UNIVERSITÄT KLAGENFURT
FAKULTÄT FÜR TECHNISCHE WISSENSCHAFTEN
INSTITUT FÜR INFORMATIONSTECHNOLOGIE

Markus Fox

Object Recognition with Mask R-CNN and Capsule Networks

MASTERARBEIT

zur Erlangung des akademischen Grades
Diplom-Ingenieurin / Diplom-Ingenieur

Studium
ANGEWANDTE INFORMATIK

Alpen-Adria-Universität Klagenfurt
Fakultät für Technische Wissenschaften

Begutachter:
Assoc.-Prof. Dipl.-Ing. Dr. Klaus Schöffmann
Institut für Informationstechnologie

Dezember 2020

Eidesstattliche Erklärung

Ich versichere an Eides statt, dass ich

- die eingereichte wissenschaftliche Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe,
- die während des Arbeitsvorganges von dritter Seite erfahrene Unterstützung, einschließlich signifikanter Betreuungshinweise, vollständig offengelegt habe,
- die Inhalte, die ich aus Werken Dritter oder eigenen Werken wortwörtlich oder singemäß übernommen habe, in geeigneter Form gekennzeichnet und den Ursprung der Information durch möglichst exakte Quellenangaben ersichtlich gemacht habe,
- die eingereichte wissenschaftliche Arbeit bisher weder im Inland noch im Ausland einer Prüfungsbehörde vorgelegt habe und
- bei der Weitergabe jedes Exemplars (z.B. in gebundener, gedruckter oder digitaler Form) der wissenschaftlichen Arbeit sicherstelle, dass diese mit der eingereichten digitalen Version übereinstimmt.

Mir ist bekannt, dass die digitale Version der eingereichten wissenschaftlichen Arbeit zur Plagiatskontrolle herangezogen wird.

Ich bin mir bewusst, dass eine tatsachenwidrige Erklärung rechtliche Folgen haben wird.

Markus Fox e. h. Klagenfurt, Dezember 2020

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der gesamten Dauer meines Studiums und besonders während der Anfertigung dieser Masterarbeit unterstützt haben. Zuerst gebührt mein Dank Herr Assoc.-Prof. Dipl.-Ing. Dr. Klaus Schöffmann, der meine Masterarbeit betreut und begutachtet hat. Für das Vertrauen in meine selbstständige Arbeitsweise, die hilfreichen Anregungen sowie die konstruktive Kritik bei der Erstellung dieser Arbeit möchte ich mich herzlich bedanken.
Anschließend möchte ich mich bei den Personen bedanken, die ich zu meiner Familie zähle. Sie haben mich in den guten wie in den schlechten Zeiten emotional und materiell stets unterstützt, wodurch ich eine sorgenfreie Mentalität aufrecht erhalten konnte. Ein besonderer Dank gilt meinen Eltern Johann und Sonja Fox, sowie meiner Partnerin Daniela Stefanics für den starken emotionalen Rückhalt über die Dauer meines gesamten Studiums.

Abstract

Visual perception and recognition is one of the central blocks towards solving artificial intelligence. We discuss the immense progress of research in computer vision, specifically the emergence of deep learning during the last decade. For evaluation we apply a state-of-the-art deep learning algorithm (Mask R-CNN) on real world medical data (cataract surgery videos) to show the effectiveness as well as limitations of convolutional neural networks in object recognition (surgical instruments). Our experiments show the importance of both data quality/quantity and reveal that in this case common data augmentation strategies do not improve the models performance. Second we reproduce and evaluate a new architectural design of neural networks, called capsule network (CapsNet), invented by the English computer scientist and cognitive psychologist Geoffrey Hinton. By extending the original matrix capsule network architecture with residual blocks, as found in the well-known ResNets, we were able to reduce the computational time by a factor greater than 4 for two medium sized image classification datasets (smallNORB, STL10), while reaching better performance as our standard CapsNet implementation. In case of STL10 the standard CapsNet did not converge but our Residual-Capsnet reached substantial performance (67,66% classification accuracy) comparable to ResNet50 (75,42%). Our result confirms the findings of related work in this area, that CapsNets might be a strong competitor in deep learning and worth evaluating when it comes to tasks that neural networks try to solve.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Computer Vision (CV) is a research area that tries to find computer-based solutions for problems that can be solved using the human visual system. It draws inspiration from many scientific fields, from the studies of cognitive- and neuroscience in psychology and biology, to information retrieval and machine learning in computer science and mathematics, through to robotics and optics in physics and engineering. Therefore it can be classified as one of the central blocks towards solving Artificial Intelligence (AI). Most of the research in recent years was done working on detecting, classifying and measuring objects in digital images for further processing. These tasks are categorized into one larger fundamental and general problem of CV named Visual Recognition. There are a good amount of visual recognition problems related to image classification, such as object detection, action recognition and scene understanding, as described in Section 2.1. Humans use all of them in some way or form and therefore it always has been a fascinating research field for every one who is inspired by AI. Due to the drastic improvements in computational power, in big parts with technical advances by companies like Nvidia[1], AI research has been able to flourish at an unprecedented rate.

At present, modern Deep Learning (DL) is the number one choice for research and applications in CV. DL architectures make perfect use of the great parallel computing power of Graphics Processing Units (GPUs) and continuous improvements in Neural Network (NN) model architectures have provided us with powerful solutions to tasks in visual recognition. The basis of such DL architectures are very often Convolutional Neural Networks (CNNs), which were loosely inspired by cognitive science and first successfully applied by [LeCun et al., 1990]. Technical details of CNNs are described in Section 2.2. The core idea goes back even many decades further to the *Mark I Perceptron* in [Rosenblatt, 1958], where the American psychologist Frank Rosenblatt built a machine

[1] `https://www.nvidia.com/`

that tied together many perceptrons to create some form of machine intelligence. Unfortunately, it was shown that his creation was not able to solve tasks that are not linearly separable, such as the XOR function. Due to his early death his perceptron idea was not properly defended against other research and it took decades before research finally picked up the idea once again and extended it to non-linear functions. Although the foundation was laid, computers in the nineties were far away from being powerful enough to process high dimensional data, such as digital images. The real breakthrough came in 2012 when [Krizhevsky et al., 2012] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) with their architecture called *AlexNet*. The architectures was eigth layers deep, out of which five were convolutional layers (some using Max-Pooling) and the rest fully connected layers. The computational improvements came in the form of an efficient GPU implementation. Impressively, Alex Krizhevsky's network outperformed the runner-up by over 10% in top-5 classification error in 2012.

In each of the following years a CNN has won the challenge, improving the accuracy of the model detecting one of a thousand objects - at some point even performing better than humans. Figure 1.1 shows the top-5 classification error rates from year 2011 up to 2016, including the results of a human test person. An interesting improvement in performance happened in 2015, where [He et al., 2016] proposed *ResNet* to outperform the human. At this point the question was how deep (total depth of layers) the networks can possibly get while still performing well. ResNets are using skip connections to improve information flow (updating weights) through the network as it gets deeper. Although there are improved architectures at the time of writing, most out-of-the-box solutions are still using ResNets in some way or form, as they are fast and usually reliable to work on many different datasets and visual recognition tasks.

Undoubtedly, CNNs have made great progress in solving CV tasks. However, we are still far away from modeling the human visual system in detail and solving the mystery that is our human brain. Therefore, our main motivation is to first implement some state-of-the-art Deep Learning (DL) techniques (e.g., Mask R-CNN) on medical data to better understand the underlying concepts, and further we do not want to stop there but keep improving upon the state-of-the-art in AI. This is why in this work we additionally evaluate the possibilities of a relatively new type of DL architecture called Capsule Network (CapsNet), introduced and made popular by Geoffrey Hinton - a British computer scientist and cognition psychologist - who has also been Krizhevsky's doctoral advisor and coauthor of the previously mentioned AlexNet paper.

For Mask R-CNN we choose to apply it to real world medical data. In the medical domain automatically detecting surgical tools in recorded surgery videos is an important building block of further content-based video analysis. In ophthalmic surgery, interventions are performed on the human eye with tiny instruments under a microscope. By far

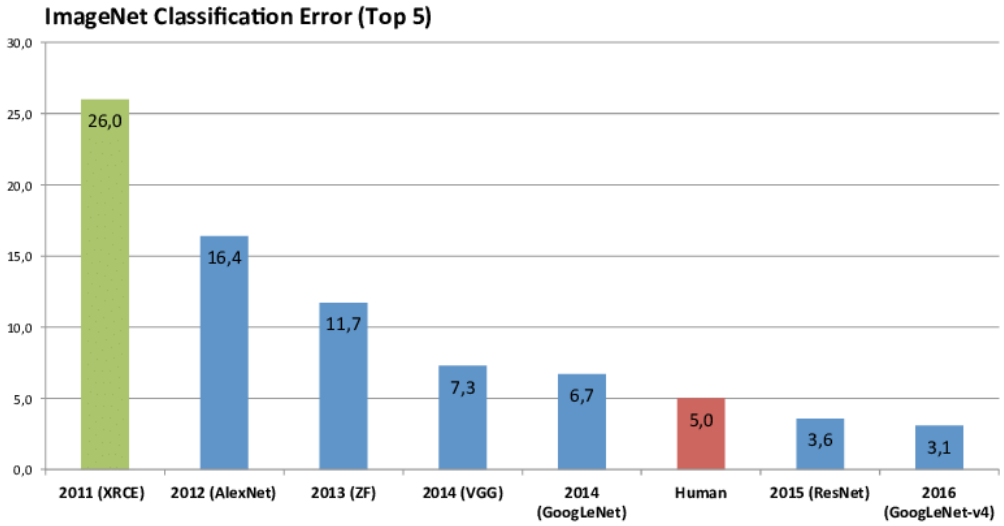

Figure 1.1: Top 5 ImageNet classification error rates from 2011 to 2016. CNNs are shown in blue, traditional algorithms in green. Source: [von Zitzewitz, 2017].

the most frequent operation in this medical field is cataract surgery (lens replacement). Ophthalmology in general is a very challenging surgical discipline, which requires special operation techniques and psycho-motor skills that need to be trained intensively. Unfortunately, due to the reason that there is not a lot of space in the operation room and the microscope only allows one additional trainee to follow the operation, an increasing number of surgeons record videos from their surgeries and use them for teaching and training.

Since surgeons are typically very busy, important moments in such ophthalmic videos (i.e., frames and segments) could be indexed automatically, in order to make the post-operative use more efficient. One way to solve this task is by providing an automatic content search feature. This is why research in this domain focuses on temporal segmentation of ophthalmic videos into surgical phases as well as on detecting the use of specific tools in frames or temporal segments ([Quellec et al., 2014], [Al Hajj et al., 2017]). However, although several works have been published for these problems (see Section 3.4), and recently even a challenge on automatic tool annotation ([Al Hajj et al., 2019]) was started, there has been no investigation of the localization of instruments in frames of the video yet.

In the object detection part of this work, we focus on automatic tool recognition and localization in cataract surgery videos on a per-pixel basis, which allows to tell whether a particular pixel in a frame of a video belongs to a specific surgical instrument or not. This instrument localization is much more fine-grained than a frame-based multi-

label classification of instruments and provides important additional context for further analysis. For example, from instrument localization we can tell whether a particular tool is used on the left or right side of the eye, if it is inside or outside the iris, and whether specific instruments overlap with each other or not. Moreover, with fine-grained localization of surgical tools it is also easier to track these objects over several frames in the videos and thereby derive further semantic information (e.g., how smoothly they are used, and which type of surgical action it could be – based on motion analysis of the instruments). The localization will be done using a state-of-the-art Mask R-CNN architecture that has been proven to produce good results in similar tasks. It is an interesting choice for object recognition as it combines a lot of fundamentals of CNN-research into one architecture. However, as we find and discuss its limitations we will also move on and evaluate a completely new DL architectural design called CapsNet.

CapsNets are trying to add additional structure to CNNs and reroute their outputs in a way that the network can learn hierarchies of objects as well as to output feature representations (in form of real values) next to the usual activation probabilities. This opens up many possibilities, such as learning to detect different representations of the same feature, and most importantly coming closer to the way the human visual system works. A detailed description of CapsNets is found in Section 3. Interestingly the first publication of this idea ([Hinton et al., 2011]) happened shortly before the success of AlexNet. This might be the reason why the idea has not been approached and extended sooner than 2017, although the inventor Geoffrey Hinton has criticized the use of Pooling operations, found in AlexNet and many other CNNs, publicly. [Sabour et al., 2017] was the first efficient implementation of a CapsNet and it has recieved public attention for some time. Due to its computationally expensive nature, [Hinton et al., 2018] have released a follow-up paper only one year later, where the center part of the architecture, the routing algorithm, was changed. Our main goal will be to reproduce and evaluate this latest approach and slightly extend it. This will open up the possibilities to improve the overall number of tasks that CapsNets can solve and allow us to judge more clearly whether this idea can have a big impact in the future of AI or not.

1.2 Contribution

The main contributions of this thesis are the following:

- We add ground-truth annotations for instance segmentation of surgical tools in cataract surgery videos and thereby create two new datasets. To make them publicly available for further research we publish them on our website[2].

[2] `http://ftp.itec.aau.at/datasets/ovid/InSegCat/`

- We evaluate a state-of-the-art instance segmentation CNN architecture (Mask R-CNN) on these datasets and compare results for different backbone networks and data augmentation strategies. We find that, in contrast to many other domains, common data augmentation methods do not necessarily improve segmentation performance.

- We analyze and explain the achieved segmentation performance with respect to each class (surgical tool) and relate results to the different visual content quality of the two datasets.

- As we move on from Mask R-CNNs we reproduce an original CapsNet architecture of author Geoffrey Hinton using a modern Machine Learning (ML)-Framework named *PyTorch* and compare our results to the one's in the original paper.

- We first extend the model architecture by adding a decoder (reconstruction) network that learns to reconstruct the input image from the outputs. This allows to visualize the learned features of the network and see what exactly the network has learned.

- Additionally we extend the architecture by adding *residual blocks* (as found in ResNets) to the head of the network. This allows us to downscale the input of CapsNet during computation and reduce the overall computational time for higher dimensional input drastically.

- The Residual-Capsnet reaches comparable performance on an image classification problem dataset (STL10), where the original CapsNet was not able to converge at all.

- All CapsNets are compared to well-understood baseline CNN architectures in terms of training- and inference time as well as classification performance. A simple three-layer CNN with pooling serves as the weak baseline and ResNet50 as the strong baseline architecture. All evaluations are done on one and the same GPU to make fair comparisons.

- The full CapsNet source code will be made publicly available for other researchers and interested parties.[3]

- Overall we strive to give a structured overview of object recognition using some methods of deep learning.

[3] `https://github.com/MarkusFox/residual-capsnet`

CHAPTER 2

Visual Content Analysis with Neural Networks

2.1 Visual Recognition Tasks

In this chapter we first want to discuss some of the main tasks in visual recognition, focusing on the bridge between how human brains and artificial neural networks process inputs. We will generate a link between the tasks and some of the famous CV datasets, to give an intuition of how and why the data was collected in such a way. We will already mention some of the neural architectures that were designed to solve a given task, but the more detailed explanations will follow in the sections afterwards.

2.1.1 Object Recognition

Object recognition is often referred to as the 'holy grail' of CV, as it is one of the most basic concepts in visual recognition and perception. As you observe an arbitrary object in nature, you immediately categorize it, usually using some language label, which can give meaning and intuition for further cognitive processes, such as a fight-or-flight response. In case of a computer the input of its visual system will be a digital image with one or more objects present. In this world we can find a vast amount of different objects and each object often consists of a hierarchy of multiple sub-objects - e.g., a face consisting of eyes, nose, mouth, wrinkles, or an eye consisting of pupil, iris, eyelid, etc. The closer one looks the more we are able to recognize and sometimes we are only able to detect something if we have previously learned about it, such as making a difference between different kinds of bird species instead of simply labeling all of them as birds. This is true for humans as well as computers. Much effort has gone into creating datasets of images where various amounts of different objects (classes) can be seen. The most famous example of datasets in this category is ILSVRC, or ImageNet, where a total of 1000 object

classes are provided on a total of 1.431.167 images. The classes are organized in the form of the *Wordnet* hierarchy, a large lexical database of the English language. Although the last challenge was held in 2017, ILSVRC is still one of the most important benchmark datasets in object recognition. More recently in [Sun et al., 2017], Google Research have published work using their own *JFT-300M* dataset which consists of 375M noisy labels for 300M images. Unfortunately, this dataset is not yet accessible to the public but we can expect data like this to be future benchmarks, especially as neural networks are getting more accurate with less time spent on computation. The ultimate goal is to reach or even exceed human-level performance but the power of the complex human brain is not easily replicated. We use large structured datasets to test our developed neural networks on subtasks like object recognition. Before we dive deeper we should mention that the reason why we split everything into smaller and smaller subtasks is the fact that the problem at hand is very complex and it has proven effective to break a problem down into its parts and solve subproblems first, before combining the solutions for the bigger picture (e.g., divide-and-conquer algorithm in computer science).

Image Classification

Object recognition can be further split into categories, most importantly classification and detection. In image classification each image is associated with labels of the objects seen in the image. The previously mentioned ILSVRC was initially designed for classification. This task can be further split into single-label and multi-label classification. Both are similar, but the latter is sometimes much more difficult to solve with CNNs. In single-label classification the observer is tasked to assign exactly one label (class) to each image. Often in such datasets each image clearly shows one object, such as a cat, and for example the label CAT is the only correct answer. Whereas in multi-label classification there can be multiple observable objects on the input image and therefore the correct answer would include a binary YES/NO for each of the possible classes. Or more simply put an enumeration of all visible objects. Classification can also be done without objects and with emotional reactions. Due to the warm colors of a sunset the image could be classified as BEAUTIFUL, CALM etc. or a piece of art could be classified as COLD or SAD although no link to an object is needed. However, most of the times we use objects to explain the content of the image and in this work we focus mainly on objects when doing image classification. We describe more datasets of this nature in Section 4.3.

Object Detection and Segmentation

Beyond pure classification it might also be useful to extract information on the precise location of the objects inside the image - e.g., to track movements of objects over consec-

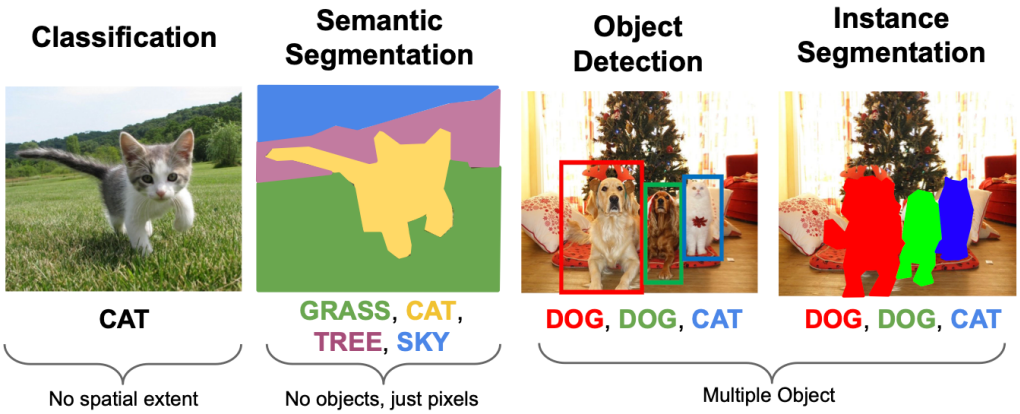

Figure 2.1: Demonstrating the main object recognition tasks in computer vision. Source: [cs2, 2020].

utive video frames. Two main approaches have been pursued in this area: 1) semantic segmentation and 2) object detection. See Figure 2.1.

In semantic segmentation each pixel in the image is classified belonging to a certain class, which creates precise pixel-by-pixel object regions on the image, if working accurately. For each image during training each pixel is labeled with a semantic category. At test time the network classifies each pixel of a new, previously unseen image. Obviously, it is not sufficient to classify each pixel individually without using some context, as for example a single black pixel can belong to almost any class. One idea is to use a sliding window and extract a patch of certain size to classify the center pixel of the patch. This was done in [Farabet et al., 2012] to produce good results, but its drawback is that it is quite inefficient when it comes to computational resources, as it does not reuse shared features between overlapping batches.

Much better was the intuitive idea to use a CNN (just as in image classification) to encode the entire image and do semantic segmentation on top. Since CNNs are using downsampling to reduce the overall computational time of deep architectures and segmentation requires the output to be of the same size as input, the idea required a method of upsampling inside network. One such method would be to do the opposite of pooling, later described in Section 2.2.2. However, a much better method was found in *transpose convolution*, a convolution operation where the input serves as weight of the filter and the output is of larger dimension as the input and the weights are trainable. Architectures for semantic segmentation that consist of a bunch of convolutional layers with downsampling and upsampling are known under the name *Fully Convolutional Networks (FCN)* ([Long et al., 2015]).

As an example for humans doing semantic segmentation imagine the point-of-view of a race car driver approaching a corner at the end of a long straight on the famous race-track Nürburgring Nordschleife. The driver needs to hit the brakes at the perfect moment to slow down his car to the right speed where he can hit the apex and carry maximum speed through the corner. He might pick an object as his reference point for the braking but the rest of his vision will be processed as segments rather than objects. In particular the track itself and the grass off-track are large segments that the driver processes to plan his steering motion for placing the car on the right line. And things like the grandstand full of fans are merely labeled as background during times of intense focus.

In the second approach called object detection the network finds the object in the image and next to the class also outputs its coordinates on the image using a bounding box. This allows for multiple objects to be located and classified correctly. In contrast to multi-label classification and semantic segmentation this type of task also supports finding multiple instances of one and the same class. This might be necessary in face detection applications, where the goal is to find all the faces on the image and their locations. Or, as seen in the third picture of Figure 2.1, finding both dogs as well as the cat.

In addition to localizing the object via bounding box, we can also apply semantic segmentation to the content of the extracted box. This combination, as shown in the fourth picture of Figure 2.1, is called *instance segmentation*. It makes our prediction of objects even more accurate and lets us extract even more information out of a single image than with all other previously mentioned approaches. In the later sections of this work we will describe and evaluate instance segmentation with a CNN.

2.1.2 Scene Understanding

So far we have focused on pure objects when explaining images. But recognizing and counting objects is hardly ever sufficient to explain the scenery of an observed image. As we might have two separate situations where a man is seen with his dog. In image number one the two seem to be calmly walking down the street, as the person probably is the owner of the dog going for the daily walk routine. On the second picture the man is lying on the floor smiling as the dog is positioned on top in a friendly gesture. In both images a man and a dog can be detected, but the context is very different. It is important to somehow link the observable objects to each other. Again, there are multiple approaches to deal with this task.

Image Captioning

In image captioning the input image is translated into a sequence of words. For processing sequences Recurrent Neural Networks (RNNs) are one of the most successful network architectures. In contrast to the networks we have seen up to this point, RNNs can process one or many inputs that result in one or many outputs. This leads to many possible scenarios. Examples include observing one image and giving a sentence of variable length as output, observing a sequence of images (e.g., video frames) and classifying them as one action phase, etc.

RNNs have hidden layers similar to feedforward neural networks but with the main difference being that these hidden layers can have connections back to themselves. The hidden layer is described as:

$$h^t = g(h^{t-1}, x^t; \theta)$$

where h^t is the state of the hidden layer at time t, x the input and θ the set of parameters. g is a non-linear activation function, often the ReLU or tanh function. The RNN output o^t at time t is then given by:

$$o^t = f(h^t; \theta)$$

where f is a recurrence formula that is used at every time step. Interestingly the same weight parameters are re-used at every time step. As we go forward through time we need the entire sequence to calculate the gradient for backpropagation. In deep RNNs with many hidden layers the gradient values might become smaller and smaller with each step until they reach zero. New ideas have emerged and solved this problem. The most famous examples are Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and Gated Recurrent Units (GRU) [Cho et al., 2014].

For image captioning a CNN is used to extract features of the image that are used as additional weight parameters of the hidden layer of the RNN. This is done by removing the last fully connected layer, which usually scales the output down to the number of classes. The RNN then outputs a sequence of words that describes the image. Examples of correct as well as incorrect results are shown in Figure 2.2.

Scene Graph Representation

[Wolfe, 1998] states that "One could imagine that a list of N objects would be sufficient to categorize a scene, but a series of thought experiments tells us that a gist is more than a list. [...] A picture of milk being poured from a carton into a glass is not the same as a picture of milk being poured from a carton into the space next to a glass, even if all of the objects are the same". But what is the information people would convey if asked to caption? One solution is found in **Objects** - *Attributes* - RELATIONSHIPS, such as in

Figure 2.2: Four examples of doing image captioning, where the input is an image and the output a sequence of words. Top two cases are accurate, bottom two make mistakes. Source: [cs2, 2020].

scene graphs. In the scene 'White llama in front of a blue wall' we can determine the objects **llama** and **wall**, the attributes *white* and *blue*, and their relationship IN FRONT OF. A scene graph captures all of them into a graph representation. Such a graph can be found in [Krishna et al., 2017] where annotations are provided that connect vision and language in such a way. In this project called *Visual Genome* there are 108k images that show around 3.8M objects, 2.8M attributes and 2.3M relationships. An example is visualized in Figure 2.3.

To tackle this task two modules are trained: 1) visual module and 2) language module. The visual module features an object detector as well as a relationship detector to make proposals about the content. The language module makes sure to feature the correct scene graph representation of the predictions.

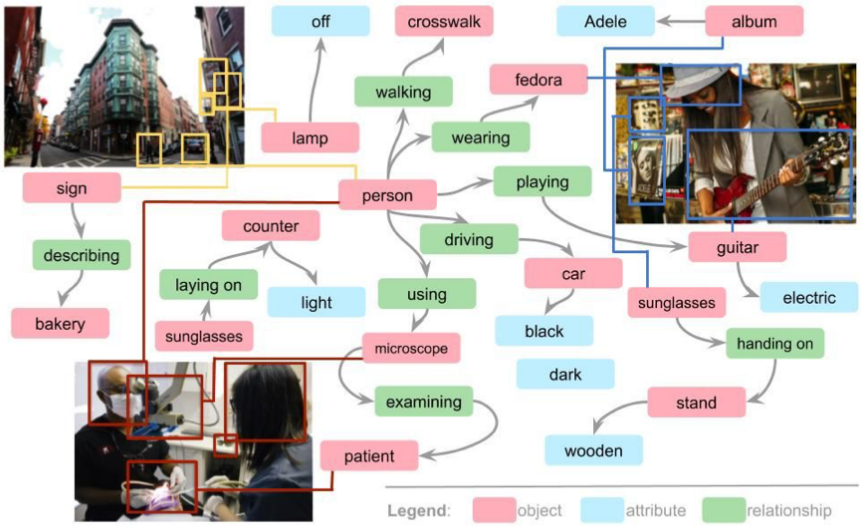

Figure 2.3: Demonstrating a small part of the scene graph of the Visual Genome project, where objects, attributes and relationships are modeled. Source: [cs2, 2020].

This approach makes more sense when comparing to the neural networks of humans, as human brains seem to work best when combining their visual and language modules. Although not covered in this work, advances in Natural Language Processing (NLP) will most likely have a strong influence in CV and vice versa, just as in the case of *transformers* ([Vaswani et al., 2017] and [Dosovitskiy et al., 2020]).

2.2 Convolutional Neural Networks (CNNs)

A CNN is a feedforward Artificial Neural Network (ANN) that typically takes as input a digital image and outputs probabilities whose meaning depend on the task it was trained on, such as the likelihoods of certain classes being visible in the image (Section 2.1.1). It is a special type of the Multilayer Perceptron (MLP), invented by Frank Rosenblatt in [Rosenblatt, 1958], so the use of multiple layers and activation functions (described in upcoming Subsections) make it possible to distinguish data that is not linearly separable. Its main building blocks are the Convolutional Layers (ConvLayers) that are stacked with other layers, such as Pooling-, Fully Connected- and Normalization Layers. The main idea was inspired by biological processes found in the visual cortex of animal brains. The first use was already in the very early 1990's but they emerged in 2012 after a CNN won the ImageNet challenge. In this section we describe the main components of CNNs. A full network very similar to *AlexNet* is later described in Section 5.1.

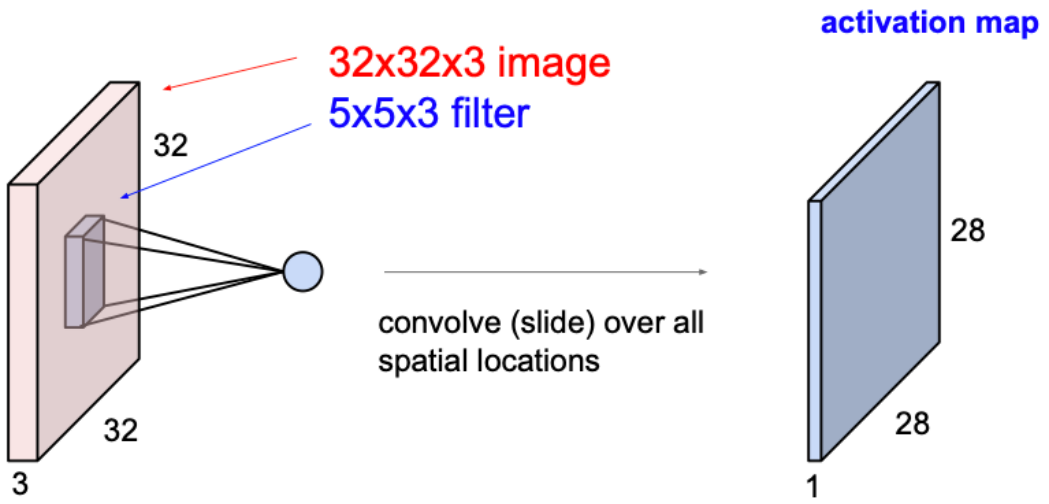

Figure 2.4: Applying a 5x5x3 filter on an image of size 32x32x3 in a ConvLayer by taking the dot product at each position results in an activation map. Here we apply a stride of 1, meaning we shift the filter by 1 for each calculation, and do not pad at the borders, so we end up with a 28x28x1 activation map. Source: [cs2, 2020].

2.2.1 Convolutional Layer

A ConvLayer is the core building block of a CNN and consists of a set of learnable weight parameters, also called a filter and/or kernel. Each filter slides (convolves) over the full width/height of the input image and calculates the dot product for the filter values and the input at each position. The depth of the input (number of image channels) must match the depth of the filter. This results in a 2-dimensional output that we call an *activation map* (Figure 2.4). Each layer can have multiple filters, each producing an activation map. By stacking all of our activation maps together we get a single three dimensional activation map. For simplicity we can think of the number of filters as the number of image channels that the next layer in the network will get as input.

The convolution operation as a whole is a linear transformation. For adding non-linearity to the network we apply an activation function after each convolution - before we send the output to the next layer. There are many possible non-linear activation functions for CNNs, with the most commonly used being Rectified Linear Unit (ReLU): $f(x) = max(0, x)$, Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ and Hyperbolic Tangent (Tanh): $tanh(x) = 2\sigma(2x) - 1$. It is critical to understand that without the use of activation functions stacking multiple linear ConvLayers in a row will collapse into a single convolution operation.

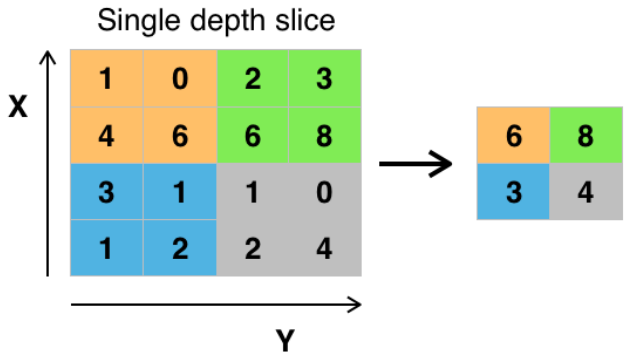

Figure 2.5: Example of max-pooling done with filter size 2x2 and stride 2. Source: [cs2, 2020].

2.2.2 Pooling Layer

To reduce overall computation a so called Pooling layer can be put after the ConvLayer. There are several types of pooling functions, with the most commonly used being the Max-pooling operation, which is supposed to simulate what neurobiologists call *lateral inhibition*[1]. It works well in a CNN because it increases the *field of view* of high level neurons (layers), making them detect more complex features in a larger region of the image. Additionally the large size reduction makes computation much more efficient. The most common practical use is a pooling of size 2x2 applied with a stride of 2, that already discards up to 75 percent of the original information (as shown in Figure 2.5). In practice the only other pooling size used is 3x2 with stride 2 (overlapping). Bigger sizes are found to be too destructive.

Although pooling leads to improved results in many CNN architectures, it has been under fire in recent years. Influental people in Computer Vision have mentioned that the pooling layer will be replaced or even discarded in favor of other techniques. For example 2018 Turing award winner Geoffrey Hinton wrote in 2014,

> *"The pooling operation used in CNNs is a big mistake and the fact that it works so well is a disaster."*

The very recent CNN architectures (such as ResNets) use pooling minimally and discarding pooling layers has also been found to be important in training good generative models, such as Variational Auto Encoders or Generative Adversarial Networks ([Goodfellow et al., 2014]). Currently it looks like we will move away from pooling lay-

[1] In neurobiology, lateral inhibition is the capacity of an excited neuron to reduce the activity of its neighbours.

ers in the future, but it is important to mention them as a valuable option for CNN architectures, especially when computational resources are limited.

2.2.3 Fully Connected Layer

A Fully Connected Layer is a layer where each neuron of layer L is connected to all neurons of layer $L+1$. The forward pass corresponds to one (large) matrix multiplication followed by the typical bias offset and an activation function, so the functional form is identical to that of a ConvLayer. In most typical CNN models it is often used as the very last layer to modify the output to have the desired size (e.g., of N classes). In some NN architectures the fully connected layers make up most of the trainable parameters, such as in VGG16 [Simonyan and Zisserman, 2014] where the first two (out of three) fully connected layers have about 100 million of the total 138 million parameters.

2.2.4 Limitations

Traditional CNNs have multiple drawbacks when we compare them to the psychology of human shape perception. First we know from neuroscience ([Rosch and Lloyd, 1978]) that it is highly likely that human brains process hierarchies of objects (although it is not yet exactly clear how). CNNs do not take spatial hierarchies into account and therefore have a very hard time solving the *Picasso problem*, which is illustrated in Figure 2.6. An image that has all the right parts of an object but that are not in the correct spatial relationship will be classified (incorrectly) as such with high confidence.

Additionally object positions and orientations are handled by using data augmentation during training. Data augmentation techniques are transforming the input images to increase the total amount and the diversity of objects in the training data. Examples of such can be rotating the image by some degrees, shifting or mirroring the image, adjusting the saturation or brightness, or simply feeding random crops of the image to the network. This has been shown to sometimes improve the networks performance on unseen data ([Perez and Wang, 2017]). However, even with the help of data augmentation, learning all changes of objects still requires a significant amount of training data, that for some tasks is not feasible to collect.

If we look at the example in Figure 2.7 our brain will recognize the letter 'R' without much effort in about 150ms or less ([Thorpe et al., 1996]). A CNN trained on recognizing letters will do the forward pass in a fraction of the time, depending on the hardware used. However, looking at the rotated R on the left and being faced with the question whether this is a mirror image of an 'R' or not is a much more challenging task. Humans might solve this by identifying coordinates of the top as well as the front of the letter, followed by a focused mental operation that rotates the whole object clockwise while

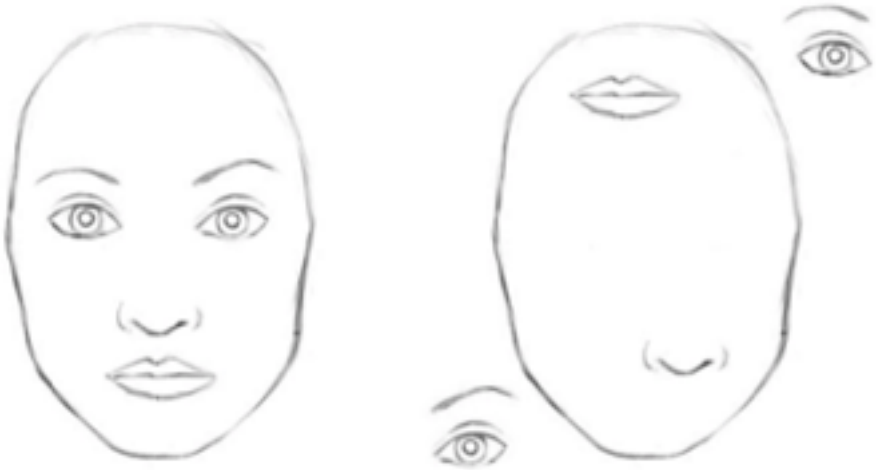

Figure 2.6: Illustrating the picasso problem, where the image on the right has all the right parts for being a face but incorrect spatial relationships. CNNs have trouble distinguishing between the two. Source: [Garg, 2017].

keeping track of the two fixed points. Finding the front at the opposite side makes us classify this correctly as a mirror image R. This process can not be solved by current CNNs as they do not have the ability to recognize features of the classified object.

Another example is shown in Figure 2.8, where two object components are combined in different arrangements to create different objects. Data augmentation methods might even confuse the network to learn correct interpretations of the different objects.

2.2.5 In Defense of CNNs

CNNs have probably been applied to all possible visual recognition tasks with model architectures getting increasingly complicated and creative. They show promising performance in most of the research tasks and it is hard to argue against the success of CNNs. In most cases the problems are very specific and data is either provided in sufficient amount or it is argued that lack of data is the reason for bad results. [Hauptmann and Adler, 2020] made the argument that CNNs are always acceptable to use by showing that a standard U-Net can be trained on invert XOR encryption, a function that is everywhere discontinuous and not translation equivariant. Their results show that training a CNN is always reasonable to use as a strong baseline model for newly proposed methods, as we will do in this work.

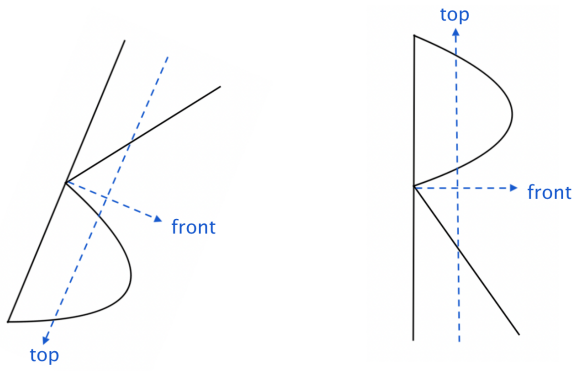

Figure 2.7: Is the letter on the left a mirror-image 'R'? To answer this question humans use coordinates (top, front) and mental rotation.

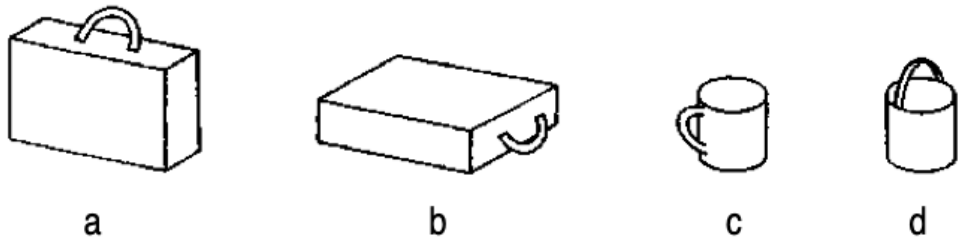

Figure 2.8: Examples of why the arrangement of components matters for object creation. Source: [Biederman, 1987].

2.3 Object Detectors (R-CNNs)

Object detectors usually apply a region proposal method to the input and pass the cropped region through a convolutional neural network. The outputs of the fully connected layers are then processed along two routes, the classification route for finding the label and the localization route for finding the box coordinates. For training the softmax loss for classification and the regression loss for localization are combined to a multitask loss. [Girshick et al., 2014] reached impressive results at that time with their end-to-end trainable object detector *R-CNN*, that extracts around 2000 bottom-up region proposals. However, this architecture was quite slow as it needed to calculate around 2000 forward passes for each individual image. [Girshick, 2015] proposed an improvement by passing the image through a CNN backbone network before doing the region cropping. The new architecture *Fast R-CNN* was significantly faster than the standard R-CNN. In theory, any CNN used in classification can be used as a backbone network. In a third iteration [Ren et al., 2015] the region proposal method was replaced by a Region Pro-

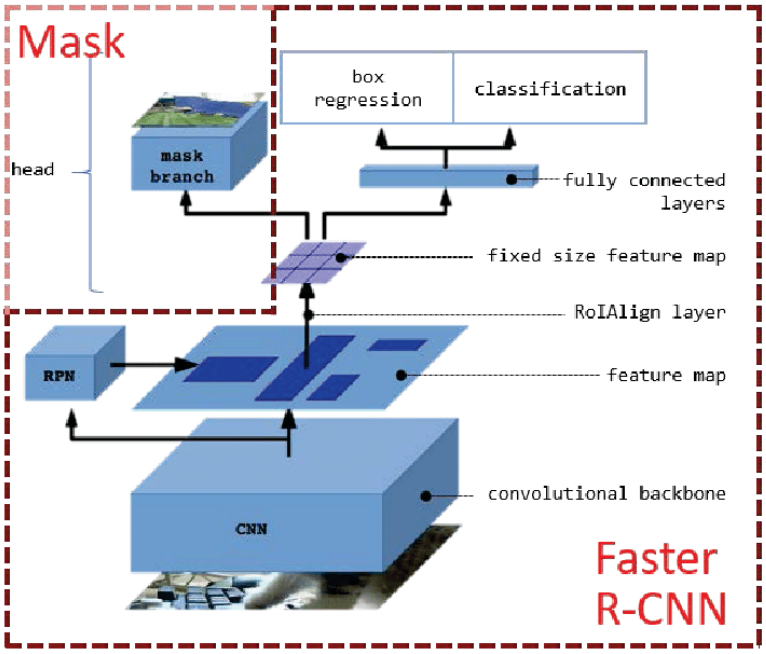

Figure 2.9: Structural design of Mask R-CNN architecture. Source: [Bienias et al., 2019]

posal Network (RPN), another CNN, that was now trained with two additional losses: RPN classify object / not object, RPN regress box coordinates. This network was even faster than the previous one and is fittingly called *Faster R-CNN*. An overview of the network is shown in the tagged part of Figure 2.9.

In [He et al., 2017] both object detection and segmentation were combined into one task and network, where objects are detected via Faster R-CNN and then each pixel inside the detection is labeled as class or background, to have more precise localizations. This instance segmentation architecture is called Mask R-CNN and we will make use of this exact design to localize surgical instruments in cataract surgery videos.

There are several other object detection architectures that are even faster, such as Single Shot Detectors (SSD) and You Only Look Once (YOLO) but they are less accurate than Faster R-CNN. An extensive comparison between speed and accuracy of all the mentioned architectures is made in [Huang et al., 2017b]. A more complete summary of object detection and segmentation is given in [Zou et al., 2019], including most to all neural network architectures mentioned in this Section.

CHAPTER 3

Capsule Networks (CapsNets)

3.1 General Idea

The idea of capsules was first introduced by [Hinton et al., 2011]. The neural networks should have vectorized outputs instead of scalar outputs to capture present features like position, orientation, scale etc. of classified objects. This could solve some of the limitations we have seen in Section 2.2.4. The goal was to have an ANN do *inverse graphics*, or *inverse rendering*. To understand this idea we can have a look at a simple example of rendering in computer graphics shown in Figure 3.1. First we define objects and assign instantiation parameters, that represent our features (e.g., coordinates, height, rotation etc.). Then we render the image with the given objects and parameters to get the desired graphics on our screen. In inverse rendering we reverse this process, meaning we are given an image and we want to extract the objects and more importantly the parameters that represent the features of it.

A capsule can be interpreted as a group of neurons that tries to output parameters for one feature (e.g., rectangle, triangle in Figure 3.1) and, just as in a traditional Neural Network (NN), an activation value in form of a scalar weight for that feature. Each of the parameter values can represent information on the feature, such as rotation angle, line thickness or location (pose) coordinates. The parameter values will be a 4x4 matrix and be referred by us as a *pose matrix*. The entire CapsNet is made up of layers of multiple such capsules.

3.2 Dynamic Routing by Agreement

The great effectiveness of CapsNets comes from the way that information is routed between the layers - also called *dynamic routing*. As each capsule represents some feature of an object we can use routing to learn part-whole relationships when classifying objects. Understanding the simplified example in Figure 3.2 will give an overall intuition. In

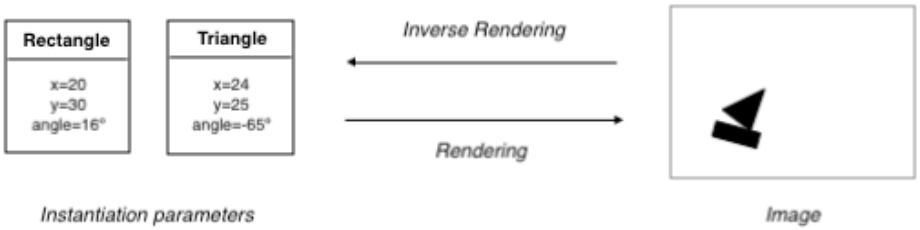

Figure 3.1: (Inverse-)Rendering in computer graphics

this case we have one hidden layer l that searches for features of the two output layer classes. Capsules 1, 2, 3 and 4 (from top to bottom) are features of the class THREE while capsules 1, 5 and 6 are features of the class TWO. Be aware that there are still connections between all capsules of layer l and all capsules of layer $l+1$. This will be clear once we go into details of the routing algorithm. For now, since capsule 1 is going to be active in cases of both classes as input, our goal is to find the probabilities of feature 1 being present *because of* class THREE/TWO of layer $l+1$. In other words we are trying to find an agreement between all the capsules in layer l. We find these probabilities by using the spatial relationships of the active features, that are captured by the previously introduced pose matrices. If the layer evaluates that the spatial relationships between feature 1 and the other features that are responsible for detecting class THREE are likely to correlate with each other, it will come to the conclusion that there is a high probability that the feature is active due to class THREE being present, and not class TWO - or vice versa. Each class gets assigned a probability that we will call *routing coefficient*. A very important concept is that these routing coefficients are not learned during training. They are calculated during each forward pass, hence the term *dynamic routing*. At this point we need to address the notion of how to actually do the calculations for the routing procedure. We will explain this detailed in the following section. Note: In the first publication ([Sabour et al., 2017]) where dynamic routing was introduced calculations were done differently by using pose vectors. More details on this can be found in Section 3.4. In this work we use the routing proposed by [Hinton et al., 2018], which is described in the following section.

3.3 Expectation Maximization (EM-)Routing

The calculations of the routing are done using a clustering algorithm called Expectation Maximization (EM) with a Gaussian mixture model, where we try to cluster data points into multiple Gaussian distributions each described by a mean μ and a standard deviation

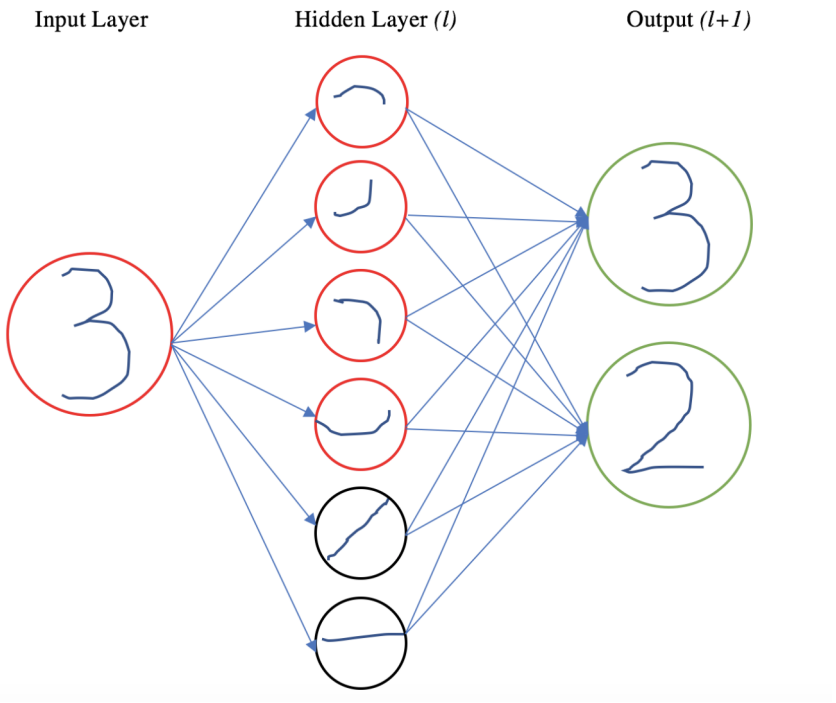

Figure 3.2: Illustration of dynamic routing-by-agreement, where each node is representing a capsule. Active features are shown in red and inactive features are shown in black. Source: [Garg, 2017].

σ. The full algorithm for a CapsNet is shown in Figure 3.3. Intuitively we think of the pose matrices as the data points and each capsule in the following layer as a Gaussian distribution of our mixture model. Each capsule i of the lower level Ω_L consists of a 4x4 pose matrix M_i and an activation value a_i. The capsule i casts a vote for each of the capsules j in the following layer Ω_{L+1}. This vote V_{ij} is calculated by the dot product of the pose matrix and a 4x4 trainable *transformation matrix* W_{ij}.

$$V_{ij} = M_i W_{ij}$$

Each of the capsule-to-capsule connections includes such a transformation matrix. Together with two learned biases per capsule they are the only stored parameters of the network. This results in small network sizes as we will see in chapter 6.2.1. The votes V_{ij} ultimately represent the data points and combine with the activation values a_i as our actual routing algorithm inputs.

EM is the procedure that alternates between two steps for a given number of iterations to calculate the pose matrices and activation values of all the capsules in next layer

procedure EM ROUTING($\boldsymbol{a}$, V)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}$, R, V, j)
    $\forall i \in \Omega_L$: E-STEP(μ, σ, $\boldsymbol{a}$, V, i)
  return $\boldsymbol{a}$, M

procedure M-STEP($\boldsymbol{a}$, R, V, j) ▷ for one higher-level capsule, j
  $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
  $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
  $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
  $cost^h \leftarrow \left(\beta_u + log(\sigma_j^h)\right) \sum_i R_{ij}$
  $a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$

procedure E-STEP(μ, σ, $\boldsymbol{a}$, V, i) ▷ for one lower-level capsule, i
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{(V_{ij}^h - \boldsymbol{\mu}_j^h)^2}{2(\sigma_j^h)^2}\right)$
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j p_j}{\sum_{k \in \Omega_{L+1}} \boldsymbol{a}_k p_k}$

Figure 3.3: Full pseudo-code example of the EM-routing algorithm. Source: [Hinton et al., 2018]

Ω_{L+1}. The E-STEP (Expectation) calculates for each data point (vote) the probability of it belonging to each of the Gaussians. These probabilities will serve as weights R_{ij} and the M-STEP (Maximization) updates the mean and variance for each Gaussian using the weighted data points. This process is done for multiple iterations until the probabilities of the mixture model fitting the data is maximized. In the very first iteration the weights are equally distributed. The probabilities are calculated using the density function of the Gaussian distribution:

$$P(x|G_n) = \frac{1}{\sigma_n\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_n}{\sigma_n})^2}$$

Let V_{ij}^h be the h^{th} dimension of the vote, the probability of the vote belonging to the Gaussian of capsule j is calculated by:

$$P_{ij}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} exp(-\frac{(V_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2})$$

The cost $cost_j^h$ for a single dimension h is calculated using the negative log probability:

$$cost_j^h = \sum_i -R_{ij} ln(P_{ij}^h)$$

And to finally determine the activation of the capsule j we use the following equation, where β_a, β_u and λ are hyperparameters:

$$a_j = sigmoid(\lambda(\beta_a - \beta_u \sum_i R_{ij} - \sum_h cost_j^h))$$

Finally the pose matrices of the capsules in layer Ω_{L+1} are assigned with the means of the Gaussians, transformed into 4x4, and with the activation value we complete our calculations for the next capsule layer.

3.4 Related Work

Due to the success of AlexNet ([Krizhevsky et al., 2012]) - a CNN architecture using pooling (see Section 2.2.2) - the idea of the original capsule network by [Hinton et al., 2011] was not further pursued by most researchers at that time and it took six more years before the first efficient implementation of a CapsNet was released in 2017. [Sabour et al., 2017] proposed a *dynamic routing-by-agreement* algorithm, where lower level capsules route their outputs and predictions to the higher level capsules based on similarity of the pose vectors. Each value of the multi-dimensional output represents a feature of the class. To get the overall likelihood of the class being present (output layer) they use the total length of the output vector. Small sized vectors represent a low probability and big size a high probability. Training was performed on the famous MNIST dataset and reached state-of-the-art results in classification of hand-written digits. By using a decoder network the authors were able to reconstruct the input images, showing that the output feature vectors do in fact represent features of the digits in the training data. We will confirm this in Section 5.3 on our own implementation.

A year later a follow-up paper ([Hinton et al., 2018]) was published where the vector outputs of the previous approach where transformed to matrices. Matrix capsules are designed to be more computationally efficient than the previous architecture. The major difference lies in the way that routing is calculated. The dynamic routing algorithm was replaced by the expectation maximization algorithm (Section 3.3). This reduced the size of the transformation matrices between the capsule layers from n^2 for vectors to n for matrices. Matrix capsules with EM routing have outperformed their dynamic routing counterpart on MNIST and were able to tackle slightly more complex datasets (smallNORB, Cifar10). Although this sounds very promising, the new approach is even more complex and most related work so far has used dynamic routing as basis of their work.

Complexity of datasets, such as high dimensional images, are still an issue for CapsNets. [Rajasegaran et al., 2019] with their network *DeepCaps* were able to create a CapsNet with higher depth (deeper, meaning more layers) without adding computational complexity. They achieved this by extending the network with skip-connections and by applying their 3D-convolution-based dynamic routing algorithm to improve the learning process. The results beat previous CapsNets in terms of accuracy on Cifar10, SVHN and FashionMNIST, but are still slightly below the CNN's ResNet ([He et al., 2016])

and DenseNet ([Huang et al., 2017a]).

CapsNets have high memory and multiply-and-accumulate operations and are therefore difficult to train in feasible time. [Marchisio et al., 2020] are developing a quantization framework for CapsNets and were able to reduce the memory footprint 6-fold by only losing 0.15% accuracy for a *DeepCaps* model trained on Cifar10.

In the field of medical imaging the datasets are often small (in terms of number of examples) and highly unbalanced. [Jiménez-Sánchez et al., 2018] show that CapsNets are better than regular CNNs when trained on small datasets and that they rely less on data augmentation. They use data from the MNIST family as well as two medical datasets for mitosis detection (TUPAC16) and diabetic retinopathy detection (DIARETDB1) and limit the amount of data during training. However, just as in most of the related work so far the baseline CNNs are very weak architectures.

[Afshar et al., 2019] have applied CapsNets to brain tumor classification where raw MRI brain images and the tumor coarse boundaries are fed to the network to identify the type of tumor. They took advantage of the spatial information given by the data and their architecture outperformed the baseline CNN. A similar approach was used in [Afshar et al., 2020], where the authors have proposed their modified CapsNet *COVID-CAPS* on computed tomography (CT) scans and X-ray images to classify whether patients are infected with the novel coronavirus disease (COVID-19). By pre–training the network with a dataset of similar nature they achieved an overall accuracy of 98.3% and specificity of 98.6%.

[de Jesus et al., 2018] have applied CapsNets in biology by doing protein structure classification. Their network outperformed traditional CNN's. It was trained on 2D and 3D structural encodings of the RAS protein family and can successfully classify HRAS and KRAS structures.

Capsnets have also been successfully applied in hyperspectral image (HSI) classification by [Deng et al., 2018]. The authors compared their CapsNet with a CNN and two baseline classifiers (RF and SVM classifier) and report better classification results for the complex data, even with limited training samples.

Although the intuition was derived from the human visual system CapsNets seem to perform especially well in the field of NLP. For example, [Rathnayaka et al., 2018] have implemented a network based on a Gated Recurrent Unit (GRU) and a CapsNet to determine the emotion of posts written on the social-media platform Twitter, where emotional keywords were removed. Their GRU+CapsNet architecture performed slightly better than the GRU+CNN counterpart. Using RNNs as the feature extractor for text has also been effective in [Srivastava and Khurana, 2019], where the goal was to classify aggressive and toxic comments in two datasets: TRAC and Kaggle's Toxic Comment Classification Challenge. In this work the results were compared to strong and recent

baseline algorithms, which makes this result especially interesting.

In the field of audio classification [Jain, 2019] has reported notable results on multiple datasets by using a CapsNet with Bidirectional LSTM layers.

[LaLonde and Bagci, 2018] have used the CapsNet idea in object segmentation where the goal is to label each pixel in an image by the appropriate class. They extend the convolutional capsules with locally-connected routing and use their own concept of deconvolutional capsules. Their architecture *SegCaps* has similarity to U-NETs, which are often used in object segmentation, and is able to handle images of up to *512x512* size. However, reproducability on other datasets have been an issue for *SegCaps* so far.

[Jaiswal et al., 2018] showed that a CapsNet can be used as a discriminator within the Generative Adversarial Networks (GAN). A GAN consists of a generator and a discriminator network, where the goal of the generator is to create artificial images and 'fool' the discriminator part. In other words increasing the error. The discriminator tries to distinguish between the real and fake images with increasing difficulty. The *capsuleGAN* was able to keep up and even outperform the *convolutional-GAN* on MNIST and CIFAR10 datasets.

The closest work to our approach is found in [Bhamidi and El-Sharkawy, 2019], where the initial convolutional layer of the CapsNet is replaced with multiple residual layers. The main difference is that their CapsNet is based on dynamic routing and the training and evaluation is done on CIFAR-10 dataset. They do not reach the performance of the original CapsNet with their architecture but they are able to reduce the parameters by a significant amount (88.32%) - when compared to the ensemble of seven capsule networks in [Sabour et al., 2017]. In a follow-up paper ([Bhamidi and El-Sharkawy, 2020]) the network architecture was extended to three levels, where each one of the three levels is similar to their previous residual capsule network. Performance was slightly better than the single level version but still short of the original CapsNet.

To study if and why CapsNets are more robust than CNNs to affine transformations of inputs, [Gu and Tresp, 2020] have (un)rolled the forward and backward passes of the dynamic routing procedure. Their work revealed that the routing procedure contributes neither to the generalization ability nor to the affine robustness of the CapsNets and that CapsNets can be successfully trained without the routing procedure. This is an interesting result, as many works have focused their efforts on analyzing and improving the routing mechanism ([Tsai et al., 2020], [Fuchs and Pernkopf, 2020], [Venkatraman et al., 2020]). They proposed their own architecture *Aff-CapsNet*, which can be trained with and without dynamic routing, and showed that it is more robust to affine transformations than the original CapsNet.

CHAPTER 4

Performance Metrics & Datasets

As we have seen so far we can train deep neural networks on a careful collection of data, making it remember the predictions of training data and updating its weights to learn representations of objects. In the best case making predictions for additional, previously unseen data will result in correct predictions. Measuring the performance of the trained models is done similarly as in traditional ML methods. For this we have to understand multiple different mechanisms.

First the full dataset should be split into three parts: training-, validation- and test set. This is an important step, as we want to do our final evaluation of the model on a test set rich of data that the model has never seen before and that is representative of as many different features as possible, which the model will have to detect if used in real world practical applications. The selection process is important, as we try to avoid any possible biases that the data could represent. How much of the data we use for testing heavily depends on the task and the data itself, but as a rule of thumb 20 to 30 percent has worked well in a great deal of research. The remaining 70 to 80 percent are used for the training process and therefore further split in training and validation data. The model will see both datasets during training but only learn to fit to the data (updating its weights) of the training set. The validation data is simply used to monitor the progress of the training. If checkpoints of the trained models are stored we can choose the checkpoints performing best on the validation data and do a final evaluation on our test set. Skipping the test set and simply using the validation set as our final results might be inefficient, as we are trying to train a model that can generalize well on future data and simply using the checkpoint where the validation score was highest might not represent the best model.

As we track the performance of our model during the training (e.g., by drawing a plot over time steps), different scenarios can occur. Figure 4.1 shows us a plot of the loss during training. The lower the loss the more accurate the predictions. As the training progresses successfully the model gets increasingly better at making the

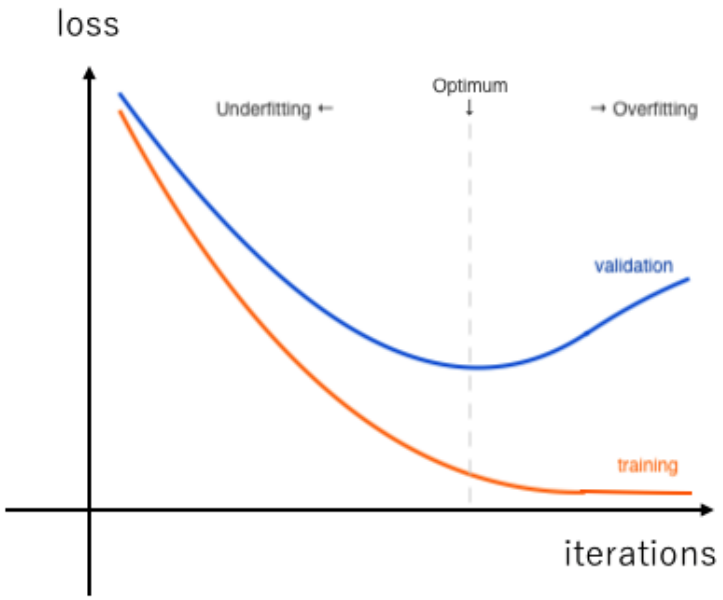

Figure 4.1: Monitoring the model performance during training on both training (orange) and validation (blue) set. For the best overall parameters we choose the optimum point where both curves are at a low point. After this point overfitting occurs, where the model is memorizing the training data too well resulting in bad generalization.

correct predictions for our training data, leading to a very steep curve that reaches a very low point at a high number of training iterations. For the validation set this is true during the beginning stages of training where the model gets better at predicting the data in general but it reaches a point where the performance gets worse and the loss starts increasing again. This is due to the network drastically memorizing the training data. We call this scenario overfitting. On the contrast, if we stopped the training too early and did not use enough iterations, we would experience underfitting. This is less of a problem, as we can simply give the model more training time to get better. The optimum point in this specific case is found where the validation performance is best and the training performance is also good. Sometimes it happens that the gap between the two curves is very high and the optimum not good enough. One solution to this might be that the amount of data is too small for the complexity of the model and we need more examples. It can also be a good idea to test a less complex model in order to reduce the overfitting to the training data.

Once we understand the learning curve of our model and find the optimum amount of iterations, we can include the validation data into our training, removing the moni-

toring but giving the model more data for training, which should result in even better performance overall.

4.1 Classification Metrics

Depending on the object recognition task, different performance metrics are used. These metrics should provide a representative measurement of how accurate the model's predictions are when compared to the ground-truth annotations of the test set. For making human interpretability more convenient these metric results are often converted into percentages, where 100% means a perfect prediction. In single-label image classification we find a very simple metric in calculating the *accuracy*, which is given by the number of correct predictions divided by the total amount of predictions made:

$$Accuracy = \frac{\#Correct\_predictions}{\#Total\_predictions}$$

This measure works well in general but we have to be careful of one thing: class (im)balance. In the best case our data sets have the same amount of examples for each class, making the set fully balanced. This gives any random model an accuracy of $\frac{1}{\#Classes}$ (e.g., if we have 4 classes and enough examples, guessing randomly will give us an accuracy around 25%). This can be very misleading for imbalanced data sets. Considering a distribution of 900 examples for class Pos and only 100 examples for class Neg a model that always predicts class Pos will reach an accuracy of 90%. To adjust to such imbalanced datasets we can look at a contingency table, better known as **confusion matrix**, which is shown in Figure 4.2. For all labeled examples the confusion matrix shows which classes were predicted by the model, making it easy to see if the model has a tendency to always predict one and the same class or if it has problems distinguishing between two classes. The True Positives (TP) are found in the diagonal of the matrix. Focusing on class 2.0 in our example we have 44% of TP in the normalized matrix for all images where 2.0 is the correct answer. We find the True Negatives (TN) in all examples where the true label is not 2.0 and the model did not predict 2.0. The False Positives (FP) are all cases where the model predicted 2.0 but the result was something different (0.00 + 0.01 + 0.21 + 0.06 = 0.28). And finally the False Negatives (FN) are the wrong predictions for images where 2.0 would have been the correct answer (0.36 + 0.18 + 0.01 = 0.55).

Calculating these values leads us to some new metrics called *Precision* and *Recall*, which are defined as:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

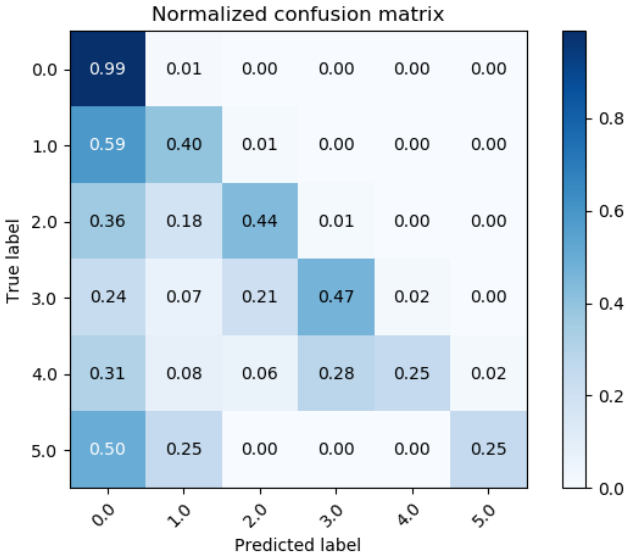

Figure 4.2: Confusion matrix for six classes. True positives are found in the diagonal. In this particular case the model has a tendency to predict class with label 0.

We can further combine these two metrics and take their harmonic mean, better known as *F-measure*. Precision and Recall can be weighted to give more emphasis on one of the two. The general formula for F-measure is the following:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$$

Most of the times in literature we find the evenly weighted version known as F_1 measure where $\beta = 1$:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2TP}{2TP + FP + FN}$$

Although not always the best option, the F_1 measure is a popular choice to evaluate models in multi-label classification, as it also takes partial correct answers into its final result.

4.2 COCO Detection Metrics

In object detection the ground-truth annotations are different, usually in form of bounding box- or segmentation mask regions. Therefore, we also need to calculate the performance metrics differently. COCO (Common Objects in Context) is the name of a large-scale object detection, segmentation, and captioning dataset with over 328k labeled images ([Lin et al., 2014]). It includes around 2.5 million labeled object instances of 91

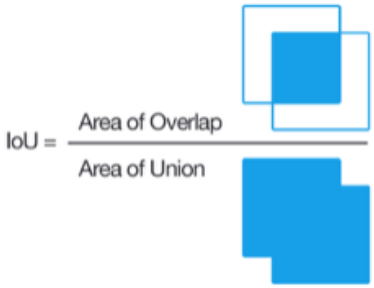

Figure 4.3: Intersection over union of two boxes. Source: [Rosebrock, 2016].

different object types. To evaluate the performance of state-of-the-art approaches on this type of data multiple metrics were used which are based on Intersection over Union (IoU) and can deal with multiple instances. The IoU, also known as Jaccard Index, is defined as the intersection of the predicted bounding box and the actual ground-truth bounding box divided by their union. This is best illustrated in Figure 4.3 where the corresponding areas and their overlapping regions are marked in blue. A prediction is considered to be a True Positive if the IoU exceeds a given threshold, and a False Positive if the IoU falls below this threshold. In COCO evaluation, the IoU threshold ranges from 0.5 to 0.95 with a step size of 0.05 represented as AP@[.5:.05:.95], where AP stands for average precision. Precision and Recall are used with the same formulas as we have seen previously in this Section. Recall is the true positive rate and takes the actual positives (TP + FN) into account, whereas precision uses all positive predictions (TP + FP).

To define the Average Precision (AP) we need to understand the Precision-Recall curve first. For this we will look at an example with a single class first. If we collect all predictions for our class X in all the images and rank them in descending order to the predicted confidence, we can calculate the Precision and Recall in every step starting from the top. This way the Recall values will increase as we go down the list, while the Precision values will change in a zigzag pattern. If we plot the Precision against the Recall values and smooth the pattern, such as in Figure 4.4, we can calculate the Area Under Curve (AUC) which represents our AP for the example class X. We can do this for all classes and average the results to get our final AP.

In COCO we have 12 similar metrics, as shown in Figure 4.5, with the most import being the mAP (mean Average Precision). The mAP is calculated as the average of the AP results for each of the thresholds between 0.5 and 0.95. For convenience in COCO the terms mAP and AP are used interchangeably, although often leading to slight confusion.

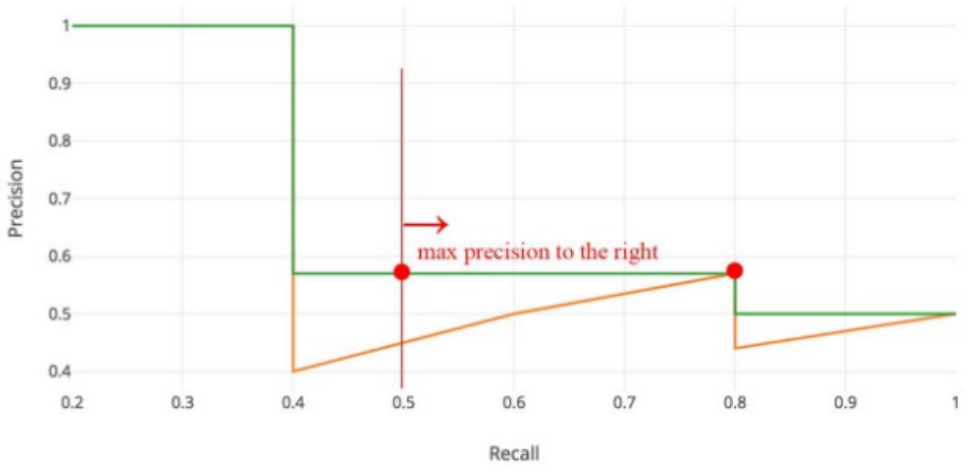

Figure 4.4: Plotting the area under curve (AUC) of precision against recall results in zigzag pattern (orange). The curve can be smoothed by taking the maximum precision to the right (green) to make computation of AUC easier. Source: [Hui, 2018].

```
Average Precision (AP):
  AP                % AP at IoU=.50:.05:.95 (primary challenge metric)
  AP^IoU=.50        % AP at IoU=.50 (PASCAL VOC metric)
  AP^IoU=.75        % AP at IoU=.75 (strict metric)
AP Across Scales:
  AP^small          % AP for small objects: area < 32^2
  AP^medium         % AP for medium objects: 32^2 < area < 96^2
  AP^large          % AP for large objects: area > 96^2
Average Recall (AR):
  AR^max=1          % AR given 1 detection per image
  AR^max=10         % AR given 10 detections per image
  AR^max=100        % AR given 100 detections per image
AR Across Scales:
  AR^small          % AR for small objects: area < 32^2
  AR^medium         % AR for medium objects: 32^2 < area < 96^2
  AR^large          % AR for large objects: area > 96^2
```

Figure 4.5: Collection of COCO metrics used in the evaluation process of models trained on COCO dataset. Source: [coc, 2017].

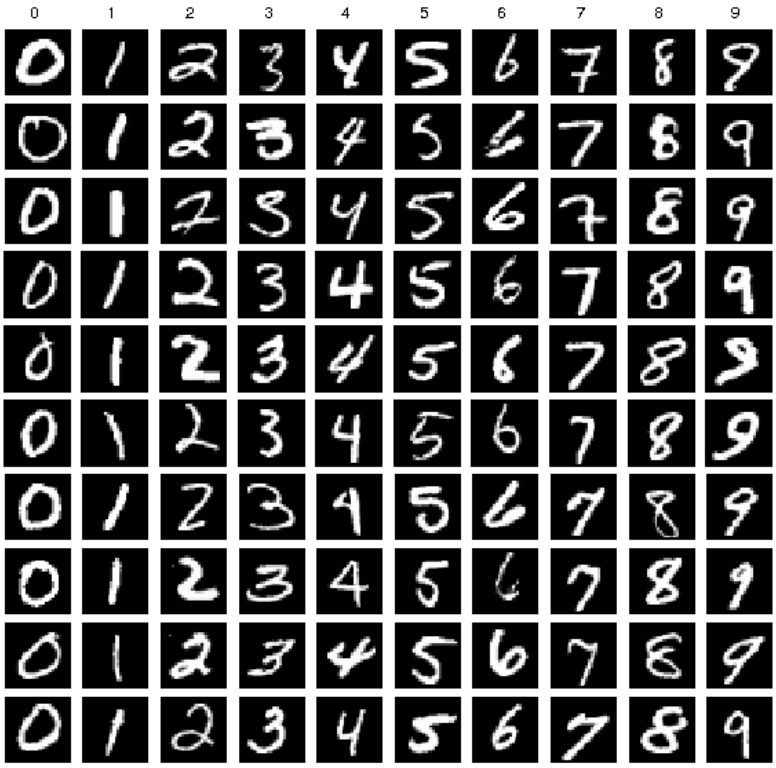

Figure 4.6: Ten examples per class of the MNIST dataset of handwritten digits. Source: [Lim et al., 2016].

4.3 Datasets

4.3.1 MNIST

The MNIST dataset is a collection of handwritten digits stored as greyscale images. Each image shows one of the 10 possible digits (0 to 9) in digital format of size 28x28. There are a total of 70.000 images that are split into two sets: 1) training- and 2) test set. The training set has a total of 60.000 images (6.000 per class/digit) and the test set the remaining 10.000 images (1.000 per class/digit). Figure 4.6 shows a number of examples for each class.

A good property is that the digits in the images are handwritten by many different people. This makes the classification task more difficult than if the digits were all of the same font. Due to its small size the MNIST dataset has been a very good fit for testing new ML algorithms on image classification in the past. In the original paper ([LeCun et al., 1990]) a support-vector machine has been trained to get an error rate of 0.8%. One negative thing to say about MNIST data is that algorithms of recent years

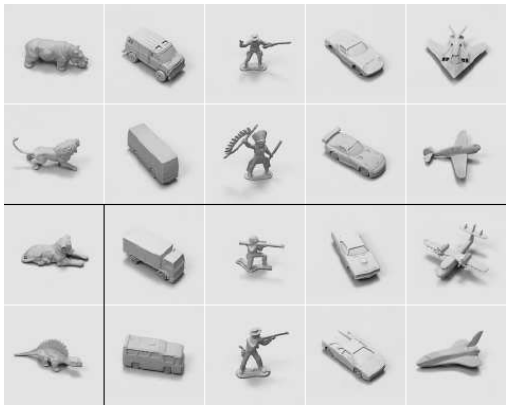

Figure 4.7: Four examples per class of the NORB dataset. Classes from left to right: Four-legged animals, Trucks, Human figures, Cars, Airplanes. Source: [Scherer et al., 2010].

have gotten so good at classifying objects into categories that MNIST is often denoted as a *solved problem* and therefore not accepted by some researchers as a benchmark anymore. Current state-of-the-art CNNs can quickly reach error rates of below 0.2%.

4.3.2 smallNORB

The small NORB database ([LeCun et al., 2004]) was intended for 3D object recognition from shape and can also be used for image classification. It contains images of toys belonging to 5 different categories - Four-legged animals, Trucks, Human figures, Cars, Airplanes - that were taken under many different conditions: 6 lighting conditions, 9 elevations (30 to 70 degrees every 5 degrees), and 18 azimuths (0 to 340 every 20 degrees). The total of 48.600 images are greyscale of size 48x48 and split into training- and test sets of 24.300 each. Examples are shown in Figure 4.7

The various conditions and poses of the toys in the images make this task much more challenging compared to MNIST, although there are only 5 classes.

4.3.3 STL10

The STL10 dataset ([Coates et al., 2011]) is another image recognition dataset similar to the others in this section, but originally designed for developing and testing unsupervised feature learning, deep learning and self-taught learning algorithms. Therefore there are a lot of unlabeled training examples in this dataset. The images are taken from labeled examples of ImageNet data and scaled to a size of 96x96. Unlike the MNIST and smallNORB datasets, STL10 has three-channel color images which adds additional

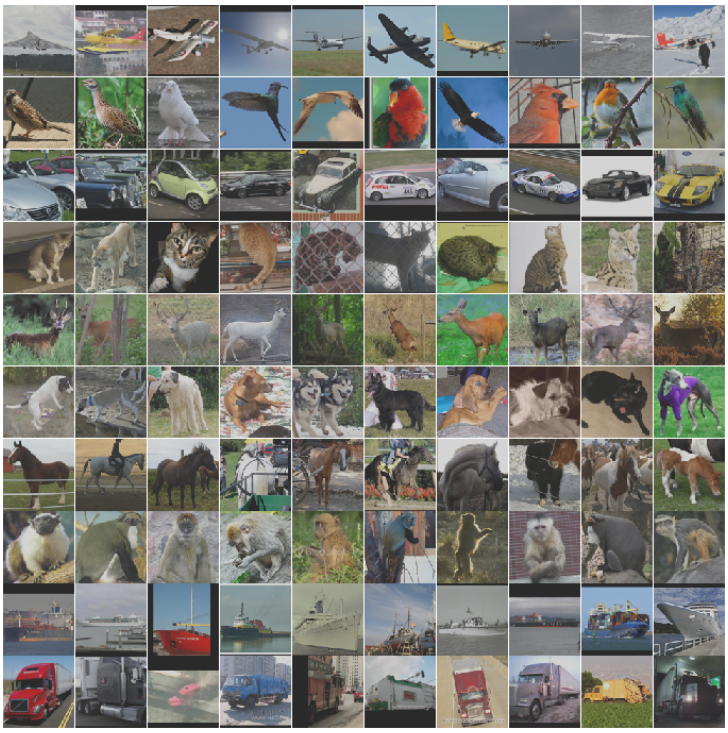

Figure 4.8: Ten examples per class of the STL10 dataset. Classes from top to bottom: Airplane, Bird, Car, Cat, Deer, Dog, Horse, Monkey, Ship and Truck. Source: [Coates et al., 2011].

depth.

In this work we do not use the 100.000 unlabeled examples since we are dealing with supervised learning. We only make use of the training- and test set that consist of 500 and 800 images per class. In total there are 10 classes: Airplane, Bird, Car, Cat, Deer, Dog, Horse, Monkey, Ship, Truck, totalling to 13.000 images. Examples are shown in Figure 4.8

4.3.4 INstance SEGmentation in CATaract Surgery Videos

For training an object detector for instance segmentation based on Mask R-CNN we collected two different datasets, both consisting of frames extracted from cataract surgery videos. In order to avoid confusion, we will refer to them as dataset 1 and dataset 2 from here onward, as we did the same thing in the original work ([Fox et al., 2020]). For dataset 1 we carefully selected frames out of a public collection of cataract surgery videos [Schoeffmann et al., 2018] and manually annotated the segmentation masks of many different instruments that are used during the surgery, using *COCO-annotator*

Dataset	Images	Train	Val	Test	Classes
DS 1	393	237	61	95	9
DS 2	4738	3582	542	614	21

Table 4.1: Numerical comparison of the created instance segmentation datasets.

([Brooks, 2019]). In the selection process we focused on frames with the best visual quality (e.g., no blurriness), as the videos of the public collection are generally of medium quality. We also made sure to have a variety of different poses for each instrument class. The final dataset includes annotations for nine important instruments in the widely-used *COCO-format* ([Lin et al., 2014]). The following classes were annotated: SLIT KNIFE, ANGLED INCISION KNIFE, KATENA FORCEPS, 27 GAUGE CANNULA, CAPSULORHEXIS FORCEPS, CANNULA, PHACO TIP, SPATULA, IRRIGATION/ASPIRATION HANDPIECE, CARTRIDGE and EYE RETRACTORS. Example annotations are shown in Figure 4.10. In total we have annotated 393 different images. For training (validation) we took 237 (61) images and for our final testset we used 95 images. This amounts to a minimum of 50 annotations per class. We make sure that the classes are balanced and that we do not use images of the same video across multiple sets.

For dataset 2 we converted the information provided with the public *CaDIS-dataset* of [Flouty et al., 2019] from a *semantic segmentation* task to an *instance segmentation* task by taking the existing mask-images and extracting bounding-boxes as well as instance masks of the desired classes (Figure 4.9). As a result we have a dataset 9x larger than dataset 1 and of the same format. In total there are 4738 annotated images and we keep the same split for training-/validation-/testset images as found in the original format: 3582 training, 542 validation and 614 testing. The images in dataset 2 have much better visual quality because of better lighting conditions and less blurry frames compared to dataset 1. Numerical comparison between the two datasets can be found in Table 4.1. It makes sense to compare these two datasets as we have multiple differences, such as visual quality, image resolution, total number of images as well as classes in all sets and different types of instruments (videos are recorded in French/Austrian hospitals). For reproducibility reasons we already released both datasets[1] with the original paper [Fox et al., 2020].

[1] `http://ftp.itec.aau.at/datasets/ovid/InSegCat/`

(a) Original image (b) Mask image (c) Annotated image

Figure 4.9: CaDIS dataset includes (a) original images and (b) mask images. Extracted information is visualized in (c).

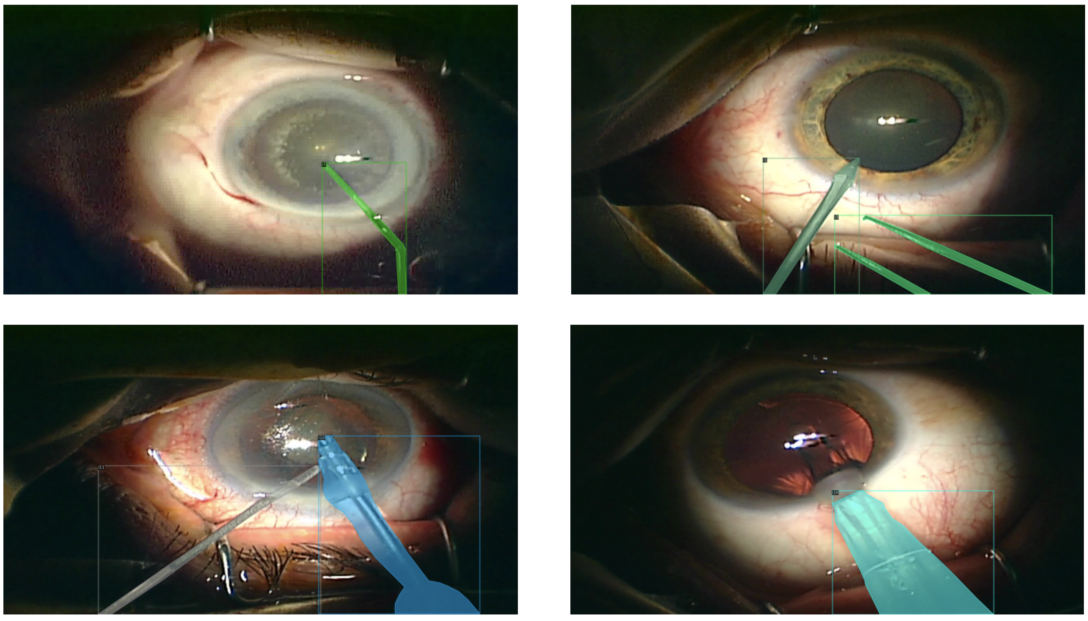

Figure 4.10: Ground-truth instance segmentation annotations of dataset 1.

CHAPTER 5 Method

5.1 Baseline CNNs

5.1.1 Weak Baseline

To compare CapsNets to regular CNNs we first define a basic CNN architecture. Since we are striving for a fair comparison between the two different approaches we will define a baseline model that has a similar amount of parameters and in terms of complexity lies in between the first successful CNN called *LeNet-5* ([LeCun et al., 1990]) and the famous *AlexNet* ([Krizhevsky et al., 2012]). This model will have three ConvLayers that are each followed by ReLU activation function and MaxPooling with stride of 2. Additionally we have two fully connected layers that ultimately transform our predictions to a size equivalent to the number of classes in the ground truth (e.g., for MNIST number of classes = 10). A softmax at the end will make sure to scale our outputs in range of [0,1] to determine confidences in our predictions. For training we are using the *negative log likelihood* as loss function and set the learning rate to 3e-3. In case of multi-label multi-class classification, we can simply change the activation function of the last layer from softmax to sigmoid, which also scales the output values to a range of [0,1] but allows multiple positive predictions by allowing a sum greater than one for the outputs. For training multi-label we also change the loss function from *negative log likelihood* to *binary cross entropy*. The full network architecture, that has a total of 1.2M trainable parameters, can be seen in Figure 5.1.

5.1.2 Strong Baseline (ResNet50)

To give our comparisons additional depth we will also use a more recent CNN architecture as our strong baseline. We choose ResNet ([He et al., 2016]), because it is a very well-known and probably the most used CNN architecture in research during the last few years. The network introduces so called *residual blocks* (Figure 5.2) that make use of skip connections to improve the gradient flow in very deep neural networks. For small

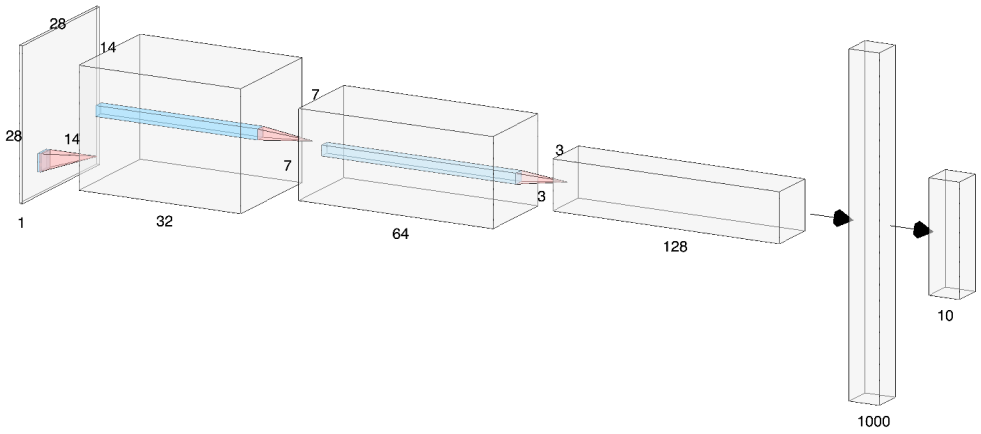

Figure 5.1: Architecture of a baseline CNN with three ConvLayers and two fully connected layers. Input images are grey-scale with a resolution of 28x28.

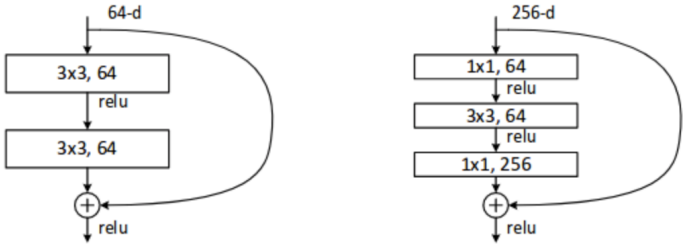

Figure 5.2: Left: Residual block of small ResNets (e.g. 18, 34 layer) with two ReLU-ConvLayers and a skip connection. Right: Residual block of large ResNets (50 layer or above) with three ReLU-ConvLayers and a skip connection. Source: [He et al., 2016].

ResNets (18, 34 layers) a residual block consists of two ReLU-ConvLayers and for larger ResNets (50 layer and above) three ReLU-ConvLayers are used. For our experiments we choose the 50 layer version of ResNet as it tends to be the best suitable option for medium sized image datasets.

For training we use cross entropy as our loss function and stochastic gradient descent (SGD) with learning rate of 0,001 and momentum of 0,9. The weights of the modified last layer are initialized with *kaiming initialization*.

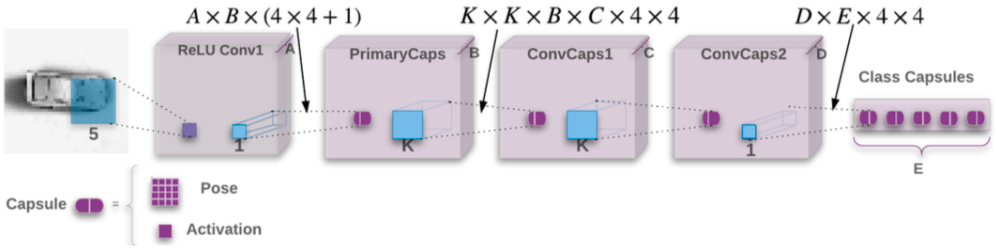

Figure 5.3: Matrix capsules network with one ReLU ConvLayer followed by a primary capsule ConvLayer and two more capsule ConvLayers [Hinton et al., 2018].

5.2 Reproducing Matrix Capsules with EM Routing

Our next objective is to implement the original architecture found in [Hinton et al., 2018] and shown in Figure 5.3. The first layer is a regular ConvLayer with A = 32 filters of 5x5 and a stride of 2, followed by ReLU, but without Pooling. The PrimaryCaps is our first capsule layer and it learns to transform the output of the lower-level layer into B = 32 capsules, that is 4x4 pose matrices and activation values. After the PrimaryCaps there are two layers of convolutional capsules (C = D = 32) that use a receptive field of 3x3 (K = 3) and strides of 2 and one, respectively. The final capsule layer has a total of E capsules - one per output class.

The network is trained end-to-end using *Spread Loss*, also known as *Squared Hinge Loss* often found in the context of *Support Vector Machines (SVM)*. If the difference of the activations between the wrong class a_i and target class a_t is smaller than a given margin m we penalize it with the squared distance to the margin. Initially the margin is set to 0.2 but it is linearly increasing to 0.9 during training.

$$L_i = max(0, m - (a_t - a_i))^2, L = \sum_{i \neq t} L_i$$

We use Adam optimizer with the same hyperparameter values as [Hinton et al., 2018]: a learning rate of $3 \cdot 10^{-3}$ with exponential decay: $decaysteps = 20000$, $decayrate = 0.96$, and a weight decay of $2 \cdot 10^{-7}$.

At the point of writing there are already many open source implementations of CapsNets, most of them implementing the dynamic routing algorithm of [Sabour et al., 2017]. We drew inspiration from many of them but we only reused some Python code from one[1] of them.

To validate the correctness of our implementation we train the network on two well-known datasets (MNIST, smallNORB) and compare the evaluation results to the ones

[1] A PyTorch Implementation of Matrix Capsules with EM Routing - `https://github.com/yl-1993/Matrix-Capsules-EM-PyTorch` (01.04.2020)

Architecture	params	iters	Error (MNIST)	Error (smallNORB)
[Hinton et al., 2018]	310k	3	0,44	1,8
Capsnet	319k	2	1,03	8,65
Capsnet + Decoder	1,73M	2	0,97	9,8

Table 5.1: Comparing results of our implementation to the original paper. In our case 2 iterations performed best. Difference in parameters might just be a case of rounding down.

in the original paper (Table 5.1). For MNIST we come very close to the reported results with 1,03 percent test error compared to the 0.44 percent and for smallNORB our results are off by 6,85 percent. However, almost all implementations we have found reported to be short by between 2 to 10 percent, with the best reproduced implementation ([Gritzman, 2019] - TensorFlow) reporting 2.4 percent difference at the point of writing.

Due to the limited amount of computational time we were able to use (hardware-related) we could not test all parameter configurations exhaustively and the reported results might not be the best possible with our implementation. Therefore, we will accept our results as sufficient to continue our investigation with our own implementation.

5.3 Reconstruction Network

[Hinton et al., 2018] mentioned the use of a Reconstruction (Decoder) network but reported no results for their modified architecture - instead they added a weight decay to the optimizer to compensate for the missing reconstruction loss as regularization. At this point we add a reconstruction network to our implementation. It will use the models final pose matrices to reconstruct the original image. We separate two parts of our reconstruction network: (1) *MaskCaps* - where we zero-mask all pose matrices of the classes that are not detected, and (2) *Decoder* - a combination of linear layers that will learn to reconstruct the image.

To train the model end-to-end we have to adjust our loss function to also include the loss for reconstruction. The reconstruction loss is calculated by taking the mean squared error between the pixel values of the original images X_j and the reconstructed images $\hat{X}_j$. The result is weighted by a hyperparameter α.

$$Loss = \sum_{i \neq t} L_i + \alpha \frac{\sum_j^n (X_j - \hat{X}_j)^2}{n}$$

In Figure 5.4 we can see the effects of three different values of α on MNIST training and reconstruction. As expected giving higher weight to the reconstruction loss will

Figure 5.4: Reconstructed images of models trained with different α values. First row shows $\alpha = 0.0005$, second row $\alpha = 0.05$ and third row $\alpha = 2.0$

improve the visuals of the reconstructions. An interesting property of the reconstruction loss is that it can be used as regularizer instead of weight decay. During our tests we have seen slightly improved results when adding a reconstruction network, even if it does not succeed at reconstructing the images.

5.4 Residual-CapsNet

Since CapsNets are having trouble dealing with higher-dimensional input (starting from 50x50 and above) as well as color images as input, we are trying to find a method that can solve this issue. Our proposed solution is making use of the residual blocks we have seen in Section 5.1.2. We change the layer(s) that come(s) before the PrimaryCaps layer - we call it the head of the CapsNet. Our architecture starts with a regular 3x3 ReLU-ConvLayer without bias, followed by three residual blocks of depth 2. In theory, the possibilities of modifying the head of the CapsNet are vast, but our initial goal is to show that modifying the head can lead to promising results. In our tests so far three residual blocks have performed best. The amount of capsules per layer is set to 32. The full architecture and the output dimensions for each layer in case of smallNORB training is listed in Table 5.2. For STL10 training the output dimensions are different but the architecture's layers stay the same.

For training we keep the hyperparameter settings of Section 5.2 and use the same spread loss function and Adam optimizer with a learning rate of $3 \cdot 10^{-3}$.

5.5 Mask R-CNN

We implement *Mask R-CNN* using the *Tensorflow Object Detection API*[2] and train it on both datasets of Section 4.3.4. The input images are resized to 960 pixels for the maximum dimension while keeping the aspect ratio, before going through a data

[2]https://github.com/tensorflow/models/tree/master/research/object_detection

layer name	output size	layer
conv1	48x48	3x3, 45, stride 1, no bias
conv2_x	48x48	$\left\{ \begin{matrix} 3x3, 45 \\ 3x3, 45 \end{matrix} \right\}$ x2, stride 1
conv3_x	24x24	$\left\{ \begin{matrix} 3x3, 45 \\ 3x3, 45 \end{matrix} \right\}$ x2, stride 2
conv4_x	14x14	$\left\{ \begin{matrix} 3x3, 32 \\ 3x3, 32 \end{matrix} \right\}$ x2, stride 2
primary_caps	14x14	$\left(\begin{matrix} pose : 4x4 \\ activation : 1 \end{matrix} \right)$ x32 capsules
conv_caps1	6x6	$\left(\begin{matrix} pose : 4x4 \\ activation : 1 \end{matrix} \right)$ x32 capsules
conv_caps2	4x4	$\left(\begin{matrix} pose : 4x4 \\ activation : 1 \end{matrix} \right)$ x32 capsules
class_caps	1x1	$\left(\begin{matrix} pose : 4x4 \\ activation : 1 \end{matrix} \right)$ x10 classes

Table 5.2: Residual-CapsNet layers when training the network on smallNORB or similar dataset. Additional residual blocks can be introduced if necessary. We achieved our best results with this setting.

augmentation step to increase the diversity of the images, as later described in detail in Section 6.3.2. Mask R-CNN consists of three stages. The first stage is a feature extractor, also called backbone network, whose trainable weights are initialized from a CNN model pretrained on ILSVRC. This leads to better results overall and makes the network less prone to overfitting the data. The resulting feature map is then fed to the region proposal network, that tries to predict the regions where objects can be found. This step is run only once per image, which makes it much faster than other region proposal methods. We have set the maximum amount of proposals to 300. The second stage uses RoI pooling to predict object classes and bounding box offsets. With the predictions we can calculate classification loss and bounding-box regression (localization) loss. We do not use dropout for the box predictor. In the third and final stage a small mask network is added that operates on each region of interest and predicts a binary mask, that we set to size 33x33.

The multi-task loss function of Mask R-CNN combines the loss of classification, localization and mask segmentation. We weighted the losses by 2.0, 1.0 and 4.0 respectively. As optimizer we use stochastic gradient descent (SGD). We have compared

different learning rates of $\eta \in \{0.06, 0.01, \ldots 0.0001\}$. The optimal learning rate for our experiments is found at steady $\eta = 0.001$ with $\gamma = 0.9$ momentum.

In the second stage post processing we have set the threshold for predictions to be at minimum of 0.6 and we only allow one detection per class and a total amount of two detections overall, as our images never show multiple instances of the same object and a maximum of two surgical instruments at the same time.

CHAPTER 6 Evaluation

6.1 Hardware

For all subsequent evaluations we use the same GPU hardware - a single Nvidia Titan RTX. The GPU was designed for AI-research and is equipped with 24GB of GDDR6 RAM and 576 Tensor cores[1]. This is supposed to make our evaluations fair and comparable. This GPU should be available to most small to medium sized research groups and represents a solid choice for analysing practicability of the evaluated approaches.

6.2 Capsule Networks

6.2.1 Training-/Inference Time

One of the major bottlenecks we find in applications of CapsNets is the time spent on computing the results, i.e., updating the trainable weights. Our experiments using the hardware of Section 6.1 confirm what we have found in related work - that CapsNets are not practical to train with high dimensional input, although the algorithm seems to converge faster than regular convolutional networks. With convergence we mean that it takes fewer epochs and often less training data to reach the optimum performance. Table 6.1 shows the measurements for training- and inference time on batches of the MNIST dataset. The CapsNets are significantly slower than the baseline-CNNs in both. There is a difference between training time and inference time as the network takes additional time to calculate the results of backpropagation. With inference only the output matters and the weights are not updated. Interestingly the decoder network in the CapsNets consists of fully connected layers that make up most of the parameters, yet it does not add much computational time overall.

The same effect can be seen in Table 6.2 where training was done on batches of the smallNORB dataset. The input dimensions increase by a factor of around three from

[1] https://www.nvidia.com/de-de/data-center/tensor-cores/

Architecture	trainable params	train-time/batch	inference-time/batch
Capsnet	316k	0,106ms	0,031ms
Capsnet + Decoder	1,73M	0,108ms	0,032ms
Weak-baseline	1,25M	0,002ms	$\leq$0,001ms
Strong-baseline	23,52M	0,017ms	0,006ms

Table 6.1: Comparing average training- and inference time on MNIST dataset batches. Input images are of size 28x28x1 and are fed to the network in batches of size 8. For the Capsnet's layers the number of capsules (A, B, C and D) were set to 32 and EM-Routing iterations were set to 2.

28x28x1 (MNIST) to 48x48x1 - so does the time spent on computation for CapsNets but not for the baselines. At this input size we start using our proposed Residual-CapsNet architecture and find that it serves as a good compromise between parameter size and time spent on computation. We can speed up training per batch from 0,391ms down to 0,079ms and testing from 0,119ms to 0,022ms, while keeping the amount of trainable parameters low (507k) compared to the baseline CNNs (4,71M and 23,52M).

When we consider that the average image resolution on ImageNet is 469x387x3 pixels we must make the conclusion that the standard CapsNet architecture is not a realistic choice for real world images. However, many optimizations have been done on CNNs that have reduced computational time and this will also be the case for CapsNets, if researchers decide to pursue the idea further. In our case we strive for a middle ground between ResNets and CapsNets without using Pooling layers and getting away from the basic theoretical principle and already gained some ground.

To increase the data complexity to the next level we train our networks on the STL10 dataset, where the images are of size 96x96x3. Table 6.3 shows the results of this experiment. ResNet50 once again computes the results very quickly but this time the weak baseline network has a strong increase in parameters, although still being fastest overall. Due to the high memory consumption we were forced to decrease the standard CapsNet's training batch size but it can be seen clearly that it loses when looking at pure computational time (and performance, as we will see in the next section).

In terms of CapsNets our proposed architecture is best with 0.404ms train-time per batch of 8 and 0.122ms inference-time per batch of 32 images, although still significantly slower than the baselines.

The overall model size on disk is determined by the amount of trainable parameters (weights) in the network, as these are the stored parts of the network. As we can see in Table 6.4 the CapsNets have a very small number of parameters and therefore they do not take up much disk space. This is a useful feature and might be especially relevant

Architecture	trainable params	train-time/batch	inference-time/batch
Capsnet	316k	0,391ms	0,119ms
Capsnet + Decoder	3,25M	0,395ms	0,119ms
Residual-Capsnet	**507k**	**0,079ms**	**0,022ms**
Weak-baseline	4,71M	0,003ms	$\leq$0,001ms
Strong-baseline	23,52M	0,017ms	0,006ms

Table 6.2: Comparing average training- and inference time on smallNORB dataset batches. Input images are of size 48x48x1 and are fed to the network in batches of size 8. For the Capsnet's layers the number of capsules (A, B, C and D) were set to 32 and EM-Routing iterations were set to 2.

Architecture	trainable params	train-time/batch	inference-time/batch
Capsnet	320k	*0,473ms**	0,569ms
Residual-Capsnet	**507k**	**0,404ms**	**0,122ms**
Weak-baseline	18,53M	0,006ms	0,008ms
Strong-baseline	23,52M	0,019ms	0,015ms

Table 6.3: Comparing average training- and inference time on STL10 dataset batches. Input images are of size 96x96x3 and are fed to the network in batches of size 8 for training and of size 32 for testing. For the Capsnet's layers the number of capsules (A, B, C and D) were set to 32 and EM-Routing iterations were set to 2. **due to high memory consumption we had to set batch size to 2 for this architecture*

for applications where disk space is limited. However, disk space of this size can be relatively cheap, making models of below 500MB of size no problem for most hardware.

6.2.2 Performance

Here we finally report on the performances of the various tested networks, making this section the most important and interesting. Obviously, a super fast model is still useless if it can not perform nearly as good as the slower counterpart. Therefore, we strive for a good balance between performance and speed. Since we deal with single-label classification tasks in three of our datasets, where the networks predict the one correct class for each image, we measure the performance by accuracy in percent of the correct predictions divided by the total amount of predictions (#True Positives / #Total Predictions).

Table 6.5 shows the results for all networks trained on smallNORB dataset for a maximum of 40 epochs. We experienced some overfitting to the smallNORB training

Architecture	trainable params	size on disk
Capsnet	316k	1,28MB
Capsnet + Decoder	1,73M	6,93MB
Residual-Capsnet	507k	2,05MB
Weak-baseline	2,15M	12,80MB
Strong-baseline	23,52M	94,40MB

Table 6.4: Model sizes in parameters and Megabytes.

Architecture	Accuracy	train-time/batch
Capsnet	91,35%	0,391ms
Capsnet + Decoder	90,20%	0,395ms
Residual-Capsnet	91,36%	0,079ms
Weak-baseline	95,83%	0,003ms
Strong-baseline	95,83%	0,017ms

Table 6.5: Comparing classification accuracies on smallNORB dataset. For the Capsnet's layers the number of capsules (A, B, C and D) were set to 32 and EM-Routing iterations were set to 2.

data in all cases but especially in training of the baseline CNNs, who performed equally well with an accuracy of 95,83%. Our Residual-Capsnet was able to beat the standard Capsnet by the slightest of margins.

A more interesting result is found in training on the STL10 dataset, as shown in Table 6.6. Our Residual-Capsnet reaches 67,66% accuracy, which sits between the weak baseline (62,72%) and the strong baseline (75,42%). Unfortunately we were not able to get the standard Capsnet to converge during training. The accuracy fluctuated around random accuracy, which is 10% as we have possible 10 classes.

6.2.3 Quality of Reconstructions

As we have already seen in Figure 5.4 our reconstruction network was able to reconstruct the original MNIST images from the features of the classification output layer. If we test the same network trained on smallNORB data we notice that the reconstructions are getting worse the higher dimensional the input images are. Figure 6.1 shows reconstructions of 32 images of the smallNORB testset. We can clearly see that the network has learned to reconstruct some parts of the images, although not in detail. We believe that a more complex and advanced reconstruction network, maybe one using transpose convolution, would be able to learn to create more detailed reconstructions. This has to

Architecture	Accuracy	train-time/batch
Capsnet	-*	*0,473ms***
Residual-Capsnet	67,66%	0,404ms
Weak-baseline	62,72%	0,006ms
Strong-baseline	75,42%	0,019ms

Table 6.6: Comparing classification accuracies on STL10 dataset. For the Capsnet's layers the number of capsules (A, B, C and D) were set to 32 and EM-Routing iterations were set to 2. **network did not converge and fluctuated around random accuracy of 10%. **due to high memory consumption we had to set batch size to 2 for this architecture*

be shown and evaluated in future work.

6.3 Mask R-CNN

We evaluate the trained Mask R-CNN models using COCO-detection [Lin et al., 2014] metrics for bounding-boxes and masks. The metrics use mean average precision (mAP) and recall (AR) at 10 different intersection over union (IoU) thresholds of .50:.05:.95. They also include mAP/AR across scales of small ($area < 32^2$), medium ($32^2 < area < 96^2$) and large ($96^2 < area$) objects.

6.3.1 Different Backbone Networks

We first compare the performance of backbone networks: *Inceptionv2*, *Inception-ResNet-v2*, *ResNet-50*, and *ResNet-101*. All models are initialized with pre-trained weights from the COCO-dataset [Lin et al., 2014], excluding the last layer. We obtain the best results with ResNet-101 for both pixel-based mask predictions and bounding-box predictions (Table 6.7). As expected, Inceptionv2 achieves the worst performance. Therefore, all subsequent evaluations are only performed with ResNet-101.

Backbone	**mAP**	mAP**@.50IoU**	**AR@1**
Inceptionv2	0.268	0.480	0.346
Inception-ResNetv2	0.324	0.528	0.416
ResNet-50	0.306	0.530	0.387
ResNet-101	**0.354**	**0.607**	**0.461**

Table 6.7: Evaluation of different backbone networks on dataset 2 for mask predictions.

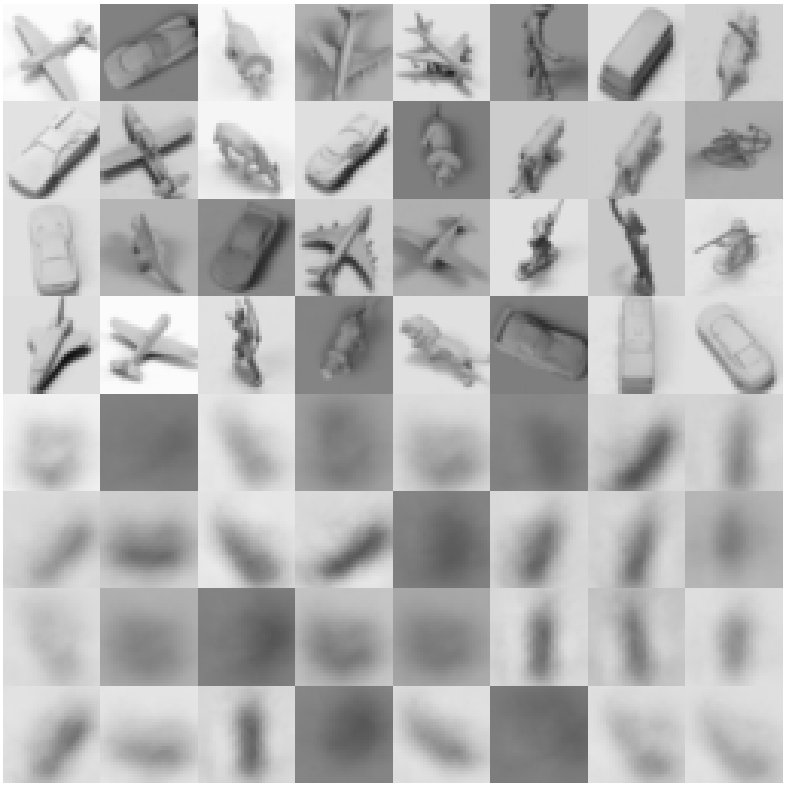

Figure 6.1: Showing reconstructions of the decoder network on a batch of smallNORB images. The decoders model complexity might not be high enough to reconstruct the full details of the images, but some success can be seen.

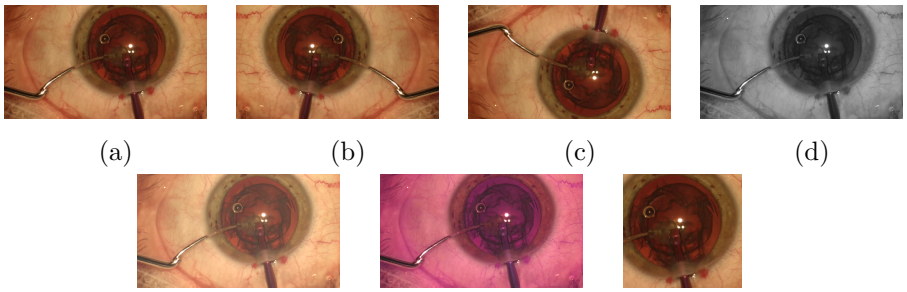

Figure 6.2: Example augmentations on an image of dataset 2: (a) original image, (b) horizontal flip, (c) vertical flip, (d) rgb-to-gray, (e) adjust brightness, (f) color-distortion and (g) random crop.

Backbone	**mAP$_{bb}$**	**mAP@.50IoU$_{bb}$**	**AR@1$_{bb}$**
Inceptionv2	0.407	0.574	0.484
Inception-ResNetv2	0.433	0.609	0.524
ResNet-50	0.423	0.604	0.499
ResNet-101	**0.473**	**0.685**	**0.576**

Table 6.8: Evaluation of different backbone networks on dataset 2 for bounding-box predictions.

6.3.2 Data Augmentation

Additionally we evaluate a variety of data augmentation methods and their effect on class predictions. For this we train five models with the following augmentation strategies: *none*: no augmentation at all, *geometric*: horizontal-/vertical flips and 90 degree rotation, *color*: brightness/saturation/rgb-to-gray/contrast/distortion, *bbox*: random jitter/crop on bounding-boxes and *combined*: a combination of all mentioned operations. Augmented examples are shown in Figure 6.2. Tables 6.9 and 6.10 show detailed results. We find that data augmentation does not necessarily improve the models' overall precision, with the model trained without augmentation performing very well in comparison to the others and the combined approach performing worse than all others. However, from all implemented data augmentation methods, bounding-box operations seem to perform best. More interestingly, as shown in Table 6.13, the augmentation strategies seem to have different effects on different classes. For example, the *aspiration/irrigation handpiece* achieves its best results with color augmentation, while the *capsulorhexis forceps*, *hydrodissection cannula*, *micromanipulator*, and *viscoelastics cannula* work best with geometric augmentation, and almost all other instruments are most accurately localized when trained with additional data using bounding-box augmentation (except for *phacoemulsification handpiece* and *lens injector*, which clearly achieve best results with a combined augmentation). Although not shown in the table, we achieve similar results with the second dataset.

6.3.3 Multi-Class and Binary Segmentation

As shown in Table 6.11 and 6.12, for multi-class instance segmentation, our best model trained on dataset 1 achieves a mAP precision score of 0.486 for detecting bounding-boxes and 0.229 for detecting masks. Our best model trained on dataset 2 has similar scores for detecting bounding-boxes (mAP precision of 0.473) but an improved score of 0.354 for detecting masks. After visual evaluation of the model predictions we suspect that the reason for this is the fact that due to the blurriness of the images in dataset 1

Augmentation	**mAP**	**mAP@.50IoU**	**AR@1**
none	0.3514	0.5531	0.4440
geometric	0.3126	0.5231	0.4082
color	0.3270	0.5391	0.4313
bbox	**0.3566**	**0.5803**	**0.4590**
combined	0.3063	0.5390	0.4085

Table 6.9: Evaluation of data augmentation strategies on dataset 2 for mask predictions with ResNet-101. Geometric uses horizontal-/vertical flip and 90 degree rotation, color uses brightness/saturation/rgb-to-gray/contrast/distortion, bounding box operations use random jitter/crop, and combined is a combination of all the mentioned operations.

Augmentation	**mAP$_{bb}$**	**mAP@.50IoU$_{bb}$**	**AR@1$_{bb}$**
none	**0.4659**	0.6507	0.5582
geometric	0.4290	0.5942	0.5216
color	0.4436	0.6382	0.5477
bbox	0.4642	**0.6703**	**0.5647**
combined	0.4054	0.5975	0.5050

Table 6.10: Evaluation of data augmentation strategies on dataset 2 for bounding-box predictions with ResNet-101. Geometric uses horizontal-/vertical flip and 90 degree rotation, color uses brightness/saturation/rgb-to-gray/contrast/distortion, bounding box operations use random jitter/crop, and combined is a combination of all the mentioned operations.

Binary	**mAP**	**mAP@.50IoU**	**AR@1**
DS 1	0.274	**0.770**	0.284
DS 2	**0.409**	0.710	**0.443**
Multiclass			
DS 1	0.229	0.559	0.299
DS 2	**0.354**	**0.607**	**0.461**

Table 6.11: Achieved performance of Mask R-CNN for binary semantic segmentation as well as multi-class instance segmentation (for mask predictions).

Binary	$\mathbf{mAP_{bb}}$	$\mathbf{mAP@.50IoU_{bb}}$	$\mathbf{AR@1_{bb}}$
DS 1	**0.718**	**0.928**	**0.612**
DS 2	0.592	0.839	0.573
Multiclass			
DS 1	**0.486**	0.656	**0.598**
DS 2	0.473	**0.685**	0.576

Table 6.12: Achieved performance of Mask R-CNN for binary semantic segmentation as well as multi-class instance segmentation (for bounding-box predictions).

it is harder to detect precise edges of the instruments and therefore the mask accuracy suffers, while the accuracy for bounding box detection stays the same. Additionally we train and evaluate *Mask R-CNN* models for binary semantic segmentation (merging all instruments into one class). At this point it is important to mention, that increasing the amount of classes might also increase the difficulty to reach the same overall precision. Evaluation of the model trained on dataset 2 has better results for detection masks despite the fact that is was trained to detect more than double the amount of classes (21) of the model trained on dataset 1 (9). This makes sense intuitively, but the real impact on object-detectors needs to be evaluated further.

Another factor to mention is the size of objects found in the images. It seems that small objects are harder to detect and therefore reduce the overall mAP. Dataset 2 includes a dozen images that show small objects, while dataset 1 does not.

Instrument	**none**	**geometric**	**color**	**bbox**	**comb.**
A/I Handpiece	0.5937	0.6277	**0.6607**	0.6314	0.6316
Bonn Forceps	0.7724	0.6361	0.7840	**0.7963**	0.617
Cap. Cystotome	0.5536	0.5521	0.5966	**0.6161**	0.5933
Cap. Forceps	0.8792	**0.8958**	0.7942	0.7476	0.7902
Charleux Cannula	0.4286	0.0	0.3571	**0.5714**	0.5429
Hydro. Cannula	0.5141	**0.5366**	0.5125	0.5052	0.4634
Lens Injector	0.4273	0.4462	0.4013	0.4389	**0.4639**
Micromanipulator	0.6371	**0.6589**	0.6443	0.6467	0.6142
Phaco. Handpiece	0.7298	0.8022	0.7904	0.7643	**0.8412**
Primary Knife	0.8610	0.8073	0.8367	**0.8766**	0.8054
Rycroft Cannula	0.5353	0.5229	0.4557	**0.5797**	0.4848
Secondary Knife	0.9126	0.8736	0.8265	**0.9619**	0.7971
Suture Needle	0.0	0.0	0.0	**0.1026**	0.0
Visco. Cannula	0.5949	**0.6098**	0.5645	0.5948	0.5542
Viter. Handpiece	0.0	0.0	0.0	0.0	0.0

Table 6.13: Comparing AP@.50IoU (masks) on classes of dataset 2, for models trained on different data augmentation strategies. Classes that are underrepresented in the testset ($\leq$8 annotations) are left out.

CHAPTER 7 Conclusion

To apply object recognition with deep neural networks we have pursued two directions. We have first taken a look at the state-of-the-art object detection where we applied an advanced CNN architecture Mask R-CNN to a problem in the medical field. We created two datasets for surgical tool detection in cataract surgery videos, which we made publicly available for further research. Our experiments with four different backbone networks based on Inceptionv2 and ResNet confirmed that ResNet-101 was most effective as expected. Evaluations with different common data augmentation strategies revealed that they did not improve overall segmentation performance, in contrast to common expectations. This is caused by strongly varying effects of each augmentation method across the different classes (surgical tools). These results suggest that data augmentation can help (e.g. through a class-specific selection of augmentation method) but need to be applied with caution.

When comparing segmentation performance on the two tested datasets, which are different in terms of visual content quality, we observe similar overall results. However, the visual quality of a dataset seems to have a significant impact on the performance of mask predictions vs. bounding-box predictions. While better results are obtained for bounding-box predictions on the low-quality dataset, the high-quality dataset enables better results for pixel-based segmentation (mask predictions). Future work in this area might focus on using semantic information about surgical tools and their locations for detecting temporal video segments like surgical phases or surgical actions.

In the second part of our object recognition journey we evaluate a rather new approach to deep learning for image classification. Capsule neural networks have been successfully applied to solve classification tasks of different kind and even reach state-of-the-art performance on small benchmark datasets. One of their main bottlenecks is the computational complexity when dealing with high dimensional inputs, such as digital color images. This is why most work so far has used data that is small in dimension and size. For our experiments we chose three datasets of different scale: MNIST, smallNORB and STL10. By reproducing the original Matrix capsule architecture with the popular

ML framework *PyTorch* we confirm the initial suspicion, that the network does not scale well and was not able to converge on the largest of the datasets STL10. By extending the architecture with residual blocks we were able to reduce the computational time by a factor greater than four for the two larger classification datasets, in particular small-NORB and STL10. More importantly our *Residual-Capsnet* was able to converge for all datasets and performed better than the weak baseline CNN and close to the strong baseline CNN *ResNet50* with 67,66% accuracy.

Future work will build on these findings and further extend the model architecture with different concepts to be able to solve more challenging real world problems such as ImageNet classification, object detection and localization, text classification and many more. We will have to figure out in the coming years whether CapsNets are just another neural network architecture in the big pool of deep learning research or if they can help us design great architectures that will come very close to the performance of a human brain.

Bibliography

[coc, 2017] (2017). Coco - common objects in context. `https://cocodataset.org/#detection-eval`. Accessed: 2020-10-13.

[cs2, 2020] (2020). Cs231n: Convolutional neural networks for visual recognition. `http://cs231n.stanford.edu`. Accessed: 2020-10-30.

[Afshar et al., 2020] Afshar, P., Heidarian, S., Naderkhani, F., Oikonomou, A., Plataniotis, K. N., and Mohammadi, A. (2020). Covid-caps: A capsule network-based framework for identification of covid-19 cases from x-ray images. *arXiv preprint arXiv:2004.02696*.

[Afshar et al., 2019] Afshar, P., Plataniotis, K. N., and Mohammadi, A. (2019). Capsule networks for brain tumor classification based on mri images and coarse tumor boundaries. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1368–1372. IEEE.

[Al Hajj et al., 2017] Al Hajj, H., Lamard, M., Charrière, K., Cochener, B., and Quellec, G. (2017). Surgical tool detection in cataract surgery videos through multi-image fusion inside a convolutional neural network. In *2017 39th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 2002–2005. IEEE.

[Al Hajj et al., 2019] Al Hajj, H., Lamard, M., Conze, P.-H., Roychowdhury, S., Hu, X., Maršalkaitė, G., Zisimopoulos, O., Dedmari, M. A., Zhao, F., Prellberg, J., et al. (2019). Cataracts: Challenge on automatic tool annotation for cataract surgery. *Medical image analysis*, 52:24–41.

[Bhamidi and El-Sharkawy, 2019] Bhamidi, S. B. S. and El-Sharkawy, M. (2019). Residual capsule network. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0557–0560. IEEE.

[Bhamidi and El-Sharkawy, 2020] Bhamidi, S. B. S. and El-Sharkawy, M. (2020). 3-level residual capsule network for complex datasets. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)*, pages 1–4. IEEE.

[Biederman, 1987] Biederman, I. (1987). Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115.

[Bienias et al., 2019] Bienias, L., n, J., Nielsen, L., and Alstrøm, T. (2019). Insights into the behaviour of multi-task deep neural networks for medical image segmentation. pages 1–6.

[Brooks, 2019] Brooks, J. (2019). COCO Annotator. `https://github.com/jsbroks/coco-annotator/`.

[Cho et al., 2014] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[Coates et al., 2011] Coates, A., Ng, A., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223.

[de Jesus et al., 2018] de Jesus, D. R., Cuevas, J., Rivera, W., and Crivelli, S. (2018). Capsule networks for protein structure classification and prediction. *arXiv preprint arXiv:1808.07475*.

[Deng et al., 2018] Deng, F., Pu, S., Chen, X., Shi, Y., Yuan, T., and Pu, S. (2018). Hyperspectral image classification with capsule network using limited training samples. *Sensors*, 18(9):3153.

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[Farabet et al., 2012] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2012). Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929.

[Flouty et al., 2019] Flouty, E., Kadkhodamohammadi, A., Luengo, I., Fuentes-Hurtado, F., Taleb, H., Barbarisi, S., Quellec, G., and Stoyanov, D. (2019). Cadis: Cataract dataset for image segmentation. *arXiv preprint arXiv:1906.11586*.

[Fox et al., 2020] Fox, M., Taschwer, M., and Schoeffmann, K. (2020). Pixel-based tool segmentation in cataract surgery videos with mask r-cnn. In *2020 IEEE 33rd International Symposium on Computer-Based Medical Systems (CBMS)*, pages 565–568. IEEE.

[Fuchs and Pernkopf, 2020] Fuchs, A. and Pernkopf, F. (2020). Wasserstein routed capsule networks. *arXiv preprint arXiv:2007.11465*.

[Garg, 2017] Garg, S. (2017). Demystifying matrix capsules with em routing. `https://towardsdatascience.com/demystifying-matrix-capsules-with-em-routing-part-1-overview-2126133a8457`. Accessed: 2020-10-10.

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gritzman, 2019] Gritzman, A. D. (2019). Avoiding implementation pitfalls of "matrix capsules with em routing" by hinton et al. In *International Workshop on Human Brain and Artificial Intelligence*, pages 224–234. Springer.

[Gu and Tresp, 2020] Gu, J. and Tresp, V. (2020). Improving the robustness of capsule networks to image affine transformations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7285–7293.

[Hauptmann and Adler, 2020] Hauptmann, A. and Adler, J. (2020). On the unreasonable effectiveness of cnns. *arXiv preprint arXiv:2007.14745*.

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton et al., 2011] Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer.

[Hinton et al., 2018] Hinton, G. E., Sabour, S., and Frosst, N. (2018). Matrix capsules with em routing.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Huang et al., 2017a] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017a). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[Huang et al., 2017b] Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., et al. (2017b). Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311.

[Hui, 2018] Hui, J. (2018). map (mean average precision) for object detection. `https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173`. Accessed: 2020-10-13.

[Jain, 2019] Jain, R. (2019). Improving performance and inference on audio classification tasks using capsule networks. *arXiv preprint arXiv:1902.05069*.

[Jaiswal et al., 2018] Jaiswal, A., AbdAlmageed, W., Wu, Y., and Natarajan, P. (2018). Capsulegan: Generative adversarial capsule network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0.

[Jiménez-Sánchez et al., 2018] Jiménez-Sánchez, A., Albarqouni, S., and Mateus, D. (2018). Capsule networks against medical imaging data challenges. In *Intravascular Imaging and Computer Assisted Stenting and Large-Scale Annotation of Biomedical Data and Expert Label Synthesis*, pages 150–160. Springer.

[Krishna et al., 2017] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., et al. (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International journal of computer vision*, 123(1):32–73.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[LaLonde and Bagci, 2018] LaLonde, R. and Bagci, U. (2018). Capsules for object segmentation. *arXiv preprint arXiv:1804.04241*.

[LeCun et al., 1990] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 2004] LeCun, Y., Huang, F. J., and Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2:II–104 Vol.2.

[Lim et al., 2016] Lim, S.-H., Young, S. R., and Patton, R. M. (2016). An analysis of image storage systems for scalable training of deep neural networks. *system*, 5(7):11.

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

[Marchisio et al., 2020] Marchisio, A., Bussolino, B., Colucci, A., Martina, M., Masera, G., and Shafique, M. (2020). Q-capsnets: A specialized framework for quantizing capsule networks. *arXiv preprint arXiv:2004.07116*.

[Perez and Wang, 2017] Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

[Quellec et al., 2014] Quellec, G., Lamard, M., Cochener, B., and Cazuguel, G. (2014). Real-time segmentation and recognition of surgical tasks in cataract surgery videos. *IEEE transactions on medical imaging*, 33(12):2352–2360.

[Rajasegaran et al., 2019] Rajasegaran, J., Jayasundara, V., Jayasekara, S., Jayasekara, H., Seneviratne, S., and Rodrigo, R. (2019). Deepcaps: Going deeper with capsule networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10725–10733.

[Rathnayaka et al., 2018] Rathnayaka, P., Abeysinghe, S., Samarajeewa, C., Manchanayake, I., and Walpola, M. (2018). Sentylic at iest 2018: Gated recurrent neural network and capsule network based approach for implicit emotion detection. *arXiv preprint arXiv:1809.01452*.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

[Rosch and Lloyd, 1978] Rosch, E. and Lloyd, B. B. (1978). Cognition and categorization.

[Rosebrock, 2016] Rosebrock, A. (2016). Intersection over union (iou) for object detection. `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`. Accessed: 2020-10-20.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Sabour et al., 2017] Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866.

[Scherer et al., 2010] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer.

[Schoeffmann et al., 2018] Schoeffmann, K., Taschwer, M., Sarny, S., Münzer, B., Primus, M. J., and Putzgruber, D. (2018). Cataract-101: video dataset of 101 cataract surgeries. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 421–425. ACM.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Srivastava and Khurana, 2019] Srivastava, S. and Khurana, P. (2019). Detecting aggression and toxicity using a multi dimension capsule network. In *Proceedings of the Third Workshop on Abusive Language Online*, pages 157–162.

[Sun et al., 2017] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852.

[Thorpe et al., 1996] Thorpe, S., Fize, D., and Marlot, C. (1996). Speed of processing in the human visual system. *nature*, 381(6582):520–522.

[Tsai et al., 2020] Tsai, Y.-H. H., Srivastava, N., Goh, H., and Salakhutdinov, R. (2020). Capsules with inverted dot-product attention routing. *arXiv preprint arXiv:2002.04764*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008.

[Venkatraman et al., 2020] Venkatraman, S. R., Anand, A., Balasubramanian, S., and Sarma, R. R. (2020). Learning compositional structures for deep learning: Why routing-by-agreement is necessary. *arXiv preprint arXiv:2010.01488*.

[von Zitzewitz, 2017] von Zitzewitz, G. (2017). Survey of neural networks in autonomous driving.

[Wolfe, 1998] Wolfe, J. M. (1998). Visual memory: What do you know about what you saw? *Current biology*, 8(9):R303–R304.

[Zou et al., 2019] Zou, Z., Shi, Z., Guo, Y., and Ye, J. (2019). Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*.

Acronyms

AI Artificial Intelligence. 5, 6, 8

ANN Artificial Neural Network. 16, 23

CapsNet Capsule Network. 6, 8, 9, 23, 25, 27, 28, 29, 41, 43, 48, 49

CNN Convolutional Neural Network. 5, 6, 8, 9, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 27, 28, 37, 41, 46

ConvLayer Convolutional Layer. 16, 17, 18, 19, 41, 43

CV Computer Vision. 5, 6, 10, 16

DL Deep Learning. 5, 6, 8

EM Expectation Maximization. 24, 25

GPU Graphics Processing Unit. 5, 6, 48

ILSVRC ImageNet Large Scale Visual Recognition Challenge. 6, 10, 11, 46

ML Machine Learning. 9, 30, 36, 59

MLP Multilayer Perceptron. 16

NLP Natural Language Processing. 16, 28

NN Neural Network. 5, 19, 23

RNN Recurrent Neural Network. 14, 28