DEGREE PROJECT

Semantic Segmentation of Iron Pellets as a Cloud Service

Christopher Rosenvall

Computer Science and Engineering, master's level
2020

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Semantic Segmentation of Iron Pellets as a Cloud Service

Master's Thesis

In collaboration with, Data Ductus

Author:

Christopher Rosenvall christopher.rosenvall@gmail.com

Supervisors:

Martin Simonsson martin.simonsson@dataductus.se

Karan Mitra karan.mitra@ltu.se

Department of Computer Science, Electrical and Space Engineering

November 29, 2020

LULEÅ UNIVERSITY OF TECHNOLOGY

Abstract

Department of Computer Science, Electrical and Space Engineering

Master of Science in Computer Science and Engineering

Semantic Segmentation of Iron Pellets as a Cloud Service

By Christopher Rosenvall

This master's thesis evaluates automatic data annotation and machine learning predictions of iron ore pellets using tools provided by Amazon Web Services (AWS) in the cloud. The main tool in focus is Amazon SageMaker which is capable of automatic data annotation as well as building, training and deploying machine learning models quickly. Three different models was trained using SageMakers built in semantic segmentation algorithm, PSP, FCN and DeepLabV3. The dataset used for training and evaluation contains 180 images of iron ore pellets collected from LKAB's experimental blast furnace in Luleå, Sweden. The Amazon Web Services solution for automatic annotation was shown to be of no use when annotating microscopic images of iron ore pellets. Ilastik which is an interactive learning and segmentation toolkit showed far superiority for the task at hand. Out of the three trained networks Fully-Convolutional Network (FCN) performed best looking at inference and training times, it was the quickest network to train and performed within 1% worse than the fastest in regard to inference time. The Fully-Convolutional Network had an average accuracy of 85.8% on the dataset, where both PSP & DeepLabV3 was showing similar performance. From the results in this thesis it was concluded that there are benefits of running deep neural networks as a cloud service for analysis and management of iron ore pellets.

Acknowledgements

I would like to start of with a special thanks to my supervisors. Dr. Martin Simonsson and Dr. Karan Mitra for all your guidance, availablity and support during this thesis work. Thank you to all teachers and acquaintences I have met during my time at Luleå University of Technology, you have all helped me prepare for future challanges. A Thank you to Data Ductus who has hosted me at their offices during this time and welcomed me with joy. Lastly, I give a special thank you to Adam Hedkvist whom have provided great feedback during this report.

Contents

List of Figures

1 Introduction

1.1 Background

Characterization of microstructures in iron ore pellets are crucial for understanding the mechanical and chemical processes in the pellet and being able to optimize its properties for different applications. The characterization is often done manually in a microscope and is time-consuming and requires skilled staff with extensive experience. Nevertheless, the characterization is above all qualitative. SSAB, LKAB and Vattenfall have embarked on an industrial development project HYBRIT where they work for a sustainable and fossil-free steel production [1]. Within the project, they try to find solutions with a hydrogen-based iron production. With a so-called direct reduction hydrogen could be used to separate the iron from the oxygen, without using carbon. This places new demands on the design of iron ore pellets and thus increased demands on being able to automatically analyze and characterize them. During a previous thesis "Semantic Segmentation of Iron Ore Pellets with Neural Networks" by Terese Svensson[2] in collaboration with Data Ductus[3] and LKAB[4], good results were shown in training and using neural networks for semantic segmentation of iron ore pellets. We now want to go further and explore how automated analysis of iron ore pellets could be developed and offered as a cloud service.

1.2 Motivation

Data Ductus works with multiple complex projects in vision technology and uses machine learning to improve production quality, production speed and for preventive maintenance. Additionally, Data Ductus has access to a large amount of microscope images of iron ore pellets through Luossavaara-Kirunavaara AB (LKAB) [4]. These images are magnified many times over resulting in files of gigapixel size each to be handled.
The aim of this thesis is to build and evaluate a system with the ability to upload a magnified image of iron ore pellets to a cloud service. Then make use of machine learning to analyze that image and return a result to user.
Software as a Service (SaaS) is a cloud service type that provides software through the internet. Using SaaS it is possible to monitor and scale the system in respect to its performance and need. The main advantage of introducing a SaaS solution is to make the system more available and reduce cost of personnel.
Moreover a local implementation requires large amount of computer power available locally and a heavy cost to keep hardware up-to-date. Moving the service to the

cloud would expectantly reduce its costs as well as remove the occasional need for local maintenance with on sight support bringing higher availability.

1.3 Problem Definition

A cloud service for automatic analysis of iron ore pellets requires several functions that work together. Amazon Web Services (AWS) offers basic services, such as networking, storage and computing capabilities, but also specialized services for image annotation, neural network training and scalable server less deployment of trained networks. The aim of the project is to evaluate various components of a cloud service through prototypes and thus guide and provide valuable information for a final design. Components that needs to be evaluated are mainly:

- Upload and store image data
- Annotate image data
- Training of networks
- Performance of different networks
- Price, Performance and scalability for the alternatives

One challenge within the project is the size of the images to be analyzed. Microscopic structures should be imaged in macroscopic pellets, resulting in images over 1 gigapixel each. For a scalable solution, the image data must be efficiently divided so that the computational burden can be parallelized and spread over several computational instances, and then compiled for a complete analysis. Another challenge is that the development of neural networks goes very fast and new architectures and ideas are constantly emerging. In order to take advantage of this, one must be able to cost-effectively train and evaluate new networks on existing annotated data. This places demands on how annotated data is stored and how trained networks are version controlled.

1.4 Equality and Ethics

As most times when developing automatic procedures there is a chance someone is loosing their job because of it. On the positive side machinery and software takes no side regarding equality.

1.5 Delimitations

There exists several different cloud platforms and machine learning tools out there, AWS, Google, Azure, Nvidia and IBM are all big time players withing the field just to mention a few. AWS has the potential of offering a complete solution using solely their systems and Data Ductus is AWS certified partners, because of this it was decided that AWS was the platform to be evaluated for this project.
To create a more tailored solution with better performance it's needed to train and tune a network outside of Amazon SageMaker. The goal of this thesis has been to evaluate mainly AWS components including but not limited to SageMaker. To train and evaluate networks and services outside of this has been out of scope.
For a finished system the current implementation is in need of a more advanced graphical user interface for the end user and possibly some tweaks regarding how data uploaded and processed is stored. In addition to this, a database should be implemented to save results and to tie saved data together with its results for easy logging and display.

1.6 Thesis structure

In the next section we will go through the related work to this problem definition. We will in short explain semantic segmentation, cover the basics of data annotation and cloud computing. Then we will move on to section 3 and dive deeper into what is going on, possible solutions and how we are going to tackle the problems. Section 4 is presenting the materials and different methods the reader comes across during the report. In section 5 implementations are described and in section 6 evaluated. The thesis is then rounded off with discussion, conclusions and future work.

2 Related Work

There already exist some implementations to automatically classify pellets, mainly the studies have been carried out at Vale's mines in Brazil. Wagner et al [5] uses digital microscopes to both acquire and process iron ore pellets. It's done by creating porosity maps and measure phase fractions with automated segmentation. Their results provide a methodology for the evaluation of non-uniform materials, such as iron ore pellets. They deem it sufficient to acquire and process a low magnification image covering the whole sample surface to obtain qualitative and quantitative information about the sample.
Augusto et al [6] classified hematite in iron ore from optical microscope images, identifying textures and shapes with an automatic analysis procedure. The combined acquisition of different analysis techniques allowed for automatic classification of hematite in iron ore.
Castellanos et al [7] utilized the software FiJi to create a semi-automatic method to process their microscope images from the same mining company. Their results showed that the phases present in optical images of iron ore pellets can be identified based on their characteristics. However, there lies a limitation to differentiate some phases from each other under different circumstances.

2.1 Performance of networks

This thesis is built upon the work by Terese Svensson [2] where the most commonly used algorithms used in Semantic Segmentation were evaluated and thoroughly tested on the very same data I am to use in this thesis. It was concluded that different algorithms had different strengths identifying classes, unfortunately no algorithm was good at identifying all classes.

2.1.1 PSPNet

PSPNet was one of the best performing CNN models during dataset size experiments and with data augmentation it was the best and fastest. It performed almost 92% average accuracy on the validation dataset with slightly tuned learning rate. PSPNet excelled at identifying pore and magnetite classes.

2.1.2 FC-DenseNet56

FC-DenseNet56 was one of the best performing CNN models during dataset size experiements. It excelled at identyfing epoxy, wüstite and magnetic iron classes.

While FC-DenseNet56 was showing promising numbers it was still outperformed by PSPNet.

2.1.3 DeepLab V3+

DeepLab V3+ delivered mediocre results during the experiments but showed increasing performance when dataset size was increased. To its benefit it was one of the fastest CNN models to train regardless of size. However, it did not manage to compete as the same levels as PSPNet or FC-DenseNet56. DeepLab V3+ excelled at identifying the olivine class but at the same time worst at identifying hematite.

2.1.4 BiSeNet

BiSeNet is developed to be faster compared to other CNN models whilst not compromising significant performance. However, in these experiments it measured as one of the slowest algorithms. Additionally it was the worst of the CNN models to correctly identify pore, epoxy, magnetite, wüstite, magnetic iron, slag and olivine classes in the images.

2.1.5 GCN

GCN performed mediocre, comparable to DeepLab V3+ but was slower to train. It excelled at identifying hematite and slag classes and was doing fairly well with all others.

2.1.6 Best Model

Based on the results and discussion Terese [2] concluded that PSPNet was the best model for the task of classifying micro structures in iron ore pellets with use of semantic segmentation. PSPNet showed to be one of the best performing ones looking at net results as well as being the fastest one to train. It also showed great potential for improvement given more annotated data to train with.

2.2 Pixel classifier

Dr. Martin Simonsson at Data Ductus has developed a pixel classifier[8] that is able to classify the different characteristics in iron ore pellets. This is done using Ilastik to train a pixel classifier for the task. Ilastik makes use of classical classification methods, such as random forest. The solution is manual and requires more time

than necessary, staff has to manually load an image into Ilastik to run in through the trained pixel classifier. The outputted result can then be run through a couple of scripts developed to show statistics of composition.

3 Theory

3.1 Semantic segmentation

Semantic Image Segmentation is the task of classifying every pixel in an image into a predefined set of categories. It can be used to classify and interpret information from images in many different areas, from satellite images to microscopical scales. Essential areas include but are not limited to: autonomous vehicles and medical diagnostics. Because it helps computers and robots in general to create a context to their environment and aids doctors in diagnostics.
Other than semantic segmentation some widely known computer vision techniques are object detection, classification, localization and instance segmentation. These are of little interest in regard to this thesis but worth mentioning in respect to following picture.

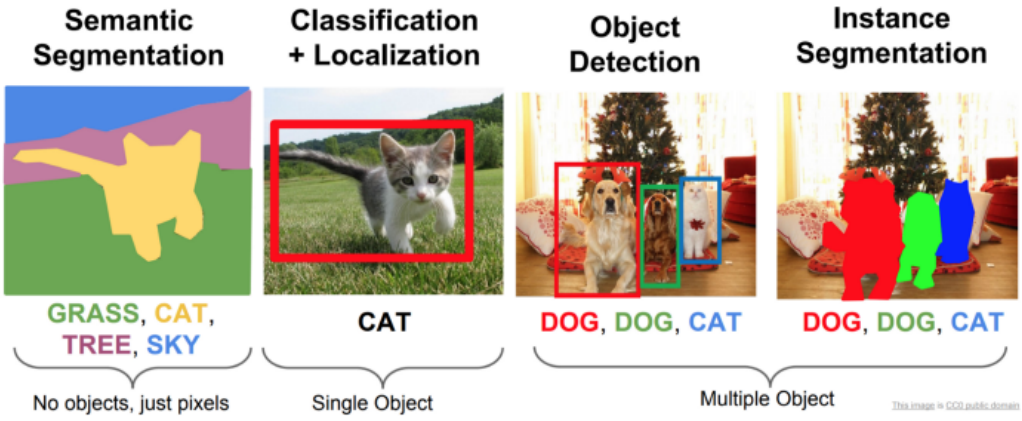


Figure 1: *Showing the differences between semantic segmentation, classification + localization, object detection & instanced segmentation.*
Licensed under: CCO Public Domain

In recent years Convolutational Neural Networks (CNN) based methods have had success in the area [9]. Some methods make use of classifying super-pixels [10], others such as Fully Convolutional Networks (FCN) classify pixels directly [11].

3.2 Neural Networks

3.2.1 Traditional Neural Networks

Multilayer perceptron (MLP) is moduled on the human brain. Neurons are stimulated by connected nodes and are only activated when a certain threshold value is reached. There are several drawbacks, biggest one concerning us is when it comes to image processing MLP use on perceptron for each input (pixel in an image, multiplied by its channels). The amount of weights rapidly becomes unmanagable for large images. For a 224x224 pixel RGB (3 channel) image there is $224 \times 224 \times 3 = 150528$ weights that must be trained. More than this MLP's have no way of handling location in the image. Lets say a bird appears in the bottom left corner of an image in one image and in the upper right in another. MLP will learn try to recognize birds in these parts of the images. If a bird were to occur in any other place of the image, the MLP would be unable to recognize it. In other words, MLP has no sense of 'where' something occurs in images.

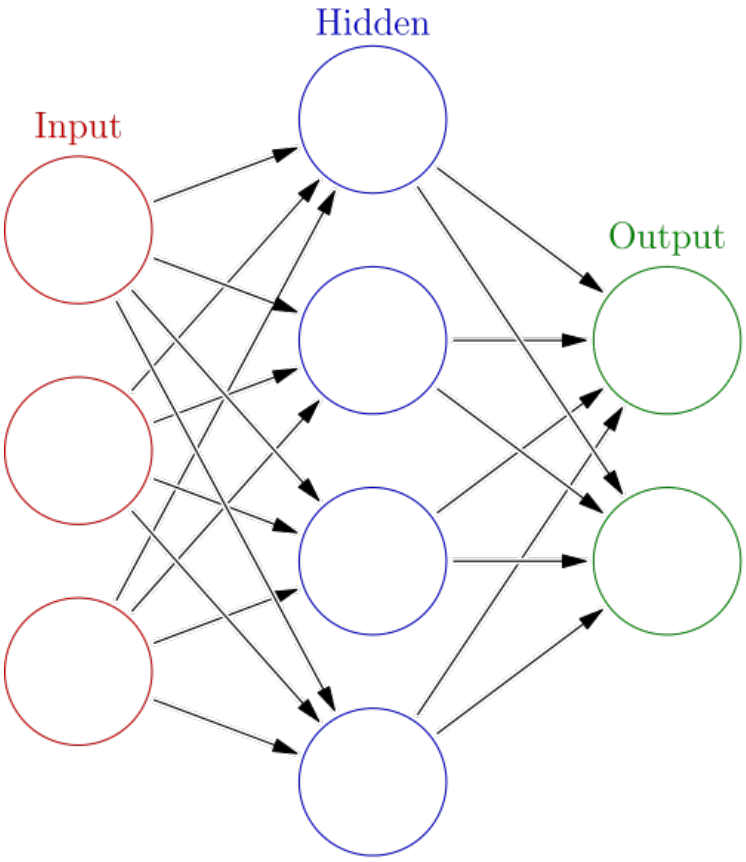


Figure 2: *A standard multilayer perceptron (traditional neural network).*

3.2.2 Convolutional Neural Networks

Convolutional networks, also known as ConvNets and convolutional neural networks, or CNNs, is a class of deep neural networks [9]. It is specialized in processing data that has a known grid-like topology like image data that can be viewed as a 2-D grid of pixels. The name indicates that the network employs the mathematical convolution operation. Convolution is a specialized kind of linear operation that replaces simple matrix multiplication performed by ANNs in at least one of the CNNs layers according to Goodfellow et al [12]. CNNs has been tremendously successful in practical applications and has an incredible impact in areas such as autonomous vehicles and medical image diagnostics.

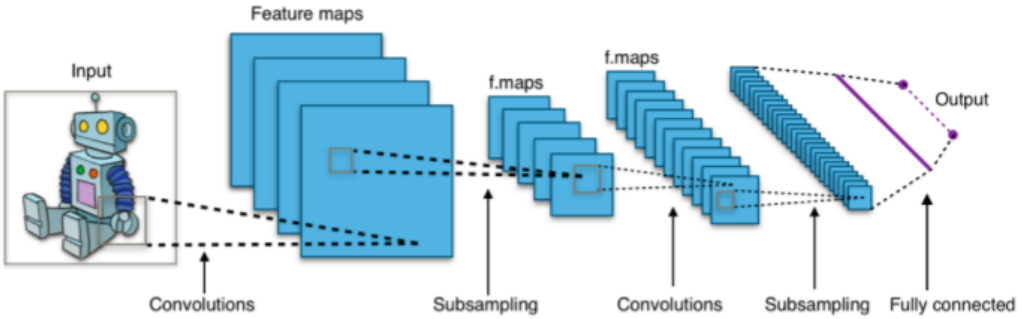


Figure 3: *A typical convolution neural network architecture.*
"Typical CNN architecture" by Aphex34 licensed under CC BY-SA 4.0

In each layer the neural network focuses on something, it could be shapes or structures in a specific region for example. It´s given a weight represented as a number between $[0 - \infty)$ where 0 represents no-match and a high value represents match. Then these pieces are added together in the last step and hopefully we get some neuron *weighted* $>> 0$ which would indicate a good match.
The weight is calculated using an activation function also known as transfer function. In early days the Sigmoid function was the dominant function but in recent years ReLU has taken over and is used in almost all convolutional neural networks or deep learning today.

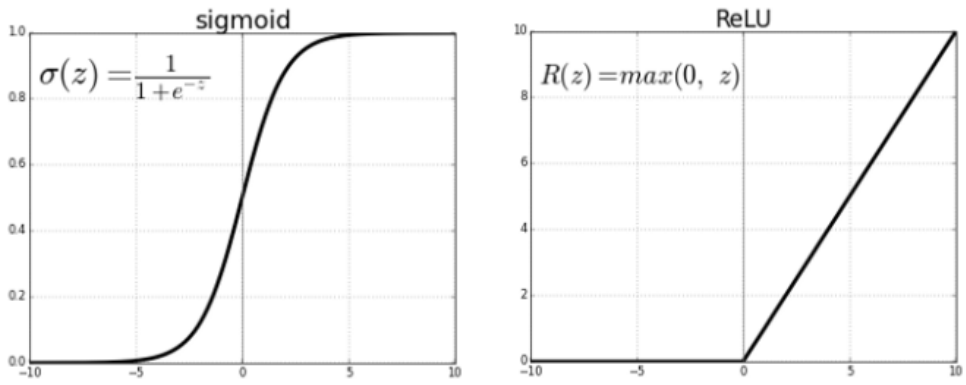


Figure 4: *Sigmoid vs. ReLU activation functions.*

The ReLU function is half rectified from the bottom. R(z) is is zero when z is less than zero and R(z) is equal to z when z is above equal or zero.
Gradient decent and batches, randomly subdivide the data into mini batches and compute each step in respect to a mini batch repeatedly going through all mini

batches making adjustments we will converge to a local minimum of the cost function. In other words, the network is going to do a really good job on the training examples. A limitation of CNNs is its fully connected layers with feature maps of fixed size. Image segmentation has input of different sizes and thus needs to be able to handle different sized inputs. This is solved by converting fully connected layers into fully convolutional ones. This make it possible to input images of any size and get corresponding predictions.

3.2.3 Fully-Convolutional Network (FCN)

The main idea behind FCN is to make the classical CNN take arbitrary-sized images as inputs, since CNNs are restricted to accept and produce labels only for specific sized inputs because of its fully-connected layers. FCNs consists of only convolutional and pooling layers which gives them the ability to make predictions on arbitrary-sized inputs [11].
Fully convolutional networks takes an input of size $W \times H \times D$ where W is width, H is height and D represents depth of an image. This input is ran through several convolution and max pooling layers applying the ReLU activation function down to 1/32 of its original size looking for higher level shapes and features. Then class prediction takes place and upsampling using billinear interpolation back to the same size as ground truth, followed by normal convolution to get the tensor output.

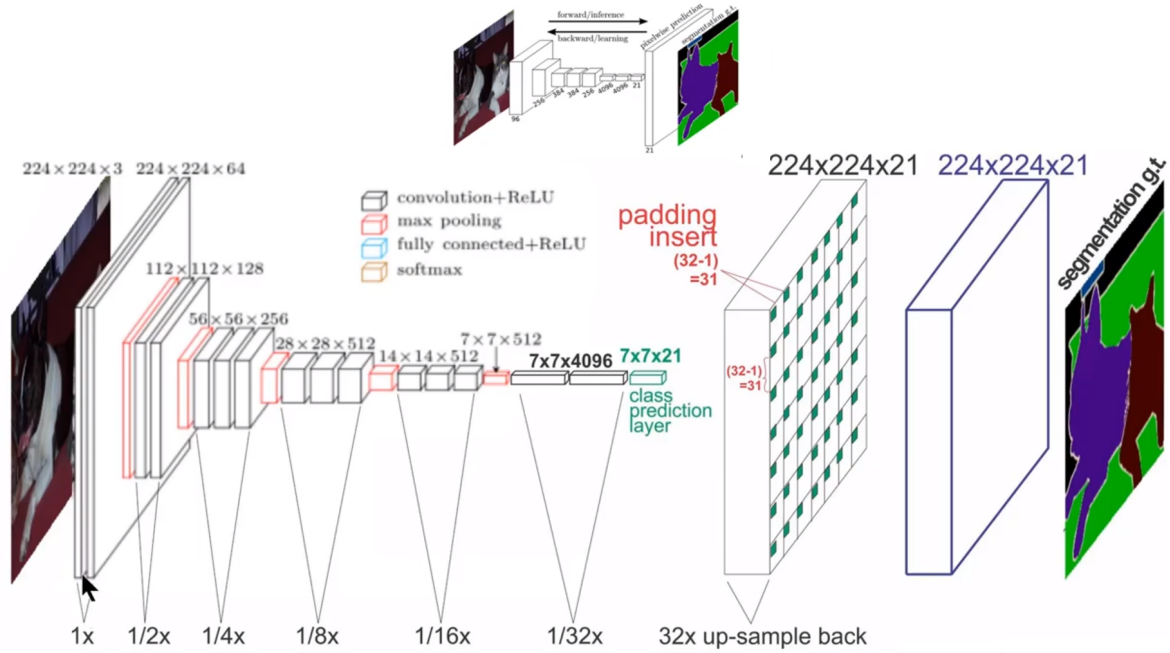


Figure 5: *Augmented Image from FCN paper [11] showing FCN structure.*
(Thanks to Ardian Umam for the image.)

Presented above is an example ran with an input of size 224×224 in 3 channels (RGB) showing how it makes its way through the convolutional and pooling layers down to $7 \times 7 \times 21$ parts (21 due to the example using the pascal VOC 2012 dataset consisting of 21 classes). Then it is up-sampled back using billinear interpolation to a tensor of size $224 \times 224 \times 21$ same size as the input but now with 21 depth representing its classes.
One issue with FCNs is that by propagating through several alternated convolutional and pooling layers, the resolution of the output feature maps is down sampled. Therefore, the direct predictions of FCNs are typically in low resolution, resulting in relatively fuzzy object boundaries.

3.2.4 Pyramid Scene Parsing (PSP)

The PSPNet architechture considers the global context of an image to predict the local level predictions thus giving better performance [13]. The model came to life because Fully-Convolutional Network based pixel classifiers were not able to take the context of the whole image into account.

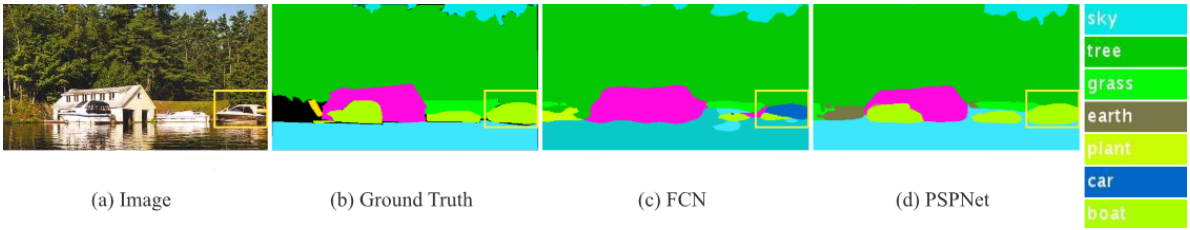


Figure 6: *Image comparing FCN and PSPNet [13].*

The boat on the right hand side of image 6 (a) is classified as a car by the Fully-Convolutional Network (FCN) (c). This happens because its shape and apperance resembles a car, but its rare to see a car in a river. If the model were to have contextual information, for example taking the water around the object into the classification, it would be able to correctly classify it. Pyramid Scene Parsing (PSP) is able to capture the context of the whole image and correctly classify the object as a boat (d).

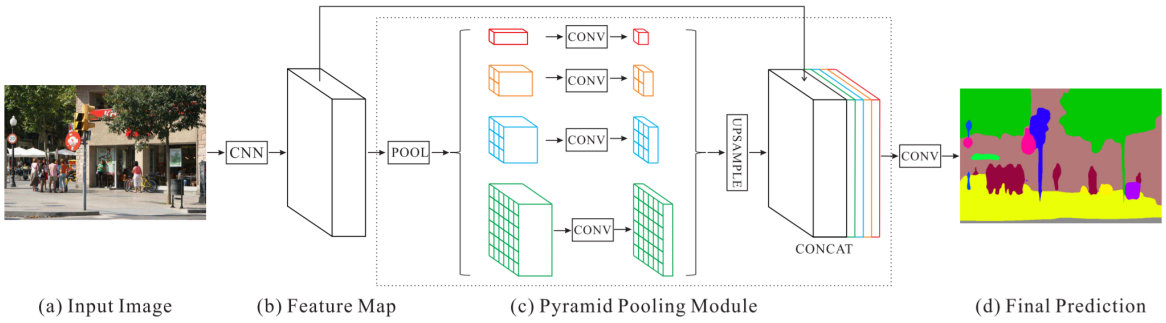


Figure 7: *Pyramid Scene Parsing (PSP)Net architecture [13].*

Looking at PSP architecture [13] in 7, an image is supplied as input (a) to get the feature map of the last convolutional layer (b). A pyramid parsing module is then applied to harvest different sub-region representations, followed by upsampling and concatenating the layers to form the final feature representation, which carries both global and local context information (c). As a last step the representation is fed into a convolution layer to get the final prediction (d).

3.2.5 DeepLab V3

As mentioned in 3.2.3 one challenge with using Fully-Convolutional Networks (FCNs) on images for segmentation tasks is that input feature maps become smaller while

traversing through the convolutional & pooling layers of the network. This ends up causing loss of information about the images and results in output where predictions have low resolution and object boundaries become fuzzy. The model addresses this challenge by using Atruous convolusions and Atrous Spatial Pyramid Pooling (ASPP) modules [14]. The DeepLab architecture has evolved over time and several generations:

- **DeepLabV1:** Uses Fully Connected Conditional Random Field (CRF) and Atrous Convolution to control the resolution at which image features are computed.
- **DeepLabV2:** Uses Conditional Random Field (CRF) and Atrous Spatial Pyramid Pooling to consider objects at different scales and segment with much improved accuracy.
- **DeepLabV3:** DeepLabV3 uses an improved ASPP module by including batch normalization and image-level features in addition to Atrous Convolution. It gets rid of CRF from V1 and V2.

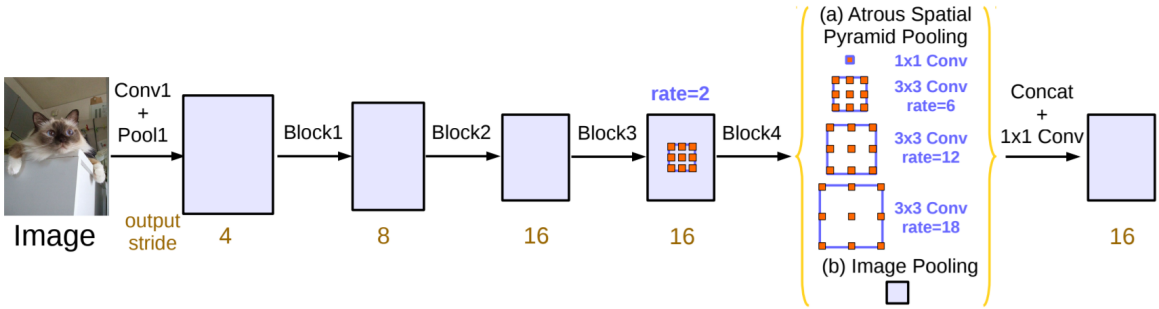


Figure 8: *DeepLabV3 architecture [14].*

Looking at DeepLabV3's architecture in 8, features are first extracted from the backbone network (VGG, DenseNet, ResNet). Atrous convolution is then applied in the last few blocks of the backbone to control the size of the feature map. On top of extracted features from the backbone, an ASPP network is added to classify each pixel corresponding to their classes. The output from the ASPP network is then passed through a 1×1 convolution to get the actual size of the image which will be the final segmented mask.
These improvements aid in extracting dense feature maps for long-range contexts. It increased the receptive field exponentially without reducing or losing the spatial dimension and improves the performance of the segmentation tasks.

Atrous Convolution Atrous Convolution can be viewed as a tool to adjust or control the effective field-of-view of the convolution. It is a simple and powerful technique to make field of view filters larger without increasing computation or the number of parameters. The main difference between traditional convolution and Atrous convolution is that the upsampling is done inserting zeros between two successive filter values along each spatial dimension.

3.2.6 Pros & Cons

Traditional neural networks (MLPs) and convolutional neural networks (CNNs) is just stepping stones in this context and will not be included in this section. Regarding which is the theoretically best option from the remaining three alternatives (FCN, PSP, DeepLab) will be covered next:

Fully-Convolutional Network and DeepLab has no concept of context compared to Pyramid Scene Parsing. FCN is also known for down-sampling the resolution of the output providing possibly fuzzy object boundaries. DeepLab introduces Atruous convolusions and Atrous Spatial Pyramid Pooling which helps aiding the problem FCN has with down-sampling. This puts DeepLab and PSP at a slight theoretical advantage compared to FCN. Which alternative is best suited for the task of segmenting iron ore pellets is to be determined in this thesis.

3.3 Metrics

3.3.1 Per Pixel Accuracy

This metric is self explanatory, since it outputs the class prediction accuracy per pixel. See equation 1.

$$acc(P, GT) = \frac{|\text{pixels correctly predicted}|}{|\text{total number of pixels}|} \tag{1}$$

3.3.2 Jaccard (Intersection over Union, IoU)

Jaccard is a per class evaluation metric. It computes the number of pixels in the intersection between the predicted and ground truth segmentation maps for a given class, divided by the number of pixels in the union between those two segmentation maps. See equation 2.

$$jacc(P(class), GT(class)) = \frac{|P(class) \cap GT(class)|}{|P(class) \cup GT(class)|} \tag{2}$$

Where P is the predicted segmentation map and GT is the ground truth segmentation map. P(class) is then the binary mask indicating if each pixel is predicted as class or not. As a general rule, the closer to 1 this metric is, the better.

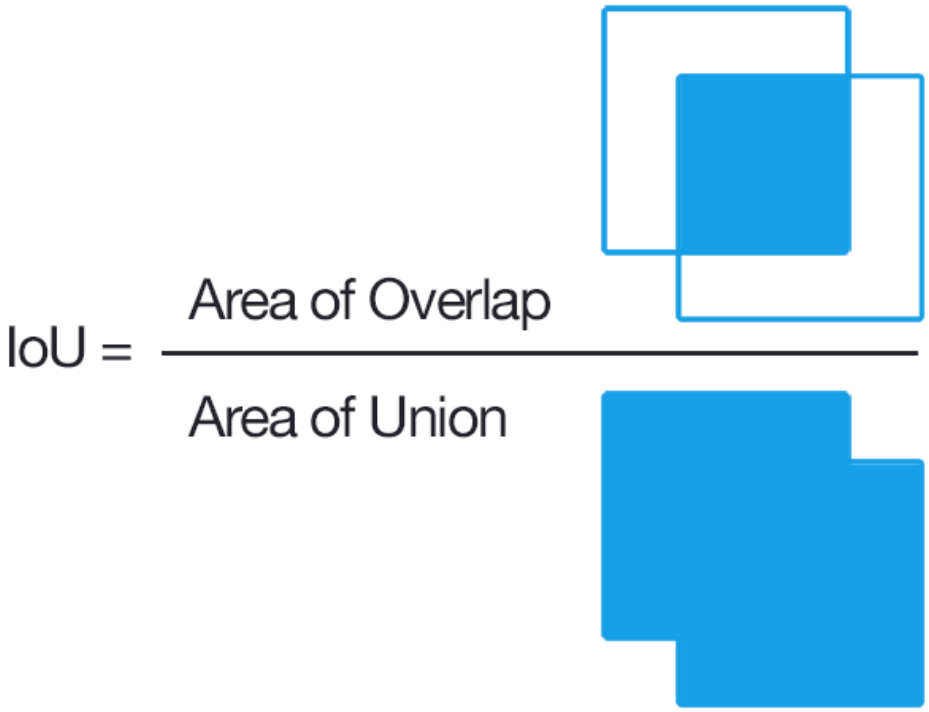
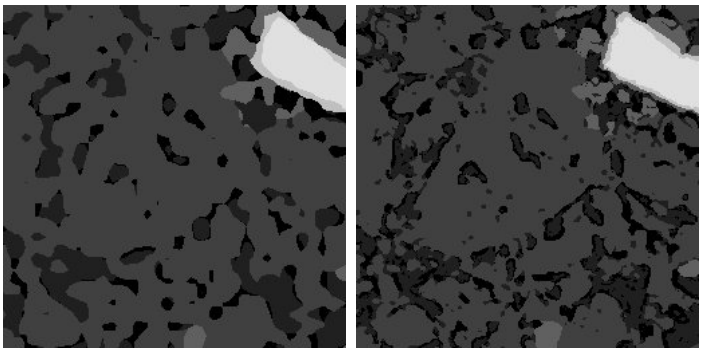


Figure 9: *Jaccard visualisation*
(Adrian Rosebrock @ pyimagesearch.com CC BY-SA 4.0.)

Taking a closer look at the equation 2 you quickly realise that Intersection over Union is simply a ratio.
The numerator computes the area of overlap between the predicted bounding box and the ground truth bounding box. The denominator is the area of union, more simply put the area encompassed by both the predicted bounding box and the ground truth bounding box.
Dividing the area of overlap by the area of union yields the final score, the Intersection over Union (IoU).

3.3.3 Confusion Matrix (CM)

A Confusion Matrix is commonly used to describe the performance of a classification model on a set of test data where there exists a ground truth to compare it to. For this thesis, 8 classes has been used to represent the composition of iron ore pellets. Each row in the matrix represents the instance in an actual class while each column represents the instance in a predicted class. As the name of the matrix suggests this makes it easy to see if the system is confusing and miss labeling two classes.



(a) *Predicted* ***(b)*** *Actual*

Figure 10: *In-data for confusion matrix explanation.*

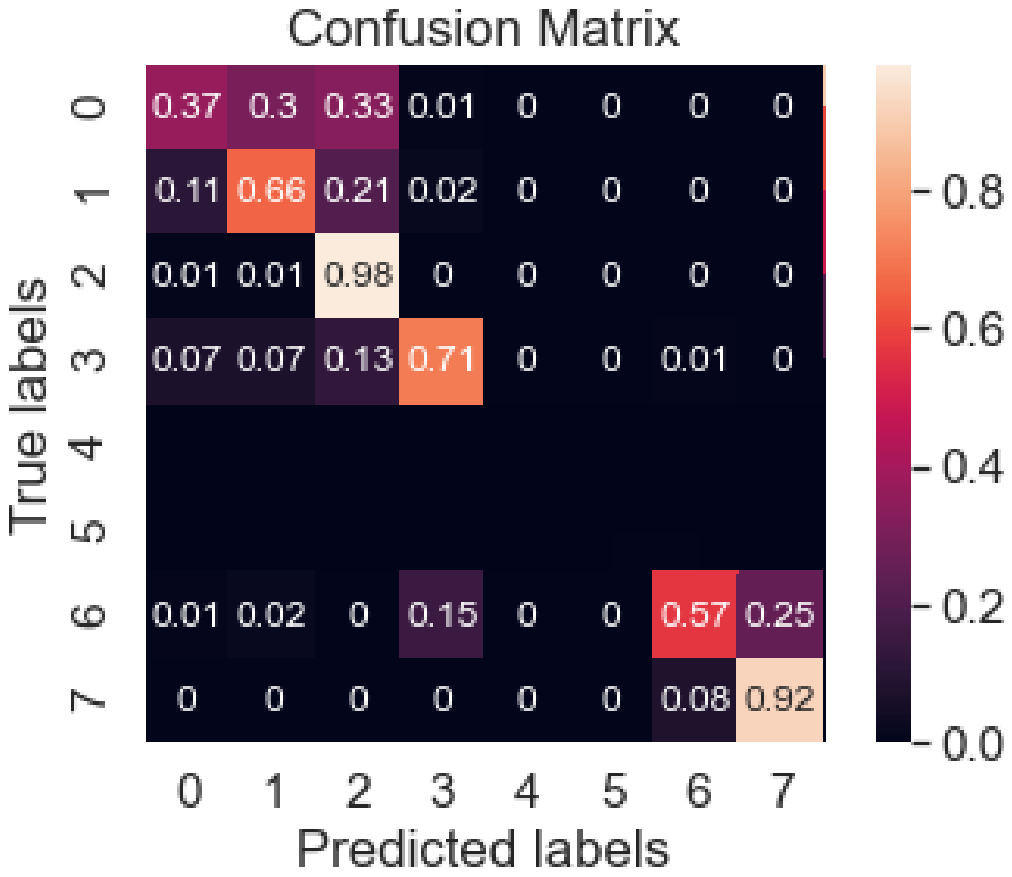


Figure 11: *Confusion Matrix of figure 10*

The confusion matrix in figure 11 was created using the data in figure 10. This example in particular happen to be missing classes 4 & 5. The main diagonal of the matrix is representing how many % of each class that was correctly classified, other values are showing errors, how predictions were confused. Inspecting the matrix more thoroughly it's possible to draw conclusions of what the network has problems predicting and what is being mixed up.

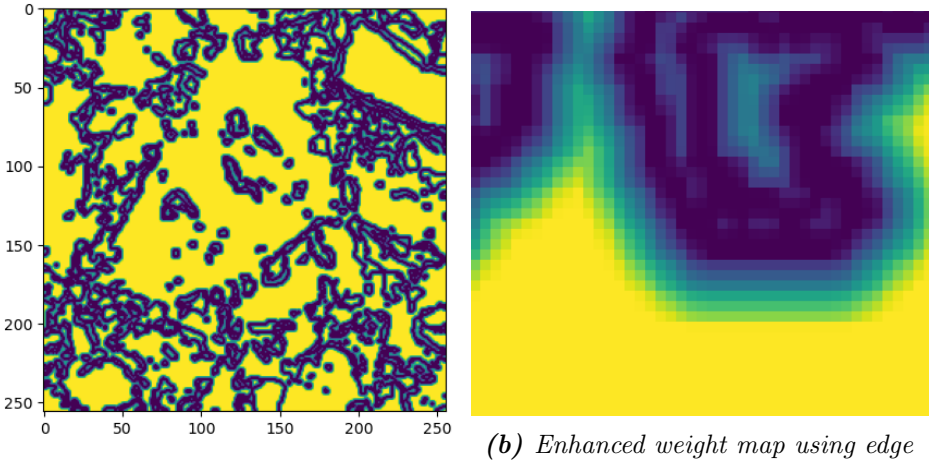
3.3.4 Distance Transform (DT)

Distance Transform or Euclidean Distance Transform (EDT), does in short describe the distance from a point within an object to its border. It essentially takes a binary image as input and outputs a distance map (Euclidean Distance). The distance map

has the exact same dimensions of the input and each pixel contains the distance to the closest border.

3.3.5 Weighted Confusion Matrix

A weighted confusion matrix is achieved by giving a confusion matrix a weight map along with its usual in-data, the weight map is created using Distance Transform. In the weight map another variable known as edge limit is introduced which is the threshold for how many pixels deviation is to be considered.



(a) *Weight map using edge limit 3*

(b) *Enhanced weight map using edge limit 6 to better show the gradients*

Figure 12: *Visual illustration of weight map.*

3.3.6 Pixel Evaluation

This is the most basic type of metric, to count the occurrence of each pixel (class) in both the predicted and ground truth data sets and then normalize it and evaluate the deviations.

3.4 Transfer Learning

Training deep neural networks from scratch is often not feasible because of various reasons: a dataset of sufficient size is required and not usually available along with reaching convergence taking too long for the experiment to be worth. Even if these things would be resolved it's often helpful to start with pre-trained weights compared to random initialized ones. Fine-tuning weights of a pre-trained network by continuing with the training process is one of the major transfer learning objectives [15].

3.5 Data Annotation

Data annotation is the task of labeling data, this thesis covers semantic segmentation, focus will be on image labelling but it could just as well be formats like text or video. In supervised machine learning labeled data is required in order to teach the machine to understand input patterns. Since dealing with vision based machine learning its important that the data is precisely annotated otherwise it might cause poor results. After processing enough annotated data the network can start to recognize the same patterns when presented with new, unannotated data.
The most common solution to annotating data for semantic segmentation is to use a pen tool to carefully outline the object. There is also tools and businesses focusing on automated annotation, this way you could get help annotating your data.

3.6 Cloud Computing

AWS provides the following short definition of cloud computing [16]:

> Cloud computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider like Amazon Web Services (AWS).

There is also a common analogy to pets and cattle when speaking about benefits of the cloud. In a non-cloud-service way of doing things you manually deploy and configure servers. It could take a very long time before a server is actually up and running when you have to wait for it to be physically delivered and setup. You could compare this to a pet, each server is unique and requires a lot of maintenance, some do even have names. In the cloud way of doing things servers are commodity resources that can be automatically provisioned in seconds. No single server should be essential to the operation of the service, like cattle.

The National Institute of Standards and Technology, U.S. Department of Commerce provides the following widely cited definition of cloud computing[17]:

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of

five essential characteristics, three service models, and four deployment models.

The five characteristics mentioned in the cite follows:

1. *On-demand self service.* A Consumer can get computing resources and services without the need of any human interaction from the service provider.

2. *Broad network access.* Capabilities are available over the network and accessible through thin and thick client platforms (e.g., mobile phones, tablets, laptops and workstations).

3. *Resource pooling.* The consumers has access to virtual resources that exists in a common pool following the multi-tenant model. The different resources can be dynamically assigned and reassigned according to consumer demand. The consumer has a sense of location (e.g. country or state) but have no control or knowledge over the exact location of provided resources. Examples of resources include storage, processing memory and network bandwidth.

4. *Rapid elasticity.* Characterized by the ability to acquire and release resources according to demand, often automatically.

5. *Measured service.* Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service. Resource usage can be monitored, controlled, and reported. Providing transparency for both the provider and consumer.

Service models:

1. *Software as a Service (SaaS).* Characterized by the offering of processing, storage, networks and other fundamental infrastructure resources. The consumer cannot manage the underlying hardware but is able to deploy arbitrary software and services on top of the provisioned resources, some times with limited configuration settings.

2. *Platform as a Service (PaaS).* Capability for the consumer to deploy onto the cloud infrastructure consumer-created or acquired applications. Limited to programming languages, libraries, services and tools supported by the provider.

3. *Infrastructure as a Service (IaaS).* Consumer provided capability to provision processing, storage, networks and other fundamental computing resources. Where the consumer is able to deploy and run arbitrary software which can include operating systems and applications.

Deployment models:

1. *Private cloud.* The cloud infrastructure is provisioned for exclusive use by a single organization.

2. *Community cloud.* The cloud infrastructure is provisioned for exclusive use by a specific community of consumers from organization that have shared concerns, e.g. security.

3. *Public cloud.* The cloud infrastructure is provisioned for open use by the public. Its often owned, managed and operated by a business, academic or government organization.

4. *Hybrid cloud.* The cloud infrastructure is a composition of two or more distinct previously mentioned cloud infrastructures, bound together by technology that enables data and application portability.

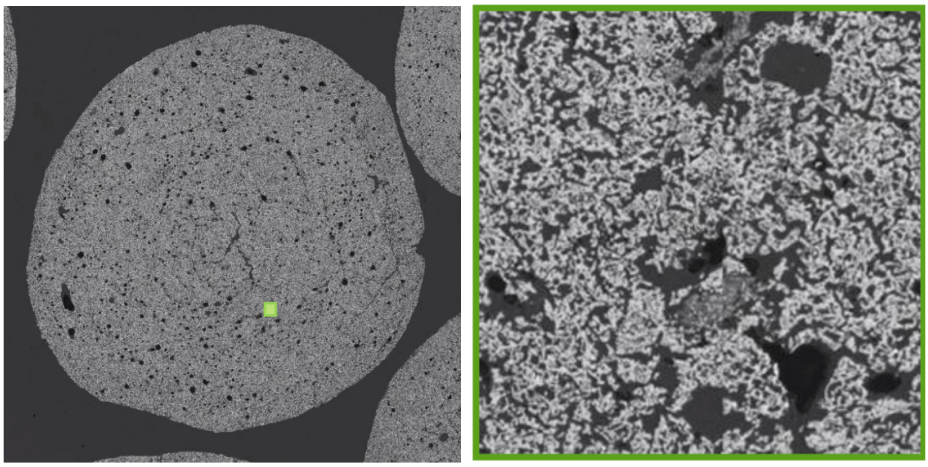
4 Material & Methods

4.1 Iron Ore Pellets

Iron ore pellets are produced at Luossavaara-Kirunavaara AB (LKAB) and used in the HYBRIT project [1] where the main goal is to find solutions for sustainable and fossil-free steel production. With so-called direct reduction hydrogen can be used to separate the iron from the oxygen without the use of carbon. This places demands on the constituents of the iron ore pellets. Iron ore pellets are at large small balls of magnetite, hematite, pore, epoxy, wüstite, olivine, slag and metallic iron fused together, see figure 13 for visual representation.

Figure 13: *Iron ore pellets.*

During evaluation some of these balls are selected for testing and fused together with epoxy, cut in half and then polished thoroughly before being photographed in a microscope.



(a) 30000×300000 ***(b)*** 1000×1000

Figure 14: *Showing a microscopic image of a pellet (a) and an enhanced piece of it (b).*

In figure 14b the composition of a iron ore can be viewed as shades of grey, where each shade represents a different constituent.

4.2 Preparing Data

The data provided from the microscopic images of iron ore pellets needs to be pre-processed before being eligible to feed to the networks for training. If provided with a gigapixel RGB image it requires downscaling to the input size of the network, in this case either squares with side 512 or 256 pixels depending on what size is used to train the network. This was achieved using the following script:

Python slicing script

```
from PIL import Image

def split(inputPath, height, width):
    img = Image.open(inputPath).convert("RGB")
    imgWidth, imgHeight = img.size
    for i in range(0, imgHeight, height):
        for j in range(0, imgWidth, width):
            box = (j, i, j+width, i+height)
            a = img.crop(box)
            a.save("./tmp/"+inputPath[:len(inputPath)-4]+\
            f"_{i}_{j}.png")

split("TIF.tif", 512, 512)
```

Amazon Web Services SageMakers Semantic Segmentation algorithm offers 1 channeled labeling style as a standard, this requires the split images to be transformed from 3-channel RGB format to single channel. This was achieved using the following script:

Python multi- to single channel script

```
from PIL import Image
import numpy as np
import cv2

color2index = { #BGR
    (0,   0,     170)  :   0,
    (0,   0,     255)  :   1,
    (0,   255,   255)  :   2,
    (255,   0,     0)  :   3,
    (255,  85,   255)  :   4,
    (255,    255,  0)  :   5,
    (0,   170,   255)  :   6,
    (0,    85,     0)  :   7
    }

def rgb2label(img, color_codes):
    result = np.ndarray(shape=img.shape[:2], dtype=int)
    result[:,:] = -1
    for rgb, idx in color_codes.items():
        result[(img==rgb).all(2)] = idx

    return result, color_codes

img, _ = rgb2label(cv2.imread("image.png"), color2index)
Image.fromarray(img).save("result.png")
```

4.3 Annotation

4.3.1 Amazon Web Services: Labeling Job

AWS offers a labeling job service where you can add data to be annotated and choose between making a private team, make use of mechanical turks or hire a vendor company to do the work for you.

- Mechanical Turk workforce consisting of over 500,000 independent contractors worldwide.

- Private workforce that you create from your employees or contractors for handling data within your organization.
- Vendor companies that you can find in the AWS Marketplace specializes in data labeling services.

The tool was reviewed using a private workforce to determine its effectiveness. It offers 3 different techniques for annotating, these are auto-segment, polygon and brush. Auto-segment takes 4 inputs, extreme points to create a mask and then you can use the brush and/or eraser to make adjustments. This works good for use cases like their supplied example with a bird on a branch in focus with a blurred background. Polygon takes any number of points connected to form a shape.

Figure 15: *AWS auto-segment tool provided example.*

4.3.2 Ilastik

Ilastik[18] is a simple, user-friendly tool for interactive image classification, segmentation and analysis. It is built as a modular software framework, which currently has workflows for automated (supervised) pixel- and object-level classification, automated and semi-automated object tracking, semi-automated segmentation and object counting without detection.

4.4 Uploading Data to S3

Amazon S3 offers a few different options to upload data to S3 buckets [19].

Single operation using AWS SDKs, REST API or AWS CLI
Using a single PUT operation you can upload files up to 5GB in size.

Amazon S3 Console
With the Amazon S3 Console you can upload large files up to 160GB in size.

Multipart upload using AWS SDKs, REST API or AWS CLI
Using the multipart upload API you can upload files up to 5TB. It's designed to improve the upload experience for larger object making it possible to upload files in parts. The parts can be uploaded independently, in any order, and in parallel. It has an supported range from 5MB to 5TB in size. Using multipart upload brings advantages such as increased performance by maximizing the available bandwith uploading parts in parallel. As well as being more resilient on a spotty network by resuming upload instead of restarting it and only needing to reupload parts that was interrupted. No need to restart uploading the file from scratch.

Presigned URLs
Presigned urls are useful if the user wants to upload a file to a bucket without AWS security credentials or permissions. When a presigned url are created you provide security credentials and specify a bucket name along with an expiration time. The url is then valid for the set amount of time and can be used to either upload a file in a single operation or utilize multipart upload. In which case all parts must have started uploading before the expiration time. Permissions to operations is required by the credentials used to create the presigned url. A big advantage using presigned urls is that there is no need for the "two-step upload" where user uploads the file to backend and backend in turn uploads it to S3.

4.5 Amazon Web Services: Lambda

AWS Lambda lets you run code without provisioning or managing servers, you pay only for the time you consume. There is also support for automatic trigger connected to other AWS services such as S3. Lambda does come with a few quotas whereas an EC2 instance does not, these quotas include but are not limited to table 1.

Resource	Quota
Function memory allocation	128 MB to 3008 MB
Function timeout	900 seconds (15 min)
Deployment package size	250 MB (unzipped)
/tmp directory storage	512 MB

Table 1: *Table showing AWS lambda quotas (25/09/2020).*

4.6 Amazon Web Services: Step Functions

AWS Step Functions is a serverless function orchestrator that makes it easy to sequence AWS lambda functions and multiple AWS services into applications. It is utilized to create and run series of checkpointed and event-driven workflows that maintain the applications state taking the output of one state as input for the next.

4.7 Security: AWS Identity and Access Management (IAM)

IAM is a free of charge security services that enables you to manage access to AWS services and resources. It empowers you to create and manage both AWS users and groups, to both allow and deny access to AWS resources.

4.8 Amazon Web Services: SageMaker

Amazon SageMaker is a fully managed service that offers developers and data scientist the ability to quickly build, train and deploy machine learning (ML) models all in one place. SageMaker is working out of the box and its not needed to stitch together different tools and workflows which normally can be very time consuming and error-prone. Amazon SageMaker provides several built-in machine learning algorithms that is available for use for a variety of different problems:

- BlazingText algorithm
- DeepAR Forecasting Algorithm
- Factorization Machines Algorithm
- Image Classification Algorithm
- IP Insights Algorithm
- K-Means Algorithm
- K-Nearest Neighbors (k-NN) Algorithm
- Latent Dirichlet Allocation (LDA) Algorithm
- Linear learner algorithm
- Neural Topic Model (NTM) Algorithm

- Object2Vec Algorithm
- Object Detection Algorithm
- Principal Component Analysis (PCA) Algorithm
- Random Cut Forest (RCF) Algorithm
- Semantic Segmentation Algorithm
- Sequence-to-Sequence Algorithm
- XGBoost Algorithm

It the built-in algorithms isn't enough in your case AWS also offer support to upload private models and algorithms to use with the SageMaker API. There is also support to upload algorithm and model package resources to the AWS Marketplace, from where it is also possible to buy them to then import either into SageMaker.

4.8.1 Built-in algorithm: Semantic Segmentation

The Semantic Segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects in the image. The segmentation output is represented as a gray-scale image, commonly known as a segmentation mask. A segmentation mask is simply a gray-scale image with the same shape as the input image. The SageMaker Semantic Segmentation algorithm is built using the MXNet Gluon framework and the Gluon CV toolkit, and provides several choices of what build-in algorithm to use for your deep neural network.

The Semantic Segmentation algorithm offers a wide range of parameters for tuning, where the most important ones has been listed below:

Backbone
The backbone to use for the algorithm's encoder component, if offers two different choices between resnet-50 and resnet-101.

Algorithm
The algorithm used for semantic segmentation, it offers 3 choices between Fully-Convolutional Network(FCN), Pyramid Scene Parsing(PSP) and DeepLab V3(deeplab).

Num_Classes
The number of classes to segment.

Num_Training_Samples
The number of samples in the training data. The algorithm uses this value to set up the learning rate scheduler.

Learning_Rate
The initial learning rate.

Crop_Size
The image size for input during training. Retrieves a random square crop with side length equal to crop size from the image during training.

Base_Size
Defines how images are rescaled before cropping. Images are rescaled such that the long size length is set to base_size multiplied by a random number from 0.5 to 2.0, and the short size is computed to preserve the aspect ratio.

Mini_Batch_size
Batch size for training, increasing it will result in faster training at the cost of memory consumption.

Epochs
The number of epochs to train.

5 Implementation

5.1 Data annotation

In addition to the solutions in 4.3 a couple of scripts was developed. Mainly to get a larger sample pool showing different techniques, but also to point out why machine learning is needed for this thesis.

5.1.1 Scripts

Python scripts, I have come up with 2 different proof of concepts to solve the problem at hand.

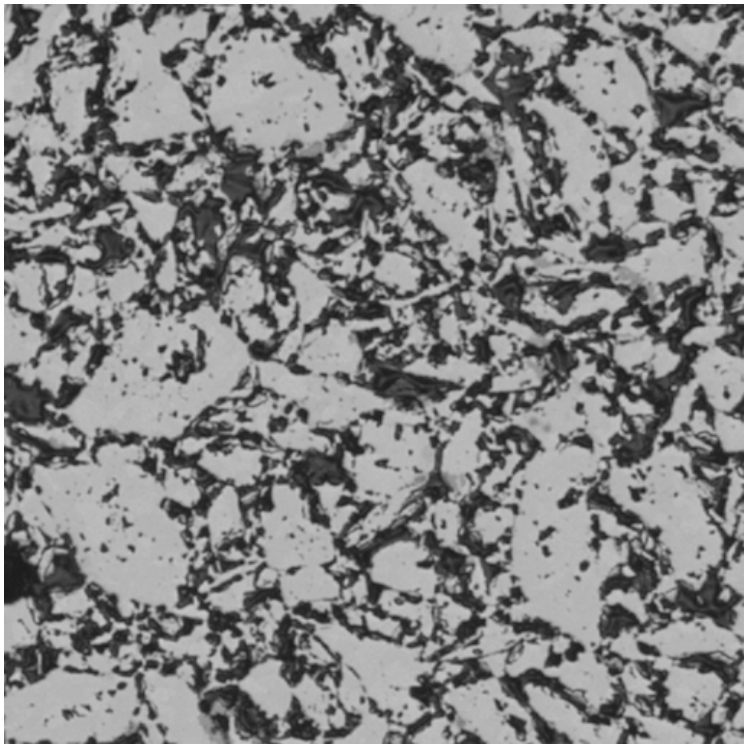


Figure 16: *Ground truth for the following 2 scripts.*

The first transforms the image to HSV format and mask one color at a time to find all pixels of approximately the same color and then colors them in a color of choice, appendix 8.0.2.

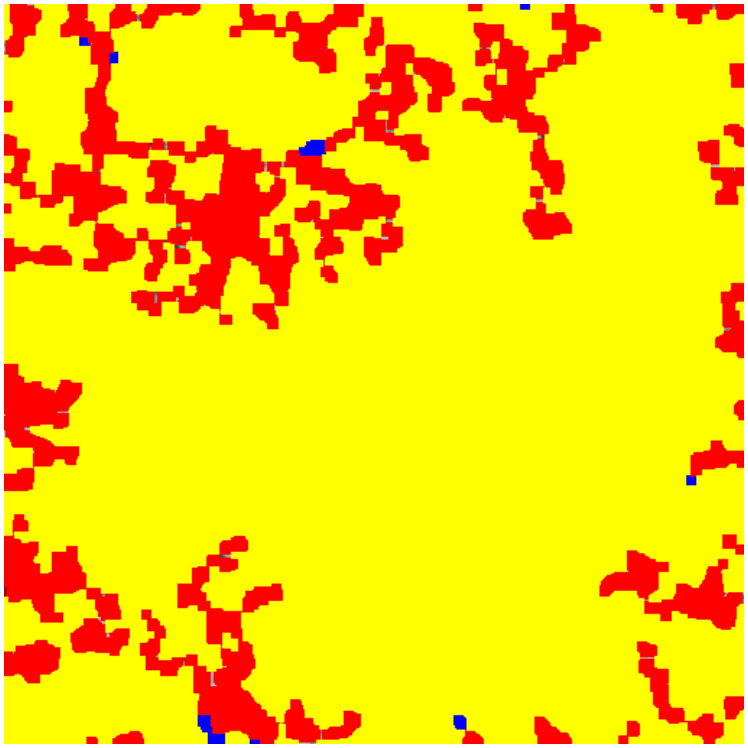


Figure 17: *Python script 8.0.2 masks all pixels within same color range to a new color of choice.*

The other solution iterates over all pixels in the picture using a color table with a threshold to find all different elements, appendix 8.0.2.

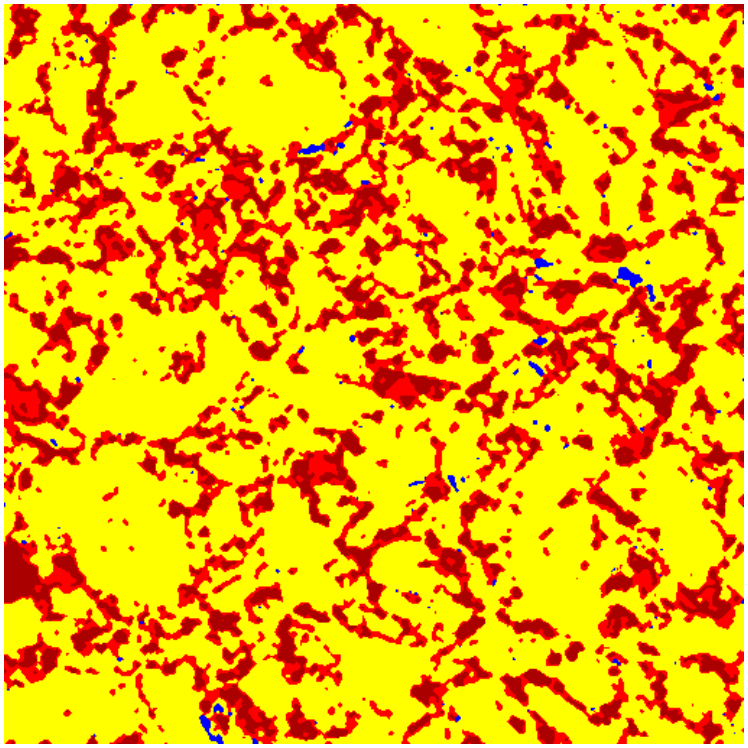


Figure 18: *Python script 8.0.2 using color tables.*

In both solutions filters are applied to remove noise.

5.2 Networks

5.2.1 Training

Three different networks has been trained using Amazon Sagemaker's built in Semantic Segmentation algorithm. All of them were trained using a ml.p2.xlarge GPU instance and 256×256 pixel sized training data with the following settings:

Table 2: *Sagemaker Settings.*

Parameter	**Setting**
Epochs	200
Batch size	8
Classes	21
Crop size	256

Base Size	256
Pre-Trained	True
Early stoppings	False

5.2.2 Dataset

The complete dataset consist out of 180 images of size 512×512, these are then split into three different categories according to the following table 3.

Table 3: *Dataset distribution.*

Dataset	**Images**
Train	91
Validation	58
Testing	31

Each image is divided into slices of 256×256 pixels to fit the input layer of the network's settings.

Table 4: *Dataset distribution of sliced set.*

Dataset	**Images**
Train	364
Validation	232
Testing	124

5.3 Cloud

5.3.1 AWS Lambda

The goal with this implementation is to determine whether its feasible to run this as a serverless application or if allocating servers are necessary or beneficial. The idea is to have an user upload data to be automatically evaluated by the neural network.

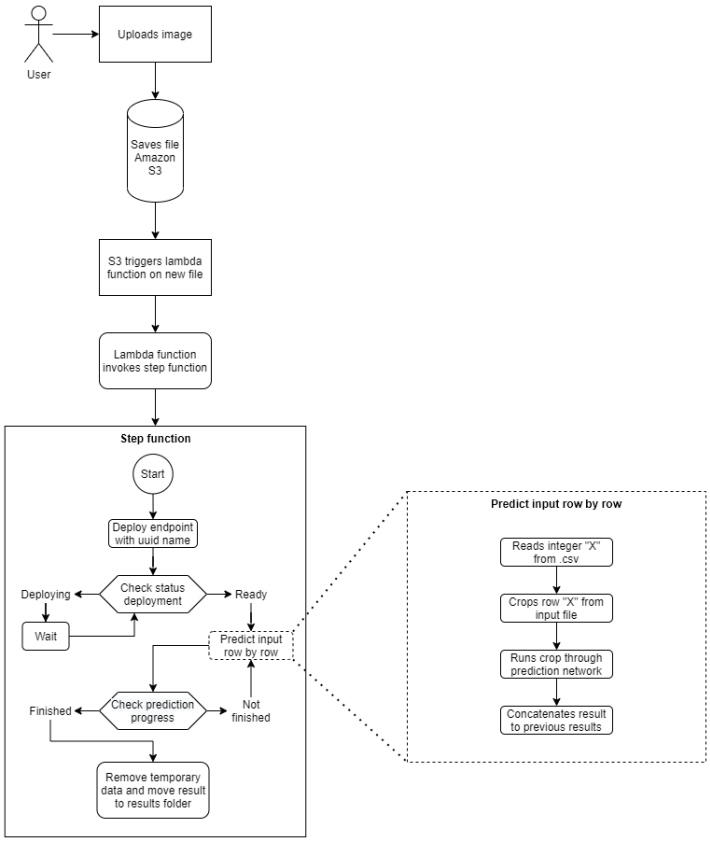


Figure 19: *System architecture.*

The system is made up of the following components:

ReactJS frontend A very minimalistic frontend written in reactJS has been implemented to be able to upload data to S3 storage by utilizing presigned URL's.

S3 The implementation takes advantage of Amazon web storage S3 and its possibility to trigger a lambda function on recognizing a new entry for a specific folder.

IAM There are a few IAM roles required for the implementation of this serverless approach, these are:

- **StepFuncInvoker:** A role which main purpose is to permit the lambda function triggered by S3 to start a step function.

- **FullS3SagemakerAccess:** A role that has full access to S3 as well as Sagemaker making it the most important role and vital to be able to run the system.
- **S3ReadWrite:** A role used to create token access for the frontend application making it able to upload data to S3 as well as read it.

Step function & lambda The step function is used as a finite state machine to sequence lambda functions. This proves especially useful when met with the restrictions put on by lambda. Instead of running the data in one go, it's divided using step functions and processed one part at a time until completed. To be able to run several predictions simultaneously and reduce costs. A new endpoint is started as the first step in the step function and then terminated in the clean up.

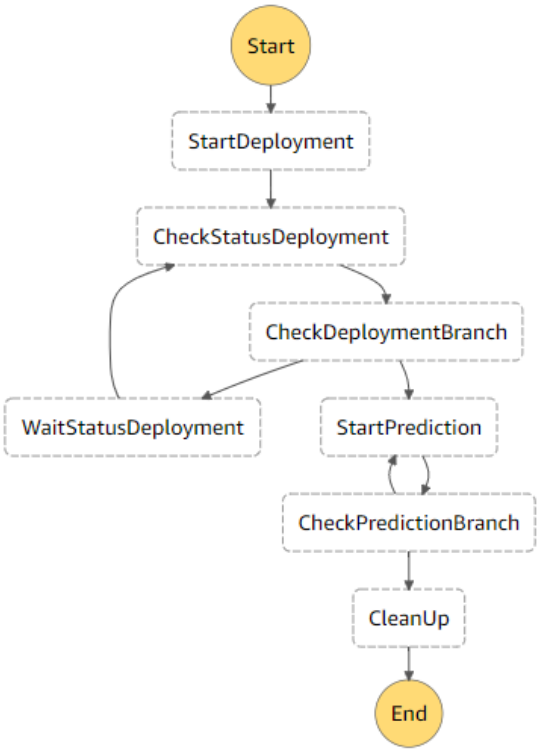


Figure 20: *AWS step function graph.*

- **StartDeployment:** Creates a new endpoint.
- **CheckStatusDeployment:** Checks status of newly created endpoint.
- **CheckDeploymentBranch:** Acts on previous functions output.
- **WaitStatusDeployment:** Waits a few seconds
- **StartPrediction:** Predicts input image row-wise.
- **CheckPredicitonBranch:** Checks status of previous functions output to determine if prediciton is finished.
- **CleanUp:** Moves result to separate folder and shuts down endpoint.

5.3.2 Amazon Elastic Compute Cloud (Amazon EC2)

Three different Amazon EC2 was setup and configured with Gluon CV to run a trained FCN model from Amazon SageMaker, as an alternative to using an endpoint.

Table 5: *Average inference time on EC2 with pre-trained FCN model from Amazon SageMaker.*

Instance	slice (sec)	Pellet (hour)	Instance cost ($/h)
p2.xlarge	5.09	1.7	0.972
c5.xlarge	2.49	0.83	0.192
t3a.medium	14.46	4.82	0.041

Figure 21: *Cost per complete pellet image using different EC2 instances.*

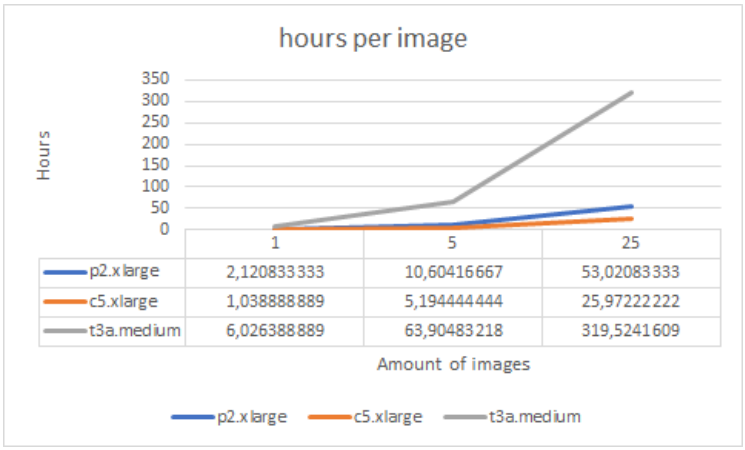


Figure 22: *Required time to run inference on a complete pellet image using different EC2 instances.*

Looking at figure 21 both c5.large and t3a.medium performs admirably in regard to cost. Taking time from figure 22 into account its no great trade off to go with the t3a.medium instance. Leaving the c5.xlarge instance as the best choice of the evaluated instances.

6 Evaluation

6.1 Annotation

Looking at our images with a lot of small important details it would require a large amount of work for each real life pellet.

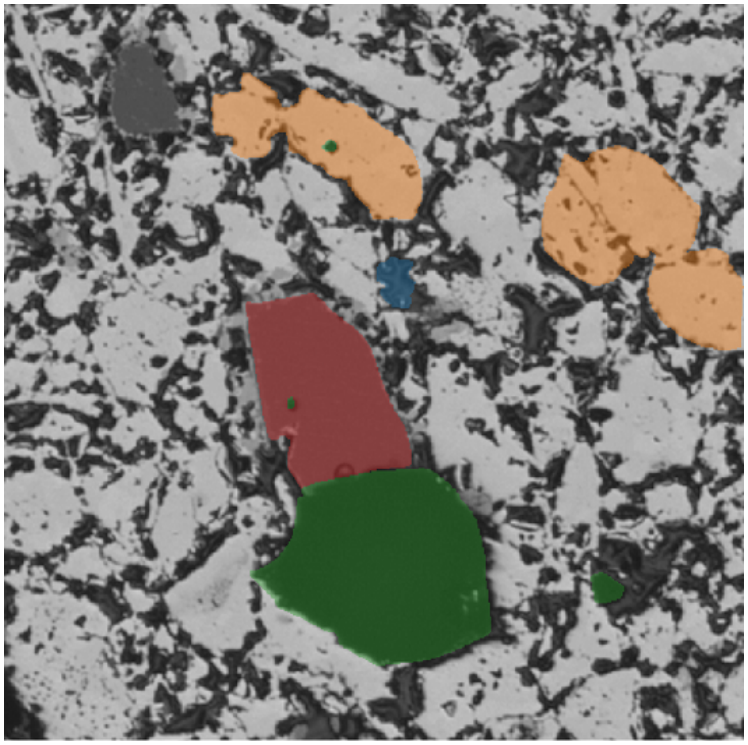


Figure 23: *AWS auto-segment tool on thesis data.*

Provided example, figure 23, is showing less then a 0.1% crop from the complete picture of one pellet and it's important to segment every pixel.
Since you have the ability to overlook and accept the work done by the Mechanical turk workforce it could be a solid option. It could end up being a lot cheaper than spending in house resources for labeling or hiring a Vendor company, at the risk of the work being low-grade. It's a tedious and time consuming task that requires focus to be done right.

6.1.1 Dataset Composition

The thesis uses a minimal amount of data to train and evaluate the networks, with more annotated data the results are bound to improve. In table 6 you can see how many % of each element there is among the training data. It can be expected that the network will have a harder time identifying and predicting the less common classes, such as slag and olivine.

Table 6: *The cut of each material in perspective to the total amount of all training data.*

Name	**Cut (%)**
Pore	10.77
Epoxy	33.75
Hematite	8.73
Magnetite	9.77
Wüstite	31.17
Iron	5.16
Slag	0.26
Olivine	0.38

6.1.2 Summary

After a careful review of the labeling job tool offered by Amazon Web Services 4.3, it was concluded that it did not bring enough automation to be of use. The functionality seemed promising at first but when inspecting it closer it was clear that it was not able to handle this type of data in a sufficient way.
Outsourcing this kind of data for annotation would require to hire experts in its field or spending a lot of valuable time educating the annotators.
Looking at the other solutions, my own developed scripts. It was concluded that these scripts was of little use because of the source image having different elements with the same color (RGB value), some kind of machine learning is needed to determine to what class the pixel at hand belongs.
Utilizing Illastik's automated but supervised pixel and object classification receives the best results from the least amount of work offered by above alternatives. You essentially color code your classes marking them in the picture and then along with user-defined filters Illastik makes a real-time prediction based on your drawings.

6.2 Networks

6.2.1 Algorithms

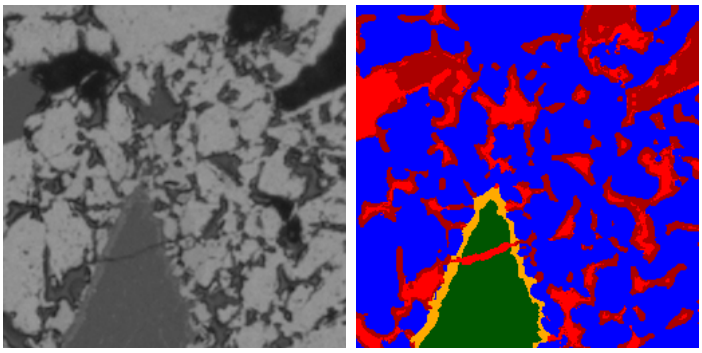
Three different networks was trained using Amazon Sagemaker's built in Semantic Segmentation algorithm. All of them was trained using a ml.p2.xlarge GPU instance and 256×256 sized images with the following settings:

Table 7: *Sagemaker Settings.*

Parameter	Setting
Epochs	200
Batch size	8
Classes	21
Crop size	256
Base Size	256
Pre-Trained	True
Early stoppings	False

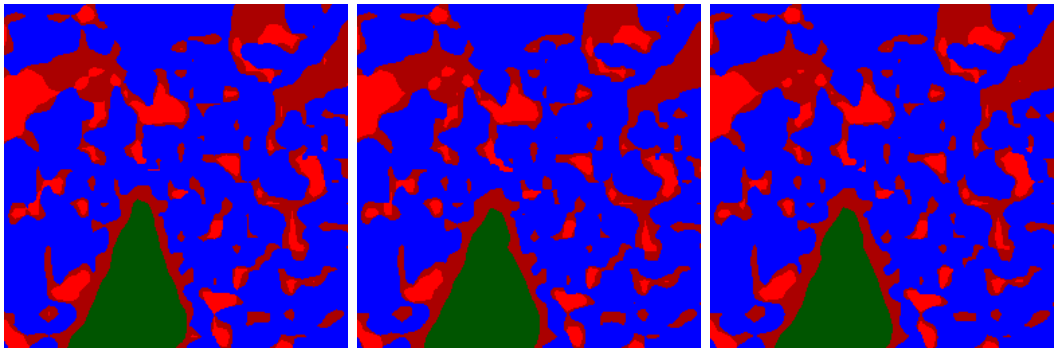
Table 8: *Validation metrics*

Algorithm	mIOU	mAcc	Throughput	Training time (min)
FCN	0.246	0.858	10.1	331
DeepLab	0.248	0.857	8.23	435
PSP	0.245	0.857	8.74	382



(a) *Input* ***(b)*** *Ground Truth*

Figure 24: *Comparing input and ground truth.*



(a) DeepLabV3 *(b) FCN* *(c) PSP*

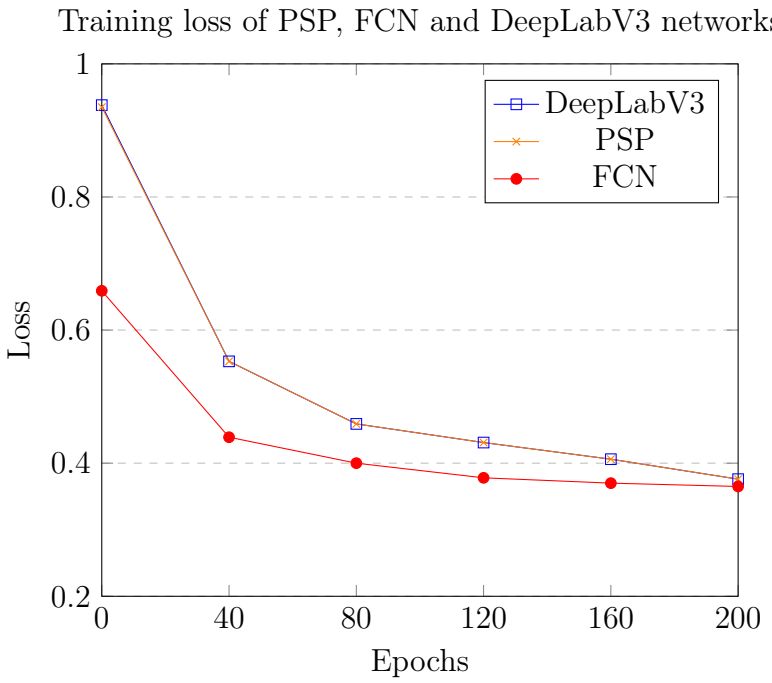
Figure 25: *Comparing results from each of the algorithms.*

Starting off its clear by table 8 that all of the networks has similar performance. The overall performance is quite low. With an average mIOU of 0.246 and mAcc of 0.857 where no network really stands out. See 24 & 25 for visual representation. However the inference time shown in table 9 varies greatly between PSP and its alternatives.

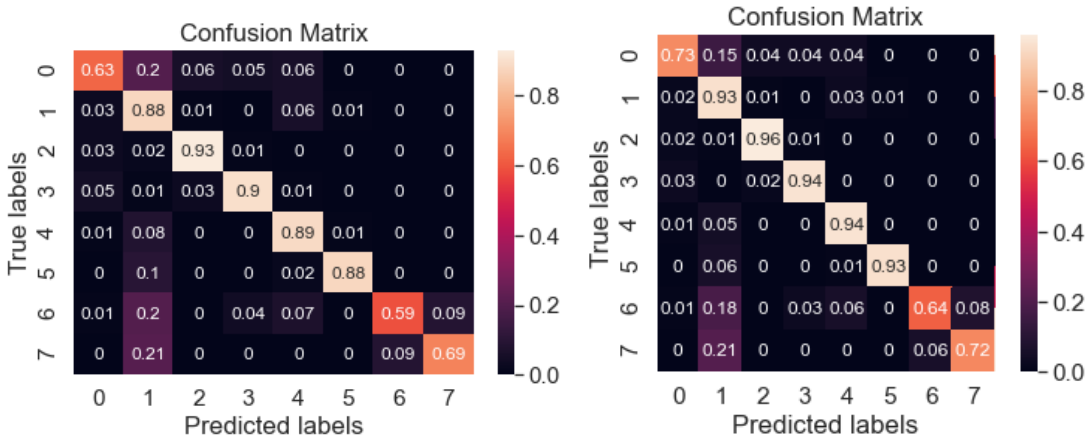
Table 9: *Mean inference time of 58 respectively 232 slices presented in seconds.*

Algorithm	512×512	256×256
FCN	0.47	0.13
DeepLab	0.50	0.14
PSP	10.5	2.42

What's intriguing is the training times, where there is almost a two hour difference between FCN and DeepLab. Looking at loss in graph 6.2.1 below FCN starts off better but all of them ends up with resembling performance once again.



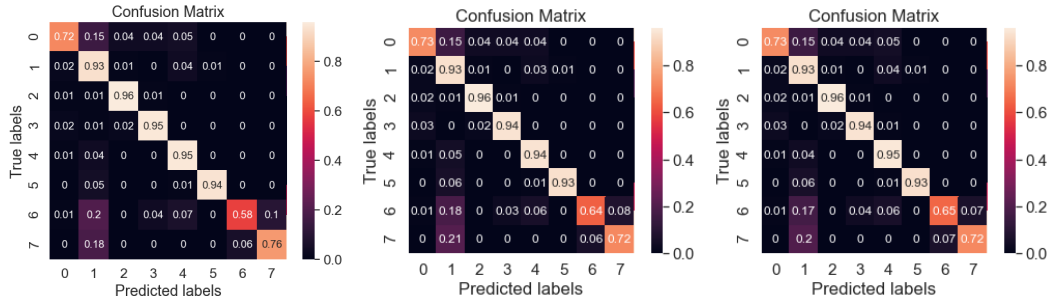
More thorough evaluations on the predicted data is done using weighted confusion matrices 3.3.5. This is because of the extremely vague borders in the data sets. The annotation in proximity to borders and edges is very tricky and the annotation of the datasets can safely be assumed to not be pixel perfect. To compensate for this, the uncertain pixels are supplied a lower weight using Euclidean Distance Transform with an edge limit of 3 pixels. The following matrices are displaying the average results on a test dataset containing 232 images. This is achieved by adding all individual non-normalized weighted confusion matrices together and then normalizing the sum.



(a) No distance transform *(b) With distance transform*

Figure 26: *Comparing a normal Confusion Matrix 26a to a weighted one 26b.*

Figure 26 shows slight but notable increased performance when utilizing the weighted CM. Below follows the weighted confusion matrices for the three different networks:



(a) DeepLabV3 *(b) FCN* *(c) PSP*

Figure 27: *Comparing results from each of the algorithms.*

To reassure the authenticity of the results displayed in figure 27 a pixel level evaluation 3.3.6 can be viewed in table 10.

Class	Labeled set	FCN	PSP	DeepLabV3
0 Pore	10.07	8.66	8.33	8.18
1 Epoxy	37.16	38.02	37.19	37.34
2 Hematite	8.33	9.10	9.11	9.18
3 Magnetite	8.32	8.37	8.34	8.47
4 Wüstite	30.61	30.41	31.58	31.29
5 Iron	4.83	4.90	4.88	4.99
6 Slag	0.21	0.20	0.21	0.18
7 Olivine	0.47	0.35	0.35	0.37
sum	100	100	100	100

Table 10: *Class occurrence % of labeled test data compared to predictions of each algorithm.*

The reason to why classes 6 & 7 are performing distinctly worse then the rest can be explained by the very low occurrence of those classes in the training data set. Furthermore the classes 0 (pore) & 1 (epoxy) are not crucial if they are being mixed up. Since, the areas showing epoxy were pores before applying the epoxy. It is being discussed if these 2 classes should be merged together to a combined class.

6.2.2 Model Tuning

Automatic parameter tuning (model tuning) is not yet supported by the semantic segmentation algorithm, it is one out of two from a total of 17 algorithms provided by Amazon SageMaker that is missing model tuning documentation.

6.3 Cloud

6.3.1 Experiment

Running one 512×512 slice through the trained Pyramid Scene Parsing (PSP) network takes at average 10.5 seconds according to table 9. An average row is consisting out of 40 slices putting the time for executing a row between 7-8 minutes. This is then repeated over 30 times to cover the complete image adding up to about 3-5 hours in time. Hosting the endpoint on different types of instances has shown only small impacts on execution time, see table 11
However, switching to either a FCN or DeepLabV3 network reduces the time from 10.5s per slice down to an average of 0.47 and 0.5 respectively (see 9). Cutting the

Instance	Test 1	Test 2	Test 3	$/h
ml.c4.xlarge	9.66	9.41	9.53	0.226
ml.c4.4xlarge	8.83	9.38	9.44	0.905
ml.p3.2xlarge	9.46	8.73	9.18	3.305

Table 11: *Table showing Amazon instance pricing (9/3/2020) and prediction time of a 512x512 image in seconds running the PSP network.*

time with a denominator of 21, putting the inference time closer to 10 minutes per pellet.
By looking at results from implementations mentioned above 6.3.1 and cost tables provided by [20] the following graphs can be presented:
Presented graphs are based purely on theoretical calculations and the mean inference times for each network. The time for slicing and concatenating images has not been included and may impact the end-cost along with start-up times when allocating resources.

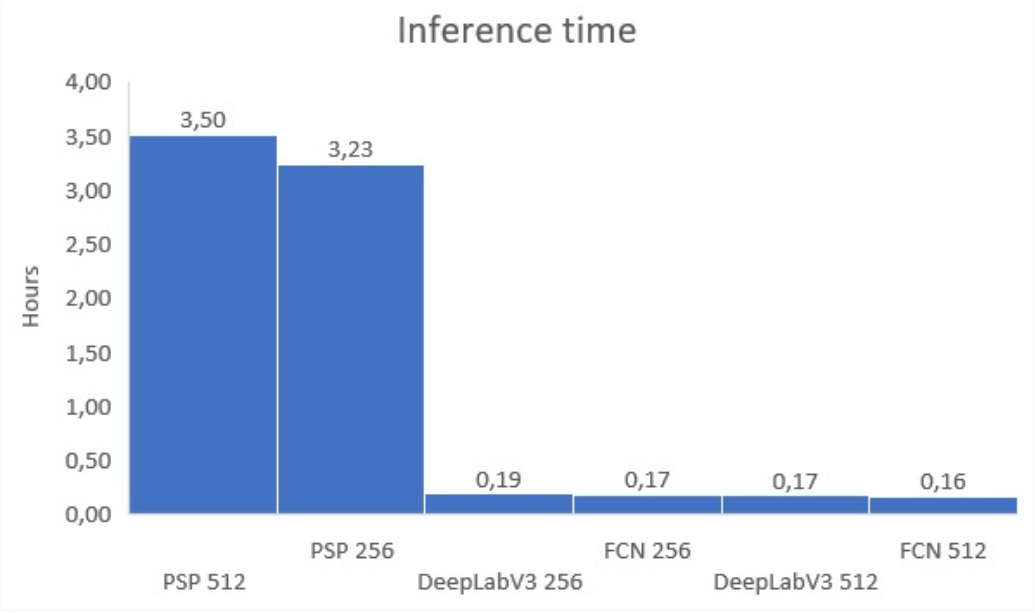


Figure 28: *Inference on a pellet using input data of* 512×512 *respectively* 256×256 *pixels.*

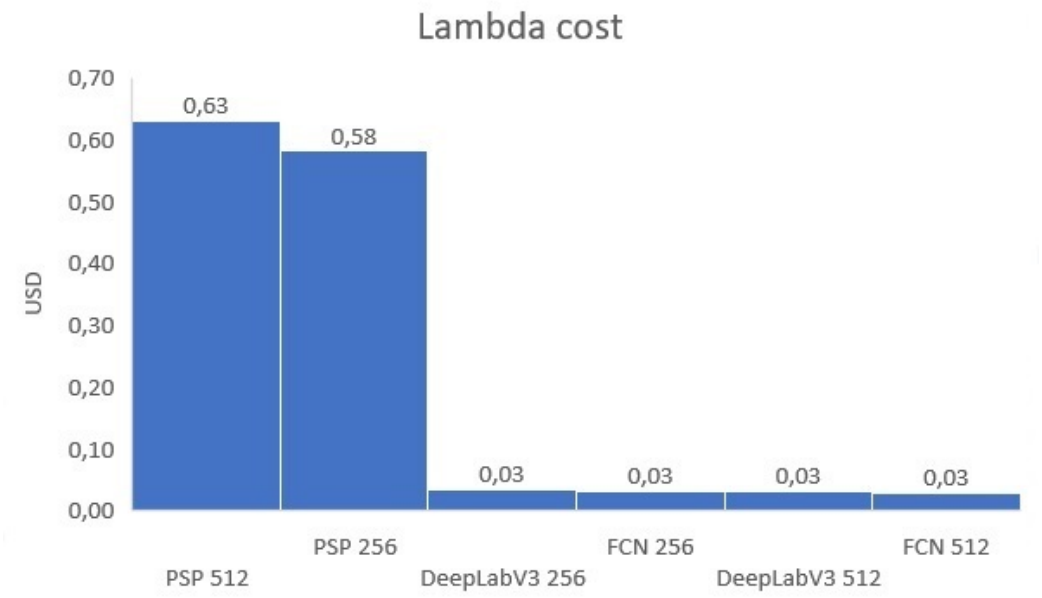


Figure 29: *The price running lambda with the different networks and input sizes.*

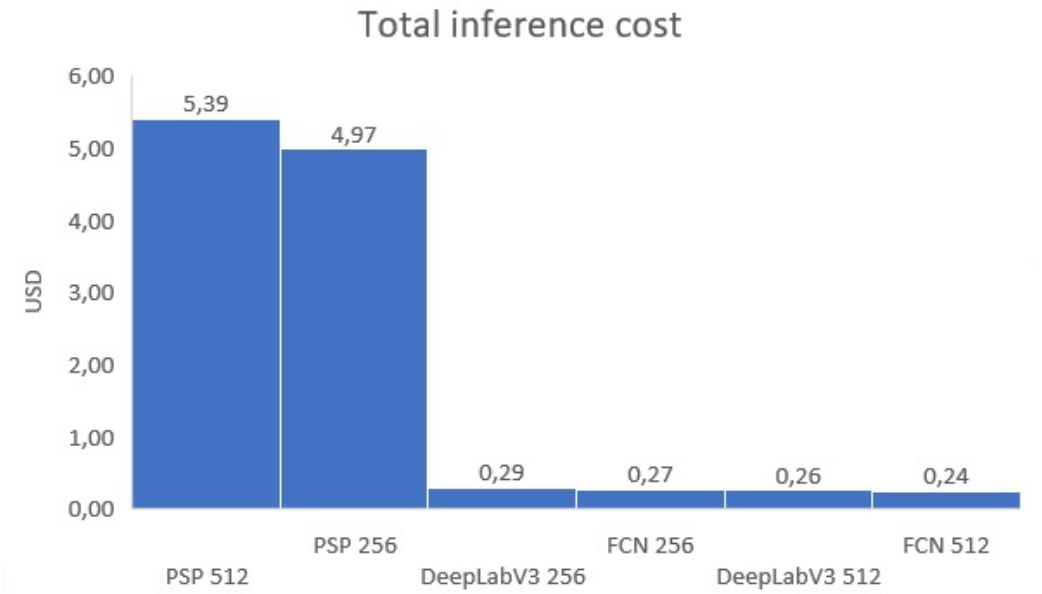


Figure 30: *The total cost of inference for one pellet using the implementation presented in 5.3.1 (ml.p2.xlarge).*

6.3.2 Cost & Time

In figures 28 & 30 presented above, it can quickly be concluded that the cost is directly tied to the inference time. This is explained by the fact that Amazon Web Services (AWS) charges for the time spent allocating their resources. The cost for making requests has been ignored in the lambda cost graph 29 due to its low cost in reference to the total cost, see graph 31. The cost of requests when running inference on one pellet is merely a few parts per thousand of the total cost.

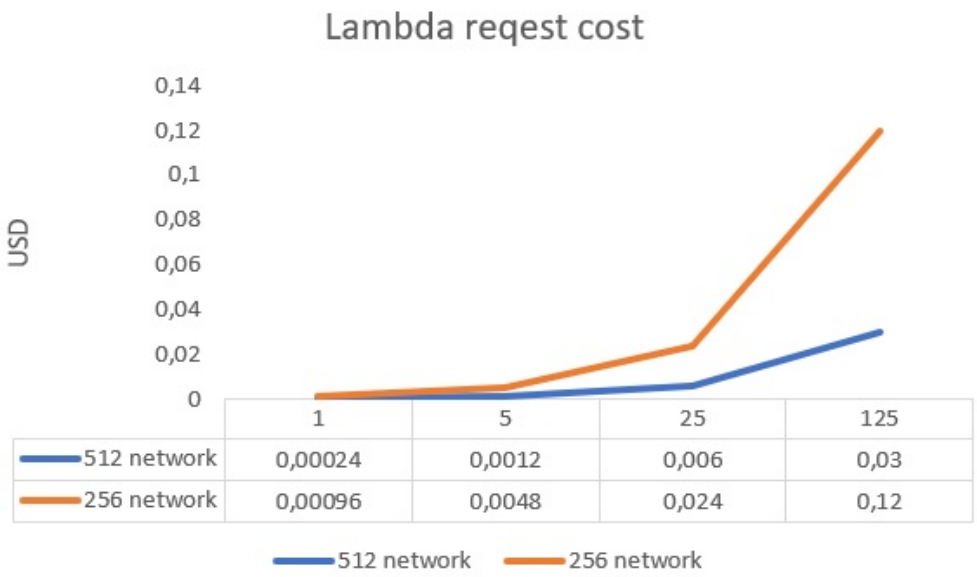


Figure 31: *The cost of lambda requests per amount of inferenced pellets (avg. 1200 & 4800 requests respectively).*

Instances for inference Amazon Web Services (AWS) offers a wide range of instances for running inference on your trained model, in table 12 the average time for three different instances is displayed. Where 2 of them is computational instances (c4 & c5) and one has GPU support (p2). As displayed there is a very slight, if any, benefit to running a GPU instance over computational ones in this case.

Table 12: *Inference on different instances using FCN network and 232 images sized* 256×256.

Instance	**Avg. inference time (sec)**	**Price ($/h)**

ml.p2.xlarge	0.13	1.361
ml.c4.xlarge	0.14	0.316
ml.c5.large	0.14	0.134

Instances for training Using AWS it's really easy to impact the training time simply by switching up instance types. For this thesis, time is however not a critical aspect when training networks. It does not matter if it takes a couple of hours to train the network, it has no critical time aspect to it nor will it be done often. It makes more sense to aim for lower costs.
The load of the system when training the three networks commonly mentioned can be viewed in the table below 13. By looking at the GPU memory we can see there being a slight potential to increase the batch size further, but in the interest of doing a reasonable similar comparisons to Terese's work [2] it was determined to settle at batch size = 8.

Table 13: *Avg. instance load during training, all metrics in % using a ml.p2.xlarge instance.*

Algorithm	GPU usage	GPU memory	CPU usage
FCN	93	61	122
DeepLab	95	65	119
PSP	91	70	122

6.3.3 Instance types & Performance

The built in Semantic Segmentation algorithm offered by Amazon SageMaker offers no parallelization support, the only way to increase training performance other than parameter tuning is by choosing a more powerful instance. Two more networks was trained using a more powerful instance (ml.p3.2xlarge - 4.131 $/h), one with the exact same parameters as in earlier networks 6.2.1 and one where the batch size has been doubled to make use of the increased GPU memory.

Table 14: *Mean performance in % comparing two trained FCN networks on ml.p3.2xlarge instances.*

Batch size	mIOU	mAcc	GPU usage	GPU memory	Training time (h)
8	27	86	73	51	1.14

16	26	85	73	66	1.14

As seen in table 14 the batch size does not correlate linearly to GPU memory, there is still plenty of memory to be used. But from what is gathered it happens to show decreased performance with an increased batch size, this could be purely situational because the table is only showing 2 different occasions and for any real conclusions to be drawn more data is required. The most interesting thing in this table is that even with a twice as big batch size the training time remains the same.

6.3.4 Usability & Maintenance

From a user's point of view there is not much to say regarding either usability or maintenance, it should be of no difference. From a developers point of view there is a lot less overseeing and management required using the lambda approach. Amazon delivers endpoints ready to run and keeps them up to date whereas EC2 instances needs to be setup and maintained. Using lambda also has the benefit of keeping the system up to date compared to running a local implementation on a EC2 instance that would force a developer to manually keep things up to date. The lambda implementation also offers an easier solution changing models, by simply changing a variable name. Compared to uploading the model to an EC2 instance and in the worst case reconfigure it to run with the new model.

6.3.5 Allocating vs. Serverless

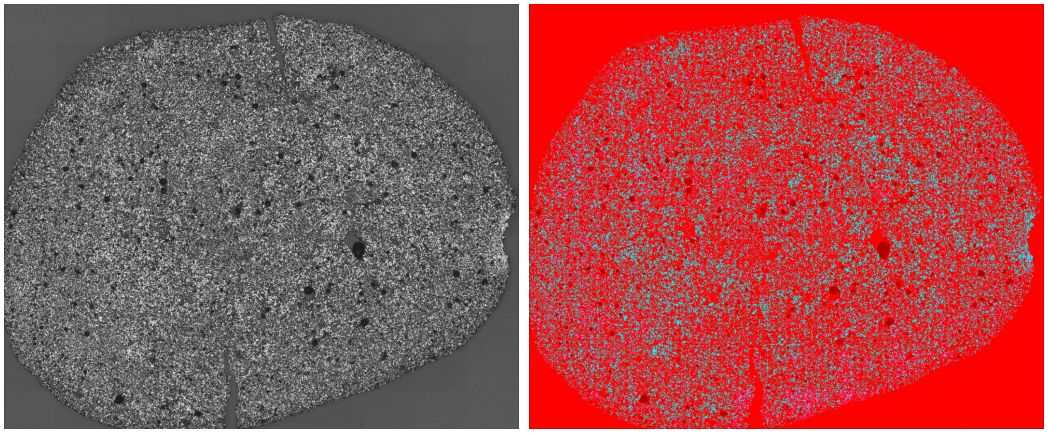
A ml.p2.xlarge instance have a cost of $1.361 per hour [20]. Making lambda requests have a set fee of $0.0000166667 for every GB-second.
Not counting for the time spent powering on/off resources and processing the data, the time for running inference on one pellet can be viewed in figure 28. Picking the FCN 512 network that runs inference on a compelete pellet in just under 10 minutes ($0.16 \text{ hours} \cdot 60 \text{ minutes} = 9.6 \text{ minutes}$). Where inference on each request takes at average 0.47s as presented in 9. The average amount of requests is 1200 after slicing a pellet into squared parts with side 512px. The system is set the use the maximum amount of memory (3GB) to be able to concatenate slices in memory. Using Amazon's provided calculator we get a hint of the real-life cost using the average 0.47s rounded:

$$\begin{aligned} 1200 \text{ requests} \cdot 0.5s &\approx 600 \quad \text{computational seconds} \\ 3GB \cdot 600 &\approx 1800 \quad \text{total compute (GB-s)} \\ 1800 \cdot 0.0000166667USD &\approx 0.03 \quad \text{USD (monthly compute charges)} \end{aligned} \tag{3}$$

About 833 images of 1200- or 208 images of 4800-slices could be processed before being charged another $0.20 for 1M more requests.
This could be considered extra costs since we are still required to run an endpoint for inference. An estimation of total cost can be viewed here 30. It comes down to deciding weather it's worth paying this $0.03 extra and utilize lambda or replace the lambda implementation with EC2 based alternatives.

6.4 Real World Experiment

A TIFF image of a pellet sized 20480×16896 pixels (330MB) was uploaded to AWS S3 using the implemented ReactJS frontend 5.3.1. The upload-time is dependant on a couple of factors where the main one is the network connection. During evaluation the upload-time never exceeded one minute to complete, this time could be decreased by utilizing multipart-upload mentioned in Material & Methods 4.4. Once finished uploading, a S3 trigger is executing a lambda function which in turn starts the step function that processes the image, see implementation 19. This experiment was conducted using the FCN network with input size 256×256. The total time to process this pellet ended up at 38 minutes, where it took 9 minutes to power on the endpoint leaving about 30 minutes for the inference. Comparing this to the theoretical value presented in figure 28, a significant difference is displayed. Theoretically the inference is supposed to take approximately 10 minutes whereas in reality it takes almost three times as long. This can be explained by a couple of things, there among the time consumed to slice and concatenate the image. Additionally the time required to concatenate each inferenced row and upload it to S3 before continuing with the next row.



(a) *Actual pellet* ***(b)*** *Predicted pellet*

Figure 32: *Showing the supplied input 32a and the cloud systems predicted output 32b.*

Figure 32 is showing the input and result from running this experiment.

7 Discussion

7.1 Annotation

There exists a few alternative solutions to the annotations problems, one of them being to hire a company specialized in annotating data to resolve the issue. Another one could be to use a trained network for supervised annotation, possibly in collaboration with the Ilastik software. Looking back at the beginning of the thesis I should have spent less time reading about the AWS annotation tool but rather started using it. It was evident that it was not sufficient once I started using it, even tho it looked promising from its documentation. It was a problem finding an annotation tool that performed according to expectation, it was decided to settle for Ilastik as its an easy to use free alternative performing relatively well for this task.

7.1.1 Dataset

All three newly trained networks (PSP, FCN, DeepLabV3) completely failed to identify slag in the test data. This was believed to be due to the small amount of annotated data used to train the network in combination with the rare occurrence of slag. Out of the images used to train the network just a few contained slag and that represented just a small part of those images seen in table 6.
However, it was later discovered that when transforming the data to 1-channeled files the conversion was corrupted. The networks completely failed to identify slag because there was no slag class in the training data, it appeared as pores. This was because the library CV2 read the image as BGR when it was believed it was in the format RBG, thus the values was mixed up which in turn lead to a faulty transformation. Towards the end of the project this was detected and adjusted, some results using the new correctly annotated data can be found in table 14. Luckily it is barely noticeable, because the slag class is consisting out of such a small part of the overall picture.
Unfortunately the area of annotating iron ore pellets is not that common and the art of automatic annotation is cutting edge making it very unlikely to find any other results to compare my findings to. Other than Terese's thesis where Ilastik was used for manual annotation.

7.2 Networks

Something that would be interesting to investigate would be to compare an alternative solution where you train a model outside of AWS and then upload the model to SageMaker and have it run in an endpoint. This along with a completely

SageMaker-free implementation using only EC2 and the same libraries (Gluon CV) that SageMaker is using for its Semantic Segmentation. Additionally there are several more interesting topics to investigate, the effects of transfer learning on the supplied data set, does this network gain from being trained on a pre-trained set like ResNet-101? How does more annotated data impact this? A more through investigation regarding the input size of the networks, ResNet-101 that has been used throughout the project has an input size of 224×224 meanwhile squared images with side 256 respectively 512 has been used to train these models. Looking back I feel I should have spent more time to structure my work, to have a clearer view of what I was looking for and how I should have treaded to get there. Also thought ahead more thoroughly of what data that was going to be of interest and how to best change variables to get desired results without having to waste time and resources. The main problems encountered has been the lack of understanding Amazon SageMaker and the varying results it has delivered. It was pure bad luck that I choose a PSP network when implementing the serverless lambda proof of concept, since the PSP network for some reason performs a lot worse then other algorithms it provided deceptive results which later was updated. Beyond this, the size of the input layer has been in question several times, looking at section 4.8 there is parameters for "base size" and "crop size". Which can only be assumed to connect to the input layer due to lack of documentation. In a recent update it was discovered that networks are no longer able to be trained using 8 classes (which is the amount of classes used in this dataset). It now requires 21 classes for the code to run, same amount as the pre-trained network was trained upon, disregarding if the pre-trained network parameter is enabled or not. More on this in section 7.2.1. Having more annotated data could bring improved results, at least up to a certain threshold. The 180 images in this dataset is in reality just a small part of one pellet. However, comparing to earlier results [2] using the very same dataset the results are significantly worse looking at mIoU and mAcc. However, the results from confusion matrices tells a different tale. Looking at the weighted confusion matrices displayed is section 6.2.1 the predictions for most classes is within the range of being great.

7.2.1 Difficulties

Amazons SageMaker tool is cutting edge software, it's developed to make machine learning more manageable and comprehensible. This however comes with a few drawbacks. The tool is a black-box design when it comes to configuration and parameter tuning. There is a lack of understanding of what each parameter actually is doing behind the scenes, how the parameters offered for configuration really affects

the network. One possibility is that the Semantic segmentation algorithm is one of the less commonly used ones and as such it's the only one that has no supported parameter tuning. The fact that the PSP network has an inference time about 21 times bigger than the two alternatives adds to this belief and makes it seem faulty.

The semantic algorithm was updated during the thesis, implemented and tested code suddenly stopped working due to more condition checks being introduced. At the start of the thesis all networks was trained using 8 classes. Later on this had to be changed to 21 because the newly added condition checks was no longer allowing 8. This is another one of these black-box matters mentioned earlier, the network that is to be trained is only using 8 classes, however the dataset used to pre-train the network is using 21. There is however no data this far pointing to this part being faulty, no output has of yet had any class outside of the 8 existing ones. This begs the question if this is done automaticly when training the network, but what would then be the reason to have a parameter named num_classes?

The parameters "base size" and "crop size" found in 4.8 says to re-scale the image to the base size and then take a random crop from this image and run in through the network. This is not of interest for this type of implementation, we do not want to scale our data, and we are already lacking training data so there is really no reason to crop but rather use the whole image. The cheaper instances showed themselves too weak to be able to run 512×512 images even with the lowest possible batch size. This was the reason that during the thesis the data sets size was changed from 512×512 to 256×256 parts. Setting the base size and crop size parameters both to 256 and thus being able to use a wider range of instances.

Using different instances and algorithms yielded infrequent behaviours. Looking at the results in the thesis its apparent that something is off. This is because of the inference time being widely infrequent. During this thesis it has been shown that models trained by the PSP algorithm requires significantly more time when running inference. Additionally during the times I trained models I received different average inference times when training the very same network on different GPU supported instances. A model trained with ml.p2.xlarge instance running inference on images with side 256 provided the average inference time of 0.13s for the FCN algorithm. Training the very same model two weeks later using a ml.p3.2xlarge (way stronger) instance yielded the average inference time of 2.7s.

AWS results compared to other results

FCN can be viewed as the best alternative due to the fact that it has the fastest average

inference time combined with lowest training time with reassembling performance of its competitors. In the thesis Semantic Segmentation of Iron Ore Pellets with Neural Networks by Terese Svensson (section 4.3) [2], similar parameters were used to train the networks. Terese managed to achieve mAcc of 91.7 and mIOU of 64.3 running a PSP network with 200 epochs, 1 batch size and a learning rate of 0.001. Amazon SageMaker was not able to train a PSP network running batch size of one but it continuously crashed until the batch size was increased to two or above. Compared to the results found presented in table 8, it can be concluded that the algorithms trained with Amazon SageMaker performs worse. It is a possibility that it is due to faulty parameter tuning, however it seems to be that the technology is too young and not really ready to compete with a fully customisable implementation.

AWS Lambda
Wheather it's a good solution or not to utilize a completely serverless lambda implementation, solely relying on lambda and endpoints without any help from Amazon Elastic Compute Cloud is to be decided by the developers. Lambda provides the benefit of not needing to setup or maintaining the compute instances, at a slight cost for a network with low inference time. The main drawback of lambda occurs when the inference time for the trained network increases. In the case provided in section 6.3.5 the overhead cost of running lambda ends up at \$0.03 per pellet. However, the cost is strictly bound to the inference time of the network and would that increase by a factor of X, so would the cost.
Taking this into account when comparing the theoretical values to the ones from the real world example 6.4, it can be of interest to evaluate how using a more performant computer instance would impact the overhead time for inference. Alternatively, if possible, improve the code to reduce the overhead time. The real implementation utilizes 3 times as much time as the theoretical implementation.

7.3 Cloud

Regarding cloud architectures, the different types generally provides a deeper understanding of the system at hand. This system is built relying solely on AWS infrastructure (IaaS). Tt utilizes several platform acquired tools, like SageMaker, for training and hosting models (PaaS). The final solution is then hosted as a software in a private cloud without any ability to change or manage the underlying hardware with some limited configuration settings (SaaS). An alternative solution to the one presented in section 5.3.1 would be to mainly use lambda and step functions to start execution. You could either skip using SageMaker completely by starting a

Amazon Elastic Compute Cloud (Amazon EC2) instance from where you run your own algorithms, or you could make use of an endpoint but have all the preprocessing happen on an Amazon EC2 instance. Middle ground would be to use SageMaker to train the model and host the model manually inside an Amazon EC2 instance. I would advice against the last option since you miss out on the customization when training the network manually without taking any real advantages of SageMaker when running it in EC2. If I were to do things differently I would have started with doing theoretical calculations of both a lambda and an EC2 solution before doing any implementation at all. This is easier said than done because you need quite extensive knowledge of AWS systems to be able to in a good way evaluate them and know if its even a possible way of implementation at all. However, it would have provided a good estimate of what the cheapest implementation would have been and I would be able to say I went the path I did because of that. I learned as I went and got reasonably good results after the discovery of PSP being a bad alternative because of its incredibly slow inference time.

There was quite a few minor problems during the implementation of the lambda system, most everyday stuff like code not running but also some major ones where I did not understand how I was supposed to handle the predicted result from the image. A lot of the documentation regarding Semantic Segmentation feels undone, it's for the most part there to be found but in some cases it's far from obvious. Another limitation was the lambda quotas found in table 1, these forced some extensive work-arounds that would not have been needed if EC2 was used. This part is hard to evaluate since it's so tightly bound to the inference time of networks. To my knowledge, no similar solution exists making it impossible to evaluate its performance in regard to others. One has to take into account as well that the cloud system is very dependant on the inference time of the network and it would be hard to purely evaluate the cloud-parts.

8 Conclusions and Future Work

8.0.1 Conclusions

Amazon Web Services annotation tool is not efficient enough for this kind of data, it would require large amount of time for whoever is doing the annotation. Illastik provides automated but supervised pixel and object classification which receives the best results from the least amount of work. Fully-Convolutional Network (FCN) shows the most potential looking at both inference time and training time, it was the fastest network to train and performed within 1% worse than the winner DeepLabV3 when comparing inference times. However, the mIOU and mAcc metrics are much lower than expected and compared to earlier thesis by Terese Svensson [2]. Training a model locally and uploading the model to SageMaker should be investigated. The serverless cloud solutions shows great potential in usability and maintainability, as long as the inference times are kept low I personally would vouch for the lambda solution. With an inference time above 2 seconds I would start looking at replacing lambda with an Amazon Elastic Compute Cloud alternative.

8.0.2 Future Work

The suggestions for future work that is given in this section is based on the work done during this thesis.

Firstly, current implementation lacks a notification system. Once a set of images has been processed the user who uploaded them or someone else might wish to be notified that there is new results to be viewed.

Secondly, the row by row execution provided by AWS Step Functions in collaboration with AWS Lambda offers a second layer of security. If the system were to crash it's possible to add an implementation to restart from the latest successful row because the progress is being written to a CSV file.

Thirdly, uploading files to AWS S3 from GUI is currently done using presigned URLs, however multipart is not implemented. Doing so would provide another level of security aiding if the network were to fail part of the upload, removing the need to re-upload the whole file again.

Fourthly, it is possible to increase the speed by forcing concurrency going around AWS SageMakers Semantic Segmentation limit of not being parallelizable by altering the Lambda code to split the work between several endpoints row-vise.

Fifthly, add support for user to choose files for inference from uploaded material rather than running inference directly on newly uploaded files. Along with support to start inference on multiple files at once without the need to power on and off

endpoints for each one.

Sixthly, a database to store old results along with metrics. Possibly tied to the end-users GUI.

Furthermore, to investigate the network load and make a cost/benefit analysis of using Amazon S3 storage.

Lastly, improving the results of trained networks by training the model locally and then hosting that model in either an endpoint or Amazon Elastic Compute Cloud instance dependant of their separate performance.

Special Terms

Amazon Web Services (AWS) Amazon Web Services (AWS) is the world's most comprehensive and broadly adopted cloud platform, offering over 175 fully featured services from data centers globally. [21]. 2

Acronyms

Amazon EC2 Amazon Elastic Compute Cloud. iv, 38, 57–60

ASPP Atrous Spatial Pyramid Pooling. 14, 15

AWS Amazon Web Services. i, 20, 25, 49, 50, 59

CM Confusion Matrix. iii, vi, 17, 45

CRF Conditional Random Field. 14

DT Distance Transform. iii, 18

FCN Fully-Convolutional Network. i, iii, vi, 11–13, 15, 46, 59

GPU Graphics Processing Unit. 42, 49

LKAB Luossavaara-Kirunavaara AB. 1, 23

PSP Pyramid Scene Parsing. iii, vi, 12, 13, 15, 46

References

[1] *Hybrit - toward fossil-free steel production*, Available at https://www.jernkontoret.se/en/vision-2050/carbon-dioxide-free-steel-production/ (2020/05/26).

[2] T. Svensson, "Semantic segmentation of iron ore pellets with neural networks", Master's thesis, 2019.

[3] *Data ductus - about*, Available at https://www.dataductus.com/about-us/ (2020/05/28).

[4] *Lkab in brief*, Available at https://www.lkab.com/en/about-lkab/lkab-in-brief/ (2020/05/28).

[5] D. W. et al, "Iron ore pellet characterization through digital microscopy", Jan. 2008.

[6] K. A. et al, "Automatic classification of hematite in iron ore", Jun. 2015.

[7] R. M. C. et al, "
characterization of iron ore pellets by multimodal microscopy and image analysis", *REM - International Engineering Journal 71*, Apr. 2018.

[8] M. Simonsson, *Quantitative characterisation of iron ore pellets with optical microscopy and machine learning*, Available at https://www.sipstrim.se/project/quantitative-characterisation-of-iron-ore-pellets-with-optical-microscopy-and-machine-learning/ (2020/06/04).

[9] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks", *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[11] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, no. 1, 2015.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network", *CoRR*, vol. abs/1612.01105, 2016. arXiv: 1612.01105. [Online]. Available: http://arxiv.org/abs/1612.01105.

[14] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation", *CoRR*, vol. abs/1706.05587, 2017. arXiv: 1706.05587. [Online]. Available: http://arxiv.org/abs/1706.05587.

[15] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J.Garcia-Rodriguez, "A review on deep learning techniques applied to semantic segmentation", *arXiv:1704.06857v1 [cs.CV] 22 Apr 2017*, 2017.

[16] *What is cloud computing?*, Available at https://aws.amazon.com/what-is-cloud-computing/ (2020/05/28).

[17] P. Mell and T. Grance, "The nist definition of cloud computing", *National Institute of Standards and Technology*, p. 3, Sep. 2011.

[18] S. Berg, D. Kutra, T. Kroeger, C. N. Straehle, B. X. Kausler, C. Haubold, M. Schiegg, J. Ales, T. Beier, M. Rudy, K. Eren, J. I. Cervantes, B. Xu, F. Beuttenmueller, A. Wolny, C. Zhang, U. Koethe, F. A. Hamprecht, and A. Kreshuk, "Ilastik: Interactive machine learning for (bio)image analysis", *Nature Methods*, Sep. 2019, ISSN: 1548-7105. DOI: 10.1038/s41592-019-0582-9. [Online]. Available: https://doi.org/10.1038/s41592-019-0582-9.

[19] *Uploading objects*, Available at https://docs.aws.amazon.com/AmazonS3/latest/dev/UploadingObjects.html (2020/08/05).

[20] *On-demand pricing*, Available at https://aws.amazon.com/ec2/pricing/on-demand/ (2020/09/21).

[21] *Cloud computing with aws*, Available at https://aws.amazon.com/what-is-aws/ (2020/08/05).

Appendix 1, Annotation POC 1

```
import numpy as np
import cv2

#dirty dirty code
image = cv2.imread('A1_s_0093_006.png')
original = image.copy()
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))

#pores
image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
lower = np.array([0, 0, 0], dtype="uint8")
upper = np.array([197, 255, 30], dtype="uint8")
mask = cv2.inRange(image, lower, upper)

opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
filtered = cv2.filter2D(opening, -1, kernel)
cnts = cv2.findContours(filtered, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

pores = 0
for c in cnts:
    pores += cv2.contourArea(c)
    cv2.drawContours(original,[c], 0, (0,0,172), thickness=-1)

#olivine
image = cv2.cvtColor(original, cv2.COLOR_BGR2HSV)
lower = np.array([0, 0, 78], dtype="uint8")
upper = np.array([179, 255, 85], dtype="uint8")
mask = cv2.inRange(image, lower, upper)

opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
filtered = cv2.filter2D(opening, -1, kernel)
cnts = cv2.findContours(filtered, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

olivine = 0
```

```
for c in cnts:
    olivine += cv2.contourArea(c)
    cv2.drawContours(original, [c], 0, (0, 85, 0), thickness=-1)


#magnetite
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
image = cv2.cvtColor(original, cv2.COLOR_BGR2HSV)
lower = np.array([0, 0, 120], dtype="uint8")
upper = np.array([179, 255, 165], dtype="uint8")
mask = cv2.inRange(image, lower, upper)

opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
filtered = cv2.filter2D(opening, -1, kernel)
cnts = cv2.findContours(filtered, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

magnetite = 0
for c in cnts:
    magnetite += cv2.contourArea(c)
    cv2.drawContours(original, [c], 0, (255, 0, 0), thickness=-1)

#epoxy
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))
image = cv2.cvtColor(original, cv2.COLOR_BGR2HSV)
lower = np.array([0, 0, 0], dtype="uint8")
upper = np.array([179, 254, 100], dtype="uint8")
mask = cv2.inRange(image, lower, upper)

opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
filtered = cv2.filter2D(opening, -1, kernel)
cnts = cv2.findContours(filtered, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

epoxy = 0
for c in cnts:
    epoxy += cv2.contourArea(c)
    cv2.drawContours(original, [c], 0, (0, 0, 255), thickness=-1)
```

```
#hematite
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
image = cv2.cvtColor(original, cv2.COLOR_BGR2HSV)
lower = np.array([0, 0, 0], dtype="uint8")
upper = np.array([179, 254, 255], dtype="uint8")
mask = cv2.inRange(image, lower, upper)

opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterations=1)
filtered = cv2.filter2D(opening, -1, kernel)
cnts = cv2.findContours(filtered, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
cnts = cnts[0] if len(cnts) == 2 else cnts[1]

hematite = 0
for c in cnts:
    olivine += cv2.contourArea(c)
    cv2.drawContours(original, [c], 0, (0, 255, 255), thickness=-1)


# print("pores: {}".format(pores))
cv2.imshow('mask', mask)
cv2.imshow('original', original)
cv2.imshow('opening', opening)
cv2.imshow('filtered', filtered)
cv2.imwrite('pore.png', original)
cv2.waitKey()
```

Appendix 2, Annotation POC 2

```
from PIL import Image, ImageFilter
import numpy as np

img = Image.open('first_train.png')
img = img.convert("RGB")
datas = img.getdata()

hematite = (180, 180, 180)
pore = (30, 30, 30)
magnetite = (145, 145, 145)
epoxy = (70, 70, 70)
border = (110, 110, 110) #this is fictional
treshold = 25


newData = []
for item in datas:
    R = item[0]
    G = item[1]
    B = item[2]
    if abs(hematite[0]-R) <= treshold or abs(border[0]-R) <= treshold:
        newData.append((255, 255, 0)) #yellow
    elif abs(pore[0]-R) <= treshold:
        newData.append((170, 0, 0)) #dark red
    # elif abs(border[0]-R) <= treshold:
    #     newData.append((0, 0 ,0))  #black
    elif abs(magnetite[0]-R) <= treshold:
        newData.append((0, 0, 255)) #blue
    elif abs(epoxy[0]-R) <= treshold:
        newData.append((255, 0, 0))
    else:
        newData.append(item)

img.putdata(newData)
img.save("no-filter.png", "PNG")
```

```
# Median filter to remove outliers
im3 = img.filter(ImageFilter.MedianFilter(3))
im3.save("filter3.png")
```