Uppsala University

Project in Computer Systems

1DT104

Deep Learning Model for Classifying HER2 Positive/Negative Breast Cancer

Authors:
Anton Eklund
(anton.eklund.1905@student.uu.se)
Markuss Sprogis
(markuss.sprogis.1122@student.uu.se)
Haubir Mariwani
(haubir.mariwani.2039@student.uu.se)
Yuhua Chen
(yuhua.chen.0070@student.uu.se)

January 17, 2020

Contents

1 Introduction

Breast cancer is the most frequently diagnosed cancer in women worldwide and the leading cause of cancer death in most countries. An amplified and/or overexpressed human epidermal growth factor receptor 2 (HER2), which appears in approximately 15-20% of breast cancer cases, has been associated with aggressive clinical behaviour. However it also has a high probability of response to HER2 targeted therapy. This makes the correct identification of HER2 positive breast cancer patients a helpful marker for therapy decision making.

This is a "Grand Challenge"[1] published for the web page grand-challenge.org where different computer vision tasks within the field of biomedical analysis are posted as challenges. The aim of the proposed challenge is to find an image analysis algorithm to identify with high sensitivity and specificity HER2 positive breast cancer from HER2 negative breast cancer[2]. Our intent is for this to be done by researching and implementing a CNN image classifier using the latest and best techniques within the computer vision field.

2 Background

Breast cancer is the most common cancer diagnosis for women. Statistics from Swedish organisation Cancerfonden states that there where 10319 cases of breast cancer out of totally 28 763 cases of cancer in Swedish women 2017 [3].

Around 20% of breast cancer cases are diagnosed with the human epidermal growth factor receptor 2 (HER2)[20]. If this type of cancer is discovered and diagnosed correctly the survival rate and curative for patients is much increased[21]. However, the HER2 positive breast cancer is very aggressive compared to other types of cancer and is not likely to respond to hormone therapy. Therefore it is crucial to diagnose the patient correctly.

In recent years the field of computer vision has exploded due to better hardware and new techniques being discovered. Image classification is to mostly done with Convolutional Neural Networks (CNNs) and to great success[1, 24, 13]. CNNs have been worked on for the last four decades[12] and were used in medical image analysis as early as 1995[17]. Unfortunately the use of CNNs did not build enough of a momentum to be seriously considered for medical applications until new techniques for efficient training of deep networks were developed and hardware advancements were made.

According to a survey[16] on deep learning in medical image analysis done in 2017, commonly used deep CNN architectures for classification include VGG19[19], ResNet[10] and Inception v3[22]. A successor architecture to Inception v3 called Xception[4] was published in an article around the time the survey was published. Pre-trained models of the mentioned networks are available on most machine learning platforms such as Keras[14] and PyTorch[18]. In medical image analysis dataset sizes are smaller than in usual image vision e.g. hundreds/thousands vs. millions of images. This makes transfer learning using pre-trained models appealing. There are two schools of thought when using transfer learning. The first is using the pre-trained model as a feature extractor, the second is fine-tuning it to medical data. There are papers that delve into which method had the better performance but are contradictory to each other. Kim et al.[2] showed that the use of a CNN as a feature extraction outperforms fine-tuning and Antony et al.[15] showed the opposite. More recent articles[5, 9] which fine-tuned Inception v3 showed superb results for fine-tuning.

3 Dataset and Evaluation Metrics

The dataset provided by the grand challenge consists of 360 Whole Slide Images (WSI) of tissue. There is 144 cases of HER2 Positive and 216 cases of HER2 Negative. The classification of this dataset used the latest American Society of Clinical Oncology/College of American Pathologists (ASCO/CAP) classification of breast cancer. The WSIs are given in MIRAX-format which can be opened in special programs like CaseViewer.

The model precision is evaluated with the F1 precision score.

$$F_1 = 2(\frac{precision * recall}{precision + recall})$$

4 Methodology

This section will entail all methods, tools and environments used for the project.

4.1 Experimental Setup

Google Colaboratory[7] was used as the development and launch environment. It was chosen due to high flexibility and ease of use as an virtual environment as well as budget limitations for choosing a fee-based virtual environment. The 25.51 GB RAM and 358.27 GB disk storage space that came with it was considered enough. Colaboratory allows execution of python files to be run on Google servers and to have the code easily accessible and modifiable for all group members online. The training data can be stored in a folder in Google Drive[8] and be synced with Google Colaboratory VM during runtime. The drawbacks are firstly that there is a execution time limit of 12h so the code needs to be developed with that in mind. Secondly that GPU and CPU allocation from the server is not guaranteed so run-time can vary. Third is that you are only allowed a limited amount of concurrent running sessions per project.

Other options that can be considered is using virtual environments like Amazon Sagemaker or Google Machine Learning Engine which are more powerful options for VM:s. In those you rent a Virtual Space of your choice and can therefore run more complex model training. However, the group decided that we did not require more computing power than what Colaboratory provided and would instead use techniques like transfer learning to make up for it. There would also be a steeper learning curve to set up and manage a VM which would not be worth the invested time and money.

Another choice would have been to set up a desktop with powerful CPU, GPU and around 32GB RAM that would be powerful enough to run the training. This is an expensive solution but gives many options on how to set up the environment to our choice with fully dedicated hardware. However, this limits flexibility and accessibility for the group. Again, because time was not highly critical and training would only be a small part of the project it was decided that this option is worse than Colaboratory and Drive.

The code is written in Python3.7[6], using various TensorFlow[23] and Keras libraries in the solution. CaseViewer is used for opening the MIRAX files and exporting them to TIFF format. OpenCV library is used for image processing (filtering and importing).

In the computer vision field there has been great progress in recent years due to the rise of Convolutional Neural Networks (CNN). Therefore, this project also tries to make use of this method. A CNN in its simplest form (seen in Figure 1) when used on images consists of a convolutional layer where a kernel is applied to every pixel, or "convoluted" with the image. Because this generates an exponential amount of new variables, usually the network also consists of a Pooling layer where some sort of algorithm (usually max pooling) reduces the network down to a manageable size. Then in the end comes fully connected or "Dense" layers which is the simple building block of neural networks in order to make a classification. When using labeled input where you can check the correct answer compared to the classification made by the network is called Supervised Learning.

4.2 Image Pre-processing

The pre-processing was performed in several steps with the aim to address the need to feed the patch-level CNN with informative patches.

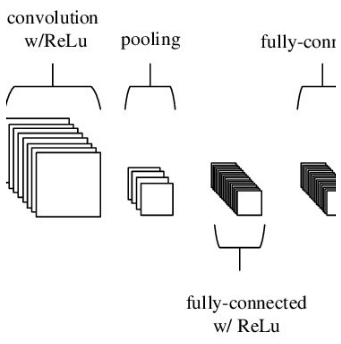

Figure 1: A simple CNN.

4.2.1 Patching the WSIs

The original WSIs are up to 7 Gigapixels (2GB) each. First step in the pre-processing pipeline is to reduce their scale. 1:4th of the original WSI reduces it to a reasonable size (50-150MB). The re-scaled WSI is still too large to train on so it is patched into smaller images as seen in the first step in Figure 2. The patch size is 296 pixels by 296 pixels. The patches are selected after running Otsu's algorithm to remove white-space from the WSIs. The patches include tumorous growths, supportive tissue and blood.

4.2.2 Blood filtration

The only information in the patches that is relevant exist in areas with tumorous growths which means that patches that contain too much blood and supportive tissue have to be filtered out from the training dataset. Due to differences in color of each WSI, they are normalized but when normalized, blood begins to look like tumorous growths. To avoid blood becoming a problem it is filtered before normalization with a red color filter. Then the patches are normalized using a Deep Convolution Gaussian Mixture Model (DCGMM). We used a stain-color normalization model created by Zanjani Farhad et al[25].

4.2.3 Tissue filtration

After the blood filtration, the next thing to focus on is making sure that the CNN is only fed with patches that contain enough cell nuclei and supportive tissue to tell anything about whether the cancer is HER2 positive or negative. Many patches did not fit this criteria, so the tissue filter was created to address this issue. It pushes a patch through a blue color mask designed to detect the blue colors that the cell nuclei are attributed with after the normalization process. When an image is pushed through a mask, a copy of the image is created where all the pixels that do not pass the mask are replaced with black pixels. Subsequently, the quota of black pixels in the copy are calculated.

$$quota = \frac{\text{No. black pixels}}{\text{total No. pixels}}$$

The process above is then repeated with a dilated version of the mask. So there will be two quotas to consider. The original mask quota makes sure that there are enough cell nuclei in a patch for it to be relevant. The dilated mask quota ensures that there are also enough relevant supportive tissue surrounding the cell nuclei in the patch.

After several test runs and observations of the algorithm, the most satisfactory filter output was achieved when the thresholds for the filter were set at:

$$quota\_mask = 0.85, quota\_dilation = 0.52$$

If any of the two quotas are exceeded for a patch, it does not pass through the filter.

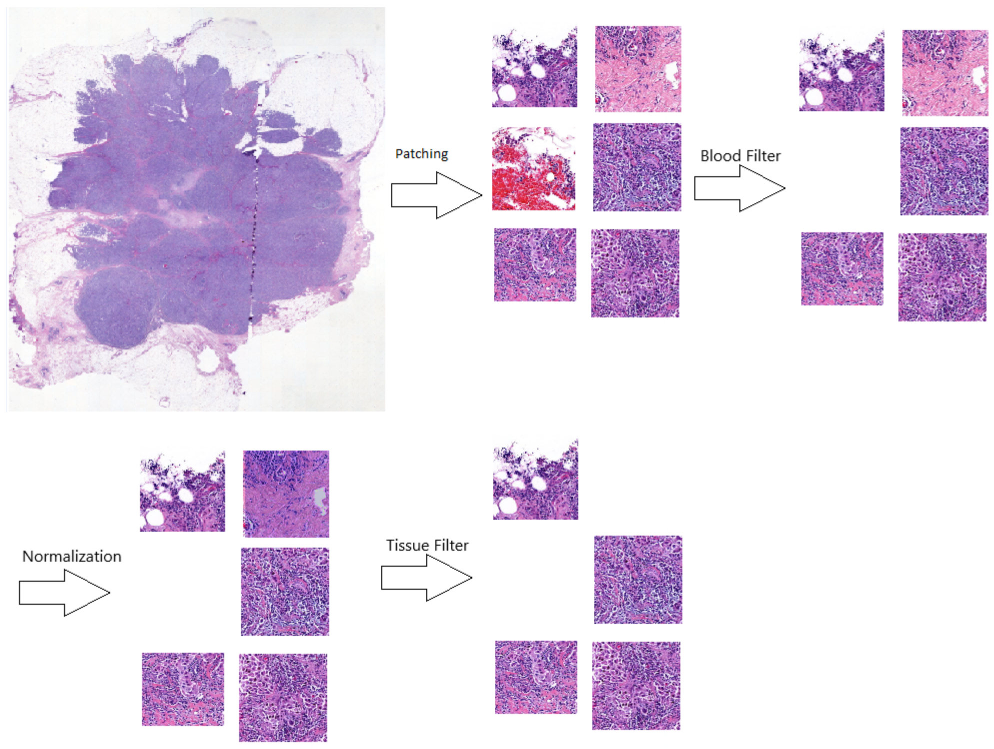

Figure 2: Pre-processing pipeline showing patching, the blood and tissue filters, and normalization steps

4.3 Transfer Learning

Transfer learning is the concept of using already trained models for tasks that they were not originally trained for. Big institutions with lots of resources and expertise design complex neural networks which they then train on an enormous general dataset. Then smaller practitioners like us can use the weights and design from these big models without having to train with millions of variables. Instead the work is to adapt the pre-trained model to fit the needs of specific projects by making the last layers trainable and to add your own layers after the pre-trained model. Hence, transferring the knowledge of the big network into your own model. The main advantage of transfer learning is that there is no need for an enormous dataset while the model can still achieve similar results as models trained from scratch and in much shorter time. As mentioned in the Background, there is no conclusive evidence on whether transfer learning models achieves better results than a model built from the ground up. This method of training has been chosen due to time constraints, hardware limitations and relatively small dataset size.

Xception[4] was used for training the patch-level neural network part of the system. It is fine-tuned by allowing training of its last layers and by adding a few layers after it in order to adapt the output for our problem. Xception is a 129 layers deep CNN and has been trained on the Imagenet dataset which is usually used as a benchmark for such networks.

4.4 Model architecture

The model architecture used for this project was based on the two-level model training architecture entailed in the paper written about Patch-based Convolutional Neural Networks for WSI image classification[11] published by Cornell University. The model consists of two levels due to the fact that WSIs are too large to be used to train the CNN. Hence, a WSI must be patched into (in pixel size) much smaller patches so that the CNN can be fed with these patches instead of the WSI. The WSIs have labels (Positive/Negative) and when the patches are created each patch is attributed with the label of the WSI that it came from. Since it is not guaranteed that an arbitrary patch contains the attributes of a HER2 positive tissue sample it is not sure to have been labeled correctly. E.g. one patch from a "positive"-labelled WSI may not contain any HER2 positive breast cancer tissue but it will still be labelled as "positive" since the WSI has that label. This is called Weak Supervised Learning as opposed to Supervised Learning since only the WSIs are guaranteed to be correctly labelled. Therefore, training of the whole model is divided into two separate models that we call the patch-level model and the image-level model.

4.4.1 Patch-level model training

The aim for this CNN is to be able to distinguish which type of patches that belong to a HER2 positive WSI. Because of the heavy filtering described in section 4.2 we end up with patches consisting mostly of tumorous areas of both HER2 positive and negative. The CNN then trains to classify each patch as either "Positive" or "Negative" correctly. This is a difficult task even with the filtering pre-processing. The CNN is not expected to do a great job at this but is rather supposed to produce some information about the patches that the image-level CNN can use.

In Figure 3 is an example of how the patch-level CNN could be designed. In most of our attempts we used transfer learning with the Xception model and then the layers after that can be experimented with. Combined with experimenting with factors like learning rate, loss function, activation function, dropout percentage, image augmentation, blurring with different intensity and more. All to achieve better scores on the validation set. This was trained on a dataset with around 3600 patches split into 80% training data and 20% validation data.

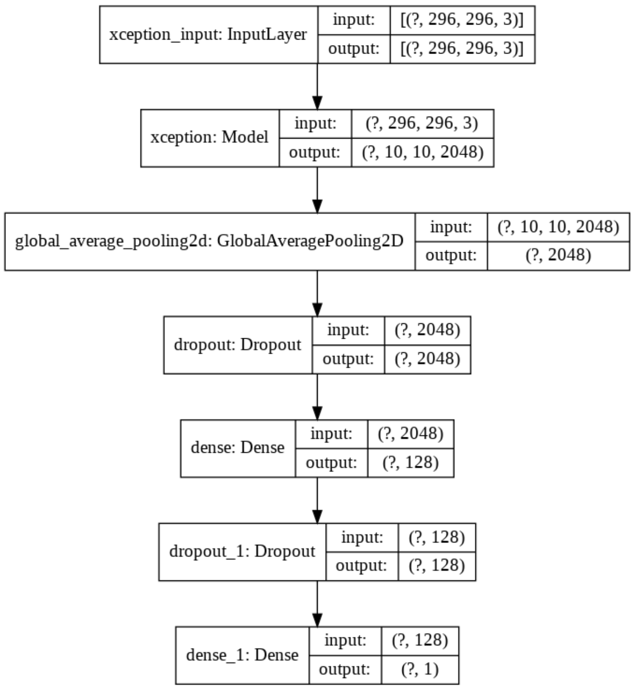

Figure 3: *The patch-level CNN model architecture. On the top there is the input layer and Xception network. Then GlobalAveragePooling2D, Dropout and Dense are our own layers added to the end of the Xception model to adapt it to our classification task. (?, x, y, z) represents the dimension of the input and output of each layer. In the beginning there is the image dimensions of 296x296x3 and it ends with a binary classification of either HER2 positive or HER2 negative. The ? is a variable input for the number of images that are fed to the network at a time.*

4.4.2 Image-level model training

The idea was to use a Support Vector Machine (SVM) to predict the WSI image-level label. The patch-level model would give the likelihood of a patch being positive or negative, when run on multiple patches it could give a vector of predictions. This vector could then be used as input for the SVM that would classify the whole WSI.

5 Results

The best validation set accuracy for the patch-level model when training on a 80%/20% data split reached 65.4%. The model consisted of a pre-trained Xception with the last layer being trainable and a smaller network that was attached to the end of it. The smaller network consisted of a dropout layer with 0.8 dropout rate, a fully connected layer with 128 nodes, another dropout layer but with 0.3 dropout rate and an output layer.

When testing different layer structures it was found that a high dropout was necessary for the model to generalize better. The dense layer was tested with 1024, 512, 256 and 128 nodes where the smaller amounts of nodes proved to train both faster and generalize better.

There were attempts at having multiple dense layers however the model tended to overfit to the training data. When the model was overfit it often resulted in a 50% accuracy on the validation set which is the same as purely guessing which label a patch should have. Many tested model designs also got the 50% acccuracy from simply weighing over to one side and classifying all patches as either negative or positive.

6 Discussion

This project has made many attempts at tackling the problem of distinguishing HER2 positive cancerous tissue from HER2 negative tissue. This has been seen as the crucial part of the bigger problem of classifying WSIs. The work has gone from very naive solutions to more and more complex ones as we progressed through failed solutions. The focus started from creating a CNN architecture that could classify the WSI but shifted into trying extract information from the patches. This was due to the discovery that even if we built more complex models they tended to overfit instead of generalizing and extracting patterns as we wanted it to do. This was a challenging task for a machine because usually when humans classify images with our eyes it is easy to distinguish patterns between e.g. cats and dogs. We have looked at patches for months now and still can not see with our bare eyes what we are looking for in the HER2 positive WSIs. Usually when pathologists make a diagnosis they stain the WSI with a color which was not present in the dataset given. It is not yet shown that finding these patterns is even possible.

Our best model achieved 65.4% accuracy but we are critical to if this reflects how it would do if we would use it in the image-level model. As mentioned the patches are heavily filtered in order to find tumorous areas which resulted in some WSIs having less than 5 patches. We don't think our patch dataset is general enough train a model general enough to classify all WSI. We would have to handpick which WSI to classify in a test situation with this patch-level model.

What we can say for sure from this project is that only building a CNN model, how advanced it may be, is not sufficient for this problem. When coming to that conclusion we saw there is two ways to approach this problem. Either work with the dataset to help the model with training and feature extraction or use a different approach that can classify the WSIs. A different approach that we researched and discussed a lot was Multiple Instance Learning (MIL) but we had difficulty setting it up. MIL is a more advanced way of choosing which patches are relevant in a WSI and is essentially an algorithm that is supposed to achieve the same thing as our pre-processing. Namely, choosing good patches for a CNN to train on. So we chose to spend our time refining and pre-processing the dataset because we saw it as being similar to what a MIL loop wold do. After working a lot on pre-processing we didn't get satisfactory results when training but still got some progress with accuracy score in the end of the project. We believe that if this problem is solvable it is in the pre-processing step more refinements and work has to be done.

7 Conclusion

In conclusion we can say that the problem of classifying HER2 positive WSIs is not solvable by trivial and naive solutions. Due to our team's previous knowledge, many approaches from simple models and pre-processing to more advanced where tested and dismissed. However, there is still a lot of techniques that we don't know about that a person with more expertise could try on this problem. In the end our model had 65.4% accuracy which could mean that the problem is solvable with more refinement work.

It has also been a crucial help to have discussed the problem with knowledgeable people in the medical field. Otherwise, there would have been no progress at all when working with the dataset. As previously mentioned, humans not trained in pathology cannot see any clear patterns in the patches which raises the need for domain knowledge to solve the problem. One conclusion to draw from this is that more areas of expertise than simply computer vision is probably required to attack these types of problems.

In hindsight we think choosing Google Colaboratory together with Google drive (Colab+Drive) was a mistake. When working with big datasets like this one you are required to be able to load, save, navigate and manage files rapidly. By choosing a cloud environment we spent huge amounts of time downloading and uploading files to Google Drive which limited our flexibility on what we could do with the dataset. Colab+Drive was chosen because we thought we were required to work on and run code to train the model from home. However, we spent most of our time working with the dataset and only a fraction of the time running training from home which we could have set up in another way with a desktop. If we were to repeat this project we would have gotten adequate hardware from the beginning in order to work with the dataset offline.

8 Future work

The results of this project show that the selection of patches of a WSI and what sort of feature extraction is used on these patches is important. More time should be spent on exploring relevant methods for feature selection. This means looking at techniques and algorithms that can be used on the patches before feeding them to the network in order to help the patch-level CNN find patterns that distinguish the different class labels.

More sophisticated filtration involving the detection of cells, cell patterns, maybe using some small WSIs of certain tumorous patterns and looking for the similarity between them and the sections of the image that was compared to them etc. In Python or object-oriented languages this could probably be achieved by implementing classes for each relevant pattern. It would require advanced knowledge about the HER2 classification and what to look for in images and patches.

The image-level classifier which is briefly described in 4.4.2 is also future work. If work can be completed on the patch-level CNN there is need for an algorithm that can use the patch-level model. This includes setting up a pipeline where one WSI can automatically be imported, patched, pre-processed, go through a patch-classification process and lastly be classified based on the information from its patches. This requires some sort of algorithm that use the information from the patch-level model. There was talk about using a support vector machine but there is of course many different techniques that can be used here.

References

[1] I. S. A. Krizhevsky and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.

[2] J. Antony, K. McGuinness, N. E. O Connor, and K. Moran. Quantifying radiographic knee osteoarthritis severity using deep convolutional neural networks, 2016.

[3] Cancerfonden. Cancerfonden. `https://www.cancerfonden.se/om-cancer/statistik`, 2018. [Online; accessed 13-November-2019].

[4] F. Chollet. Xception: Deep learning with depthwise separable convolutions, 2016.

[5] A. Esteva, B. Kuprel, and R. e. a. Novoa. Dermatologist-level classification of skin cancer with deep neural networks, 2017.

[6] T. Foundation and P. Software. Python 3.7.5 documentation. `https://docs.python.org/3.7/`. [Online; accessed 25-November-2019].

[7] Google. Colaboratory. `https://research.google.com/colaboratory/faq.html`. [Online; accessed 25-November-2019].

[8] Google. Google drive: Free cloud storage for personal use. `https://www.google.com/drive/`. [Online; accessed 18-November-2019].

[9] V. Gulshan, L. Peng, M. Coram, M. C. Stumpe, D. Wu, A. Narayanaswamy, S. Venugopalan, K. Widner, T. Madams, and J. e. a. Cuadros. Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *Jama*, 316(22):2402–2410, 2016.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[11] L. Hou, D. Samaras, T. M. Kurc, Y. Gao, J. E. Davis, and J. H. Saltz. arxiv:1504.07947v5 [cs.cv]: Patch-based convolutional neural network for whole slide tissue image classification. `https://arxiv.org/pdf/1504.07947.pdf`, Mar 2016. [Online; accessed 25-November-2019].

[12] F. K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybern*, 36(4):193–202, 1980.

[13] S. R. K. He, X. Zhang and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *ICCV*, 2015.

[14] Keras. Applications. `https://keras.io/applications/`. [Online; accessed 18-November-2019].

[15] E. Kim, M. Corte-Real, and Z. Baloch. A deep semantic mobile application for thyroid cytopathology. In *Medical Imaging 2016: PACS and Imaging Informatics: Next Generation and Innovations*, volume 9789, page 97890A. International Society for Optics and Photonics, 2016.

[16] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42:60–88, Dec 2017.

[17] L. S.-L. L. J.-S. F. M. T. C. M. V. M. S. K. Lo, S.-C. Artificial convolution neural network techniques and applications for lung nodule detection. *IEEE Trans Med Imaging*, 14:711–718, 1995.

[18] PyTorch. Torchvision models. `https://pytorch.org/docs/stable/torchvision/models.html`. [Online; accessed 18-November-2019].

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. `https://arxiv.org/pdf/1409.1556.pdf`, 2014.

[20] D. J. Slamon, W. Godolphin, L. A. Jones, J. A. Holt, S. G. Wong, and e. al. Studies of the her-2/neu proto-oncogene in human breast and ovarian cancer. *Science*, 244(4905):707, May 12 1989.

[21] D. J. Slamon, B. Leyland-Jones, S. Shak, H. Fuchs, V. Paton, A. Bajamonde, T. Fleming, W. Eiermann, J. Wolter, M. Pegram, J. Baselga, and L. Norton. Use of chemotherapy plus a monoclonal antibody against her2 for metastatic breast cancer that overexpresses her2. *New England Journal of Medicine*, 344(11):783–792, 2001. PMID: 11248153.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. `https://arxiv.org/pdf/1512.00567.pdf`, 2015.

[23] TensorFlow. Tensorflow. `https://www.tensorflow.org/`. [Online; accessed 25-November-2019].

[24] Y. B. Y. LeCun, L. Bottou and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[25] Z. S. B. B. E.-. v. d. L. J. A. Zanjani, F. G. Histopathology stain-color normalization using deep generative models. `https://openreview.net/pdf?id=SkjdxkhoG`, 2018.

A Learning outcomes

This section describes the attempts at solving the challenges that working with big data and image analysis has given us and their outcomes. As a group we have touched many aspects of this field of study ranging from advanced neural network design to specific image pre-processing. The main lesson we learned was that the quality of the data has much bigger impact on the accuracy of a CNN model than how the model is designed.

A.1 First Attempt: The Naive Version

As the title of this section implies, due to lack of experience in working with big data and image analysis in general, we failed to solve the problem multiple times. This attempt had two main steps: *patching* and creating a model for patch classification.

The images that were to be classified are huge. During the research stage we found papers describing the patching method when classifying these enormous images. The basic thought was to take an WSI, remove white-space from it with an algorithm, in out case we used the Otsu's algorithm, and then divide the WSI into multiple smaller (299px x 299px) images(patches).

These patches were then fed into a pre-trained model (Xception) of a convolutional neural network (CNN) for classification. We got it up and running. During this attempt patching was done per WSI directly on Google Colaboratory.

The Naive Version resulted in models that were not able to learn anything. These models chose one of the labels and always selected them as the prediction. What we learned was that most of the WSIs contained large amounts of unnecessary data. We had effectively fed our CNN with so diverse and mostly irrelevant data that it couldn't do anything at all with it. What we did not know, until we got in contact with the Computer Vision division here at Uppsala University, was what data in the WSIs was simple tissue/inflammation/artifacts and what was malignant tumorous cells. The amount of patches in each class was also unbalanced which resulted in the model favoring the label of the class with more patches.

A.2 Second Attempt

On November 27, our group met with Carolina Wählby from the Centre for Image Analysis and PhD/pathology student Renata Papatella at Uppsala University. At the meeting we learned that our general method was fine. The method at the time was not very specific as we were only patching and training. We discussed different pre-processing techniques that are used in image pre-processing and how their PhD students have gone about doing their image analysis projects. Renata showed us what the actual tumorous growths in a WSI were and we realized that our pre-processing was not nearly good enough. However, they seemed to think our general approach of just training a CNN could be enough if we worked more on pre-processing. We also got the information that if HER2 positive tumerous areas exist in a WSI then it is unlikely to to contain HER2 negative areas. Therefore we planned to reduce the dataset to patches with only tumerous areas to feed the CNN. A couple of pre-processing techniques for removal of unnecessary patches were discussed and we began the our second attempt at a solution for the challenge.

Since the meeting we have been working on filters and smarter ways to patch the WSIs. After a lot of trial and error we have came to a final method which we thought should work in theory. The method is described under our method section in the main article.

Sadly, the above did not turn out to be sufficient enough for a good validation score on the validation set when we trained the CNN. Hence we have chosen to not download the test set published by the arrangers, since we know nothing will come out of it and we all have other studies to attend to at this time.

B Problems

B.1 Data

The amount of data we got was small compared to the size of it. The downloading process took around a week and a half. In the beginning the challenge providers had an FTP server where the data could easily be downloaded. It proved to be slow and they moved the data to Google Drive which is a cloud storage solution. At first it seemed like a good idea but we soon realized that Google Drive was not capable of sending large files without splitting them up in multiple archives. This is not bad per se however Google dropped some of the archives making us keep tabs on what files arrived and what files had to be re-downloaded. It was a frustrating process to say the least.

Once the data was acquired the next problem was the data size. We had decided early on that Google Colabaratory (Colab) was the way to go since the computer that we borrowed only had a 256GB SSD on it. To use Colab we needed to move the data to Google Drive but 750GB (total dataset size) is a lot of data and would have cost us 100kr per month for each person that wanted to work with the data. Then there is the problem of loading 2GB to 5GB files into memory which was not reasonable. So we scaled down the pictures with 1:4 scale factor.

B.2 Patching

After the first week, during which we read multiple articles related to medical image analysis, we decided that we had to divide the big picture into smaller pictures. During our first attempt we tried doing it on the fly i.e. loading a WSI into memory, patching it and train a model. This tended to crash Colab due to memory limitations. Once we figured out that this would not work and why it would not work we began creating an algorithm that would patch pictures and save the patches to Google Drive. The patches themselves were small and could be loaded into memory without any problems. Two potential issues arouse. First, it took around two days to do it. Second, once done we were stuck with those patches and had to do the best we could with them. We really did not have a choice. During the project we did this twice. First time we were laid back and filtered the whitespace using Otsu's algorithm with a high threshold letting patches with questionable amounts of whitespace through. This lead to us having around 130000 patches. The second time we did the patching we were a lot more strict with the threshold for whitespace and ended up with around 37000 patches.

B.3 Tissue filtration

We initially assumed it would be enough to just filter out the whitespace from each WSI in the patching process, and then let the CNN work out what is what in the patches by itself. However, that proved to be insufficient amount of pre-processing. The CNN learnt nothing from that input since both classes contained large amounts of irrelevant tissue (blood, supportive tissue, fat) compared to actual tumorous growth tissue. So we figured we needed to expand the filtration process and identified, with our own eyes (instructed by Renata, a pathology student), the parts of the patches that did not contain any cell nuclei or their surroundings as irrelevant tissue that the CNN has no use of in the training phase. So we started working on producing a tissue filter that would make sure that we only use patches with at least a certain amount of relevant tissue. That is when we created the color filter, which at first focused on different shades of purple. It was however ineffective since in some images, the shades of purple that the filter focused on was the color of the relevant parts, meanwhile in other images that particular color was in fact the color of the irrelevant parts. Luckily, we were tipped by Renata to look at a color normalization algorithm for cell scans similar to ours that existed online. After implementing it on our patches, we found that the resulting patches all had the same shades of the color blue on the relevant parts of the tissue. This helped us finalize the tissue filter and using two versions of the same filter mask, the second one being a dilated version of the first, we managed to only focus on the patches that contained a high amount of relevant tissue. The resulting dataset from this filtering process was now reduced to around 3600 patches from the 360 WSIs. This resulted in some improvement in the training but the model was still not accurate enough. Given that we had chosen highly specific patches to train on also made us believe that this model would not do well on a more general dataset. Renata also pointed out that we still had patches that contained too much non-tumorous tissue. It would probably have been better to feed the CNN with only tumorous tissue since the challenge is to distinguish between two types of cancer. This was however not possible for us to come up with and implement during the time frame of this project. This was nearing the end of the project and therefore we did not have time to research further about methods and techniques that could be useful.

C CODE

C.1 Patching Code

```
# -*- coding: utf-8 -*-
"""patch.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1jpPU7qCTV1z68rYqkqlpVDyxRQGc70q7

# Mount Drive
"""

from google.colab import drive
drive.mount('/content/drive/', force_remount=True)

"""# Utils"""

import cv2
import pandas as pd
import glob
import ntpath
import numpy as np
import os

X_PIXEL = 299
Y_PIXEL = 299

def get_paths():
    return glob.glob("/content/drive/My Drive/Colab Notebooks/PCS-slides/*.tif")

def get_labels():
    return pd.read_csv("/content/drive/My Drive/Colab Notebooks/her2_labels")

def import_slide(path):
    img = cv2.imread(path, cv2.IMREAD_COLOR)
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGBA2GRAY)
    th, ret = cv2.threshold(gray_img,0,255,cv2.THRESH_OTSU)
    return img, gray_img, ret, th

def count_threshold_pass(patches, th):
  sum = 0
  for p in patches:
    if p[1] < th:
      sum += 1
  return sum

def filter_patch(img):
  hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
  mask1 = cv2.inRange(hsv, (170, 2.55*50, 2.55*0), (180, 2.55*100, 255)) #red filter
  mask = mask1
  res = cv2.bitwise_and(hsv,hsv, mask=mask)
  res = cv2.cvtColor(res, cv2.COLOR_HSV2RGB)

  # Calculate how much of the res is blackspace
  res_rows, res_cols, res_depth = res.shape
  blk_pxl_ctr = 0
  for i in range(res_rows):
    for j in range(res_cols):
          if res[i,j,0] == 0 and res[i,j,1] == 0 and res[i,j,2] == 0 : blk_pxl_ctr += 1

  not_blood = blk_pxl_ctr / (299*299)
  return not_blood

def patch_image(img, ret, th):
  patches = []
  for x in range(0, img.shape[0], X_PIXEL):
    for y in range(0, img.shape[1], Y_PIXEL):
      if x+X_PIXEL < img.shape[0] and y+Y_PIXEL < img.shape[1]:
        if np.sum(ret[x:x+X_PIXEL, y:y+Y_PIXEL])/(X_PIXEL*Y_PIXEL) < th*0.2 :
          patch = img[x:x+X_PIXEL, y:y+Y_PIXEL]
          if filter_patch(patch) > 0.98:
            patches.append( (patch, 0, (x, y)) )

  #patches.sort(key=take_second)
  return patches

"""# Run Patching"""

paths = get_paths()
labels = get_labels()
labels = labels.set_index("Case")
for path in paths[74:]:
    print("Patching for path: " + path)
    file_name = ntpath.basename(path)
    label_i = file_name[:len(file_name)-4]
    label = labels.loc[int(label_i)]["HER2 Status"]
    print("Label is: " + label)
    img, gray_img, ret, th = import_slide(path)
    patches = patch_image(img, ret, th)
    print(len(patches))
    if label == "Positive":
        for i, patch in enumerate(patches):
            p = "/content/drive/My Drive/Colab Notebooks/Patches/Pos/" + str(label_i) +
    "_"+ str(i) + ".tif"
            cv2.imwrite(p, patch[0])
    else:
        for i, patch in enumerate(patches):
            p = "/content/drive/My Drive/Colab Notebooks/Patches/Neg/" + str(label_i) +
    "_"+ str(i) + ".tif"
            cv2.imwrite(p, patch[0])
```

patch.py

C.2 Training Code

```
# -*- coding: utf-8 -*-
"""STrain.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1nCEXhE9amqvEBUa97AFnZkLxmXsWrYD2

# **Breast cancer HER2 positive/negative classification using Deep Learning**

## **Mount drive**
"""

from google.colab import drive
drive.mount('/content/drive/', force_remount=True)

```

```python
"""## **Imports, global variables and utils functions**"""

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 2.x

import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
from PIL import Image
import glob
import ntpath
import numpy as np
import tensorflow as tf
import pandas as pd
import random as r
from tensorflow.keras.applications import Xception
from tensorflow.keras.applications.resnet_v2 import ResNet152V2
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D, Flatten, MaxPooling2D
    , Conv2D, Dropout
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.python.framework.ops import disable_eager_execution
import tensorflow.keras.regularizers

disable_eager_execution()


X_PIXEL = 296
Y_PIXEL = 296

path = "/content/drive/My Drive/Colab Notebooks/PCS-slides/500.tif"
LABEL_PATH = "/content/drive/My Drive/Colab Notebooks/her2_labels"

print(tf.__version__)

"""## **Download and modify Xception**"""

base_model = Xception(include_top=False, weights='imagenet', input_shape=(X_PIXEL,
    Y_PIXEL,3))

for layer in base_model.layers[:]:
    layer.trainable=False
for layer in base_model.layers[129:]:
    layer.trainable=True

"""## **Model specification**"""

from keras import regularizers
from keras import metrics

ru = tf.keras.initializers.RandomUniform(minval=-0.05, maxval=0.05, seed=None)

model=Sequential()
model.add(base_model)
model.add(GlobalAveragePooling2D())
model.add(Dropout(.8))
model.add(Dense(128,activation='relu', kernel_regularizer=regularizers.l2(0.001),
    activity_regularizer=regularizers.l1(0.001)))
model.add(Dropout(.5))
model.add(Dense(1, activation="sigmoid"))

adam = Adam(learning_rate=0.00005, beta_1=0.9, beta_2=0.999, amsgrad=False)

model.summary()
model.compile(optimizer='Nadam',loss='binary_crossentropy', metrics = [metrics.
    binary_accuracy])

"""## **Training discriminative network**
Discriminative = A patch has the same label as the whole slide

**Put into console to run for 12h (CTRL+SHIFT+I -> console)**

function ClickConnect(){
console.log("Working");
document.querySelector("colab-toolbar-button#connect").click()
}
setInterval(ClickConnect,60000)
"""

from keras.callbacks import EarlyStopping, ModelCheckpoint

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=3)
mc = ModelCheckpoint('/content/drive/My Drive/Colab Notebooks/best_model2.h5', monitor=
    'val_loss', mode='min', save_best_only=True)

aug = ImageDataGenerator(horizontal_flip=True,
                         vertical_flip=True,
                         rotation_range=180,
                         rescale=1./255,
                         width_shift_range=0.1,
                         height_shift_range=0.1)

EPOCHS = 100
BATCH_SIZE = 92
STEPS_EPOCH = 32

train_generator = aug.flow_from_directory('/content/drive/My Drive/Colab Notebooks/
    Patches/Data/Train',
                                          target_size=(X_PIXEL, Y_PIXEL),
                                          batch_size=BATCH_SIZE,
                                          class_mode='binary')
validation_generator = aug.flow_from_directory('/content/drive/My Drive/Colab Notebooks
    /Patches/Data/Validation',
    target_size=(X_PIXEL, Y_PIXEL),
    batch_size=BATCH_SIZE,
    class_mode='binary')

model.fit_generator(train_generator, steps_per_epoch=STEPS_EPOCH, validation_data =
    validation_generator, validation_steps = 32, epochs=EPOCHS, callbacks=[es,mc])

"""# Load Saved Model"""

from tensorflow.keras.models import load_model

model = load_model('/content/drive/My Drive/Colab Notebooks/best_model.h5') #<---------
    Choose which model to continue train ##############

model.fit_generator(train_generator, steps_per_epoch=STEPS_EPOCH, validation_data =
    validation_generator, validation_steps = 32, epochs=EPOCHS)
```

strain.py

C.3 Blood Filter

```
# -*- coding: utf-8 -*-
"""ignore.ipynb

Automatically generated by Colaboratory.

```

```
Original file is located at
    https://colab.research.google.com/drive/1agBC24VA23tyw71dVKj7VwF3evUEtaKw

# **Breast cancer HER2 positive/negative classification using Deep Learning**

## **Mount drive**
"""

from google.colab import drive
drive.mount('/content/drive/', force_remount=True)

"""# Get Data"""

TrainNegPaths = glob.glob('/content/drive/My Drive/Colab Notebooks/Patches/Train/Negative/*.tif')
print("TrainNeg: " + str(len(TrainNegPaths)))

TrainPosPaths = glob.glob('/content/drive/My Drive/Colab Notebooks/Patches/Train/Positive/*.tif')
print("TrainPos: " + str(len(TrainPosPaths)))

ValNegPaths = glob.glob('/content/drive/My Drive/Colab Notebooks/Patches/Validation/Negative/*.tif')
print("ValNeg: " + str(len(ValNegPaths)))

ValPosPaths = glob.glob('/content/drive/My Drive/Colab Notebooks/Patches/Validation/Positive/*.tif')
print("ValPos: " + str(len(ValPosPaths)))

"""# Red Color Filter"""

from skimage.morphology import erosion, dilation, opening, closing, white_tophat
from skimage.morphology import black_tophat, skeletonize, convex_hull_image
from skimage.morphology import disk, square
from IPython.display import clear_output
def plot_comparison(original, filtered, filter_name):

    fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12, 6), sharex=True,
                                   sharey=True)
    ax1.imshow(original, cmap=plt.cm.gray)
    ax1.set_title('original')
    ax1.axis('off')
    ax1.set_adjustable('box')
    ax2.imshow(filtered, cmap=plt.cm.gray)
    ax2.set_title(filter_name)
    ax2.axis('off')
    ax2.set_adjustable('box')

def filter_patch(path):
  img = cv2.imread(path)

  gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
  img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

  mask1 = cv2.inRange(hsv, (170, 2.55*50, 2.55*0), (180, 2.55*100, 255)) #red filter
  mask = mask1

  res = cv2.bitwise_and(hsv,hsv, mask=mask)
  res = cv2.cvtColor(res, cv2.COLOR_HSV2RGB)

  # Calculate how much of the res is blackspace
  res_rows, res_cols, res_depth = res.shape
  blk_pxl_ctr = 0
  for i in range(res_rows):
    for j in range(res_cols):
            if res[i,j,0] == 0 and res[i,j,1] == 0 and res[i,j,2] == 0 : blk_pxl_ctr += 1

    trash_quota = blk_pxl_ctr / (299*299)
    plot_comparison(img, res, 'filtered')
    print(trash_quota*100,'% of the res is not blood')

filter_patch('/content/drive/My Drive/Colab Notebooks/Patches/Test/Negative/11.tif_287.
    tif')

#!bash -c 'ls /content/drive/My\ Drive/Colab\ Notebooks/Patches/Filtered'
!mv '/content/drive/My Drive/Colab Notebooks/Patches/Train/Positive/57.tif_79.tif' '/
    content/drive/My Drive/Colab Notebooks/Patches/Filtered'
```

ignore.py

C.4 Tissue Filter

```
# -*- coding: utf-8 -*-
"""dark_filter.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1TeDY1sG4hq7QJqQUF1VcCFDT2NF6r6b_

# Mount drive
"""

from google.colab import drive
drive.mount('/content/drive', force_remount=True)

"""# Imports"""

import numpy as np
import cv2
import os
import glob
import random as r
from os import listdir, walk
from os.path import isfile, join
from google.colab.patches import cv2_imshow

"""# Set color ranges"""

lower_red = np.array([150,150,50])
upper_red = np.array([180,255,150])

# This color range should be good enough:
lower_purple = np.array([100,75,100])
upper_purple = np.array([304,166,170])

lower_dark_pink = np.array([160,150,0])
upper_dark_pink = np.array([170,255,255])

lower_pink = np.array([145,150,0])
upper_pink = np.array([155,255,255])

lower_cyan = np.array([85,150,0])
upper_cyan = np.array([95,255,255])


#lower_dark_blue = np.array([115,150,0])
lower_dark_blue = np.array([115,150,0])
upper_dark_blue = np.array([135,255,255])
#upper_dark_blue = np.array([140,255,255])
```

```python

lower_black = np.array([0,0,0])
upper_black = np.array([250,255,30])

"""# Patch filter"""

def patch_filter(FILE_NAME, SRC_FOLDER, REG_FOLDER, DEST_FOLDER, DENIED, RET_FLAG=False):
  kernel = np.ones((5,5),np.uint8)

  # Load the patch and convert it to HSV
  patch = cv2.imread(join(SRC_FOLDER,FILE_NAME), cv2.IMREAD_COLOR)
  patch_in_hsv = cv2.cvtColor(patch, cv2.COLOR_BGR2HSV)

  # Dark blue mask
  mask = cv2.inRange(patch_in_hsv, lower_dark_blue, upper_dark_blue)

  #mask += cv2.inRange(patch_in_hsv, lower_purple, upper_purple) # Purple mask added
  dilation = cv2.dilate(mask,kernel,iterations=1)
  clean_mask = cv2.bitwise_and(patch, patch, mask=mask)
  clean_dilation = cv2.bitwise_and(patch, patch, mask=dilation)

  # Calculate how much of the patch has been cleaned up
  rows, cols, depth = clean_dilation.shape
  blk_pxl_ctr_mask = 0
  blk_pxl_ctr_dilation = 0
  for i in range(rows):
    for j in range(cols):
        for d in range(depth):
          if clean_mask[i,j,d] == 0: blk_pxl_ctr_mask = blk_pxl_ctr_mask + 1
          if clean_dilation[i,j,d] == 0: blk_pxl_ctr_dilation = blk_pxl_ctr_dilation + 1

  quota_mask = float(blk_pxl_ctr_mask) / float(clean_mask.size)
  quota_dilation = float(blk_pxl_ctr_dilation) / float(clean_dilation.size)

  if quota_dilation < 0.52 and quota_mask < 0.85:
    print(FILE_NAME)
    print('quota_mask:', str(round(quota_mask*100,2)),"%,",'quota_dilation:', str(round(quota_dilation*100,2)),"%")
    #cv2.imwrite(join(DEST_FOLDER,FILE_NAME), patch)
    # Regular, un-normalized patch saved below
    try:
      reg_patch = cv2.imread(join(REG_FOLDER,FILE_NAME), cv2.IMREAD_COLOR)
      cv2.imwrite(join(DEST_FOLDER,FILE_NAME), reg_patch)
    except:
      print("Could not write file.")

  else:
    DENIED.append('{}: {} <- quota_mask: {}%, quota_dilation: {}%'.format(len(DENIED)+1, FILE_NAME, str(round(quota_mask*100,2)), str(round(quota_dilation*100,2))))

  if RET_FLAG:
    return clean_mask

"""# Run on data"""

def filter_both():
  denied_patches = []
  src_FOLDERS = ['NormPos/','NormNeg/']
  reg_FOLDERS = dest_FOLDERS = ['Pos/','Neg/']
  for j in range(0,2):
    SRC_FOLDER = join('/content/drive/My Drive/Colab Notebooks/Patches/', src_FOLDERS[j])
    REG_FOLDER = join('/content/drive/My Drive/Colab Notebooks/Patches/', reg_FOLDERS[j])
    DEST_FOLDER = join('/content/drive/My Drive/Colab Notebooks/HistopathologyStainColorNormalization/Out/Filtered_Original/', dest_FOLDERS[j])
    paths = glob.glob(join(SRC_FOLDER, '*.tif'))
    print('Working on',src_FOLDERS[j],'...')
    for i in range(0, 15000):
      path = r.choice(paths)
      patch_filter(path[len(SRC_FOLDER):], SRC_FOLDER, REG_FOLDER, DEST_FOLDER, DENIED=denied_patches)

    denied_log = open(join(DEST_FOLDER,'denied_patches.txt'),'w')
    for dp in denied_patches:
      denied_log.write(dp+'\n')
    print('For',src_FOLDERS[j],':',len(denied_patches),'patches were denied, peep denied_patches.txt in',DEST_FOLDER,'for more info.')


def filter_specific():
  denied_patches = []
  SRC_FOLDER = join('/content/drive/My Drive/Colab Notebooks/Patches/', 'NormNeg/')
  REG_FOLDER = join('/content/drive/My Drive/Colab Notebooks/Patches/', 'Neg/')
  DEST_FOLDER = join('/content/drive/My Drive/Colab Notebooks/HistopathologyStainColorNormalization/Out/Filtered_Original/', 'Neg/')

  paths = glob.glob(join(SRC_FOLDER, '*.tif'))
  denied_patches = []
  for i in range(0, 15000):
    path = r.choice(paths)
    patch_filter(path[len(SRC_FOLDER):], SRC_FOLDER, REG_FOLDER, DEST_FOLDER, DENIED=denied_patches)

  denied_log = open(join(DEST_FOLDER,'denied_patches.txt'),'w')
  for dp in denied_patches:
    denied_log.write(dp+'\n')
  print('For NormNeg:',len(denied_patches),'patches were denied, peep denied_patches.txt in',DEST_FOLDER,'for more info.')

filter_specific()
#detect_cells(FILE_PATH=path)
```

dark_filter.py