Improving training of deep learning for biomedical image analysis and computational physics

Karl Bengtsson Bernander
Improving training of deep learning for biomedical image analysis and computational physics

Karl Bengtsson Bernander
karl.bengtsson.berlander@it.uu.se

December 2021

Centre for Image Analysis
Department of Information Technology
Uppsala University
Box 337
SE-751 05 Uppsala
Sweden

http://www.it.uu.se/

Dissertation for the degree of Licentiate of Philosophy in Computerised Image Processing

© Karl Bengtsson Bernander 2021
ISSN 1404-5117
Printed by the Department of Information Technology, Uppsala University, Sweden
Abstract

The previous decade has seen breakthroughs in image analysis and computer vision, mainly due to machine learning methods known as deep learning. These methods have since spread to other fields. This thesis aims to survey the progress, highlight problems related to data and computations, and show techniques to mitigate them.

In Paper I, we show how to modify the VGG16 architecture to be equivariant to transformations in the p4 group, consisting of translations and specific rotations. We conduct experiments to investigate if baseline architectures, using data augmentation, can be replaced with these rotation-equivariant networks. We train and test on the Oral cancer dataset, used to automate cancer diagnostics.

In Paper III, we use a similar methodology as in Paper I to modify the U-net architecture combined with a discriminative loss, for semantic instance segmentation. We test the method on the BBBC038 dataset consisting of highly varied images of cell nuclei.

In Paper II, we look at the UCluster method, used to group subatomic particles in particle physics. We show how to distribute the training over multiple GPUs using distributed deep learning in a cloud environment.

The papers show how to use limited training data more efficiently, using group-equivariant convolutions, to reduce the problems of overfitting. They also demonstrate how to distribute training over multiple nodes in computational centers, which is needed to handle growing data sizes.
Acknowledgments

This research was funded by WASP — the Wallenberg AI, Autonomous Systems and Software Program.

A big thanks to those who helped supervise my thesis and help me write the papers: Ingela Nyström, Joakim Lindblad, Robin Strand, Carolina Wahlby and Nataša Sladoje. Thank you to Liselott Dominicus van der Bussche and the rest of the student advisory group for their support. Thank you to Fred Hamprecht, Ewert Bengtsson, Håkan Wieslander, Johan Öfverstedt, Elisa-beth Wetzer and others who helped me at various stages of the project. Thank you to Olga Gudnasdottir and the other collaborators for widening my knowledge in the particle physics project. Thank you to all of the WASP course mates, travel buddies and organizers. Thank you to all the people I’ve met working at CBA, the IT department and to the research community in general.

Thank you to my friends and family - my mother and her husband, my brother, my aunt and my grandparents. Thank you to all of the friends I’ve met over the years - you are too many to list, but your support was invaluable. Nothing is ever accomplished alone.
List of Papers

This thesis is based on the following papers:


III K.B. Bernander, J. Lindblad, R. Strand, and I. Nyström. ”Rotation-Equivariant Semantic Instance Segmentation”. Submitted manuscript, 2021
# Contents

1 Introduction .................................................. 1
   1.1 Deep learning ............................................. 2
       1.1.1 CNN classifier example ............................ 3
       1.1.2 Problems and limitations .......................... 5
   1.2 Aim ....................................................... 6
   1.3 Equivariance ............................................. 6
   1.4 Cloud machine learning ................................... 8
   1.5 Papers included in this thesis ........................... 9
       1.5.1 Replacing data augmentation with rotation-equivariant CNNs in image-based classification of oral cancer ........................................ 9
       1.5.2 Distributed training and scalability for the particle clustering method UCluster ............................. 9
       1.5.3 Rotation-Equivariant Semantic Instance Segmentation 10

2 Classification .................................................. 11
   2.1 Diagnosing oral cancer .................................... 11
   2.2 Classifying using CNNs and G-convolutions ............... 13
   2.3 CNN vs GCNN classifiers ................................... 14

3 Segmentation .................................................. 17
   3.1 The BBBC038 dataset ...................................... 18
   3.2 Segmentation by border delineation ........................ 18
   3.3 Semantic segmentation by CNNs ............................ 19
   3.4 Rotation-equivariant U-net ................................ 20
   3.5 Experiments ............................................... 20

4 Clustering .................................................... 21
   4.1 Instance segmentation with U-net and a discriminative loss function ........................................ 22
   4.2 Equivariance of instance segmentation ........................ 23
   4.3 Standard vs rotation-equivariant U-net ........................ 24
   4.4 Jet classification with UCluster ............................. 26
4.5 Distributed deep learning ....................... 26

5 Discussion and Conclusions .......................... 29
5.1 Future work ........................................ 30

References ............................................. 31
Chapter 1

Introduction

Imagine yourself in the position a space engineer, designing the navigational system of a satellite orbiting the earth. Such a satellite typically orients itself by imaging the stars around it, analyzing them, and matching them against an internal database of known constellations and galactic coordinates. To achieve this, many technologies are needed. One of them is computerised image analysis — the extraction of meaningful information from images.

In 2014, while writing my master thesis, I worked on extending such a system to also detect the presence of other man-made objects orbiting the earth [2]. These objects would show up as straight lines in the images, since imaging in space typically uses long exposure times. The system uses a star tracker camera, which is a camera adapted for navigation in space. I took an image with the camera and added different types of noise to simulate the harsh conditions that sometimes occur in space, such as cosmic rays. See an example in Figure 1.1. Then, I designed a method that would filter the images in steps and register only the strong responses. These responses were evaluated against the ground truth, indicating the presence of an object in the scene. The purpose behind the system was to assist in cataloging objects orbiting the earth — so called space debris. This debris poses a threat by possible chain collisions of debris and active satellites.

Back then, a workflow like this was typical for object detection classifiers. You would typically design a series of filters, inspecting each step visually to see that the result was in line with the expectations of what the filter was designed to do. Since then, a lot has happened.
1.1 Deep learning

The ImageNet database consists of not only one class, like in the previous section, but multiple, including golf balls, planetariums and flamingos \[12\]. Furthermore, the objects in the images occur in very different settings - e.g. differing in lighting, positioning, orientation, and occlusion. Designing an algorithm to account for all the different classes and conditions was extremely challenging by following the traditional methodology. Usually, you would find yourself fine-tuning the sensitivity of each step of the algorithm, by e.g. adjusting the parameters of a laplace filter for an edge detector. Usually there would be tradeoffs - you could design a classifier that was extremely good at detecting flamingos in bright lighting with high contrasts, but very bad at detecting them in darker and noisier conditions. Humans do not find this difficult, so something could be done differently.

In 2012, the problem was solved in a very different way. AlexNet used deep Convolutional Neural Networks (CNNs), reducing the misclassification rate from the previous state of the art (26.2%) to a new record (15.3%) \[20\]. CNNs and related methods based on machine learning collectively called deep learning would go on to consistently outperform other practices in computer vision. In the following years, deep learning would spread to
1.1. Deep learning

Figure 1.2: The VGG16 convolutional neural network architecture. The white blocks are combined convolutional and ReLu layers, the red blocks are max pooling layers, and the green blocks are combined fully connected and ReLu layers. Image sourced from [10].

In other fields, including biomedicine, particle physics and natural language processing. Also, combining deep learning with reinforcement learning has seen computers beat human players in games of Go, using AlphaGo [28] for the first time in history.

1.1.1 CNN classifier example

CNNs for 2D image processing are constructed by stacking layers of neurons, one after another, in input, hidden and output layers [15]. An example can be seen in Figure 1.2 [10], illustrating the VGG16 architecture [29]. The input layer is the input image. The hidden layers are connected to previous hidden layers, or the input image for the first hidden layer. Each neuron has a local receptive field, which connects it to a small 2D patch of neurons in the previous layer. At the other end of the network, the neurons are fully connected as each neuron connects to all the neurons of the previous layer. These final layers are also linear, meaning there is no spatial 2D concept of connectivity. The output layer outputs probabilities for each class occurring in the image.

To classify an image, the image is input to the network. The neurons in the first hidden layer processes the input layer by filtering with multiple convolution kernels. These multiple kernels constitute the channels of the layer. Each kernel is used in a convolution operation. Each neuron receives its value by filtering the values in the neurons of its receptive field with the kernel. This is performed for each hidden layer. There are also pooling layers, like the max layer, which outputs only the maximum value of the neurons in the receptive field. Activation layers are additionally used to discard certain values, like the ReLu layer. In the fully connected layers, a 1D kernel is used instead of a 2D kernel.
1.1. Deep learning

Figure 1.3: CNN architectures tend to learn a hierarchy of features. Earlier layers tend to learn simpler features, like edges, while later layers tend to learn more abstract features. This mechanism for visual perception is similar to the visual cortex in the brain. Image sourced from [31].

To learn the weights, the network needs to be trained. First, the input images are fed to the network and classified. The output classes are then compared with a set of ground truth labels, and a loss function is calculated, indicating the size of the error. Then, the error is back-propagated. During this phase, the error function is derived with respect to its weights. This is achieved by computing the gradient layer-wise, using the chain rule. The weights are then updated using gradient descent or similar methods. This process is repeated until the loss does not decrease, or for a fixed number of epochs. An epoch is a complete pass of the training set. After the training phase, images in the test set are classified by the network.

During training, earlier layers will tend to learn simple features like edges, while later layers will tend to learn more complex features. This is illustrated in Figure 1.3 [31]. In the fully connected layers, the combination of features are compared to a number of class templates, and the highest scoring class obtains the highest probability using a softmax function. Returning to the example of detecting the presence of objects orbiting the earth, a deep learning approach would seek to learn the filters that were manually designed and tested.
1.1. Deep learning

1.1.2 Problems and limitations

Deep learning approaches are not always straightforward to implement. For them to be effective, typically you need a lot of labelled data. This is not always easy to find, especially if you are working with sensitive data relating to personal information, or data needing expert labelling, such as annotations by pathologists for automated diagnosis of cancer based on imaged cell samples.

Then, while this is not a unique problem to deep learning, there is the problem of underfitting and overfitting. Underfitting usually means having low accuracy on both the training and test sets, usually resulting from the model being too simple. Overfitting means the network has learnt the training examples, but fails to generalize to unseen examples in the test set. This can come from having a too complex model using many weights, and can be mitigated with techniques such as regularization. The balance between underfitting and overfitting is also known as the bias-variance tradeoff.

Furthermore, deep learning does not align very well with how humans learn to distinguish objects. Small children do not need to see thousands of objects to be able to accurately classify new objects. Usually a few examples are sufficient. Therefore, many are advocating for approaches based on unsupervised learning, which relies on the statistics of the input data as a supervisory signal [17]. This would also be more consistent with how humans learn from a neuro-scientific viewpoint. In human brains, activation patterns are usually associated with a concept — thinking about a lemon produces similar patterns as smelling it or seeing it. These activation patterns are usually formed by association of the different signals over time. This learning rule is called Hebbian learning [16], or ”cells that fire together, wire together”, and is an example of timing-dependent unsupervised learning.

Finally, you need powerful computational systems. These systems are becoming more available, but can still be hard to get access to, even in established research environments. For many problems, you also need time, since the training phase can take several days to finish. Research over the last 70 years has shown that methods in machine learning that are better able to leverage computation outperform those that do not [30]. A common criticism to deep learning is that there is little intelligence to it - you simply push a lot of data into a network, tune a lot of weights until the results look right, while using a lot of computational power. Regardless of these viewpoints, it is clear that there is a need for more intelligent methods in their use of limited resources.
1.2. Aim

The main aim of this thesis is to show how the needs for massive datasets as well as powerful computational setups can be mitigated when performing deep learning. Another aim is to investigate how overfitting can be reduced by using the weights of CNNs in a more structured way. To do this, we introduce the concept of equivariance in a CNN setting, as well as the ecosystem of current cloud and distributed deep learning technologies.

1.3. Equivariance

Imagine we have a classifier CNN that is trained to detect flamingos. If we shift the flamingo in the input image, i.e., move it sideways, up or down, the classifier will still recognize the flamingo. This property is called translation invariance, meaning no matter how the input is shifted, the output is identical. This property follows from the way convolutions are performed. When convolving inside a CNN, the kernel is shifted over the feature map.
in a left-to-right, up-to-down type of motion. This means that the kernel, which is adapted to detect a specific feature, covers the entire feature map.

This property does not hold for other rigid transformations, such as reflections and rotations. A rigid transformation is defined in this context as any transformation that preserves the Euclidean distance between pixel coordinates. The consequence of the lack of such a property is that any network will need to learn each such transformation separately. If the flamingo is rotated in the input image, the network needs to learn a kernel that detects the rotated version of the flamingo. This holds for all levels in the network - any rotated features need to be learnt separately. This implies that the minimum size of the number of weights need to increase as well. This is also in stark contrast to how humans learn to recognize objects and understand concepts.

Invariance is a special type of equivariance. The layers inside CNNs are equivariant to translations. This means that if you shift the input feature map and then perform the convolution, you will get the same result as if you perform the convolution and then shift the resulting feature map. We say that the convolution operation \( C \) commutes with the translation transformation \( T \):

\[
T(C(x)) = C(T(x))
\] (1.1)

where \( x \) is the feature map used as input. Another example of a commutative operator is multiplication: \( 2 \times 3 = 3 \times 2 \). However, the convolution operator does not commute with rotations. To construct a rotation-invariant classifier, this property needs to hold for any layer in the network.

CNNs have been extended to provide these properties, using, e.g., Steerable Filters [33] or Scattering Convolution Networks [5]. The focus in this thesis is Group Equivariant Convolutional Networks, which employs G-convolutions [9]. A G-convolution has two main properties. First, it is defined on a specific symmetry group. An example is the p4 group, which consists of all combinations of translations and 90-degree rotations. Secondly, a G-convolution is performed on this symmetry group. This means that the kernel is not only shifted across the feature map. It is shifted across the four multiples of 90-degree rotated versions of the feature map. Combined with other special layers like a pooling layer over groups, this makes it possible to construct network architectures that are equivariant to rotations of 90 degrees or other rigid transformations. An example can be seen in Figure 1.4. The theory of group equivariant networks has been extended to a framework called General E(2) - Equivariant Steerable CNNs [32], which forms another important foundation for this thesis. An implementation of E2CNN is available as a library on top of common deep learning libraries.
1.4 Cloud machine learning

Many projects are today driven by the need to analyze big datasets. This holds for computerised image analysis and many other application domains. This need puts higher and higher demands on the hardware required to store and analyze the data. As costs for these setups increase, as well as the complexity in managing them, cloud computing becomes increasingly attractive. This can come with many advantages for fast-moving projects: you can rent as much resources as you need when you need it. Accessing the data and files is usually as simple as working on your local computer as high Internet speeds have become mainstream.

If you are working with large datasets, you can distribute the computations over multiple clusters - this speeds up the computations, and usually has built in failure tolerance mechanisms for hardware failures. When the project is completed, the hardware is free for others to use, and you are not left with redundant equipment. The downsides is that you lose control over the project - typically you have less freedom in how to configure the system. The code and data is uploaded to the Internet, which can be problematic if you are working with sensitive data, since it is well known that intelligence agencies and corporations surveil internet communications.

Ericsson OpenStack and SNIC (the Swedish National Infrastructure for Computing) offer different computing platforms. Databricks, which is built on top of Apache Spark [35], is another cloud computing platform and was developed as an alternative to Google’s MapReduce [11]. It provides programming environments in Python, R, Scala, and SQL. The code and data can be deployed to the major cloud centers: Microsoft Azure, Amazon Web Services and Google Cloud. It also provides a machine learning environment, with the possibility to perform distributed deep learning using HorovodRunner. Another example is Google Colab, which execute on Google Cloud. This provides access to projects centered around notebooks running on a configurable Linux environment, and is typically used for collaborative prototyping.
1.5 Papers included in this thesis

The remainder of this thesis is structured around the three papers, which are listed below. The following chapters are dedicated to the investigating the aims of the thesis in detail, applied to problems mainly in biomedical computerised image analysis, but also particle physics. The final chapter summarizes the results, and concludes with a discussion about how the aims of the thesis have been met, as well as possible future research directions.

1.5.1 Replacing data augmentation with rotation-equivariant CNNs in image-based classification of oral cancer


This paper investigates how CNNs equivariant to the p4 group can replace data augmentation schemes with multiples of 90-degree rotations. It also shows that these networks reduce overfitting quantitatively, and investigates how much data is needed for sufficient accuracy in an image classifier based on the VGG16 architecture. The data used is images of cell nuclei from patients who are healthy or in the process of developing oral cancer. The main idea originates from Natasà Sladoje and Joakim Lindblad. I designed the experiments and implemented the methods. I wrote the paper jointly with Joakim Lindblad, Robin Strand and Ingela Nyström.

1.5.2 Distributed training and scalability for the particle clustering method UCluster


This paper explores how Horovodrunner in a Databricks environment can be used to speed up training of the deep learning method UCluster. This also provides scalability, adapting the method to larger datasets. The data is particle collisions from the Large Hadron Collider of CERN. Rebeca Gonzales Suarez and Raazesh Sainudiin were responsible for coordinating the project, as well as providing access to the databricks environment. I suggested the use of the Horovodrunner method and worked on the evaluation script. Daniel Gedon and Colin Desmarais implemented the core of the method. Olga Sunneborn Gudnadottir sourced the idea, worked on the evaluation script and wrote the paper. All participants reviewed the paper.
1.5.3 Rotation-Equivariant Semantic Instance Segmentation


This paper shows how to make a semantic instance segmentation network that is equivariant to rotations of multiples of 90 degree rotations, based on the U-net architecture. It also provides experiments to show this property empirically, as well as comparisons to an ordinary CNN, with and without data augmentation consisting of rotations of multiples of 90 degrees. The main idea was jointly developed among me, Robin Strand, Joakim Lindblad and Ingela Nyström. I implemented the method, designed the experiments and wrote the paper. The co-authors reviewed the paper.
Chapter 2

Classification

Classification, or object recognition, is one of the core problems in the field of computerised image analysis [13]. The task is simple: given a set of items, what is most likely to be represented by them? Most commonly the input is in the form of images, but it could also be features, regions of images, or even individual pixels. The output of the classifier is usually a physical object, e.g. a flamingo, a car, or a tree. The granularity of the classes depends on the specific classifier and on what it is trained to detect. The classifier could learn to further separate a tree into different species, but it could also learn to detect something else entirely, such as the season of the year based on the number and colours of the leaves.

In this chapter, we will look at a specific classification problem in biomedical image analysis. This mostly relates to Paper I. We will introduce the dataset - how it was prepared and preprocessed, how the labels were chosen, how it was partitioned, and how it was augmented. We will then look at a standard CNN architecture for classifying unseen data. After that, we will modify the architecture to be equivariant to more rotations of multiples of 90 degrees in addition to translations. Following that, we compare the performance between the architectures related to data augmentation and overfitting. We close the chapter by summarizing what we have learnt.

2.1 Diagnosing oral cancer

Today, classification is used in more and more medical settings, e.g., radiology and pathology. An example of the latter is the early diagnosis of cancer in the oral cavity. To screen for this, medical professionals scrape cells from the inside of a patient’s mouth. The sample is then put in liquid vials, stained and placed on a glass slide. Thereafter, it is inspected through a microscope. An experienced pathologist can diagnose the sample
2.1. Diagnosing oral cancer

Figure 2.1: Example images from the oral dataset, consisting of cell nuclei from patients who are healthy or diagnosed with cancer in the oral cavity.

as cancerous or healthy by characterizing, e.g., the ratio of the cell nuclei compared to the cytoplasm, the shapes of the cells, and the number of cells in different stages of the cell cycle. This process can be tedious and time-consuming, and it can be difficult to scale up the screening as experienced pathologists are not always easy to find. Hence, automating the process is highly desirable and could save many lives.

To automate the diagnosis of oral cancer, the oral dataset was created \cite{21} \cite{34}. Here, samples came from patients with various traits, some healthy and some diagnosed with oral cancer. The cell samples were imaged, and each image was centered around a nucleus with the size of 80 by 80 pixels. Example data can be seen in Figure 2.1. The images were also converted to grey-scale before training and testing.

Since a diagnosis of cancer is given to a patient based on many factors, the images are weakly labelled. This means that all of the individual cells are given the same label, even if some cells coming from a cancerous patient might be healthy and vice versa. On the other hand, an automated method might discover malignancy associated changes that are difficult for humans to detect, given the knowledge that the patient developed cancer at a later stage \cite{18}. The images are partitioned into a training set of 8508 images from six patients and a test set of 9942 images from six other patients. Half of the twelve patients are healthy and half have a cancer diagnosis. These types are balanced evenly among the training and test set.
2.2 Classifying using CNNs and G-convolutions

A standard approach to create a classifier for the oral dataset today is to use a CNN. There are two important aspects in designing the network. First, it needs to have sufficient depth in order to fit the hierarchical representation of features. The VGG16 architecture, introduced in Chapter 1 and in Figure 1.2, is an excellent candidate. Secondly, it is very important that the network is trained with a sufficient amount of data, which usually means thousands of examples.

A common strategy for increasing the amount of training data is to use data augmentation [27]. The main idea behind data augmentation is that transforming the data will not change the labels. If we rotate a picture of a flamingo, the image should still be classified as containing a flamingo. The same holds for other transformations, such as reflections or scalings. This makes it possible to increase the amount of training data by creating copies of the images, transforming them and keeping their labels intact. The process is visualized in Figure 2.2 [23]. However, some of these augmentations can introduce interpolation artifacts. One example is rotations by 45 degrees. There is also a risk of overfitting, meaning there is a high accuracy score on the training set, but a low accuracy score on the test set. This is due to the need to increase the model capacity to learn all the additional transformations of the data.
2.3 CNN vs GCNN classifiers

Table 2.1: Overfitting measurements

<table>
<thead>
<tr>
<th>Network</th>
<th>Overfitting ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN with data augmentation</td>
<td>1.82</td>
</tr>
<tr>
<td>GCNN without data augmentation</td>
<td>1.69</td>
</tr>
</tbody>
</table>

Another way to achieve the same effect as data augmentation is to design the networks to be equivariant to those same transformations. One way to attain this is to replace each layer with group-equivariant convolutions, together with their corresponding representations and pooling layers throughout the network. Then, as explained in Chapter 1, the transformations do not need be learnt during training. This comes with an additional advantage. Since only one kernel with weights is learnt instead of one for each transformation, there is an increased weight sharing among filter kernels in the network, which can reduce the risk of overfitting.

2.3 CNN vs GCNN classifiers

A network like VGG16 is fairly straightforward to implement in python libraries such as tensorflow and pytorch. This network can then extended using the e2cnn library, which provides the possibility to build networks that are equivariant to e.g. rigid transformations. We call the first network the baseline network, or CNN, and the second network, which uses group-equivariant convolutions, the GCNN. Both networks use the same hyperparameters and are trained in an identical fashion. Paper I outlines these experiment in detail.

To verify that the GCNN is at least as accurate as the CNN combined with data augmentation, we are interested in performing four comparisons. Our hypothesis is that the CNN with no data augmentation performs the worst, and that adding data augmentation improves the accuracy. The GCNN should be at least as accurate as the CNN with data augmentation, and adding data augmentation to the GCNN should provide no further benefit.

From the result section about phase 1 experiments in Paper I, illustrated in Figures 2.3 and 2.4, it is clear that the GCNN performs the best, with or without data augmentation, in line with our expectations. It seems however that adding data augmentation to the CNN has a marginal effect on the results. It is unclear why, but it could be that more data augmentation is needed to improve the results for the baseline network.
These results are further quantified by measuring the overfitting ratio, defined as the accuracy on the training set divided by the accuracy on the test set, for the last epoch. The results are seen in Table 2.1. It is evident that the GCNN without data augmentation yields a lower overfitting ratio than the CNN with data augmentation, indicating less overfitting in the rotation-equivariant network.

Figure 2.3: Classification testing accuracy for an ordinary CNN. The plots show the experiments with a) no data augmentation and b) with data augmentation. Dashed lines are runs with identical settings, the solid line is the mean of these runs.

Figure 2.4: Classification testing accuracy for a GCNN. The plots show the experiments with a) no data augmentation and b) with data augmentation. Dashed lines are runs with identical settings, the solid line is the mean of these runs.
In Chapter 2, we discussed classification in image analysis. In this chapter, we move on to segmentation, a related, but more advanced problem. Here, the main task is to partition an image into different regions, which can facilitate easier analysis of the components [13]. The components can be anything that share some similar characteristic on a pixel-level. To separate apples from oranges in a color image, we can construct an algorithm that denotes all green and red pixels as being part of the apple class, and orange pixels as being part of the orange class. The components can also be grouped together based on a more holistic understanding of the image, such as delineating the sky from a foreground in a picture of a landscape. This can be performed by looking for the sharp difference of intensity at the edge of the sky and the foreground.

We need to define three terms before we continue. By edge-based segmentation, we mean intensity-based methods such as the separation of sky from foreground mentioned above. By semantic segmentation, we mean classification of pixel regions based on some defined criteria, like the apple versus oranges example above. By instance segmentation, we mean uniquely identifying each part of the image, e.g., labelling each specific apple and orange in the apple versus oranges example.

In this chapter, we will show the evolution from edge-based segmentation algorithms to semantic segmentation methods based on deep learning. We will show a state-of-the-art semantic segmentation CNN, the U-net. We will then show how to make it equivariant to rotations of multiples of 90 degrees, in order to reduce the need for data augmentation and reduce overfitting. We conclude by summarizing what we have learnt. We will continue with instance segmentation in Chapter 4.
3.1 The BBBC038 dataset

![Image of cell nuclei and segmentation mask](image)

Figure 3.1: An image from the Broad BBBC038 dataset of a) cell nuclei and b) its segmentation mask. The contrast has been increased for clarity.

The BBBC038 dataset contains images of cell nuclei [6], see Figure 3.1. The cells originate from different animals and tissues and are in various states, such as cell division or genotoxic stress. The samples are imaged under different conditions, resulting in different illuminations, colors and sizes. Being able to automatically segment such images is of high interest to assist pathologists in various tasks. Imagine having a tool that could identify only those cells that have been infected with a specific pathogen.

This dataset shares the same underlying philosophy as ImageNet. For an algorithm to perform well, it needs to be able to generalize to many different conditions. This is similar to how a human is able to adapt their understanding of very different visual inputs. This makes for a challenging segmentation problem.

3.2 Segmentation by border delineation

One way to perform segmentation is by border delineation, using the watershed algorithm [13]. It can be conceptually understood by first considering watersheds on earth. When rain falls, water will collect in regions called watersheds, based on the topography of the landscape. Image segmentation watersheds work similarly, where the intensities of the image form the basis of the watersheds. Each region needs to be initialized by a seed, or marker.
In 2012, I worked in a project extending stochastic watersheds [3]. Here, the markers are distributed according to some stochastic function. In the context of the BBBC038 dataset, one would need one seed per cell, which is not trivial to achieve. Also, there needs to be clear intensity differences between cells, which is not always the case when they are adjacent to each other. Today, much of the field has moved on to methods based on deep learning.

### 3.3 Semantic segmentation by CNNs

In 2014, the U-net architecture was developed [24], illustrated in Figure 3.2. It is a CNN based on encoding the image into a compact representation. The decoder part of the network learns from this compact representation, while also using skip connections to gradually restore the image into its original size. The final layers outputs a semantic segmentation map in the same size and orientation as the input. Training the U-net is performed similarly as for classifier networks, with the difference that the labels are not classes but segmentation masks. This method, like all CNNs, need large amounts of data to function well. Therefore, data augmentation is commonly used to train it, with the same limitations as for classifier networks.
3.4 Rotation-equivariant U-net

Creating a semantic pixel-wise U-net that is also equivariant to rigid transformations has been accomplished in various ways [8]. Here, I provide an overview for how to create a network that is equivariant to the p4 group of translations and multiples of 90-degree rotations. This architecture is an important foundation for Chapter 4. There, a method is introduced that, in addition to semantic segmentation, also performs instance segmentation.

Again, I used the e2cnn library on top of pytorch to replace each convolutional and pooling layer with its e2cnn counterpart. One must also take care to convert the input to the layers to the correct representations, as well as ensuring to pool over the rotation channels at the end of the network.

Like in Chapter 2 on classification, a version of U-net that is equivariant to the p4 group does not need data augmentation to the transformations of the group when training. Also, it is less sensitive to overfitting since it only needs to learn one filter kernel per transformation.

3.5 Experiments

To verify that the rotation-equivariant version of U-net actually is equivariant to rotations of 90 degrees, the following procedure is followed. First, an image is segmented by the network, producing output 1. Then, the image is rotated, and then segmented by the network, producing output 2. Output 2 is then rotated by -90 degrees, and both outputs are compared pixel-wise. If the network is equivariant to rotations of 90 degrees, there should be no difference between the outputs. This is exactly the result for the rotation-equivariant version of the U-net. However, for the baseline network, on average 167.0625 pixels (i.e., 0.25 % in the 256 × 256 sized images) are different. These experiments were performed on a dataset of synthetic scattered sticks. This is reported in detail in Paper III, along with experiments on instance segmentation further described in Chapter 4.
Chapter 4

Clustering

It is possible to modify the U-net architecture by adding an additional head. This head is identical in structure to the head that outputs the semantic segmentation map, with one crucial difference. Instead of a 2-channel representation, a 16-channel representation is learnt. This representation has a greater capacity for learning more complex features, which makes it suitable for more refined processing using, e.g., clustering methods.

This is a common approach in machine learning. The main idea is to learn a latent representation with a balanced capacity, that can capture the important aspects of patterns efficiently [1]. This representation is then further analyzed to extract the information we are interested in. One such 'bottleneck' can be found between the encoder and decoder of the U-net architecture. It is also a crucial design component of auto-encoders [25] (used for compression and reconstruction) and generative adversarial networks (GANs) [14] (used for generating similar data).

In this chapter, we will look at two different applications of clustering methods in the context of deep learning. In the first part of the chapter, we will extend the U-net from Chapter 3 to also provide instance segmentation. We will also analyze its properties in terms of equivariance to different transformations. In the second part of the chapter, we will look at the UCluster method, used to group sub-atomic particles with similar properties. We will close the chapter by showing how to leverage cloud computation and distributed deep learning over multiple GPUs to speed up training.
4.1 Instance segmentation with U-net and a discriminative loss function

In Chapter 3, we defined semantic instance segmentation as not only classifying each region in the image according to class, but also according to its instance of that class. In an image of apples and oranges, each fruit pixel would be labelled both by class (apple or orange) and unique fruit. There are many ways to achieve such an instance segmentation. Here, we will use the modified U-net introduced in Chapter 3, along with a discriminative loss function \[^4\] at training time, and a clustering method at inference time.

When performing segmentation as described in Chapter 3, the cross-entropy loss function is used to calculate the misclassifications for each pixel. When performing semantic instance segmentation, we add another loss term to the cross-entropy to compute a total loss. This added term is called the discriminative loss function, and is itself divided into three parts. The discriminative loss can intuitively be explained as follows. The goal of training is to enforce that all the pixels originating from the same instance form tight clusters, and that different clusters should be separated from each other.

Figure 4.1: Clustering with a discriminative loss. Each colored circle represents a pixel that has been mapped to the instance representation. The colors represent the different instances of the image. The black circle in the middle of each cluster is the cluster center of that instance. The inner dashed circles represent the force pulling the pixels towards the cluster center. The outer dashed circles represent the push force, separating the different clusters from each other.
other. Since each pixel is associated with an instance label, the center of the corresponding cluster center is straightforward to calculate, as well as the distances to each pixel belonging the cluster. From this, the loss term $L$ can be calculated:

$$L = \alpha \cdot L_{\text{var}} + \beta \cdot L_{\text{dist}} + \gamma \cdot L_{\text{reg}}$$  \hspace{1cm} (4.1)

where $\alpha$, $\beta$ and $\gamma$ are hyper-parameters, $L_{\text{var}}$ is a term that enforces the separations of clusters, $L_{\text{dist}}$ is a term for minimizing the distance of pixels from their cluster center, and $L_{\text{reg}}$ is a term that penalizes clusters from maximizing the distance to the origin. As with any other loss term, this is used to update the weights of the network by back-propagation. The mechanics of the loss function is illustrated in Figure 4.1.

At inference time, the pixels of the images are mapped to the 16-channel instance representation. The semantic segmentation map is used to mask the relevant pixels. These are then assigned to instances by using a clustering method. Here, we use $k$-means clustering, but any clustering method can be used. $k$-means clustering is a fast and commonly used method for clustering, but the number of clusters $k$ needs to be pre-assigned. In this context, we need to know the number of instances beforehand.

### 4.2 Equivariance of instance segmentation

In Chapter 3, we showed how to modify the U-net architecture to become equivariant to the transformations in the p4 group, consisting of translations and rotations of multiples of 90 degrees. We can use this same procedure for the U-net variant used for semantic instance segmentation. Here, we replace the standard convolutions in the instance representation head with the corresponding group-equivariant operations from the e2cnn library. Again, we have to add a pooling layer over the four rotations at the end of the network, and ensure to format the input representations in the correct way. Details about how to construct this network can be found in Paper III.

As explained previously, the instance representation needs to be clustered into separate instances. To design a network that is fully equivariant to transformations in the p4 group, we need to ensure that the clustering step preserves the equivariance to these transformations. If we rotate the input image to the network by 90 degrees, the pixel coordinates, and the features from the group-equivariant convolutions, are all rotated by 90 degrees throughout the layers of the network. This will propagate to the instance representation.

At the clustering step, all of the pixels are mapped to the 16-channel representation space, and a clustering algorithm partitions the space to out-
4.3 Standard vs rotation-equivariant U-net

In this section, we compare the baseline U-net for semantic instance segmentation with the variant that is equivariant to transformations in the p4 group. To do this, we need to specify the dataset and how to evaluate the results. We again use BBBC038, the highly varied dataset of cell nuclei. We modify this dataset by cropping the images to only contain a fixed number of instances, to comply with the predetermined number of clusters. We set this number to four.

To evaluate the experiments, we use the DICE score, defined as follows:

$$\frac{2 \cdot TP}{2 \cdot TP + FP + FN} \quad (4.2)$$

where $TP$ is true positives (the number of pixels with correct instance segmentations), $FP$ is the false positives (the number of pixels incorrectly classified as the current instance), and $FN$ is the false negatives (the number of pixels incorrectly classified as another instance). Equation (4.2) is calculated for each instance, in addition to the background, and the end result is averaged. The output instances are permutation-invariant, i.e., the specific marker an instance is tagged with is without meaning. To handle this, the

Figure 4.2: The full instance segmentation method is equivariant to rotations of multiples of 90 degrees. The input is fed into the rotation-equivariant U-net, which outputs a binary segmentation and a 16-valued instance representation. The instance representation is then clustered into instances. Rotations act only on pixel coordinates. No matter where a rotation of 90 degrees is applied, the final output is identical.
4.3. Standard vs rotation-equivariant U-net

Figure 4.3: Example from the modified BBBC038 dataset: a) input image, b) an instance segmentation from for the baseline architecture, and c) an instance segmentation from the rotation-equivariant architecture.

Instances are compared with all of the labels, the best overlap is chosen, and then both the instance and the label are removed. This is repeated until no instances and labels are left.

Again, we want to investigate the effect of replacing standard U-net architectures using data augmentation with the architecture equivariant to the p4 transformations. The rotation-equivariant architecture should yield a DICE score at least as good as the baseline network with data augmentation to those same transformations. In addition to this, it should be less sensitive to overfitting since it uses a higher degree of weight sharing, which should improve the DICE accuracy.

The results for these experiments, which are further outlined in Paper III, can be seen in Figure 4.4. From this, it is unclear if the rotation-equivariant network offers any advantage to the baseline network combined with data augmentation. The DICE scores are similar for all four cases. The experiments should be run longer than 100 epochs, repeated and averaged to fully confirm the hypothesis. A similar experiment on the scattered synthetic sticks yields similar results - in this problem setting, it is unclear if there is an advantage to use the rotation-equivariant U-net instead of the baseline network combined with data augmentation.
4.4 Jet classification with UCluster

Figure 4.4: Experiments to measure the effects of data augmentation. The images from the BBBC038 dataset were rotated 90, 180 and 270 degrees, keeping the labels intact. Both the original data and the augmented data were then used as input to the two networks.

4.4 Jet classification with UCluster

We now switch focus to particle physics. Specifically, we will consider jets, i.e., the resulting evolution from high energy collisions of protons. An example can be seen in Figure 4.5, a visualization of experiments that supports the existence of the Higgs boson. To facilitate analysis of the jets, they often need to be classified. This can be performed with the graph neural network ABCnet, to generate a suitable latent representation. This can then be partitioned using clustering methods, such as \( k \)-means clustering. The clusters can represent particles, and be grouped together based on, e.g., masses or event types. This forms the basis for the UCluster method \[22\].

4.5 Distributed deep learning

The HLS4ML LHC dataset \[19\] consists of jets generated from proton-proton collisions from the Large Hadron Collider at CERN. This dataset can be partitioned using UCluster. Here, we show how to distribute the training of UCluster over multiple GPUs in a cloud computing environment.

For the distributed deep learning, Horovodrunner \[26\] was selected. This framework allows us to distribute the calculations over several GPUs. The typical workflow is to first get the model running on a single node, then allocate the same operations over multiple nodes. Horovodrunner facilitates easy integration with Apache Spark clusters \[36\], an engine for large-scale
4.5. Distributed deep learning

Figure 4.5: A collision event producing jets. This event is consistent with Higgs boson decay. Image sourced from the ATLAS detector at CERN [7].

Table 4.1: Training times for the single machine codes and the distributed code.

<table>
<thead>
<tr>
<th>Code</th>
<th>Training time per epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code</td>
<td>4 minutes</td>
</tr>
<tr>
<td>TensorFlow 2 version of original code</td>
<td>4 minutes</td>
</tr>
<tr>
<td>Distributed training, 2 GPUs</td>
<td>2 minutes</td>
</tr>
<tr>
<td>Distributed training, 4 GPUs</td>
<td>1 minute</td>
</tr>
<tr>
<td>Distributed training, 8 GPUs</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

Data processing. We chose to develop and deploy our models in the cloud computing environment Databricks, which is built on top of Spark. These frameworks necessitated a code migration from the python deep learning library TensorFlow v1.14 to v2. More details, such as training settings and hyper-parameters, can be found in Paper II.

The results from the experiments showed a validation accuracy of around 50–60 %, for all combinations of hyper-parameters and number of nodes. This is in contrast to validation accuracy reported in the original UCluster paper of around 81 %. It is unclear what causes this discrepancy. In Table 4.1, the training times are reported. It is clear that the training time is inversely proportional to the number of nodes.
Chapter 5

Discussion and Conclusions

In Chapter 2, we delved into the details about what a classifier in image analysis aims to achieve. We introduced the problem of oral cancer diagnosis based on images of cell nuclei, and showed how a convolutional neural network, the VGG16 architecture, can be designed and trained to classify new images. We also introduced the concept of data augmentation, i.e., increasing the amount of training data by transforming it. In Paper I, we showed how to modify the VGG16 architecture to become equivariant to rigid transformations. Finally, we showed that rotation-equivariant networks, GCNNs, outperform baseline CNNs combined with data augmentation in terms of accuracy. We also showed quantitatively that this results in less overfitting to the training data.

Chapter 3 introduced semantic segmentation in an image analysis context. We showed how to segment a varied dataset of cell nuclei using the common U-net deep learning architecture. Then, in Paper III, we showed how to make this architecture equivariant to rotations and translations in the p4 symmetry group. We showed empirically that this property holds in experiments for the rotation-equivariant architecture, in contrast to the baseline network.

In Chapter 4, we looked at clustering methods. We began by showing how the U-net architecture can be combined with a discriminative loss and a clustering method to provide semantic instance segmentation. Furthermore, in Paper III, we showed how to modify this method to also be equivariant to rotations and translations in the p4 symmetry group. We showed, by reasoning and analysis, how the equivariant properties of the network continue to hold when performing these modifications. Looking at the experiments from this paper, we concluded that the performance in terms of DICE scores were similar for the rotation-equivariant network and the baseline network combined with data augmentation using the same transformations.
Also in Chapter 4, we looked at how another combined deep learning and clustering method, UCluster, could be used to group sub-atomic particles with similar properties. Then, in Paper II, we showed how to leverage a cloud computing framework, and how to distribute the training over multiple GPUs using the Horovodrunner framework. This led to significantly faster training, and provided scalability for much larger datasets than a single node can handle.

We have shown that modifying deep learning architectures to be equivariant to the p4 symmetry group of transformations can result in higher accuracy and less overfitting in classification problems. We have not seen the same results for semantic instance segmentation - here, more research is needed. Returning to the aim of the thesis, we have shown how to use limited training data more efficiently to reduce the problems of overfitting. We have also shown how to migrate training to multiple nodes in computational centers, mitigating the problems of handling growing data sizes.

5.1 Future work

From Paper III, it is evident that more experiments are needed to verify that the rotation-equivariant network can replace a baseline CNN combined with data augmentation in semantic instance segmentation. The experiments from Paper I and Paper III could also be extended to include more transformations, other architectures and other datasets.

Distributed deep learning will become increasingly relevant as data sizes continue to grow. Even though many current experiments in computerised image analysis can be handled on single-machine setups, it could be of interest to perform further experiments in environments that are prepared for scalable and distributed training.
Bibliography


Replacing data augmentation with rotation-equivariant CNNs in image-based classification of oral cancer

Karl Bengtsson Bernander, Joakim Lindblad, Robin Strand, and Ingela Nyström

Centre for Image Analysis, Dept. of Information Technology, Uppsala University, Sweden

Abstract. We present how replacing convolutional neural networks with a rotation-equivariant counterpart can be used to reduce the amount of training images needed for classification of whether a cell is cancerous or not. Our hypothesis is that data augmentation schemes by rotation can be replaced, thereby increasing weight sharing and reducing overfitting. The dataset at hand consists of single cell images. We have balanced a subset of almost 9,000 images from healthy patients and patients diagnosed with cancer. Results show that classification accuracy is improved and overfitting reduced if compared to an ordinary convolutional neural network. The results are encouraging and thereby an advancing step towards making screening of patients widely used for the application of oral cancer.

Keywords: machine learning · accuracy · cell image analysis · malignancy

1 Introduction

In the last years, convolutional neural networks (CNNs) have been increasingly applied to classification of biomedical data. Our application concerns cell samples collected from patients’ oral cavity used to efficiently screen for oral cancer [4]. Performing analysis of the cell images by experienced pathologists is time-consuming and thereby costly. In addition, the expertise may not always be readily available to classify whether a cell is malignant or not. In order to make the screenings widely used, it is therefore desirable to perform these screenings automatically [9]. In this study, we make an advancement towards feasible automated classification of cell images.

It is well-known that despite being popular, the CNNs need a large amount of varied training data to avoid overfitting. One approach to increase the data during training and thereby enrich the data is to use data augmentation [6]. For image analysis tasks, the input data is typically altered with different transformations such as rotations, scalings, and reflections [12]. One example is when the amount of data is increased four-fold by rotating the images by 90 degrees. By
using such a scheme, even though being unvaried, new data is retrieved for training of the network while still keeping the ground truth classification labels intact. However, this comes with the cost of an extra step in the data preparation. In addition, interpolation artifacts may be introduced in the images.

CNNs are, with some exceptions such as edge effects, equivariant to translations [11], which means that if an object is shifted in the image, the resulting features after the convolution operation are shifted as well. In more mathematical terms, the translation operator $T$ commutes with the convolution operator $C$:

$$T(C(x)) = C(T(x))$$ (1)

However, this relation does not hold for other transformations such as rotations. In order to achieve this handy property, several methods have been proposed and tested. One of the most common ways to achieve rotation-equivariant CNNs is to generalize the convolution operator [3, 17, 18]. Most of these approaches are limited to specific discrete rotations. Other advances in the field extend to arbitrary angular resolutions using atomic steerable filters [15]. Another alternative is to use conic convolutions and discrete Fourier transforms for rotation-invariant classification [2]. Recently, a framework was released to aid in the development of $E(2)$-equivariant steerable networks (E2CNN) [14].

One domain where CNNs are heavily used is cytopathology, where cell samples from patients are collected, imaged and diagnosed for diseases. When these samples are collected from, for example, the oral cavity, they can serve as biomarkers for cancer development. For these images, we make the assumption that the cells are rotationally invariant on a global scale, implying that regardless how the sample is rotated during the microscopy imaging process, the classification output should be identical. However, on a smaller scale, the rotations of parts of the cells (for example, organelles and chromosomes) could be an indicative of cancer, which is what we expect the CNNs to reveal.

Our hypothesis is that by replacing ordinary CNNs with equivariant CNNs in our application, the data augmentation step could be skipped since the networks in effect perform the corresponding transformations at each layer of the network. That is, instead of performing data augmentations by rotating the data when training the CNNs, we can use rotation-equivariant networks with no data augmentations during training and testing. This should improve the expressive capacity of the network while reducing the number of trainable weights.

To verify our hypothesis, we compare the accuracy of a rotation-equivariant CNN with that of a standard CNN on a large oral cancer dataset which we use as a reference. Previous works indicate it would be interesting to investigate these networks further on this type of dataset, due to the importance of texture-based features in cell images [16]. In this study, we also vary the size of the training datasets from a few hundred up to several thousands of images, to verify that the accuracy is comparable to the results for the baseline network.

Section 2 introduces the methodology of the experiments. Section 3 details the oral cancer dataset, while Section 4 outlines the architectures and training procedures for classifiers on this dataset. The results are then presented in Section 5 and interpreted in Section 6.
2 Methodology

In this work, we focus on how data augmentation, the amount of training data, and equivariance properties of the architecture affect the ability of the network to generalize. We also inspect and compare the models in terms of sensitivity and specificity. These statistical measures are calculated in the following way:

\[
\text{Sensitivity} = \frac{TP}{TP + FN}, \quad (2) \\
\text{Specificity} = \frac{TN}{TN + FP} \quad (3)
\]

where TN is the sum of true negatives, FN is the sum of false negatives, TP is the sum of true positives, and FP is the sum of false positives.

The core idea is however to investigate overfitting. According to the classic bias-variance trade-off, an optimally trained network strikes a good balance between underfitting and overfitting [6]. In this region, the network has captured the characteristics of the underlying distribution and is able to generalize to unseen data from similar distributions. To measure overfitting, we compare the empirical risk with the testing accuracy.

The empirical risk for a classifier \( R_{\text{emp}}(h) \) [13] is defined as:

\[
R_{\text{emp}}(h) = \frac{1}{n} \sum_{i=1}^{n} L(h(x_i), y_i) \quad (4)
\]

where \( h \) is the hypothesis (i.e., our network), \( L \) is the loss function, and \( x_i \) and \( y_i \) are \( n \) independent input and output samples, respectively.

The accuracy is defined as the percentage of examples classified with the correct label. Since a low empirical risk is directly proportional to a high classification accuracy on the training set, we can use the training accuracy as a proxy for the empirical risk. Based on this, we form the following metric of overfitting:

\[
\text{Overfitting Ratio} = \frac{\text{Training Accuracy}}{\text{Testing Accuracy}} \quad (5)
\]

A high number will then indicate high overfitting, while a low ratio will indicate low overfitting.

3 Oral Dataset

Screening for malignant cancer is a highly specialized task, requiring good sample quality, experience, and patience. The procedure for oral cancer diagnosis is similar to the one for cervical cancer [1]. One can expect issues of very different origin in the procedure, for example, staining artifacts, bacterial and fungal infections, ruptured cells, or a combination thereof. Indicators of cancer include atypical ratios of the nucleus to cytoplasm, the presence of several nuclei, or
asymmetrical shapes of the cell. Recently, a dataset of images has been prepared to automate this diagnostic process [9] (we use images from the liquid-based prepared Dataset 3). A number of colored cell samples from this dataset are exemplified in a mosaic setting in Figure 1a and one of the cells is selected for larger illustration in Figure 1b. The corresponding gray-scale image is shown in Figure 1c. The single cell images are 80 x 80 pixels, centered on cell nuclei. We use the same dataset in this study, with a few modifications.

We opt to remove the color information the three bands RGB carry, in order to simplify the image processing and analysis for our prototype system for malignant cancer classification. While the gray-scale conversion eliminates hue and saturation, our aim is to keep the luminance intact. The conversion is formed by a weighted sum of the red, green, and blue channels:

\[ 0.289R + 0.587G + 0.114B. \]  

This way, the green channel is given the highest impact, which is reasonable since a wavelength of around 530 nm (i.e., green) is well-known in the literature to produce the best contrast for this type of biological material [5, 7]. This conversion is achieved by the Matlab standard function rgb2gray.

In phase one, we partition the dataset into training data and test data as follows, in two phases. For phase one, the training data consist of 4,254 images balanced evenly between healthy and malignant samples. The test data consist of 9,942 images, divided into roughly one third healthy and two thirds malignant samples. For phase two, we investigate the effects of varying the amount of training data. Here, the number of cells in the training set is increased from 266 to 8,508 in steps. We use the same test set as in phase one.

The data set contains samples from 12 patients, 6 healthy and 6 with confirmed cancer diagnosis. The data are split on a patient level, so that the test set and the training set include cells from three healthy and three sick patients, respectively. No cells from an individual patient appear in both training and test sets, to avoid information leakage. Despite having annotations only on patient level, we aim to classify each cell nucleus as originating either from a sick or a healthy patient. That is, if the patient was diagnosed with cancer, then we associate all the individual cell images from that patient with the cancer label. However, it is not necessarily true that all the individual cells are malignant cells, or even indicative of cancer, and therefore it is not expected to near a 100 % classification accuracy.

4 Experiments

The experiments were performed in two phases following the methodology described in Section 2. We investigated the effects on empirical risk and accuracy by varying data augmentation, the equivariant properties of the network, and the amount of data. For all experiments, we modified the VGG16 architecture, seen in Figure 2 to work with greyscale images of size 80x80 pixels.
The training hyper-parameters are listed in Table 1 along with additional details of the network. Note that the standard number of channels was reduced to speed up the experiments. We also adapted the network to be equivariant to 90-degree rotations, by replacing the ordinary convolutions with group-equivariant convolutions on the C4 group. We call this architecture GCNN. The architecture with ordinary convolutions is named CNN. For the GCNN, we also introduced an additional group-pooling layer before the fully connected layer, which was not present in the ordinary CNN architecture. This functioned as a max-pooling layer over the rotations, and ensured a compatible input to the fully connected layer. This resulted in an architecture invariant to 90-degree rotations. We implemented our networks in the E2CNN framework, which is based on pytorch [14].

In the first phase, we investigated how the ordinary CNN performed compared to our new rotation-equivariant network. Here, we also investigated the effect of the use of data augmentation. We rotated the input images by 90, 180, and 270 degrees, increasing the amount of data fourfold, before starting the training. For each experimental setting, we performed five runs in order to account for possible outlier results.

In phase two, we investigated how the amount of training data affected accuracy. The amount of training data was increased from 266 to 4254 in steps, following a law of $266 \times 2^x$, where $x$ is step. The final step diverged from this law and used 8508 samples, which was the full amount of data available. This ensured use of all the samples as well as intermediary multiples of the smallest number of samples. Data augmentation by multiples of 90 degrees was also used.
Table 1: Settings for the CNN architecture and training procedure.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss function</td>
<td>Cross entropy</td>
</tr>
<tr>
<td>Weight initialization</td>
<td>He</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
<tr>
<td>Learning rate</td>
<td>0.00001</td>
</tr>
<tr>
<td>Batch normalization</td>
<td>Batches of size 128</td>
</tr>
<tr>
<td>No. of epochs</td>
<td>200</td>
</tr>
<tr>
<td>Validation frequency</td>
<td>1/5 epochs</td>
</tr>
<tr>
<td>Activation functions</td>
<td>ReLu</td>
</tr>
<tr>
<td>Dropout</td>
<td>0.5 between linear layers</td>
</tr>
<tr>
<td>No. of channels</td>
<td>16-16-32-32-64-64-128-128-128-128-128</td>
</tr>
<tr>
<td>Convolution layers (size, stride, padding)</td>
<td>Layers 1-12: (3,1,1)</td>
</tr>
<tr>
<td></td>
<td>Layer 13: (4,0,0)</td>
</tr>
<tr>
<td>Maxpooling layers (size, stride, padding)</td>
<td>Layers 1-4: (2,2,0)</td>
</tr>
<tr>
<td></td>
<td>Layer 5: (2,1,0)</td>
</tr>
<tr>
<td>Linear layer parameters (input, output sizes)</td>
<td>(128,4096) -</td>
</tr>
<tr>
<td></td>
<td>(4096,4096) - (4096,2)</td>
</tr>
</tbody>
</table>

5 Results

The results from the first phase are presented in Figures 3, 4, and 5. Figure 3 shows the training results for the experiments with the CNN. The training accuracy quickly converged to 100 %. This was true for the GCNN as well.

Figures 4 and 5 show the accuracy on the test set for the CNN and GCNN, respectively. It can be seen that the accuracy for the CNN on the test set stabilizes at around 55–56 %. This is true regardless of whether data augmentation is used or not. Using the GCNN increases the test accuracy to around 59–60 %. Again, this holds regardless the use of using data augmentation. In Table 2 the sensitivity and the specificity for the last epoch for one instance of the experiments are reported. For both architectures, data augmentation by 90-degree rotations is performed. It is again evident that the GCNN outperforms the CNN.

The results from the second phase are presented in Figure 6. For both architectures, the accuracy on the test set quickly reaches 100 %. It converges quicker for increasing amounts of data. On the test set, for both architectures, the accuracy increases when more data is added. The GCNN again outperforms the CNN.
Fig. 3: Classification accuracy during training for an ordinary CNN. The figures show the experiments with a) no data augmentation b) with data augmentation.

Fig. 4: Classification testing accuracy for an ordinary CNN. The plots show the experiments with a) no data augmentation b) with data augmentation. Dashed lines are runs with identical settings, the solid line is the mean of these runs.

Fig. 5: Classification testing accuracy for a GCNN. The plots show the experiments with a) no data augmentation b) with data augmentation. Dashed lines are runs with identical settings, the solid line is the mean of these runs.
Fig. 6: Training and testing accuracy when varying the amount of training data. Figures a) and b) show the training and testing accuracy respectively for a CNN, while figures c) and d) show the training and testing accuracy respectively for a GCNN.

### Table 2: CNN confusion matrix

<table>
<thead>
<tr>
<th>Network</th>
<th>TN</th>
<th>FN</th>
<th>TP</th>
<th>FP</th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>2980</td>
<td>1991</td>
<td>2630</td>
<td>2341</td>
<td>0.569</td>
<td>0.560</td>
</tr>
<tr>
<td>GCNN</td>
<td>3249</td>
<td>1722</td>
<td>2756</td>
<td>2215</td>
<td>0.615</td>
<td>0.595</td>
</tr>
</tbody>
</table>

### 6 Conclusions and Future Work

In this study, we investigated classifiers on a biomedical dataset. Our hypothesis was that replacing CNNs using data augmentation by rotations with architectures equivariant to those same rotations would decrease overfitting. Specifically, we experimented with 90-degree rotations.

By looking at Figure 4, we see that performing data augmentation by multiples of 90-degree rotations does not improve test accuracy, for an ordinary
Table 3: Overfitting measurements

<table>
<thead>
<tr>
<th>Network</th>
<th>Overfitting ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN with data augmentation</td>
<td>1.82</td>
</tr>
<tr>
<td>GCNN without data augmentation</td>
<td>1.69</td>
</tr>
</tbody>
</table>

CNN. However, designing the network to be equivariant to those same rotations by the use of the GCNN increases the test accuracy. By inspecting the values in Figures 4b and 5a, we estimate the mean test accuracies for the CNN with data augmentation versus the GCNN with no data augmentation. For the last epoch, these values are 0.55 and 0.59 respectively. From Figure 3b, the training accuracy for the CNN is 1.0. The training accuracy for the GCNN is also 1.0. This corresponds to an empirical risk of 0. Following Equation 5 we calculate overfitting in Table 3. Since a lower value means less overfitting, we conclude that using a GCNN instead of a CNN with data augmentation reduces overfitting in this setting.

The second phase of the experiments saw the variation of the amount of training data. From Figure 6, we conclude that the minimal number of samples for the highest possible test accuracy is around 4000. Secondly, the GCNN yields a higher accuracy on the test set for all amounts of data, indicating less overfitting.

In order to increase the accuracy on the test set and decrease overfitting, we added $L_2$ regularization. Weight decay was varied between 0.5 and 256, doubling in size for each experiment. For a weight decay above 8, the training accuracy decreased significantly. However, this came at the cost of decreased accuracy on the test set.

A promising way forward is to repeat the experiments using other types of neural network architectures, particularly those who already have shown to yield high test accuracy on the dataset. Future work could also compare other rotations besides the current multiples of 90 degrees, as well as other types of transformations, such as reflections. Finally, considering other, highly realistic datasets is of great interest for determining if equivariant networks can improve automatic diagnostics in more general biomedical settings.

Acknowledgment

We thank Gabriele Cesa et al. for their contributions during our tailoring of the framework to the specific application of cancer classification for oral images. We are grateful for Professor Emeritus Ewert Bengtsson’s valuable input to this project on cell image analysis in general and color conversions in particular. We especially thank Professor Nataša Sladoje for initiating the project in conjunction with the Wallenberg AI, Autonomous Systems and Software Program.
References

Distributed training and scalability for the particle clustering method UCluster

Olga Sunneborn Gudnadottir1,*, Daniel Gedon2, Colin Desmarais4, Karl Bengtsson Bernander3, Raazesh Sainudiin4,5, and Rebeca Gonzalez Suarez1

1Department of Physics and Astronomy, Division of High Energy Physics, Uppsala University
2Department of Information Technology, Division of Systems and Control, Uppsala University
3Department of Information Technology, Division of Visual Information & Interaction, Uppsala University
4Department of Mathematics, Uppsala University
5Combient Competence Centre for Data Engineering Sciences, Uppsala University

Abstract. In recent years, machine-learning methods have become increasingly important for the experiments at the Large Hadron Collider (LHC). They are utilised in everything from trigger systems to reconstruction and data analysis. The recent UCluster method is a general model providing unsupervised clustering of particle physics data, that can be easily modified to provide solutions for a variety of different decision problems. In the current paper, we improve on the UCluster method by adding the option of training the model in a scalable and distributed fashion, and thereby extending its utility to learn from arbitrarily large data sets. UCluster combines a graph-based neural network called ABCnet with a clustering step, using a combined loss function in the training phase. The original code is publicly available in TensorFlow v1.14 and has previously been trained on a single GPU. It shows a clustering accuracy of 81% when applied to the problem of multi-class classification of simulated jet events. Our implementation adds the distributed training functionality by utilising the Horovod distributed training framework, which necessitated a migration of the code to TensorFlow v2. Together with using parquet files for splitting data up between different compute nodes, the distributed training makes the model scalable to any amount of input data, something that will be essential for use with real LHC data sets. We find that the model is well suited for distributed training, with the training time decreasing in direct relation to the number of GPU’s used. However, further improvements by a more exhaustive and possibly distributed hyper-parameter search is required in order to achieve the reported accuracy of the original UCluster method.

1 Introduction

Although machine-learning methods have been used in high energy physics for more than 50 years, recent years have seen a substantial increase in their variety and prevalence (see e.g. [1]). This can be connected to different factors, such as, the increased need for more precise methods in experiments, the fast-paced development of novel machine-learning methods

*e-mail: olga.sunneborn.gudnadottir@cern.ch

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).
coming from both within the academic setting and the private sector, and improvements in computer hardware leading to a larger computing capacity. Within the Large Hadron Collider (LHC) experiments, several research problems are already solved with machine-learning; neural networks and boosted decision trees are used for e.g. flavour tagging jets, separating signal from background in analysis, and particle identification [2]. Improving these methods, extending their reach and developing new ones is currently an active field of study. In addition to the above, machine-learning has been proposed to solve such diverse problems as collision and detector simulation, trigger decision-making, model-independent searches for new physics, jet substructure studies and much more [3].

A lot of the development in machine-learning methods for High Energy Physics is done on data sets that have already undergone a great deal of event selection, and are small enough to allow for training on a single Graphical Processing Unit (GPU) used as the compute node. For more complex models, such as graph networks, generative adversarial networks, or any large enough model, this type of training is limited by its lack of scalability, i.e. its capacity to handle growing amounts of data. This problem can be ameliorated by distributing the training over multiple workers in a cluster of several compute nodes, and thereby increasing the capacity for data ingestion by utilising the distinct storage devices and the working parallel random access memory of several GPU nodes. Using distributed training algorithms also reduces the overall training time by effectively increasing the batch size, which in turn mitigates the problem of prohibitively long training times. In this paper, we apply distributed training to the recently proposed UCluster method [4] for unsupervised clustering of particle physics data, with the goal of both speeding up the training and making it scalable to arbitrarily large data sets.

UCluster is a neural network for unsupervised clustering on particle collider data. It creates a latent space using a classification network and then clusters particles, that are close in the latent space, together. Depending on the desired properties of the clusters, different classification objectives can be used to create different latent spaces. By choosing jet mass classification, the model produces clusters of jets with the same mass, by choosing event classification the model produces clusters of events with similar properties, etc. This makes UCluster general and highly adaptable, and it has the potential to be useful for several physics problems relating to particle or event classification. So far, it has shown promising results when applied to multi-class classification of jets and anomaly detection at the event level. However, with all training and evaluation executed on a single GPU, the size of the input data is limited to the number of events that can be loaded onto one GPU memory simultaneously. This excludes modifying UCluster, as implemented in a single machine setup [4], to any task that requires training on bigger sample sizes, e.g. full data samples from the LHC experiments.

Unsupervised multi-class classification is something that could be of interest in precision measurements, e.g. in cases where simulated data are not precise enough to be used for background estimation. In this case, data-driven methods, i.e. methods that use real data to estimate the background, are commonly used. This approach can quickly become involved if it has to be done for more than one background process. Instead, an unsupervised multi-class classification method could be applied directly to data, labelling the processes without the need for multiple background fits. UCluster is reported to have an 81% clustering accuracy when applied to the problem of classifying the fat jets of the HLS4ML LHC Jet data set [5] into clusters in which the member jets all originate from the same particle, making use of particle mass classification. However, for data-driven background estimation, the data sets could become several orders of magnitudes bigger.

Anomaly detection is interesting in particle physics, since it is model-independent by nature and can be used to find deviations from the Standard Model, which can then be used as the
basis for new studies. UCluster has been applied to the R&D Dataset for the LHC Olympics 2020 Anomaly Detection Challenge [6] and reports an increase in signal-to-background ratio from 1% to 28%, in which the signal represents the anomalies. This is accomplished through all anomalies ending up in the same cluster. Making this setup scalable would open up the possibilities of looking at larger data sets or even the full experimental data samples of the large LHC collaborations, as has been proposed in reference [7].

To achieve distributed and scalable training, we made use of Apache Spark [8], an open-source distributed general-purpose cluster-computing framework, which creates an architecture over a cluster of multiple compute nodes for distributed data processing through a distributed file system, that splits and stores the data for processing in a fault-tolerant manner. We set this up on the Databricks [9] platform, which allows for easy creation of Apache Spark clusters. This setup bypasses many of the challenges of processing large data sets such as cluster management, unreliable hardware or running out of memory on a single GPU. We used the distributed deep learning framework Horovod [10], which contains a wrapper to run distributed training in Spark clusters, to run the training. Once a training algorithm has been set up with Horovod, it can be run on any number of GPUs (including only one) without any changes in the code. Distributing the training across multiple GPUs means an effective increase in the batch size of the underlying stochastic gradient descent optimisation algorithm, leading to a faster convergence of the optimisation functions. We expect the training time to be in inverse proportion to the number of GPUs.

2 Model and data

The details of the UCluster model can be found in the original reference [4], but some elements needed to understand the rest of this paper will be repeated here. Consider a data set from a particle collider, which has already gone through digitisation and object reconstruction. The reconstructed objects are then represented as nodes in the graph-based neural network known as ABCnet [11], a classification net that aids in the over-arching classification problem.

This classification net needs to be optimised to create a suitable latent space on the over-arching clustering problem of the UCluster model, i.e. a space in which particles with similar properties are close to each other, making clustering possible. The ABCnet is pre-trained for a number of epochs and then a classical k-means algorithm is applied to the latent space. The resulting clusters are used to initialise the cluster centroids in the full model, which is a Deep k-means algorithm that combines the classification net with clustering [12]. The full model is trained end-to-end, with the combined classification and clustering loss, and the trained model assigns every data-point to a cluster. The code is written in TensorFlow v.1 [13].

In this paper, we will mimic the first use case demonstrated in the original paper – multi-class classification of fat jets from the HLS4ML LHC Jets data set – using distributed training. The classification objective is mass classification of jets. The goal is to produce three clusters of jets, each of which contains only jets originating from either a W-boson, Z-boson or top quark.

This HLS4ML data set contains high \( p_T \) jets originating from W-bosons, Z-bosons and top-quarks from simulations of LHC proton-proton collisions at \( \sqrt{s} = 13 \) TeV, which have subsequently been run through a parametric description of a generic LHC detector. They have then been reconstructed using the anti-k_{T} algorithm [14] with radius parameter \( R = 0.8 \). The data consist of various high-level features of each of up to 100 constituent particles of each jet. Table 1 shows the data features of each constituent used for this project.

The data are stored in two HDF5 files [16]: one training data set and one validation data set. Following the unconventional nomenclature of the HLS4ML challenge [5], we call the
Table 1: Data features used as input in the UCluster method.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta \eta$</td>
<td>Difference in pseudorapidity $\eta$ between jet constituent and jet.</td>
</tr>
<tr>
<td>$\Delta \phi$</td>
<td>Difference in azimuthal angle $\phi$ between jet constituent and jet.</td>
</tr>
<tr>
<td>$\log(p_T)$</td>
<td>Logarithm of the constituent transverse momentum $p_T$.</td>
</tr>
<tr>
<td>$\log(E)$</td>
<td>Logarithm of the constituent energy $E$.</td>
</tr>
<tr>
<td>$\log\left(\frac{p_T}{p_T(jet)}\right)$</td>
<td>Logarithm of the constituent transverse momentum relative to the jet transverse momentum.</td>
</tr>
<tr>
<td>$\log\left(\frac{E}{E(jet)}\right)$</td>
<td>Logarithm of the constituent energy relative to the jet energy.</td>
</tr>
<tr>
<td>$\Delta R$</td>
<td>Distance defined as $\sqrt{\Delta \eta^2 + \Delta \phi^2}$</td>
</tr>
<tr>
<td>PID</td>
<td>Particle ID [15]</td>
</tr>
</tbody>
</table>

two data sets, used for development, the training and testing data sets and the data set we test our final model on as the validation data set.

3 Modifying the code for scalability and distributed training

To make use of the Horovod framework, we needed to migrate the code to Tensorflow v2. We then extended the model to allow for distributed training. We thus have three incarnations of the code:

The original code which is obtained from the GitHub of the original authors [17]. This was used to validate our setup against.

The original code migrated to TensorFlow v2. This was done with an automated function supplied by TensorFlow and relies on TensorFlow 1 compatible functions. This code was validated against the original code and shows comparable results in both training behaviours, results and execution time.

The distributed model. This is described in the next section.

3.1 Distributed training

With the Tensorflow v2 code as a starting point, we used the HorovodRunner, the Horovod Databricks Application Programming Interface (API), to be able to run distributed training. This was done by creating a HorovodRunner instance and passing it the training function. The training function had to be modified for use with Horovod, including changes, such as, increasing the learning rate to compensate for the bigger effective batch size and handling of checkpoints to ensure consistent saving to and initialisation from them. With our data on the local driver, we initialise the HorovodRunner instance, which copies the data to each GPU. The Horovod framework takes care of the distributed training using a distributed optimiser to accumulate gradients across multiple GPUs, and saves model checkpoints at regular intervals. The learning rate is scaled by the number of GPUs used. Currently, all data is copied onto each GPU, which limits the scalability to what can be fit into a single GPU memory. This will be addressed by copying only fractions of data to each GPU at a time.

3.2 Scalability

The UCluster repository includes a pre-processing script specifically for the HLS4ML data set which we use to extract the relevant features from it (see Table 1). After pre-processing,
Table 2: Hyper-parameters used in training.

<table>
<thead>
<tr>
<th>Hyper-parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch size</td>
<td>1024</td>
</tr>
<tr>
<td>Inverse temperature $\alpha$</td>
<td>Starting at 1, increasing linearly by 2 every following epoch</td>
</tr>
<tr>
<td>Proportionality constant between classification loss and clustering loss $\beta$</td>
<td>10</td>
</tr>
<tr>
<td>Focal loss hyperparameter $\gamma$</td>
<td>2</td>
</tr>
<tr>
<td>Learning rate</td>
<td>Starting at 0.001 and decreases by 2 every three epochs until it has reached $10^{-5}$. Multiplied by a factor equal to the number of GPUs.</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam</td>
</tr>
</tbody>
</table>

the data are contained in a single file and will have to be loaded in their entirety onto the local driver before training. This currently puts limits on the scalability of our setup, and will be addressed in future work by writing data loaders directly into the distributed file system.

4 Training and evaluation

The model was trained with the same hyper-parameters as found in the original paper, summarised in Table 2. It was first pre-trained for 20 epochs with only the classification net, before being trained end-to-end (with the combined classification and clustering loss) for a total of 100 epochs. The training was done on GPU clusters with either 2, 4 or 8 NVIDIA T4 GPUs, each with 16 GB of memory, running Apache Spark 3.0.1 (Amazon EC2 G4dn.xlarge instance). The training time per epoch for different number of GPUs can be seen in Table 3, compared to the training time for the single machine codes. The training time can be seen to inversely scale with the number of GPUs. Each of the models, the distributed code as well as the single machine codes, has been trained several times, and the testing accuracy during training has been recorded. A representative plot for each model can be seen in Figure 1. The models display very similar training behaviour, which is to be expected if the distributed training works as it should: For all models, the accuracy lies in the interval 0.5 to 0.6 throughout the training, usually increasing slightly with training time and occasionally decreasing over a number of epochs before returning to the stable behaviour. It can be noted that the testing accuracy is far from the 81% validation accuracy obtained in the original paper, which is true for all models, including the original one. Since there is virtually no improvement with training, the optimisation algorithms might have found a local minimum. Figure 2 shows the clustering loss during training of the distributed model on 4 GPUs. Here we can see a minimum at around epoch 28, and then a slight increase until epoch 65. Some hyper-parameter optimisations was done, testing $\alpha$ (inverse temperature) values that changed by a factor of (1,2,5,10) every epoch following the starting value of 1, changing the batch size to 512, changing the proportionality constant $\beta$ between the classification loss and the clustering loss to $(10^{-1},1,10,100)$ and lowering the learning rate. All trials gave worse or comparable results to those in Table 2. Furthermore, using the same hyper-parameters consistently shows the same behaviour as displayed in Figure 1 across a number of trials, independent on the number of GPUs used for training.
Table 3: Training times for the single machine codes and the distributed code.

<table>
<thead>
<tr>
<th>Code</th>
<th>Training time per epoch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original code</td>
<td>4 minutes</td>
</tr>
<tr>
<td>TensorFlow 2 version of original code</td>
<td>4 minutes</td>
</tr>
<tr>
<td>Distributed training, 2 GPUs</td>
<td>2 minutes</td>
</tr>
<tr>
<td>Distributed training, 4 GPUs</td>
<td>1 minute</td>
</tr>
<tr>
<td>Distributed training, 8 GPUs</td>
<td>30 seconds</td>
</tr>
</tbody>
</table>

Figure 1: Testing accuracies. The first 20 epochs were pre-training without any clustering, so the clustering accuracy is set to 0.

5 Next steps

To make the model fully scalable, we need both the data ingestion and the training to be scalable. We will accomplish this by loading data directly into the distributed file system, bypassing the memory limitations of the main memory, as well as copying only subsets of data onto each GPU. For the data loading, we will use file formats designed for distributed processing of large data sets such as the open-source column-oriented data storage format.
Parquet or the to high energy physicists well-known ROOT format. For the data distribution, we will compare the Horovod framework to the Maggy [18] framework to see if there are already tools implemented that can be used for this purpose.

We are also actively looking into how the accuracy can be improved to that which was shown in the original paper. Since the optimisation might be stuck in a local minimum, a natural path forward would be to investigate the inverse temperature and potential energy surface defined by the optimisation problem. If the weights of the fully trained, accurate model can be made available to us, this could provide valuable insight without having to look into the inverse temperature. We could initialise our weights with the fully trained parameters, or slight perturbations there of, and see if our training reaches the same parameters again.

6 Conclusion and outlook

We have implemented distributed training for the UCluster method and are in the process of making it scalable to any input data size. We migrated the UCluster model to TensorFlow v2 and added distributed training using the HorovodRunner. After this, the training behaviour of the model is very similar to that of the original code, and this behaviour is consistent over a large number of trials. However, we see a significantly lower accuracy than that reported in the original paper. We are in the process of making the setup fully scalable, as well as troubleshooting the lowered accuracy. The UCluster method is a very general method in the sense that it can be modified for any task in which unsupervised clustering of particles could be used. The generality of the original model together with the scalability added in this project has the potential to be very powerful in processing large amounts of data for a wide variety of tasks at the LHC. The distributed model can already be used as is for very fast training with HDF5 data, a format commonly used in particle physics machine-learning challenges open to researchers outside of the big LHC collaborations. Once it has been made fully scalable, it will be able to train directly on both experimental data and simulated data from the LHC – possibly requiring some pre-processing of the files.
7 Acknowledgements

This research was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation, the Center for Interdisciplinary Mathematics at Uppsala University and Combient Competence Centre for Data Engineering Sciences at Uppsala University. This project originated in the Scalable Data Science and Distributed Machine Learning course of the WASP graduate school that was supported by Databricks University Alliance with an infrastructure grant from AWS. We would also like to thank Vinicius Mikuni for help in getting the model set up and Tilo Wiklund for his support as an industrial mentor sponsored by Combient Mix AB, Stockholm.

References

Paper III
Rotation-Equivariant Semantic Instance Segmentation

Karl Bengtsson Bernander  
Centre for Image Analysis, Uppsala University  
Uppsala, Sweden  
karl.bengtsson.bernamder@it.uu.se

Joakim Lindblad  
Centre for Image Analysis, Uppsala University  
Uppsala, Sweden  
joakim.lindblad@it.uu.se

Robin Strand  
Centre for Image Analysis, Uppsala University  
Uppsala, Sweden  
robin.strand@it.uu.se

Ingela Nyström  
Centre for Image Analysis, Uppsala University  
Uppsala, Sweden  
ingela.nystrom@it.uu.se

Abstract

Advances in image segmentation techniques, caused by convolutional neural network architectures like U-Net, show promise for tasks such as automated cancer screening. Recently, these methods have been extended to detect different instances of the same class, which could be used to, for example, characterize individual cells in whole-slide images. Still, the amount of data needed and the number of weights in the network are substantial. To alleviate these problems, we modify the method of semantic instance segmentation with a discriminative loss function to also preserve equivariance to the p4 symmetry group of 90-degree rotations. We perform three experiments on a synthetic dataset of scattered sticks and a subset of the Kaggle 2018 Data Science Bowl, or BBBC038, consisting of segmented nuclei images. Results indicate that the rotation-equivariant architecture yields DICE scores similar to a baseline architecture. However, in contrast to the baseline architecture, the rotation-equivariant architecture provides identical pixel classifications when comparing an input image and copies which have been rotated by multiples of 90 degrees. This is a promising step towards reducing the amount of training data in biomedical settings, as well as reducing overfitting. The method can be applied to other tasks involving pixel-level instance segmentation.

1. Introduction

Semantic segmentation and labelling of instances is sometimes said to be one of the most difficult problems in the field of image analysis. Each pixel in the image needs to be classified, and different instances separated from each other. This presents many problems, even for humans, when attempting to isolate objects with similar characteristics close to each other. Consider a pathologist looking in a microscope, trying to diagnose cancer by observing a sample of cells. The cells are numerous, often occluding each other, varying in shape, color and size, and accompanied by waste from the staining process. The diagnostic process can be time-consuming and tedious even for an experienced pathologist. Automation of challenging processes like these is highly desirable.

In recent years, automated semantic segmentation has made great progress, mainly due to convolutional neural networks such as U-Net [15] and The One Hundred Layers Tiramisu [13]. However, typically for all deep learning methods, there is a need for a lot of training data. Suitable data can be difficult to find, and the more data used in the training, the longer the training takes. One typical strategy for reducing the amount of training data is data augmentation, e.g. rotating the data while keeping the labels intact, thereby forcing the network to map rotated versions of the input to the same output. This is not without problems however - rotating the data can introduce interpolation artifacts, and each rotation needs to be learnt separately by the network.

Equivariant neural networks offer solutions to these problems. The core idea of these methods is to enforce equivariance to transformations such as rotations (R) throughout the network:

$$R(C(x)) = C(R(x))$$  \hspace{1cm} (1)\

where C is a convolution operation and x is the input to the corresponding layer. Put simply, rotating the input to a layer yields identical results as rotating the output of the layer. One way to achieve this is to use G-convolutions [6]. Other methods for achieving rotational equivariance include enforcing it through a loss function [14]. By using equivariant networks, data augmentation steps can be skipped [2].
While enforcing rotation equivariance for semantic segmentation has seen great progress in the last years [5], the same is not true for semantic instance segmentation, where each instance should be labelled in addition to its class. The aim of this paper is to combine the worlds of semantic instance segmentation with equivariant neural networks in order to reduce the amount of data needed for training and reduce the risk of overfitting. This is performed primarily through three experiments, outlined in Section 5. This method could be used in pathology settings as well as similar situations, such as object detection for autonomous vehicles or remote sensing.

2. Related work

Deep learning-based instance segmentation methods can loosely be grouped into several main categories [9].

The first group is the classification of mask proposals. This typically consists of two stages: first, regions where objects are likely to occur are generated and thereafter these are classified into the corresponding class, along with its bounding box. An example is the R-CNN [8].

The second group is the detection followed by segmentation. This can, very simplified, be seen as an extension of the first group. An important difference is that it further processes the mask proposals into segmentation masks. An example is the Mask R-CNN [11].

The third group is the labelling of pixels followed by clustering. These methods generally use pixel-wise segmentation architectures such as U-Net to also output an instance representation map. This map is then clustered into separate instances. An example is to use a Discriminative Loss Function [3], which this paper builds upon.

The fourth group is the methods of dense sliding windows. These methods generate object mask proposals or probability masks, which are then compared against the entirety of the image. An example is the Proposal-Free Volumetric Instance Segmentation from Latent Single-Instance Masks [1].

A fifth group consists of those methods related to cellpose method [16]. Here, the instances are labelled using a simulated diffusion process to produce spatial gradients directed to the center of the instance. These flows are then predicted by a neural network, and during test time each pixel is labelled according to the fixed points of the system.

ReDet [10] is a novel method with a similar aim as ours, but with some important differences. ReDet provides rotation-invariant instance segmentation on a local scale, by classifying and refining regions of interest. In contrast, our method provides a pixel-wise instance map over the whole image, and enforces equivariance to rotations in the p4 symmetry group on a global scale. We believe this makes our approach more suited for images with a dense distribution of objects, such as in the analysis of whole-slide images of cell nuclei, where the global orientation under the microscope is irrelevant.

3. Methods

The baseline architecture consists of the U-Net architecture [15] modified to have two output heads. The first head outputs a segmentation map, while the second head outputs an instance map. The architecture is presented in detail in Figure 1.

The instance map is used for clustering into separate instances using e.g. k-means or mean shift clustering. In this paper, we use the k-means clustering at inference time and set the number of pre-determined clusters to four. We extend a popular implementation [17] using the CC BY-SA 4.0 license [7]. The extended code can be found here.

3.1. Loss function

To train the network, we add the cross-entropy loss from the semantic segmentation to the discriminative loss from the instance segmentation. The loss function [3] enforces separation between the instances in the 16-channel representation:

\[
L_{\text{var}} = \frac{1}{C} \sum_{c=1}^{C} \sum_{i=1}^{N_c} \| \mu_c - x_i \| - \epsilon_c^2 \tag{2}
\]

\[
L_{\text{dist}} = \frac{1}{C(C-1)} \sum_{c_A=1}^{C} \sum_{c_B=1}^{C} \left[ 25d - \| \mu_{c_A} - \mu_{c_B} \| \right] \cdot \left( c_A \neq c_B \right) \tag{3}
\]

\[
L_{\text{reg}} = \frac{1}{C} \sum_{c=1}^{C} \| \mu_c \| \tag{4}
\]

\[
L = \alpha \cdot L_{\text{var}} + \beta \cdot L_{\text{dist}} + \gamma \cdot L_{\text{reg}} \tag{5}
\]

where we set the constants in the loss function \( L \) to \( \alpha = 1.0, \beta = 1.0, \) and \( \gamma = 0.001 \). Equation (2), \( L_{\text{var}} \), corresponds to a term that encourages pixels from the same instance to minimize the distance to their center. Equation (3), \( L_{\text{dist}} \), pushes different clusters apart from each other. Equation (4), \( L_{\text{reg}} \), prohibits cluster terms from growing too large. The number of clusters \( C = 4 \), \( N_c \) is the number of elements in cluster \( c \), \( x_i \) is an embedding, and \( \mu_c \) is a cluster center. \( \| \cdot \| \) is the L1 or L2 norm, and \( [x]_\beta = \max(0, x) \). The margins are defined as \( \delta_{\text{var}} = 0.5 \) and \( \delta_{\text{dist}} = 1.5 \).

3.2. Rotation-equivariance of architecture

From the selected baseline network, we construct a network that is equivariant to rotations by design, using the E2CNN library [18]. First, the input image is converted
Figure 1. The modified U-Net architecture, with its two output heads. The upper 2-channel head is the segmentation output, while the lower 16-channel head is the instance representation output. The numbers inside the colored boxes indicate the output number of channels for the corresponding operation. All convolution and pooling parameters are from the default pytorch and e2cnn settings unless otherwise noted.

Figure 2. The full instance segmentation method is equivariant to rotations of multiples of 90 degrees. The input is fed into the rotation-equivariant U-Net, which outputs a binary segmentation and a 16-valued instance representation. The instance representation is then clustered into instances. Rotations act only on pixel coordinates. No matter where a rotation of 90 degrees is applied, the final output is identical.

from a standard torch tensor to a GeometricTensor. Standard convolutions are replaced with group-equivariant convolutions. The symmetry group is the set of the four discrete 90-degree rotations in a circle, also known as the p4 group. Activations, poolings and normalizations make use of the respective equivariant functions from the E2CNN library.

For the upscaling layers in the upblock (the decoder), the corresponding R2ConvTransposed function is used with a sigma parameter of 0.2. For the skip-connections, where the layers from the down-block and up-block are merged, the direct sum function is used. Finally, group pooling layers are added to the two output representations, and the resulting GeometricTensors are converted back to torch tensors.
3.3. Rotation-equivariance of clustering

During inference, after a forward pass in the rotation-equivariant network, the instance representation needs to be clustered into separate instances. For the complete method to be equivariant to the $p4$ group of multiples of 90-degree rotations, this clustering step must be shown to be equivariant to rotations as well. Figure 1 illustrates the situation for a generic clustering algorithm.

At the clustering step, the feature map has the same number of pixels as the input. Each pixel is associated with 16 scalar values, which are invariant under rotation of the pixel coordinates. The clustering algorithm operates only in the space spanned by these 16 scalar values, which looks the same regardless of the rotation of the underlying pixels. Therefore, the clustering step is invariant to rotations. This holds for both $k$-means and the clustering in the discriminative loss, as well as any other clustering algorithm. This also holds for any transformation acting on the pixel coordinates. Hence, in this case, the complete method is equivariant to rotations of multiples of 90 degrees.

4. Datasets

To evaluate and compare the baseline and rotation-equivariant methods, two datasets were used. Both datasets consist of sets of one input image, one segmentation mask and one image stack with a mask for each instance.

4.1. Synthetic scattered sticks

The synthetic scattered sticks consists of a fixed number of rectangular sticks of identical size, but with varying positions and orientations. The sticks are arbitrarily placed in the image and can also occlude each other, meaning that any instance segmentation method must take into account the detailed shape of the objects and cannot only consider adjacent pixels.

The dataset was partitioned into a training set of 512 images and a test partition of 16 images. For both partitions, the data was dynamically generated for each run of the experiment.

4.2. BBBC038

The BBBC038 dataset from the Kaggle 2018 Data Science Bowl [4] consists of a collection of biological images of cell nuclei. The samples have been treated and imaged under highly varied conditions, differing in illumination, magnification, and image modalities. The cells themselves come from different animals and tissues in different states, for example, cell division or genotoxic stress. This dataset offers a highly diverse set of shapes, colors and distribution of objects for semantic segmentation algorithms.

We modified the images in several ways to align them with our experimental setup. First, the dataset is intended for semantic segmentation, while we are interested mainly in semantic instance segmentation. Therefore, we extracted both segmentation and instance masks. Secondly, the clustering methods need the number of instances or clusters as a constant parameter. Since the number of cells in the images vary, the images were cropped to ensure they contained the predetermined number of cells. The image pixels outside the cropped area were padded with the mean intensity of the original image to a fixed image size of 256 by 256 pixels. We use 500 images for the training set and 16 images for the test set. Pixel intensities were converted to greyscale. An example can be seen in Figure 3.

5. Experimental setup

We performed three main experiments. The first tested if the rotation-equivariant architecture would perform similarly as the baseline network in terms of pixel-wise classification accuracy. To achieve this, both the baseline and rotation-equivariant networks were trained for 100 epochs. Once during each epoch, instance segmentation DICE scores were calculated. Experiment details such as hyperparameters are outlined in Table 1. The tests were re-
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss Function</td>
<td>Cross-entropy, Discriminative</td>
</tr>
<tr>
<td>Weight Initialization</td>
<td>He</td>
</tr>
<tr>
<td>Optimizer</td>
<td>SGD</td>
</tr>
<tr>
<td>Momentum</td>
<td>0.9</td>
</tr>
<tr>
<td>Weight Decay</td>
<td>0.001</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>0.01, dynamically decreasing</td>
</tr>
<tr>
<td>Batch Normalization</td>
<td>No</td>
</tr>
<tr>
<td>Number of Epochs</td>
<td>100</td>
</tr>
<tr>
<td>Activation Functions</td>
<td>ReLu</td>
</tr>
<tr>
<td>Dropout</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1. Hyperparameters and details of the networks.

The second experiment examined if data augmentation would yield higher accuracy when using data augmentation for the baseline method. Also, it tested if the rotation-equivariant method could match these results without data augmentation. First, 100 images from the BBBC038 training set were selected and rotated by 90, 180 and 270 degrees while keeping their labels intact. Then, both the unaugmented and the augmented datasets were input to the baseline and rotation-equivariant architectures.

The third experiment tested empirically that the rotation-equivariant architecture is equivariant to rotations. Details for how to perform the measurements are outlined in the following subsections.

5.1. Evaluation of segmentation

To evaluate the instance segmentation, a metric that yields a value of 0 when there is no overlap between labels and a value of 1 when there is perfect overlap is desired. To achieve this, we used the DICE score:

\[
\text{DICE} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}
\]  

This was performed instance-wise, in addition to the background. Since the instances belong to the same class, there exists a multitude of combinations of possible matches between predicted and ground truth labels. To handle this uncertainty, the first ground truth label was compared pixel-wise with each predicted label, excluding the background. The predicted label with the highest overlap was considered the most likely match. From this, the DICE score for the first instance could be calculated. Then, both the ground truth label and the matching label were removed from the lists of possible matches. This was repeated until all the labels and instances had been processed. Then, the background DICE score was calculated. Finally, the total DICE score of the segmented image compared to the ground truth was calculated by taking the mean of all the instance and background DICE scores.

5.2. Evaluation of equivariance to rotations

To test equivariance to rotations, the following procedure was observed. First, one image and a 90-degree rotated copy of the same image was sent through the model, generating output one and two, respectively. Then, output two was rotated by -90 degrees. Finally, both outputs were compared pixel-to-pixel. The final output segmentations needed to overlap exactly for perfect equivariance to 90 degree rotations.

6. Results

The first experiment involved testing both architectures on both datasets. The DICE scores for the experiments on the synthetic scattered sticks dataset are shown in Figure 4. Both the baseline and the rotation-equivariant models reached DICE scores of around 0.88 in the last epoch. The rotation-equivariant model converged in fewer epochs compared to the baseline model. Example outputs can be seen in Figure 5. They are selected to be as close as possible to the mean DICE score of the 16 images in the test set.

The DICE scores for the experiments on the modified BBBC038 dataset are shown in Figure 6. The baseline reached a DICE score of around 0.673 for the last 20 epochs, and the rotation-equivariant reaches a score of around 0.638. The average over all epochs is 0.626 for the baseline model and 0.613 for the rotation-equivariant model. Visual results of the instance segmentation in the inference phase after training can be seen in Figure 7. The examples are again selected to be as close as possible to the mean DICE score of the 16 images in the test set.

The second experiment was to test the effects of data augmentation by rotations of multiples of 90 degrees on both architectures. The results can be seen in Figure 8. The baseline network with data augmentation performed the best with a DICE score of 0.67 at the last epoch. The baseline network and the rotation-equivariant network came next, at 0.64 and 0.63, respectively. The rotation-equivariant network with data augmentation scored 0.59.

The third experiment was to test for equivariance, outlined in Section 5.2, on the synthetic scattered sticks dataset. Here, on average 167.0625 pixels (that is, 0.25 % in the 256 x 256 size images) differed between the two outputs using the baseline network. The input was the 16 images in the test set. For the rotation-equivariant network the difference was 0 for all images, indicating perfect equivariance.
7. Discussion and conclusions

From Section 6, it is clear from the first experiment that the baseline and the rotation-equivariant networks are comparable in terms of DICE scores and visual interpretations of the instance segmentations. The DICE score on the scattered sticks dataset is high, and lower on the modified BBBC038 dataset. We believe BBBC038 is a challenging and highly varied dataset, with cells being adjacent to each other and in different shapes, sizes and colors.

The second experiment investigated if rotation-equivariant networks could replace baseline networks with data augmentation. From Figure 8, we conclude that the network would need to be trained longer to fully verify this. The experiments should also be repeated and averaged.

We have shown that the rotation-equivariant architecture is indeed equivariant to rotations from the p4 group, both by reasoning and by experiment 3, including the final clustering step. This is a promising step towards reducing overfitting. Also, there is no need to perform data augmentation using the same transformations as in the symmetry group of the equivariant network. To the best of our knowledge, this is the first implementation of a pixel-level rotation-equivariant semantic instance segmentation network.

The current equivariant architecture has some limitations and can be extended in useful ways. First, it could use other symmetry groups in the E2CNN library, making it equivariant to the corresponding transformations. Secondly, the clustering methods in the discriminative loss and the post-processing step could be modified to not require a predetermined number of clusters. This would enhance its usability in settings where the number of instances is unknown.

The method warrants some ethical considerations. Improving automation in pathology could lead to faster and more efficient diagnosis of medical conditions, which could save many lives. However, it could also lead to redundancy among trained pathologists. What we believe is more likely is that these methods will assist them, freeing up time for
Figure 6. DICE scores for the BBBC038 dataset during training for a) the baseline network and b) the rotation-equivariant network. Each color represents one experimental run with identical setups, while the black dots represent the mean of the five experiments. For a), the mean DICE score is 0.626 for all epochs and 0.673 for the last 20 epochs. For b), the mean DICE score is 0.613 for all epochs and 0.638 for the last 20 epochs.

Figure 7. Example instance segmentations on the modified BBBC038 dataset. Images show a) the input image b) an instance segmentation from the baseline architecture, and c) an instance segmentation from the rotation-equivariant architecture.

other tasks. This is similar to how search engines have not made traditional diagnosis by medical doctors obsolete.

Furthermore, the method could be used for other purposes, such as navigation by autonomous vehicles. However, it could also be used for more precise surveillance of individuals or target tracking in weapons systems. A continuous ethical discussion as well as legislation is required as a complement to technical developments [12].

Acknowledgments

We thank Professor Fred Hamprecht, University of Heidelberg, Germany, for providing guidance on instance segmentation.

References

Figure 8. Experiments to measure the effects of data augmentation. The images from the BBBC038 dataset were rotated 90, 180 and 270 degrees, keeping the labels intact. Both the original data and the augmented data were then used as input to the two networks.
Recent licentiate theses from the Department of Information Technology

2021-001 Niklas Gunnarsson: *On the Registration and Modeling of Sequential Medical Images*

2020-006 David Widmann: *Calibration of Probabilistic Predictive Models*

2020-005 Anna Wigren: *Exploiting Conjugacy in State-Space Models with Sequential Monte Carlo*

2020-004 Muhammad Osama: *Machine Learning for Spatially Varying Data*

2020-003 Christos Sakalis: *Securing the Memory Hierarchy from Speculative Side-Channel Attacks*

2020-002 Ulrika Sundin: *Global Radial Basis Function Collocation Methods for PDEs*


2019-006 Kristiina Ausmees: *Efficient Computational Methods for Applications in Genomics*

2019-005 Carl Jidling: *Tailoring Gaussian Processes for Tomographic Reconstruction*

2019-004 Amin Kaveh: *Local Measures for Probabilistic Networks*

2019-003 Viktor Bro: *Volterra Modeling of the Human Smooth Pursuit System in Health and Disease*