathologist have managed to annotate these nuclei can be seen in figure 3.3.

3.3 Annotated data

The cells in the images is annotated by two pathologist. Both pathologist have annotated the same slices and separately chosen where they wanted to annotate in each slice.



(a) Annotation from pathologist 1

(b) Original slice

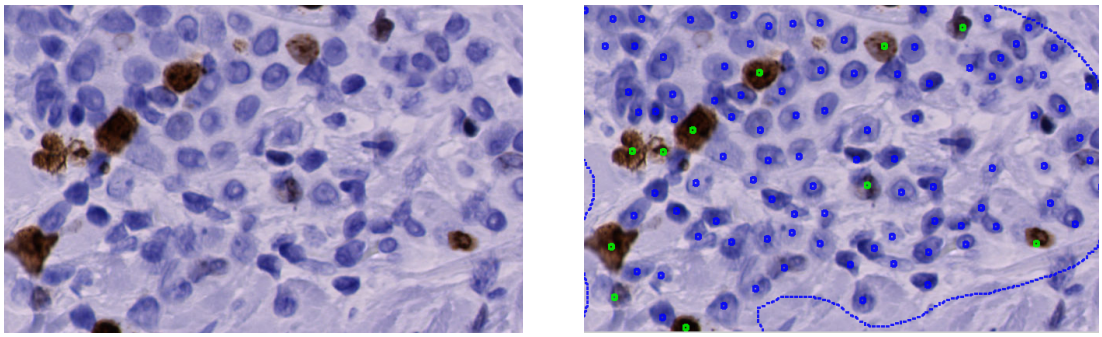
(c) Annotation from pathologist 2

Figure 3.2: Slices with annotation for the 2 different pathologist. Area for annotation is marked with blue and green.

From figure 3.2 one of the slices is displayed and the annotation for pathologist 1 and 2 is shown separately as well. It can be seen that each pathologist have chosen their own areas to annotate. Some of them overlap but most of them don't. The two pathologist annotations can therefore not be used to comparison of each other.

In the annotated data from pathologist 1 there is both green and blue areas. The colour just gives an indication of the ratio between positive and negative cells. Therefore it doesn't mean anything for the project.

Another rather important detail is the quality of the annotations. For the Faster R-CNN to work the annotation of the nuclei have to be consistent and precious. By this means that nuclei in a selected area should be annotated and as much in the middle of the nuclei as possible. These annotations make the foundation for where the boundary boxes will have their center. How pathologist 2 have annotated can be seen in figure 3.3. From looking through the annotations from pathologist 2 it seems quite consistent in the quality and nicely done.

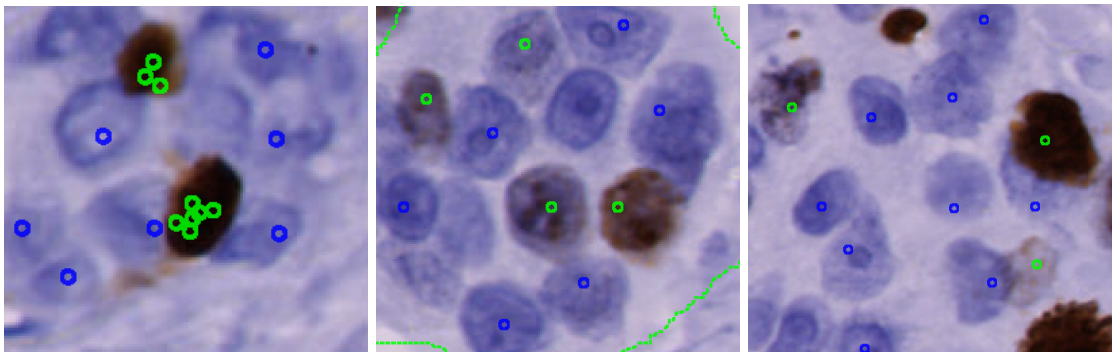


(a) Nuclei without annotations

(b) Nuclei with annotations

Figure 3.3: Nuclei with and without annotation from pathologist 2. Blue dots is negative nuclei and green is positive nuclei

When looking at the annotations from pathologist 1 these are much more varying and the quality isn't at the same high level as for pathologist 2. This is quite clear to see in figure 3.4. These examples isn't just some coincidences but a generally lack in the quality of the annotations from pathologist 1. It is therefor decided to discard the annotations from pathologist 1 to be used for training nor testing.

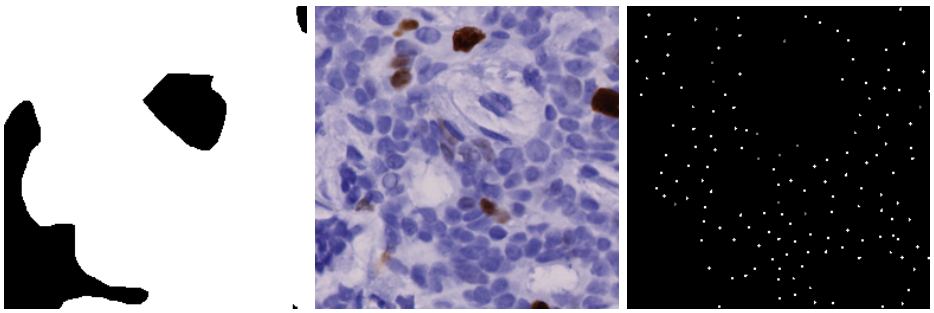


(a) Multiple annotations in the same nuclei **(b)** Missing annotation of nucleus **(c)** Annotations not consistent in the center

Figure 3.4: Different annotation for pathologist 1 with different errors

3.4 Data to analyse

The exact data received from Visiopharm can be seen in figure 3.5. As seen the data consists of three images which each contain information. The first figure 3.5a tells where the annotated area is. Figure 3.5b is the RGB image itself and figure 3.5c contains labels of the annotations. In the label data each label is either a pointset of 1s or 2s telling if the annotated nuclei is positive or negative.



(a) Mask for the image **(b)** Original image **(c)** Labels for the image

Figure 3.5: A pair of 3 images for 1 data image.

A thing to note in relation to the data is that not all images contain areas which isn't annotated while others primarily contains these areas.
The annotations in the label data can vary in size from image to image. The size of the annotation doesn't provide any information. This has to be handled in the pre-processing so it's size independent.

Further to mention is that each image can overlap with other of the images. This is definitely not ideal but have been necessary to generate plenty of data. When two image overlap it's only a part-wise overlap and not a full overlap.

Size of data

The provided data from Visiopharm consists of above 350,000 training images and their corresponding mask and label data. Further there is above 50,000 images for validation and test, and their corresponding mask and label data. The validation and test data is extracted from a completely different whole slice than the training data.

The presented data in figure 3.5 is 256x256x3 in size and at magnification x20. This data will be used to set the baseline for the model.

3.5 Assumptions

To make sure that the data is usable and that the model perform properly some assumptions related to the data have to be made. Most of the assumptions is related to the label and nuclei data since these easily can effect the performance of the deep learning model.

Labels is in the center of a nucleus.
This may seem quite trivial when looking at figure 3.5c, that each label is a center of a nuclei. This assumption is important since these labels set the base for the boundary boxes for each nucleus. If this assumption doesn't hold the ground truth boundary boxes will be placed incorrectly which leads to incorrect training.

All nuclei is annotated and in the same way.
This assumptions is related to the work of the pathologist. It's assumed that pathologist 1(the preferred pathologist) have been consistent in her way to annotate nucleus. This is both for annotating in the center of each nucleus, correct labelling of the annotations and annotate all visible nuclei. The last one also includes that if any nucleus isn't annotated because it's too obscure all similar nuclei shouldn't be annotated either.

Images have to be independent of each other.
Generally for data to a deep learning model it is expected that each input data is independent of the rest of the data. This also implies that each data input is unique and only exist once. Even though it is already known that the data overlap with each other, the data is still accepted since it's only a part-wise overlap.

Each class have nuclei in the same size.
A reason to have this assumption is to make the entire process as fully automated as possible and thereby making equal size boundary boxes. Since no information related to the size of the nuclei is provided a generally size for each class is required. When looking at figure 3.3 it seems to hold for most of the nuclei. If each nucleus had to be fitted with a costume boundary box the time consumption would explode and it would be too ineffectively. Therefore it is require to make the assumption that all nuclei within a class have the same size.

The model is background independent.
As it can be seen in figure 3.5 the data have certain areas which isn't annotated. These areas could possible contain nuclei which therefore haven't been annotated. To ensure that these nuclei doesn't affect the the model they have to be removed in the pre-processing phase. Since there already is a mask, figure 4.6, the easiest way would be to fill the non annotated areas with one color. This color should therefore be considered as a background color. The assumption is that the model is independent on which background color that is applied to these non annotated areas.

Chapter 4

Baseline

4.1 Data

When configuring data to Faster R-CNN the following list needs to be full filled.

- Image data(preferably in RGB format)
- File with list of all classes
- Training, validation and test files:
  - File with list of each set of images
  - File with list of all roi for each set of images

This sums up to have an arbitrary number of image files and 7 text files.

To generate the text files there is three options: Visual Object Tagging Tool(VOTT)[11], Annotation scripts[12] or to do it manually.

The two first option requires that the user manually select the area on the object. This is an easy way to make sure that the marked areas is in fact the exact area of the object which is ideal when an image contains few objects. For these histopathological images the number of objects/nuclei in each could easily be above 100 as seen in figure 3.3.

Since the data provided already contains annotated points the manual way is chosen.

First off the file with a list of all classes can be created.

```
1  __background__  0
2  avocado 1
3  orange  2
4  ketchup 3
5  onion   4
6  eggBox  5
7  joghurt 6
8  gerkin  7
9  pepper  8
10 champagne   9
11 orangeJuice 10
12 tomato  11
13 tabasco 12
14 milk    13
15 butter  14
16 water   15
17 mustard 16
18
```

Figure 4.1: Class file for Grocery data set

In the class file for Grocery data set in figure 4.1 it can be seen that the class `__Background__` is the first class to be included and therefor have the number 0. This is required for CNTK to filter away all predicted objects which don't belong to any other class. Here after follows any other class that the

images may contain. At the bottom it is very important to include an empty line. If the first line is not `__Background__` and the last line is not empty the script cannot run.

The way CNTK gets the path for an image is through a file with path for every single image.

```
0   testImages/WIN_20160803_11_28_42_Pro.jpg    0
1   testImages/WIN_20160803_11_42_36_Pro.jpg    0
2   testImages/WIN_20160803_11_46_03_Pro.jpg    0
3   testImages/WIN_20160803_11_48_26_Pro.jpg    0
4   testImages/WIN_20160803_12_37_07_Pro.jpg    0

```

Figure 4.2: File with path to all test images for Grocery data set

It can be seen in figure 4.2 that each line contains a number, the path to an image file and 0. Here the first number in each line correspond to the same number in the roi files, linking the image to its roi's. The 0 indicates that no other input is given. Once again the last line should be empty.
This file should exist for both training, validation and test data.

For the roi file each line should include all the roi's for one image and correspond to the path line in the path file.

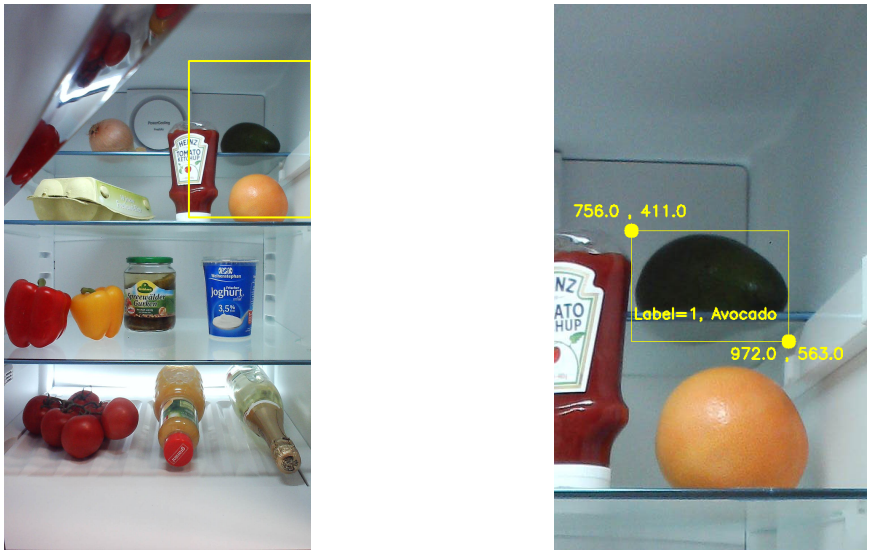
```
0 |roiAndLabel 756.0 411.0 972.0 563.0 1.0 789.0
1 |roiAndLabel 649.0 340.0 814.0 612.0 7.0 634.0
2 |roiAndLabel 616.0 680.0 831.0 785.0 14.0 424.0
3 |roiAndLabel 607.0 687.0 858.0 793.0 14.0 200.0
4 |roiAndLabel 728.0 367.0 856.0 582.0 16.0 730.0

```

Figure 4.3: Small cut out of the file with roi's to all test images for Grocery data set

As seen in figure 4.3 every line start with a number(that correspond in the path file) and `|roiAndLabel`. This indicates that roi's and labels comes.

To understand how the roi's and labels are structured a small example is beneficial.



(a) Original grocery image with cutout. **(b)** Cutout of original image with roi and label.

Figure 4.4: Test image from Grocery data set with roi and label for 1 object.

From figure 4.4b it's clear that the coordinates from the first line in 4.3 correspond pairwise to the two corners that is diagonally to each other and thereby marks the entire area. There is no order in which corner pair that comes first in the lines. This area is called the roi(Region of Interest). After the roi comes the label for that roi. For the first line this label is `1` which correspond in figure 4.1 to Avocado. After this comes the next roi and its label and so further.
This can be summarised in this way: `n` $|$`roiAndLabel` X_1^1 Y_1^1 X_2^1 Y_2^1 `label`^{1} X_1^2 Y_1^2`.....`

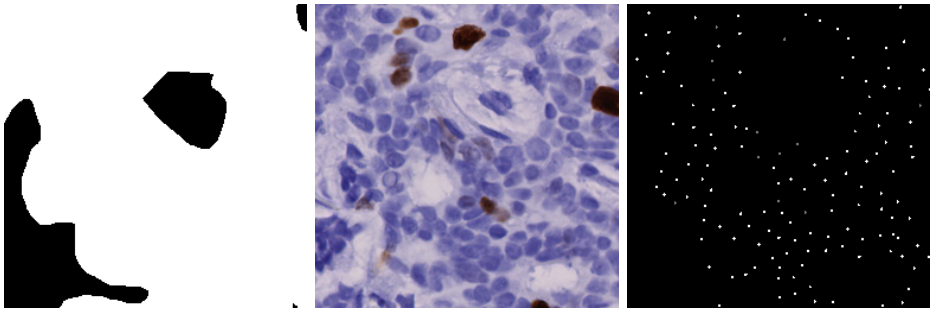
A quite important difference between this file type and the other file types is that the separator here space while it is tab for the other two. This make a difference for Faster R-CNN when it read each element. This file should also exist for both training, validation and test data.

All these 3 file types should be placed in the directory from where the path file points from.
The path to each of these files along with the number of images should be specified/modified in the util config file for the data. For the grocery data this file is called `Grocery_config.py`.

4.1.1 Pre-processing of data

Before the data can be put in to Faster R-CNN a pre-processing need to be made.

As shown in a previous chapter each image file comes in pair of 3 images.



(a) Mask for the image **(b)** Original image **(c)** Labels for the image

Figure 4.5: A pair of 3 images for 1 data image.

Figure 4.5a contains a mask of the area where nuclei have been annotated(white areas). Figure 4.5c is annotated in each pixel with a class label. 0 is background(black), 1 is positive nuclei(grey) and 2 is negative nuclei(white).

From this point the pre-processing can be split up in 2 tasks.

- Remove negative mask from the image and save the new image
- Find the centre of the annotated nuclei and save the annotated data in file format as described above

These 2 steps can easily be implemented in Python.

Masking the image

The goal masking the image is to remove any part that may or may not contain nuclei which haven't been annotated. If these areas isn't remove CNTK will try to correctly detect/classify those nuclei and mark them as missed detected/classified and add to the loss/error.

This process can be split up into a few smaller tasks.

- Load image(in RGB) and mask(in grayscale)
  `img = cv2.imread(ImgPath,x)`, x∈ (0,1), gray or RGB
- Find the areas that need to be masked away
  `i,j = np.where(mask==1)`
- Set the value for these areas to a uniform arbitrary value(white in this case)
  `img[i,j,1:3] = [255,255,255]`
- Save the new image
  `cv2.imwrite(NewImgPath,img)`

These steps will make a image that can be put into Faster R-CNN.

(a) Original image **(b)** Mask for the image **(c)** Original image masked

Figure 4.6: Masking the original image and the outcome.

As it's illustrated in figure 4.6 it shouldn't be possible to mistakenly detect or classify any object in the masked areas.

This step of masking can be done completely separated from the rest of the pre-processing.

Find annotated centre

From figure 4.5c it can be seen that the annotation comes as a grayscale image with each pixel label with either 0, 1 or 2. The annotation itself is then some circles with the same label, 1 or 2. These circles can vary in size from image to image.

Once again the main task can be broken into fewer smaller tasks.

- Load label image as grayscale image
  `img = cv2.imread(ImgPath,0)`
- Find all pixels belonging to label x
  `labelX = (img==x)`
- Find all objects/annotations for each class
  `im, contours, hi = cv2.findContours(labelX, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`
- Calculate center of each annotations, mean of each contours
  `valM = contours[i].mean(0)`
- Set boundary boxes using a scalar from center. Avoid having boxes outside of the image.
  `xmin = max(0,valM[0]-scale)`
  `xmax = min(img.shape[0],valM[0]-scale)`
- Write boundary boxes to file
  `file.write("0 |roiAndLabel "+str(xmin)+" "+str(ymin)+" "+str(xmax)+" "+str(ymax)+"\n")`

Magnification, resolution and ground truth boxes

For the data a baseline is set at resolution 256x256x3 at 20x magnification. The ground truth boundary boxes is set to 26x26 around each nuclei. These images with ground truth boundary boxes looks like figure 4.7.

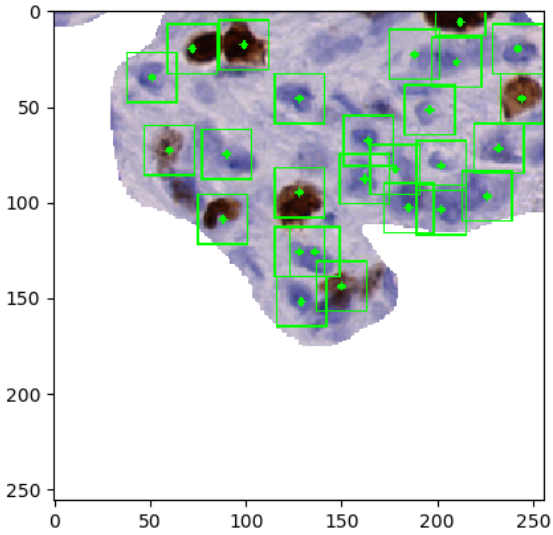


Figure 4.7: Image with annotation and ground truth boundary boxes

Even though it seems in figure 4.7 that these ground truth boundary box size doesn't fit perfectly to every nucleus, they generally fit. As this projects aims at doing everything as automated as possible and no information about size for each nucleus is provided there needs to be a fix size for each nuclei class. At this point it is decided to use an equal fixed size for both classes.

4.2 Model

The model along with data is now ready and a baseline of the parameters can be found. The first parameter to secure is one of the most essential, the feature extractor network.

As a general guideline for the baseline settings of various parameters, is the same as for the original provided code for Faster R-CNN by CNTK[12]. Some of these parameters have been modified slightly to suit the nuclei data better.

Feature extractor network

From the original paper they used two different pre-trained feature extractor networks(CNN), ZFNet and VGG16[1]. ZFNet was used to win the ILSVRC 2013 competition and is generally a deeper version of AlexNet where parameters have been fine tuned[13]. In the paper they found that VGG16 was superior to ZFNet but also that ResNet101 performed even better.

CNTK provides a couple of pre-trained networks for example VGG16, AlexNet, ResNet50[14]. One problem occurs when it comes to ResNet, dimensions of the layers doesn't fit to the standard of Faster R-CNN. Therefore average pooling of rois is necessary. Unfortunately this haven't been implemented yet[12].

Since ResNet is not possible to use directly and AlexNet have been shown multiple times not to be as efficient as VGG. It have been decided that VGG16 is the feature extractor network to use.

Layers in VGG16

As a standard from CNTK the layers used in Faster R-CNN for VGG16 is the following[12].

- Start training at `Pool2`
- Take out feature map at `relu5_3`
- Use layer `Pool5` to `drop7` for classifying

These settings make the number of parameters to train 136,449,349 in 32 parameter tensors.

As a reference to where those layers is placed in the VGG16 network, the network can be seen in figure 4.8.

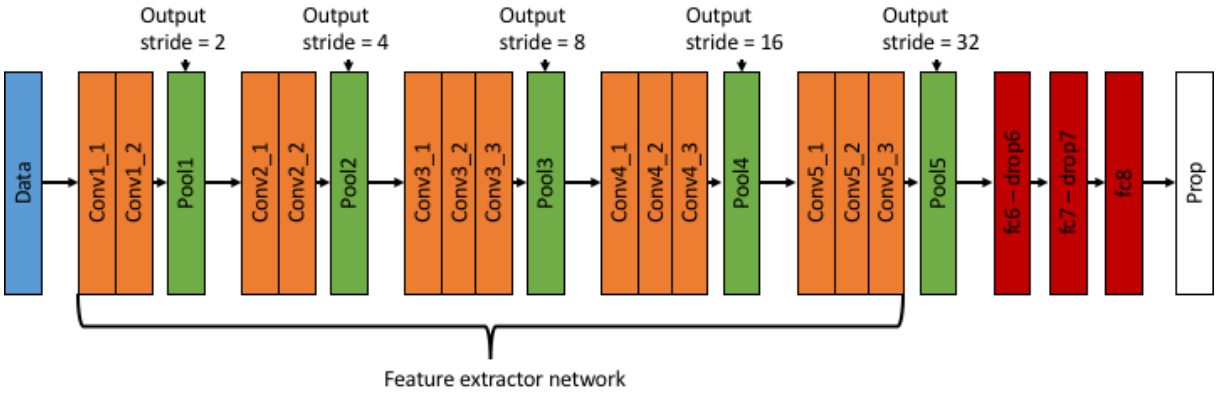


Figure 4.8: Overview of VGG16

Starting at a later point, use an earlier feature map or reduce the classifier part would result in fewer parameters to train. By starting training at `Pool3` or `Pool4` the number of parameters to train would reduce with 1.5 and 7.4 million. Even though this is quite few parameters to lose compared to the total amount it make a significant impact on the results. The mAP decreased with 0.05-0.15(from around 0.63). Since the number of parameters saved compared to the total number is quite low and results became significant worse it was decided to keep starting training at `Pool2`.

In the other end of the training is the classifier. By exiting at `drop7` to `drop6` the number of parameters could be reduced with nearly 16.8 million parameters. When trying to train with this setting it seemed that the network stopped improving after a couple of iterations. This combined with a very poor mAP indicated that it was better to keep `drop7` than trying to improve with `drop6`.

Samples per epoch

The chosen feature extractor, VGG16, takes around 5 fps(frames per second)[1]. There is approximately 350,000 images in total. If all images is included in each epoch, one epoch would last around 20 hours without validation images. This would be necessary if the model should learn more than 50 object-classes but in this case the number of object-classes is 2(3 including background). Further it can be seen that each image easily can contain above 100 nuclei while Pascal or COCO normally contains less than 10 objects in each image.

After having tried some different settings varying from 500 images per epoch to 50,000 images per epoch it became clear that the size of samples per epoch shouldn't be too high. At 50,000 over fitting occurred

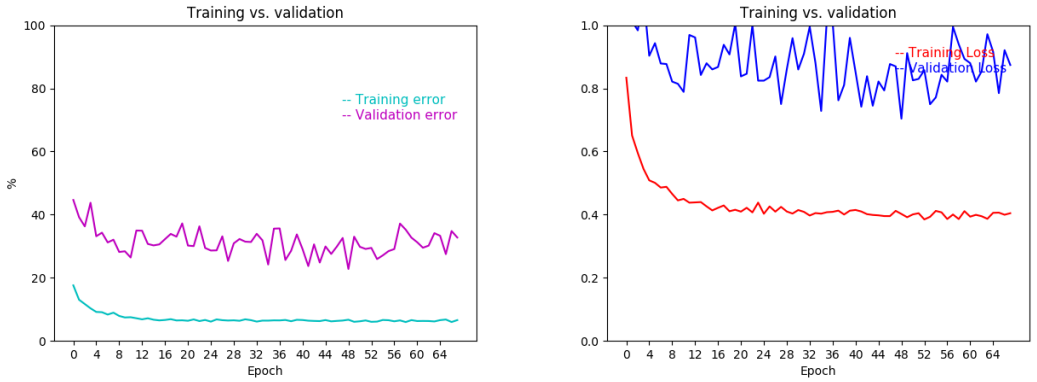
after the first epoch which took around 3 hours to run. Based on the images this also makes sense. Each image is quite similar to each other and contain at least 100 nuclei in each. This sums up to above 5,000,000 nuclei for each epoch. For a sample size per epoch at 500 the outcome could vary a bit between each run.

It was decided to set the size at 2,000 training images per epoch. This is set to be random from the 350,000 images. This gave a fair fit(without too much over fitting), stability and a reasonable run time. At this setting it still converges quick with above 200,000 nuclei per epoch.

Early stopping

Since the amount of images used in each epoch isn't that high compared to the amount in total it can be difficult to predict when to stop training. An easy way to determine when to stop training is when the model doesn't optimise anymore is to use early stopping[15]. There isn't a clear way to set a maximum number of epochs or when to use early stopping except it shouldn't under or over fit. This is of course also dependent on the data that the network is training on.

Normally early stopping is based on accuracy/prediction error. For competitions like COCO and Pascal this also makes sense since there is multiple object-classes, few objects per image and each object can look very differently from each other. In this case there is 2 object-classes, plenty of objects per image and each nuclei doesn't look completely different from each other.



(a) Prediction error for a training session. **(b)** Loss for a training session

Figure 4.9: Prediction error vs loss

It is clear from figure 4.9 that it seems like from the prediction error that the model starts to over-fit after 9 epochs while the loss starts to over-fit after 48 epochs. For Faster R-CNN the loss includes how well it predict each class, objects but also regression for each boxes. This could therefore imply that the model gets better at the regression of these boxes.

Based on these experiences it was decided to continue with the loss as the early stopping criteria. Because of the small amount of data(compared to the total size) in each training epoch early stopping is activated when the validation loss didn't change after 20 epochs and the max number of epochs to 200. The number of validation images is set to 1,000 images.

Tuning L2

From Figure 4.9 it seems that the model doesn't converge well but keeps bouncing up and down. This could indicate that the L2 regularization doesn't regulate enough. It was found that decreasing the L2 regularization made the output much smoother.

Figure 4.10: Loss curve for a model with lower L2 regularization

Figure 4.10 shows that it definitely improves the curve compared to figure 4.9. In this case the L2 regularization was decreased from 0.0005 to 0.00005 which is chosen for baseline.

Small changes

Other small changes that was made is to set maximum number of ground truth rois in each image to 200 instead of 50. Number of rois proposals was changed from 128 to 512. Image size was set to 256x256. Minimum size of a roi was set to 8x8 from 16x16.

These changes was primarily made to suit the dataset better.

4.2.1 Stability

4.3 Hardware

Chapter 5

Results

5.1 Baseline

5.2 σ and center weights

σ is used in the calculation of the loss function alongside with the center weights.

Table 5.1: Positive and negative overlap

Center weight	0.01		0.1		1		10		100	
σ	MAP	Time	MAP	Time	MAP	Time	MAP	Time	MAP	Time
0.01										
0.1										
1										
10										
100										

5.3 Learning parameters

L2 reg weight

Chapter 6

Discussion and conclusion

6.1 Discussion and future work

6.2 Conclusion

Bibliography

[1] Ren, Shaoqing, Kaiming He, Ross Girshick, and Jian Sun: *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017, ISSN 01628828. https://arxiv.org/pdf/1506.01497.pdf.

[2] Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik: *Rich feature hierarchies for accurate object detection and semantic segmentation*. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014, ISBN 9781479951178. http://www.cs.berkeley.edu.

[3] Uijlings, J. R.R., K. E.A. Van De Sande, T. Gevers, and A. W.M. Smeulders: *Selective search for object recognition*. International Journal of Computer Vision, 104(2):154–171, 2013, ISSN 09205691. https://koen.me/research/pub/uijlings-ijcv2013-draft.pdf.

[4] Santos, Leonardo Araujo dos: *Object Localization and Detection*, 2017. https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/object{_}localization{_}and{_}detection.html, visited on 2018-04-28.

[5] Girshick, Ross: *Fast R-CNN*. 2:9, 2015, ISSN 15505499. https://arxiv.org/pdf/1504.08083.pdf.

[6] Rey, Javier: *Faster R-CNN: Down the rabbit hole of modern object detection*, 2018. https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/, visited on 2018-02-26.

[7] Simonyan, Karen and Andrew Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition*. sep 2014. https://arxiv.org/abs/1409.1556v6http://arxiv.org/abs/1409.1556.

[8] Github: *VGG ILSVRC 16 layers*. http://ethereon.github.io/netscope/{#}/preset/vgg-16, visited on 2018-04-10.

[9] Karpathy, Andrej: *Convolutional Neural Networks (CNNs / ConvNets)*, 2018. http://cs231n.github.io/convolutional-networks/, visited on 2018-04-30.

[10] Karpathy, Andrej: *Visualizing what ConvNets learn*, 2018. http://cs231n.github.io/understanding-cnn/, visited on 2018-05-07.

[11] Özercan, Sertaç: *VoTT: Visual Object Tagging Tool*, 2017. https://github.com/Microsoft/VoTT/blob/master/README.md.

[12] Kranen, Philipp: *Object detection using Faster R-CNN*, 2017. https://docs.microsoft.com/en-us/cognitive-toolkit/Object-Detection-using-Faster-R-CNN, visited on 2018-03-19.

[13] Das, Siddharth: *CNNs Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more ...*, 2017. https://medium.com/@siddharthdas{_}32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5, visited on 2018-04-23.

[14] Zhang, Cha and Philipp Kranen: *CNTK Pre-trained Image Models*, 2017. https://github.com/Microsoft/CNTK/blob/master/PretrainedModels/Image.md.

[15] Radhakrishnan, Pranoy: *When to stop Training your Neural Network?*, 2017. https://medium.com/@pranoyradhakrishnan/when-to-stop-training-your-neural-network-174ff0a6dea5, visited on 2018-04-24.