DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Face Detection with Capsule Networks

Lucas Wolf

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Face Detection with Capsule Networks

Gesichtserkennung mittels Capsule Networks

Author:	Lucas Wolf
Supervisor:	Prof. Dr.-Ing. habil. Alois Knoll
Advisor:	M. Sc. Alexander Kuhn
Submission Date:	15. September 2018

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, Lucas Wolf

Abstract

Capsule Networks are a novel Computer Vision architecture which learn to decompose the scene into a hierarchy of entities. Recent publications [SFH17; HSF18] reported state-of-the-art performance on standard image classification datasets, while significantly reducing the number of model parameters compared to a baseline Convolutional Neural Network. In this work, we apply Capsule Networks to Face Detection in order to determine whether these results can be extrapolated to more demanding Computer Vision tasks. To this end, we combine the YOLO object detector [Red+15] with such a network and train the resulting architecture on the WIDER FACE dataset [Yan+16]. Based on these experiments, we identify a number of computational issues that impede the use of deep Capsule Networks with high-resolution inputs resolution in practice.

Contents

1 Introduction

The human brain is remarkably apt at processing visual information. Although our eyes provide high resolution exclusively within the *fovea centralis*, a small patch at the center of the retina, the brain is able to assemble a full mental picture by integrating stimuli from a sequence of eye movements into a coherent whole. From this, it reliably extracts scene information such as the identity and location of contained objects without any notable conscious effort.

Computers, on the other hand, still struggle with the task of making sense from visual data. While the problem was initially believed to be solvable within a few months (see the infamous "Summer Vision Project" in 1966 [Pap66]), one soon had to come to terms with the non-triviality of the endeavour. The results were six decades of joint research effort from various scientific disciplines including Computer Science, Physics, Biology, and Psychology, coalescing in the field of *Computer Vision*.

Recently, Computer Vision experienced a series of breakthroughs driven by advances in the training of deep *Convolutional Neural Networks* (CNNs), a class of Machine Learning models that outperformed existing Computer Vision algorithms by a considerable margin across a range of tasks. Unlike the brain however, CNNs require copious amounts of training data in order to generalize to previously unseen viewpoints. For this reason, Geoffrey Hinton, one of the leading figures in the Machine Learning community, proposed *Capsule Networks* – a novel vision architecture designed to rectify these shortcomings of CNNs.

1.1 Motivation

As of July 2018, Capsule Networks have been applied to standard image classification problems with great success, yielding results on a par with state-of-the-art CNN architectures while reducing the number of model parameters to a fraction [SFH17; HSF18]. In this work, we apply Capsule Networks to a more ambitious Computer Vision task in the hope of observing similar effects.

We choose *face detection* as a representative for such a task, since images of human faces exhibit a large amount of variation and the detection task requires the model to not only identify present faces but also to learn their position within the image. Additionally, the precise localization of small faces invariably demands the network to process input images at a high resolution. In conjunction, the inherent complexity of the task and the large input size should necessitate training a model with a higher capacity (i.e. a deeper Capsule Network) than in prior publications. Note that we deliberately restrict our architecture to the detection of faces, rather than arbitrary objects. While the latter would certainly have a wider range of applications in practice, this simplification spares the model from discerning various types of objects.

1.2 Scope

In this thesis we make the following contributions:

Chapter 2 provides a survey of current approaches to Computer Vision, in particular a discussion of the theoretic underpinnings of CNNs and Capsule Networks. A brief introduction to classical Computer Vision algorithms is given for context. We then elaborate on the discipline of face detection, the relevant datasets and metrics as well as state-of-the art algorithms.

Chapter 3 discusses *YOLOphem*, our architecture for face detection with Capsule Networks based on the *YOLO* object detector [Red+15]. In particular, we describe the design of YOLOphem, its network architecture and the loss function used during training. We also provide details on the implementation of the underlying capsule routing mechanism.

Chapter 4 documents our experiments conducted on YOLOphem. We train two variants of our architecture on the WIDER FACE dataset [Yan+16], evaluate the resulting models, and investigate the computational issues we observed during training.

Chapter 5 summarizes our findings. In particular, we attempt to answer the following research questions:

1. Are Capsule Networks suitable as a drop-in replacement for Convolutional Neural Networks in existing architectures?
2. Can larger Capsule Networks be trained efficiently?
3. Are Capsule Networks a suitable model (component) for the detection task?

Lastly, chapter 6 concludes this work by providing pointers to opportunities for future research.

2 State of the Art

In this chapter, we present state-of-the-art approaches to Computer Vision and face detection. Section 2.1 gives a general overview of the field of Computer Vision, describing classical approaches (2.1.1), Convolutional Neural Networks (2.1.2), Capsule Networks (2.1.3) and other frontiers in modern Computer Vision research (2.1.4). Building on this information, section 2.2 illustrates the discipline of face detection in further detail, summarizing datasets, benchmarks and metrics (2.2.1), as well as state-of-the-art algorithms (2.2.2).

2.1 Computer Vision

The human brain is able to perform a wide range of visual functions without substantial conscious effort. In the broadest sense, the field of *Computer Vision* aims to replicate these cognitive functions in software. More specifically, Computer Vision is concerned with the extraction of semantic information from visual input signals, such as images, videos or volumetric data. Typical tasks include image classification (assigning the dominant object in an image to one of several categories), object detection (classifying and localizing objects within an image) or optical flow analysis (determining the apparent movement of objects relative to the camera).

Since its inception in the 1960s, Computer Vision has become a major area of research within Computer Science, which led to the development of various algorithms following different approaches. This picture changed in 2012 when AlexNet [KSH12], a deep learning architecture for image classification, outperformed existing models in the ImageNet Large Scale Visual Recognition Challenge [Rus+15] by a considerable margin. Since then, a remarkable portion of Computer Vision research shifted focus towards end-to-end learning-based methods. We therefore refer to vision models that do not fully rely on machine learning as *classical approaches*.

2.1.1 Classical Approaches

Computer Vision research has experienced a constant influx of ideas over the past half-century, leading to a host of algorithms that satisfy our definition of “classical” (see e.g. [Sze10] for reference). Since an in-depth discussion of these approaches exceeds the scope of this thesis, we restrict our attention to common aspects among them.

Classical algorithms in Computer Vision frequently rely on heavy pre-processing of input data. In this step, higher-level visual information, such as the location and orientation of edges, corners or connected regions of the same color is extracted from the image. We henceforth refer to this kind of semantic information as *features*. The extracted features then serve as input to some task-specific computation, e.g. a learning-based classifier such as a support vector machine in the case of image classification. A key concern in this process is the robustness of the feature extraction algorithm; features should be reliably detected under varying external factors, e.g. varying camera viewpoints, object scales, and lighting conditions.

Algorithms for feature extraction can be broadly categorized into two classes: *Feature detectors* compute per-pixel information about the presence of the respective feature. The output has a similar shape to the input image and preserves its spatial structure. An example for this is the Canny edge detector [Can86], which computes the first derivative of pixel intensities in order to detect discontinuities in image brightness that might constitute edges. *Feature descriptors*, on the other hand, compress the input image into a dense representation. The output is typically vector-valued which either encodes or neglects spatial structure. The histogram of oriented gradients (HOG) [DT05], which concatenates histograms of intensity gradients in regularly-spaced image cells into a feature vector, is a representative of this group.

As an almost universal rule, the quality of a classical Computer Vision algorithm depends directly on the quality of the features used. This has been a major impedance to consistently satisfying results, as feature extraction algorithms often depend on expert knowledge or borrow ideas from other domains such as calculus, differential geometry or physics. Also, almost all feature extraction mechanisms require themselves prior pre-processing, such as a Gaussian blurring of the image. As a result, many classical approaches have been superseded by end-to-end learning-based techniques, which learn feature representations from the input directly. We therefore narrow our focus on pure machine learning algorithms for the remainder of this thesis.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) (for reference, see e.g. [GBC16]) refers to a family of artificial neural network architectures particularly suited for processing image data. Their overall design originated from Fukushima's *neocognitron* [Fuk80], an early hierarchical model inspired by the visual cortex. LeCun *et al.* [LeC+89] later provided the extension to CNNs, most notably supervised training via backpropagation and stochastic gradient descent.

Convolutional Layers

The primary building block of a CNN is the *convolutional layer*. These build on the notion that the detection of visual features (e.g. edges, corners or higher-level entities) should be independent of their location within the image. In other words, detecting, for example, horizontal edges in two different parts of the image should require the same computations at the respective image regions. Convolutional layers exploit this property by replicating learned local feature detectors across space. To this end, fixed-sized, rectangular patches are extracted from the image in regular intervals in a left-to-right, top-to-bottom fashion. We refer to the spatial extent of a patch as the layer's *kernel size* K and to the offset between two (possibly overlapping) adjacent patches as the layer's *stride* S. The patch-extraction process is illustrated in Figure 2.1

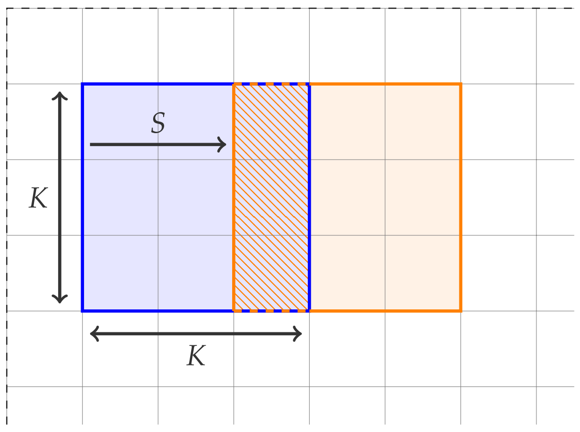

Figure 2.1: Two successively extracted patches, shaded in blue and orange. A grid cell signifies a pixel in the input image. Note that the hatched region is contained in both patches. The kernel size and stride in this example are $K = 3$ and $S = 2$, respectively.

Note that a patch spans the entire depth C_{in} of the input, e.g. the red, blue, and green channels in the case of a colored image. A patch can thus be thought of as a $K \times K \times C_{in}$ tensor. The extracted patches then serve as input to an artificial *neuron*, i.e. an affine transformation with some subsequent nonlinear activation function (see Equation 2.1). The weights of the neuron, typically referred to as the *kernel* or *filter*, are learned during training and shared among all input patches. Intuitively, this process can be thought of as "sliding" the neuron over the input image, similar to the discrete convolution operation frequently used in signal processing. It should, however, be noted that convolutional layers compute cross-correlation rather than a discrete convolution.

$$f(\mathbf{x}; \mathbf{w}) = \sigma\left(\sum_{i=1}^{K^2} w_i x_i + w_0\right) \tag{2.1}$$

Given a fixed kernel, a convolutional layer acts as a local nonlinear feature extractor where the kernel determines the nature of the feature being extracted. The activation computed over a given patch indicates to what extent that particular feature is present at the corresponding image location. For a given filter, the computed activations are arranged into a two-dimensional *filter map*, according to the locations from which the input patches were extracted (see Figure 2.2 for illustration). Put another way, adjacent input patches lead to adjacent output activations, thereby preserving the spatial relations between the extracted features. As a result of both the preservation of spatial structure and the sharing of weights across input patches, convolutional layers exhibit *translational* and *rotational equivariance*, meaning that shifting or rotating features in the input's spatial domain results in a similar translation or rotation in activations within the corresponding filter map.

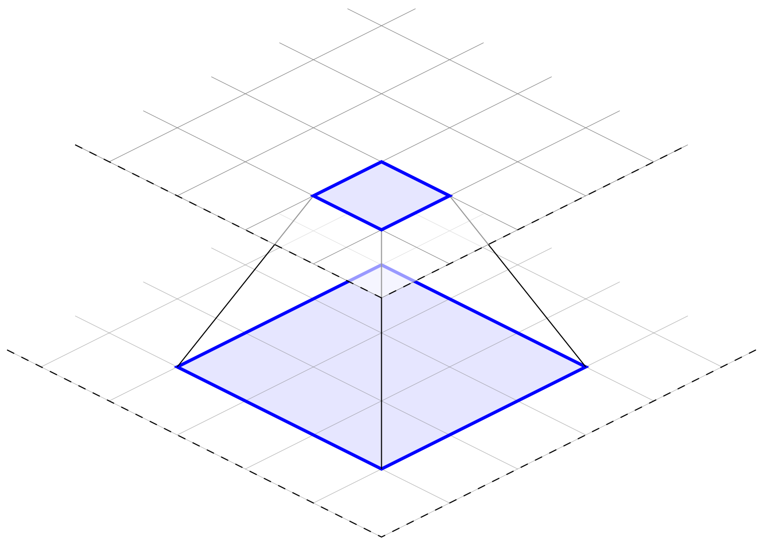
Figure 2.2: Relation of an extracted patch (lower layer) to the computed activation in the corresponding filter map (upper layer).

To extract more than one feature, a single convolutional layer typically comprises several kernels of identical size and stride, which independently compute a corresponding number of filter maps. We refer to these as the layer's *(output) channels* and denote their number by C_{out}. The total count of trainable parameters in a convolutional layer thus amounts to $(K \cdot K + 1) \cdot C_{in} \cdot C_{out}$ (the "plus one" stems from the additive bias in a neuron's affine transformation).

Following the idea of deep neural networks, convolutional layers in CNNs are stacked upon one another in order to increase model capacity. In such a configuration, the input to a higher-level convolutional layer is not an array of pixels but an array of features, i.e. the filter map computed by the lower-level convolutional layer. This enables the network to learn complex features in a hierarchical fashion, e.g. corners from features encoding oriented edges. From this perspective, convolutional layers can thus be thought of as a variant of fully-connected layers, where a) weights are shared between all output neurons for an output channel and b) a particular neuron is not connected to the entire input but only to a K-$\times$-K-sized *receptive field* (across all input channels). This dramatically reduces the number of parameters to learn during training, rendering CNNs computationally feasible.

Pooling Layers

The second most common component of modern CNN architectures is the *pooling layer*, which performs down-sampling of the input tensor. This is achieved by extracting patches at a fixed kernel size and stride, similar to convolutional layers. For a given patch, each channel is then *pooled* into a single scalar value. This reduction is typically not learned, but realized via some fixed function such as the maximum of the values in the respective channel patch (see Figure 2.3 for an example). Pooling layers therefore effectively "scale down" the input without introducing additional model parameters, keeping the training of larger networks feasible. To a small degree, pooling layers also exhibit *translational* and *rotational invariance*, meaning that slight translations and rotations in the input do not alter the output, as the maximal value within a patch still remains within that patch. While this property is believed to contribute to the model's stability, the overall effect is of minor significance.

Deep CNNs

Abundance of (labeled) data and the discovery of non-saturating activation functions (see e.g. [NH10; CUH15]) and efficient nonlinear optimization procedures ([Sut+13; DHS10; KB14]) facilitated the training of convolutional networks with many layers, commonly referred to as *deep CNNs*. Prominent examples include the architectures published by Krizhevsky *et al.* [KSH12] (8 layers, 60M parameters) and Simyonian and Zisserman [SZ14] (16 layers, 138M parameters).

A frequent design choice in these architectures is to decrease the spatial extent of

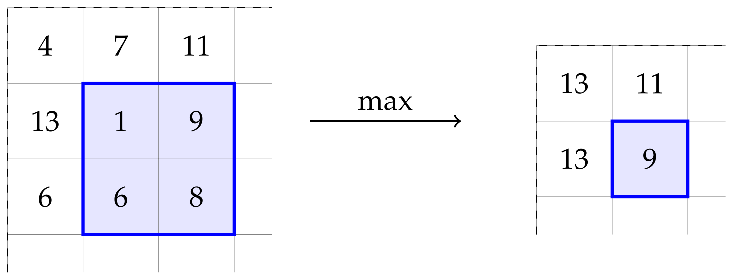

Figure 2.3: *Max–Pooling* applied one input channel. Patches are extracted with $K = 2$, $S = 1$ for each channel separately. Each patch is then reduced to its maximum activation.

the layers with the network depth by interleaving (blocks of) convolutional layers with pooling layers. Simultaneously, the number of filter maps is increased. Intuitively, this follows the observation that higher-level features are more diverse but appear less frequently in an image. Furthermore, the effective receptive field of a downstream convolutional layer compounds with depth, since it comprises features that have been computed from patches of features, that have been computed from patches of features, etc., which also justifies smaller layers deeper in the network.

Since 2012, deep CNNs have been successfully employed in a wide range of disciplines, such as image classification [KSH12; SZ14; He+15; Sze+15], object detection [Gir+13; Red+15; Liu+16] or semantic segmentation [LSD14; He+17]. Applying these models to a specific task can however be difficult, as deep architectures require a large amount of data and training iterations in order to converge. For this reason, researchers and practitioners often resort to a technique called *transfer learning*, in which a pre-trained CNN is adapted to the task at hand. To this end, the layers of an existing network, typically trained on a large image classification dataset such as ImageNet, are copied into the new architecture in order to extract general-purpose features from the input. The new network is then fine-tuned on application data for a number of additional training epochs. The specific mechanics of this process, such as whether to include the imported weights in the optimization procedure may vary from instance to instance. In any case, transfer learning can help to speed up training and reduce the amount of required data to a significant degree.

Recent developments proposed a variety of minor modifications to the architectures described above. Most notably among them are the use of dropout regularization [Sri+14], batch normalization [IS15] and residual learning [He+15]. The specifics of these techniques, however, lie beyond the scope of this thesis.

2.1.3 Capsule Networks

Despite their success across various tasks, CNNs are not unanimously considered as an optimal model for vision. *Capsule Networks*, a recent architecture developed by the renowned Geoffrey Hinton (see e.g. [RHW88; AHS88; KSH12; Sri+14]), attempt to rectify these shortcomings. To this end, Hinton identified a set of flaws in modern CNN architectures in a talk given at the Massachusetts Institute of Technology in 2014 [Hin14]:

(1) Lack of structure According to Hinton, the structural hierarchy in contemporary CNNs (i.e. neurons, layers and networks) does not reflect the complexity of the task, especially when compared to the amount of structure present in the human cortex. Moreover, CNNs lack the explicit notion of entities in the scene, which may impede generalization.

(2) Lack of equivariance While convolutional layers exhibit translational and rotational equivariance, the invariances introduced by pooling layers prevent these properties to inherit on the entire network (see 2.1.2). Hinton hypothesizes that this is detrimental to the models performance as changes in viewpoint are not reflected by the neural activities within deeper layers of the network.

(3) Mismatch to human psychology Humans perceive shapes by imposing rectangular coordinate frames upon individual objects in the scene. Hinton argues that this process is likely to result in a hierarchy of interlinked coordinate systems, similar to the notion of a scene tree in Computer Graphics. Further, the relationship between these coordinate frames are represented by several neurons in the cortex, similar to a transformation matrix. This explains why humans are able to identify the type and orientation of objects quickly, but need significantly longer in order to determine their handedness, which would require computing the determinant of said transformation matrix (compare Figure 2.4). CNNs however lack mechanisms for imposing coordinate frames (apart from "accidentally" emulating it as part of the learned computation).

(4) Improper routing of pixel information The task performed by the layers of a neural network can be understood as the solution to a routing problem; pixel information from the image has to be directed to those neurons that are specialized in processing this information. From this standpoint, finding the optimal route through the network can be regarded as "parsing" the image. CNNs implement this procedure via pooling layers, which route information solely depending on the strength of the input signal. In Hinton's view, this is suboptimal, as it fails to handle “dimension hopping”, i.e. the shift of information to a different set of pixels as the viewpoint changes.

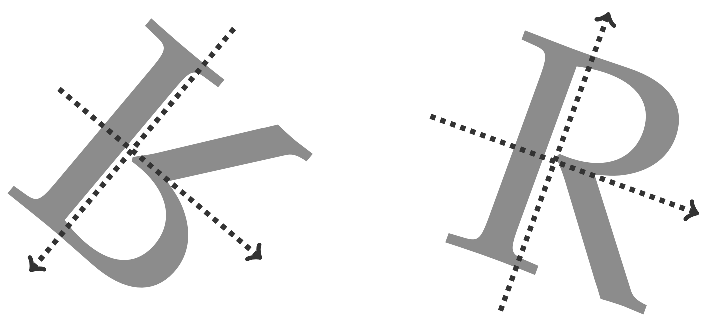

Figure 2.4: Two objects and their imposed coordinate frames. While it is easy for humans to infer their identity (the letter “R”) and orientation (the direction of the coordinate axes), it takes a substantially longer amount of time to determine that the left “R” has been mirrored. (Illustration adapted from [Hin14])

According to Hinton, CNNs fail to generalize to previously unseen viewpoints due to these architectural flaws. Current applications circumvent this problem by simply training the model on a range of different viewpoints, which in return demands for large sets of training data.

Capsule Networks try to alleviate these issues by grouping neural activities within a layer into *capsules*. A capsule is meant to represent an entity in the scene by encoding the probability of its presence (its *activation*) as well as properties of that entity, thereby addressing aspect (1) in the enumeration above. The properties captured by a capsule, to which we refer as its *(generalized) pose*, are chosen by the network via discriminative learning and in practice often relate to distinctive visual parameters of the entity, such as its thickness, skew or scale (see [SFH17]).

Capsules are organized into hierarchical layers, similar to neural networks. In this arrangement, a higher-level capsule represents an entity composed of other entities represented by lower-level capsules, thereby capturing part-whole relationships in the scene. To this end, a learned transformation matrix $\mathbf{A}_{ik}$ encodes the relation between the pose of a higher-level capsule k and the pose of a constituent part i (see Figure 2.5 for an example of a capsule hierarchy). This serves as an inductive bias for the model to capture the translations between the "coordinate frames" of individual objects as outlined in (3).

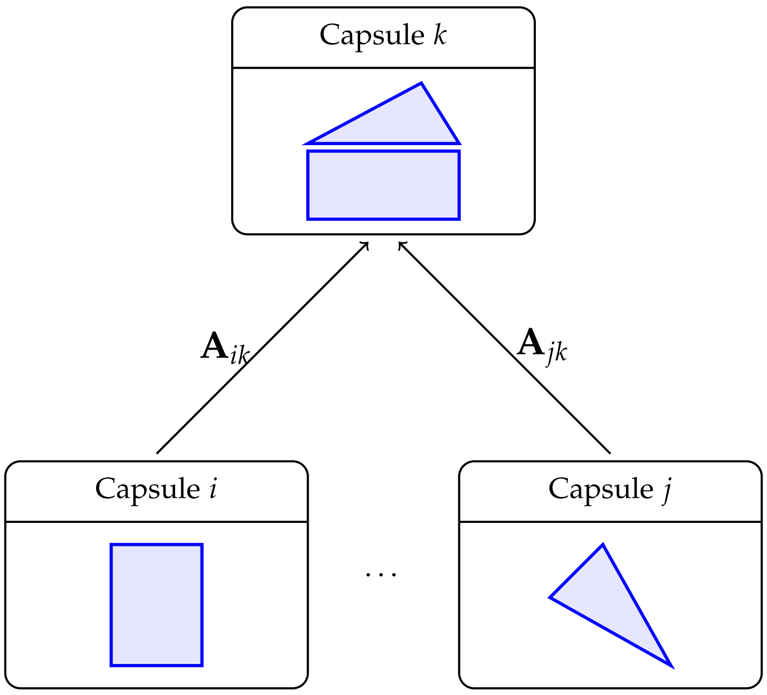

Figure 2.5: A hierarchy of (idealized) capsules. In this example, the lower-level capsules i and j represent abstract shapes from which the object detected by the higher-level capsule k is composed. The details of this part-to-whole relationship are captured by the respective transformation matrices A_{ik}, A_{jk}. (Illustration adapted from [Hin14])

The pose and activation of a higher-level capsule are computed from the poses and activations of lower-level capsules via *coincidence filtering*: Each lower-level capsule predicts the pose of the higher-level capsule by multiplying its own pose with the corresponding transformation matrix mentioned above. The higher-level capsule then searches for tight clusters in the resulting set of predictions. If a cluster is found, the higher-level capsule becomes active and assumes a pose that describes the cluster (e.g. its centroid). An illustration of this process can be found in Figure 2.6 Intuitively, this amounts to inspecting the lower-level capsules and asking "If this object was a part of a higher-level object, how would the higher-level object look like?"; if the answers look similar for many lower-level objects, the probability that these objects are parts of a composed entity is high. In order to filter out those predictions that do not agree (i.e. lie within the vicinity of the cluster), the input predictions from the lower-level capsules are moderated by a set of weights that is adapted during clustering via top-down feedback. This method of assigning parts to wholes can be thought of as an explicit routing process in the spirit of aspect (4), where the converged weights indicate the optimal route.

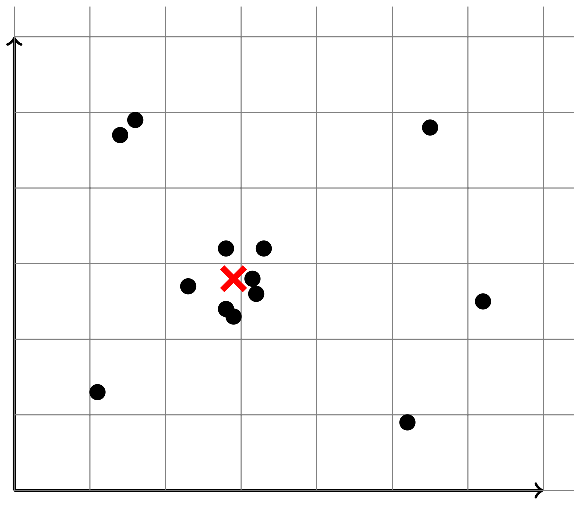
Figure 2.6: An illustration of the clustering process for one higher-level capsule. Black dots signify the pose predictions of a lower-level capsules. The final pose of the higher-level capsule is depicted as a red cross. Note that although plotted on a two-dimensional plane, the points symbolize vectors in the (high-dimensional) pose space of the higher-level caspule. (Illustration adapted from [Hin14])

Tracing the optimal route through the network yields a tree of capsules (and thus entities) that decomposes the input image. From this perspective, Capsule Networks can be understood as performing "Inverse Graphics"; contrary to Computer Graphics where a tree of objects with local coordinate systems is rendered into an image, the tree is reconstructed by detecting the individual parts and the transformations by which they relate to another. In doing so, the network converts the information from the input space of pixel intensities into the space of generalized pose parameters of the particular capsules. As an effect, the manifold of images of objects of the same rigid shape, which might be highly nonlinear in the input space becomes linear in the transformed space. This, according to Hinton, enables Capsule Networks to easily extrapolate to novel viewpoints while relying on far less trainable parameters than CNNs. To put this claim into perspective, the Capsule Network published in [SFH17] was able to outperform a competitive CNN baseline with five times as many parameters.

At the time of writing Capsule Networks are still an area of active research. However, most published architectures [SFH17; HSF18] follow the same general design: Initially, the input image is fed through a small number of convolutional layers in order to "derender" the image from pixel intensities to pose parameters. A subsequent layer of *primary capsules* then groups these activations into capsules. The remainder of the network consists of either *fully-connected capsule layers*, in which each lower-level capsule predicts the pose of each higher-level capsule or *convolutional capsule layers* which behave similar to convolutional layers in CNNs (i.e. higher-level capsules receive predictions exclusively from lower-level capsules within their receptive field and the layer output is arranged into two-dimensional filters). Note that convolutional capsule layers are fully equivariant (see issue (2)), as translations and rotations in the input image result in either a change of the pose within the affected capsule (*rate-coded equivariance*) or in moving pose and activation information to another capsule in the same filter map (*place-coded equivariance*), depending on whether the change is contained within the visual field of a particular capsule.

We defer the discussion of further implementation details such as the programmatic representation of capsules or the precise mechanics of the routing algorithm to section 3.1 where we describe the Capsule Network architecture used in this thesis.

2.1.4 Other frontiers in Computer Vision

We want to end our preliminary discussion of Computer Vision by providing a non-exhaustive overview of other research areas besides Capsule Networks. Promising ideas include generative models and spiking neural networks, which we will now briefly introduce:

Generative Models are a class of machine learning models that generate new images by capturing underlying structure in the training data. This is done by learning a probability distribution from which the data is assumed to be sampled. Some approaches model the density function explicitly, for example via decomposition into a product of conditional densities (*Fully Visible Belief Networks*, see e.g. [OKK16; Oor+16]) or through the introduction of latent variables (*Variational Autoencoders* [KW13]). *Generative Adverserial Networks* (GANs) [Goo+14] follow a different approach and model the density implicitly, by learning a transformation from a uniform distribution to the desired data distribution. This transformation is modeled with a *generator* neural network that is pitted against a *discriminator* network, similar to a zero-sum game. During training, the generator is trained to produce new images from the input distribution, while the discriminator has to discern generated "fake" images from "real" input samples. This is achieved by optimizing a joint objective function in an alternating min-max fashion. GANs have been demonstrated to produce very convincing imagery [BSM17; Iso+16], while their major downsides are a high demand of training data and convergence issues during training. However, recent work keeps introducing improvements in that regard [ACB17; Gul+17; BSM17].

Spiking Neural Networks (SNNs) (see e.g. [GK02]) comprise a range of neural network models with a higher degree of biological realism than that found in artificial neural network architectures. This is primarily achieved by modelling the dynamics of membrane potentials found in biological neurons with a set of differential equations. Neuron models employed in SNNs range from simplified, phenomenological descriptions to high-fidelity models that capture the concentration of sodium, chlorine and potassium ions in the cell. SNNs have been successfully applied to Computer Vision problems, such as image classification [Sen+18]. Their main benefit, however, is tied to recent ventures in developing specialized hardware (see e.g. [Fur+14; Sch+10]). These platforms allow simulating large neuron populations at a low energy consumption, which renders SNNs as a potential candidate for robotic or embedded applications.

2.2 Face Detection

Research suggests that the ability to identify facial shapes begins to develop already prior to birth [Rei+17], emphasizing its significance for social interaction. It is therefore not surprising that the visual processing of faces is a central pursuit in Computer Vision.

Among this set of tasks, the discipline of *face detection* is concerned with the detection and localization of faces in images. More specifically, we refer to face detection as the prediction of a) one or more (or none) bounding boxes framing the faces present in a given image together with b) a confidence score within the interval $[0, 1]$ for each predicted bounding box. Each bounding box should contain a single, entire face and nothing but that face. An example of this can be seen in Figure 2.7. Further, we distinguish face detection from other face-centered tasks such as *face recognition*, *face alignment* or *face parsing*; for state-of-the-art methods for these tasks see e.g. [SKP15; Zha+14; LWT12], respectively.

2.2.1 Datasets, Benchmarks and Metrics

Recent advances in face detection must not only be attributed to algorithmic improvements but also to the availability of large-scale datasets. These simultaneously serve as training source for learning-based detectors as well as evaluation benchmarks for detection performance. Publicly available datasets include AFW [Ram12], FDDB [JL10], MALF [Yan+15a], WIDER FACE [Yan+16] and datasets from the IARPA Janus Benchmark (e.g. IJB-A [Kla+15]). However, AFW, FDDB and MALF contain merely a small number of images with limited variation in pose, scale or facial expression,

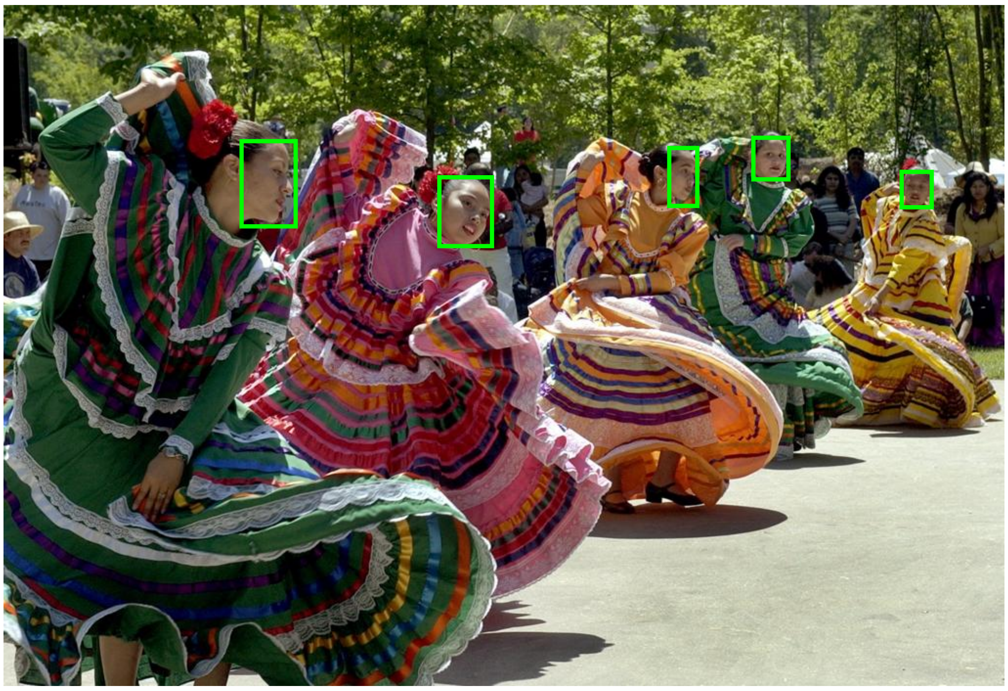

Figure 2.7: Exemplary output of a face detector on a picture from the WIDER FACE dataset [Yan+16]. Confidence scores for the particular bounding boxes are not shown. Note that the algorithm fails to detect the smaller faces in the background.

which caused algorithm performance to saturate over the recent years (for a detailed discussion, see [Yan+16]).

The WIDER FACE dataset comprises 32, 023 images containing 393, 703 faces annotated with ground-truth bounding boxes. Samples are tagged with meta-information about blur, occlusion, illumination, facial expression, pose and situational context, allowing for fine-grained error analysis. Additionally, images are categorized by difficulty as either “Easy”, “Medium” or “Hard”, based on the detection rate of a general-purpose object proposal algorithm. The data is divided into training, validation and test sets using a 40%/10%/50% split, respectively.

Detection performance on WIDER FACE is evaluated by means of precision-recall curves. In the context of face detection, *precision* is defined as the ratio of true positives over the total number of faces predicted for a given level of confidence. In a similar vein, *recall* is defined as the ratio of true positives over the total number of faces present. Equation 2.2 summarizes these definitions, where TP denotes the number of true positives (faces in the image detected as such), FP denotes the number of false positives (detections of faces that are not present in the input), and FN denotes the number of false negatives (faces in the image that were not detected) for a fixed confidence threshold.

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN} \tag{2.2}$$

A prediction is considered a true positive if the predicted bounding box *pred* achieves an *Intersection-over-Union* (IoU) larger than 0.5 with the respective ground-truth box *truth*. IoU is computed according to Equation 2.3, where A_b denotes the area in pixels covered by a box b.

$$\text{IoU}(pred, truth) = \frac{A_{pred} \cap A_{truth}}{A_{pred} \cup A_{truth}} \tag{2.3}$$

Precision-recall curves are generated by interpreting the precision achieved at a certain level of confidence as a function of the corresponding recall and plotting the resulting points. To directly compare two detectors, these curves are often condensed into a single scalar value by integrating over the area under the curve. In this aspect, WIDER FACE follows the PASCAL VOC object detection challenge [Eve+10] and resorts to approximating this quantity by discretizing the integral. The resulting metric, commonly known as *average precision* (AP) is computed by averaging the maximum precision for eleven uniformly-distributed levels of recall. The precise formulation is given in Equation 2.4, where $p(r)$ denotes the precision achieved for a particular level of recall r.

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} \max_{\tilde{r}: \tilde{r} \geq r} p(\tilde{r}) \approx \int_0^1 p(r) dr \tag{2.4}$$

Precision-recall curves and AP scores of various algorithms are published on the WIDER FACE website[1]. We therefore restrict our discussion to WIDER FACE, although substantially larger datasets (e.g. IJB-C with over 140,000 images) have been published since.

2.2.2 Face Detection with Convolutional Neural Networks

Similar to many other Computer Vision disciplines, classical approaches to face detection (notably [VJ01; FMR08]) have been almost entirely replaced by Convolutional Neural Networks over the recent years (cf. section 2.1.2). At the time of writing, only two out of 18 entries in the WIDER FACE leaderboard are utilizing classical face detection algorithms, ranking in places 14 and 18.

[1] `http://mmlab.ie.cuhk.edu.hk/projects/WIDERFace/index.html`

Among the host of CNN-based face detectors, the majority casts face detection as a special case of *object detection*, i.e. the prediction of bounding boxes and confidence scores for various object classes (although [Yan+15b] represents a notable exception). Using transfer learning, networks for face detection are typically trained by fine-tuning architectures pre-trained on general-purpose object detection datasets like PASCAL VOC [Eve+10] or Microsoft COCO [Lin+14]. We will therefore devote the remainder of this chapter to the discussion of current CNN-based object detection models.

In the following, we will discuss the two major streams of convolutional object detection models: *region-based* and *single-shot* models. To compare these architectures, we include the *mean average precision* (mAP) scores and execution times as reported by the respective authors in Table 2.1 at the end of this chapter. mAP on PASCAL VOC is computed by averaging the achieved AP scores over all object classes. MS COCO recently proposed a new version of the mAP metric that additionally averages over the AP scores generated by varying the IoU threshold for true positives in the interval [0.5, 0.95] in steps of 0.05. To disambiguate between these two flavours of mAP, we will refer to them as mAP@0.5 and mAP@[.5, .95], following [Ren+15].

Region-based Convolutional Neural Networks

Region-based models generally work in a two-step fashion: First, rectangular candidate regions are extracted from the image. These *proposals* are then fed into a CNN which classifies the extracted patches as either one of the respective object categories or background. Some architectures additionally refine the predicted bounding box by regressing correction terms relative to the position and dimension of the proposal.

Arguably, the easiest approach to proposal extraction is by sliding a rectangular mask over the image (similar to a convolution kernel) and crop proposal regions at fixed intervals. While these *sliding-window detectors* are simple in terms of implementation, they suffer from substantial drawbacks:

(1) Speed Sliding-window detectors are slow and computationally expensive as each proposal requires a full network evaluation. This prevents the use of deeper networks downstream.

(2) Shared Information Since each image patch is analyzed separately, shared information between adjacent or overlapping patches is ignored.

(3) Object Scale Objects may appear in multiple sizes, depending on camera viewpoint and perspective. As a result, the detection network may not recognize objects that are very small or very large relative to the window size.

(4) Multiple Detections Objects may be fully visible in several overlapping proposals. This may result in multiple detections of the same instance.

Issue (3) is sometimes circumvented by extracting patches at various window sizes or image resolutions (see e.g. [Yahoo]). However, this increases the number of required network evaluations by a linear factor and thus only contributes to issue (1). Issue (4) is typically alleviated via a technique called *non-maximal suppression* (NMS) which greedily selects the bounding-box with the highest confidence score among a set of predictions with high overlap.

Over the recent years, a series of improvements has been developed that made region-based detectors with deep CNNs feasible. *Regions with convolutional features* (R-CNN) [Gir+13] addressed issues (1) and (3) by replacing the sliding-window paradigm with proposals from a object-independent region proposal algorithm (Selective Search [Uij+13]) and regressing bounding-box corrections as described above. This reduced the number of network evaluations to roughly 2000 per image and also allowed for proposals of variable spatial dimensions. By fine-tuning a re-implementation of AlexNet [KSH12] pre-trained on ImageNet [Rus+15] as classification backbone, R-CNN was able to achieve 53.7% mAP@0.5 on the VOC 2010 test set at the time of publication. This amounts to an improvement of 13.3 percentage points compared to the then top entry. Still, R-CNN takes a long time during execution (47s/image on a GPU with VGG-16 [SZ14] as CNN backbone) due to the numerous network evaluations.

Fast R-CNN [Gir15] further improved on issue (1). Instead of analyzing each region individually, the entire image is fed through the convolutional layers of the CNN backbone. The region proposals generated by Selective Search are then projected onto the output feature maps of the network where activations within the projected regions are converted into fixed-length vectors via max-pooling. These feature vectors serve as input to fully-connected networks that predict classification scores and perform bounding-box regression. The reversal of proposal extraction and feature computation effectively eliminated issue (1): Faster R-CNN achieves a test-time performance of 1,83s/image on a GPU, using VGG-16 as a feature extractor. Computing convolutional features over the entire image has the additional benefit of allowing the network to encode global image information about objects in the scene (counteracting problem (2)), leading to improved detection accuracy (66.1% mAP@0.5 on VOC 2010).

Faster R-CNN [Ren+15] improves the model even further, by replacing Selective Search with a fully-convolutional *Region Proposal Network* (RPN) which generates proposals relative to reference boxes called anchors. Anchors vary in dimension and aspect-ratio (thereby addressing issue (3)) and are placed over the image on a regular grid. Sharing the convolutional layers of Fast R-CNN with the RPN enabled the authors to reduce execution time to a mere 198ms/image (on a GPU with VGG-16 as feature extractor). Feature sharing additionally allows Fast R-CNN and the RPN to jointly reason about global image information, raising detection accuracy to 67.0% mAP@0.5 on VOC 2012 while achieving near real-time performance.

Region-based models represent the current state-of-the-art in object detection. At the time of writing, most leading entries on the MS COCO leaderboard[2] are variants of R-CNN (see e.g. [Liu+18], [He+17]). Their major drawback however lies in their inherent complexity; Faster R-CNN for example has to be trained in a complicated four-step alternating optimization procedure and requires non-trivial sampling strategies in order to converge.

Single-Shot Approaches

Single-Shot approaches are considerably simpler than region-based models, since they process the entire image and predict bounding boxes and class probabilities simultaneously. Examples of single-shot predictors include SSD [Liu+16], YOLO [Red+15], RetinaNet [Lin+17] and their respective variants. Nevertheless, we will limit our discussion on the YOLO family as it forms the foundation for the models we present in this thesis.

You Only Look Once (YOLO) [Red+15] divides the input image into a regular grid of $S \times S$ cells. A grid cell that overlaps the center of a particular ground-truth bounding box is responsible for detecting the respective object. To this end, two sets of predictions are made for each cell:

1. B bounding boxes and objectness scores[3]. Box locations are encoded as offsets relative to the location of the cell. Similarly, box dimensions are parametrized relative to dimensions of the image. Both quantities are normalized to the unit interval. "Objectness" encodes the probability that a predicted box contains an object, multiplied with the IoU between prediction and ground truth box.

2. C class probabilities. These probabilities are conditioned on the the grid cell containing an object. The confidence level for a particular predicted bounding box is computed by multiplying the objectness score of the box with the conditional class probability of the cell.

YOLO relies on a CNN architecture similar to GoogLeNet [Sze+15] pre-trained on ImageNet. The final output is a tensor of shape $S \times S \times (B * 5 + C)$ containing

[2]`http://cocodataset.org/#detection-leaderboard`

[3]Note that the authors of YOLO refer to these as "confidence scores". However, we will use to the term "objectness" in order to prevent ambiguity with the notion of "confidence" of bounding box predictions.

the aforementioned predictions for an input image. The model is trained via the l_2-loss with minor modifications for gradient stability (e.g. decreasing the loss from objectness predictions for boxes that do not contain objects by a constant factor). To foster specialization among the bounding box predictions for a particular cell only "responsible" predictors, i.e. the bounding box with the highest current overlap with the ground truth box among the B predicted boxes, contribute to the loss. The single-forward-pass architecture allows for very short evaluation times. YOLO achieves 63.4% mAP@0.5 on VOC 2007 with 0.02s/image (45Hz) on a GPU. *FastYOLO*, a variation of YOLO built on a shallower network is able to run at 155Hz, achieving 52.7% mAP@0.5. While YOLO is fast, its detection performance is impaired by the coarse spatial resolution of the grid, causing localization errors to be the dominant error mode.

YOLOv2 [RF16] alleviates this issue by replacing unconstrained bounding box prediction with an anchor approach, similar to Faster R-CNN's RPN. Instead of predicting box dimensions directly, each cell regresses logarithmic correction factors for the width and height of B anchor boxes. Box position is still represented as an offset from the cell's coordinates, however the predicted values are fed through sigmoid activations in order to constrain the value to $[0, 1]$. Deviating from the hand-picked anchors used in [Ren+15], YOLOv2 uses anchors generated by k-means clustering of training set bounding boxes. This improves quality of the anchor boxes, allowing to restrict bounding box regression to a small amount of anchors during inference and training. In addition, YOLOv2 introduces changes to underlying CNN, among them the replacement of dropout by batch normalization and the removal of fully-connected layers. The latter alteration enables YOLOv2 to operate at multiple input resolutions, which is also exploited during training. The resulting detector yields 76.8% mAP@0.5 on VOC 2007 at an input resolution of 416×416 at 67Hz (GPU) and 21.6% mAP@[.5, .95] on COCO. On a higher input resolution (544×544), YOLOv2 is able to achieve 78.6% mAP@0.5 on VOC 2007 while the framerate drops to 40Hz.

Other improvements on the YOLO paradigm include *YOLO9000* [RF16] and *YOLOv3* [RF18]. YOLO9000 extends YOLOv2 to over 9000 object classes by using a semantic tree hierarchy of labels. YOLOv3 expands YOLOv2 by introducing a larger network and feature extraction at several scales, following the concept of feature pyramid networks [Lin+16]. YOLOv3 achieves 33.0% mAP@[.5, .95] at an input resolution of 608×608 at 19Hz.

	R-CNN		Fast R-CNN	Faster R-CNN		YOLO		FastYOLO	YOLOv2	YOLOv3
	AlexNet	VGG 16	VGG 16	ZFNet	VGG 16	Untitled	VGG 16	Untitled	Darknet-19	Darknet-53
VOC ’07 mAP@0.5	58.5	66.0	66.9	59.9	69.9	63.4	66.4	52.7	76.8	N/A
VOC ’10 mAP@0.5	53.7	62.9	66.1	N/A	N/A	N/A	N/A	N/A	N/A	N/A
VOC ’12 mAP@0.5	53.3	62.4	65.7	N/A	67.0	57.9	N/A	N/A	73.4	N/A
COCO mAP@0.5	N/A	N/A	35.9	N/A	42.1	N/A	N/A	N/A	44.0	57.9
COCO mAP@[.5,.95]	N/A	N/A	19.7	N/A	21.5	N/A	N/A	N/A	21.6	33.0
s/image	$1.3 \cdot 10^1$	$4.7 \cdot 10^1$	$1.83 \cdot 10^0$	$5.9 \cdot 10^{-2}$	$1.98 \cdot 10^{-1}$	$2.22 \cdot 10^{-2}$	$4.76 \cdot 10^{-2}$	$6.45 \cdot 10^{-3}$	$1.49 \cdot 19^{-2}$	$5.1 \cdot 10^{-2}$

Table 2.1: Mean average performance (in percent) and runtime performance of various object detection architectures. All figures have been extracted from the respective original publications; “N/A”s indicate results not published by the authors.

This concludes our discussion of face detection. Notably, the leading entry on the WIDER FACE leaderboard [Tan+18] falls into the class of single-shot approaches. However, other entries, including region-based architectures, follow closely within a range of one percentage point of AP. Thus, a general rule regarding which paradigm is better suited for face detection cannot be inferred.

3 Face Detection with Capsule Networks

In this chapter, we present *YOLOphem*[1], our capsule-based architecture for face detection. YOLOphem follows the overall design of the YOLO object detector see 2.2.2), but replaces its core Convolutional Neural Network with a Capsule Network. All models were implemented in Python using TensorFlow [Mar+15] version 1.8. The source code is publicly available on GitHub[2].

YOLOphem's Capsule Network architecture is described in detail in section 3.2. Section 3.1 provides details on our implementation of capsules and their associated routing procedure. Lastly, section 3.3 discusses our adaption of the YOLO loss function for face detection.

3.1 Capsules and Routing

Our implementation of capsules and coincidence filtering follows the design published by Sabour *et al.* [SFH17][3]: A single capsule is represented as a state vector $\mathbf{s}$ in some higher-dimensional space. We interpret the direction of the vector as the generalized pose of the capsule and its length as the capsule's activation. Since capsule activations denote probabilities, the length of a pose vector is restricted to the unit interval by applying the "squash" non-linearity.

$$squash(\mathbf{s}) = \frac{||\mathbf{s}||^2}{1 + ||\mathbf{s}||^2} \frac{\mathbf{s}}{||\mathbf{s}||} \tag{3.1}$$

As described in section 2.1.3, each lower-level capsule i computes a prediction of the pose of each higher-level capsule j whose receptive field comprises capsule i (or all higher-level capsules in the case of a fully-connected capsule layer) by multiplying its state vector $\mathbf{s}_i$ with a learned transformation matrix $\mathbf{W}_{ij}$.

From the resulting set of prediction vectors $\hat{\mathbf{s}}_{j|i}$ the state vector of the higher-level capsule $\mathbf{s}_j$ is determined by an iterative routing procedure called *routing by agreement*

[1]in homage to the blinded cyclops Polyphemus from Homer's *Odyssey* [HM19], although we have not conducted any experiments as to whether our model could distinguish Odysseus from a sheep.

[2]`https://github.com/iamlucaswolf/Face-Detection-with-Capsule-Networks`

[3]Note that some of the mathematical formulations have been modified for clarity

(see algorithm 1). To this end, $\mathbf{s}_j$ is represented as a (squashed) linear combination of the predictions.

$$\mathbf{s}_j = squash\left(\sum_i c_{ij}\hat{\mathbf{s}}_{j|i}\right), \qquad \hat{\mathbf{s}}_{j|i} = \mathbf{W}_{ij}\mathbf{s}_i \tag{3.2}$$

The weights c_{ij} of the linear combination are referred to as *coupling coefficients* and are iteratively adjusted such that s_j converges towards the center of a subset of prediction vectors that agree strongly (i.e. that lie close to each other in all dimensions). A coupling coefficient can therefore be thought of as the "contribution" of the lower-level capsule's prediction to the pose of the higher-level capsule. Coupling coefficients are computed from a set of *coupling logits b_{ij}* by applying the softmax function to the logits belonging to the predictions of a lower-level capsule (see Equation 3.3). This constrains the contributions of a lower-level capsule to sum to one, thereby implicitly forcing the model to assign each lower-level part to a higher-level whole.

$$c_{ij} = \frac{\exp\left(b_{ij}\right)}{\sum_k \exp\left(b_{ik}\right)} \tag{3.3}$$

The coupling logits are initialzied to zero (which results in initially equal coupling coefficients) and updated by adding the scalar product between $\mathbf{s}_j$ and the prediction $\hat{\mathbf{s}}_{j|i}$ corresponding to a particular coefficient. Here, the scalar product measures the "agreement" between the two state vectors, as $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|\|\mathbf{b}\|\cos(\alpha)$, where α denotes the angle between the vectors $\mathbf{a}$ and $\mathbf{b}$. In the next iteration, $\mathbf{s}_j$ is recomputed from the adapted coefficients and the update procedure begins anew.

This process of iteratively updating coupling logits can be understood as reinforcing the contribution of those predictions that are similar to the current state vector of the higher-level capsule and weakening those that differ. If many lower-level capsules agree in their predictions, $\mathbf{s}_j$ should eventually converge towards their center, as these predictions dominate the initial input signal. In practice, two to three routing iterations are reportedly sufficient for convergence.

Note that Hinton *et al.* have since published a newer version of Capsule Networks that encode capsule activations explicitly via sigmoidal units and formulate routing in the expectation-maximization paradigm (see [HSF18]). We however did not implement these ideas as the paper is still undergoing peer review at the time of writing.

3.2 Network Architecture

The overall structure of YOLOphem's Capsule Network is similar to that of the CNN in YOLO: An input image is first re-scaled to a fixed input size of 448×448 pixels and

Algorithm 1: Routing by agreement

```
input  : Prediction vectors $\hat{\mathbf{s}}_{j|i}$ from layer $l$ capsules, number of routing iterations $r$
output: State vectors $\mathbf{s}_j$ of capsules in layer $(l+1)$
begin
    foreach capsule $i \in$ layer $l$ and capsule $j \in$ layer $(l+1)$ do
        initialize $b_{ij} \leftarrow 0$
    end
    for $r$ iterations do
        foreach capsule $i \in$ layer $l$ and capsule $j \in$ layer $(l+1)$ do
            $c_{ij} \leftarrow \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$
        end
        foreach capsule $j \in$ layer $(l+1)$ do
            $\mathbf{s}_j \leftarrow squash\Big(\sum_i c_{ij}\hat{\mathbf{s}}_{j|i}\Big)$
        end
        foreach capsule $i \in$ layer $l$ and capsule $j \in$ layer $(l+1)$ do
            $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{s}}_{j|i} \cdot \mathbf{s}_j$
        end
    end
    return $\mathbf{s}_j$
end
```

fed through two initial convolutional layers with ReLU activation functions in order to "de-render" the image from pixel intensities to some feature space. The output of the last convolutional layer is then transformed into a grid of primary capsules by grouping activations at the same location across filter maps and applying the *squash* function. Subsequent convolutional capsule layers yield a final set of B filters containing $S \times S$ output capsules that (hopefully) represent detected faces in the scene.

Converting the poses and activations of the output capsules into bounding box predictions and confidence scores requires introducing some minor adjustments to the original YOLO architecture. To evaluate the necessary amount of adaptation, we propose two variants of our design, *YOLOphem A* and *B*. The output of both models are a $S \times S \times B \times 5$ tensor, where S denotes the resolution of the grid imposed on the input image and B denotes the number of predictors for each grid cell. A capsule in the final layer thus corresponds directly to a cell and predictor in the output tensor. Following the original YOLO architecture, we choose $S = 7$ and $B = 3$ for both of our models. We however do not predict a set of class probabilities for each cell as a face detection architecture does not need to discriminate between various types of objects. Instead, we interpret the "objectness" score of each cell and predictor as the confidence for the corresponding bounding box.

Both YOLOphem variants extract the output tensor by utilizing the inherent properties of capsules to a different degree. In the following, we describe these mechanisms for the two models. Specifics, such as the precise dimensioning of layers or the total number of trainable parameters are listed in table Table 3.1.

YOLOphem A In YOLOphem A, we aim to closely reproduce the approach of YOLO, which computes the output tensor from a feature vector extracted from the output layer's filter maps. We therefore concatenate the state vectors of the last capsule layer and feed the resulting vector into a fully-connected layer with ReLU non-linearity. The resulting 512-dimensional output serves as our feature vector, which we use as input for a second fully-connected layer (without any subsequent activation function). The eventual prediction tensor is obtained by reshaping the output of this final layer accordingly.

YOLOphem B YOLOphem A makes little use of the properties exhibit by capsules, as it discards any structure in the network's output. Furthermore, it ignores capsule activations (i.e. the lengths of their state vectors) entirely. In YOLOphem B, we attempt to rectify these shortcomings by computing each of the $S \times S \times B$ bounding boxes and confidence scores from exactly one of the $S \times S \times B$ output capsules. To this end, we use the state vector of each capsule separately as input to a fully-connected layer (without activation function) with four output units and interpret the resulting vector as the predicted bounding box for the corresponding grid cell and predictor. By using merely a linear transformation to recover box coordinates and dimensions, we hope to implicitly bias the model to encode sufficient information in the capsule's pose vector. In a similar vain, we treat the capsule's activation as the confidence score associated with the respective bounding box, thereby "encouraging" the capsules in the final layer to detect faces in the scene.

	Type	Input Size	Dimension	#Params
1	Conv	$448 \times 448 \times 3$	$7 \times 7 \times 256$, s-2	38,400
2	PrimaryCaps	$221 \times 221 \times 256$	$5 \times 5 \times (16 \times 8)$, s-2	851,968
3	ConvCaps	$109 \times 109 \times (16 \times 8)$	$5 \times 5 \times (16 \times 8)$, s-2	409,600
4	ConvCaps	$53 \times 53 \times (16 \times 8)$	$5 \times 5 \times (8 \times 12)$, s-2	307,200
5	ConvCaps	$25 \times 25 \times (8 \times 12)$	$5 \times 5 \times (8 \times 12)$, s-2	230,400
6	ConvCaps	$11 \times 11 \times (8 \times 12)$	$3 \times 3 \times (3 \times 16)$, s-1	41,472
7	ConvCaps	$9 \times 9 \times (3 \times 16)$	$3 \times 3 \times (3 \times 16)$, s-1	20,736
		YOLOphem A		
8	Dense	$7 \times 7 \times (3 \times 16)$	512	1,204,736
9	Dense	512	$7 \times 7 \times 3 \times 5$	377,055
				$= 3,481,567$
		YOLOphem B		
8	Dense	16	5	85
				$= 1,899,861$

Table 3.1: A detailed specification of the YOLOphem architecture(s). The "Dimension" column specifies the configuration of the respective layers in the format "*kernel size* $\times$ *kernel size* $\times$ *output filters*, s-*stride*" for convolutional layers ("Conv") and "*kernel size* $\times$ *kernel size* $\times$ (*output filters* $\times$ *capsule dimensions*) , s-*stride*" for convolutional capsule layers ("ConvCaps"). For fully-connected layers ("Dense"), the "Dimension" column specifies the size of the output vector. Note that for layer 8 in YOLOphem 8, each of the $7 \times 7 \times 3$ state vectors computed in layer 7 serve as a separate input. In practice this is implemented as a convolutional layer with a 1×1 kernel.

3.3 Loss Function

Both variants of the YOLOphem architecture are trained via an adapted version of YOLO's sum-of-squares loss computed over the predicted bounding box $\left(\hat{x}_{ij}, \hat{y}_{ij}, \hat{w}_{ij}, \hat{h}_{ij}\right)$ and confidence scores C_{ij} for each grid cell i and predictor j (see Equation 3.4).

$$\begin{aligned} L = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \delta_{ij}^{obj} \left[\left(x_{ij} - \hat{x}_{ij}\right)^2 + \left(y_{ij} - \hat{y}_{ij}\right)^2 + \left(w_{ij} - \hat{w}_{ij}\right)^2 + \left(h_{ij} - \hat{h}_{ij}\right)^2 \right] \\ + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \left[\delta_{ij}^{obj} \left(C_{ij} - \mathrm{IoU}_{ij}\right)^2 + \lambda_{noobj} \left(1 - \delta_{ij}^{obj}\right) C_{ij}^2 \right] \end{aligned} \tag{3.4}$$

Here, $\hat{x}$ and $\hat{y}$ denote the center coordinates of the particular box relative to its containing grid cell whereas $\hat{w}$ and $\hat{h}$ encode its width and height relative to the input image. All predicted quantities should therefore lie within the unit interval. Note that rather than regressing on w and h directly, the error function instead considers deviations between the square roots of this quantity in order to prevent large bounding boxes from dominating the loss. Further, a predicted bounding box contributes to the loss if and only if its predictor is "responsible" for detecting a face within the respective grid cell. To this end δ_{ij}^{obj} denotes the indicator function which assumes a value of one if predictor j in cell i is responsible and zero otherwise. Recall that, if a predictor cell is responsible, its confidence output should equal the current intersection-over-union between the predicted and ground truth bounding boxes, here denoted as IoU_{ij}. Following [Red+15] we introduce two additional hyper-parameters, λ_{coord} and λ_{noobj} to balance the contributions of box predictions and confidence predictions to the objective function.

4 Experiments

We trained both YOLOphem models on WIDER Face in order to determine their practicability; this chapter documents these experiments. Section 4.1 describes the settings we used during training. Results of the training procedure are given in section 4.2 and interpreted in section 4.3.

4.1 Setup

We originally intended to train and evaluate both YOLOphem variants on the WIDER FACE dataset (see section 2.2.1). However, we saw ourselves unable to do so, as training either model exceeded our computational resources (two NVIDIA GTX 1080 GPUs with 8GB of graphics memory per card), due to the high memory requirements of capsule layers. A thorough analysis of this issue is given in section 4.3. To nevertheless test our architecture, we reduced the size of both YOLOphem variants by removing layers 3 and 4 (see Table 3.1) and readjust the size of an input image to 109×109 pixels.

The resulting smaller models are trained on the WIDER FACE training partition for 50 epochs each, using a batch size of 16 samples per iteration. To this end, input images and bounding box annotations are warped linearly to the required input resolution. Additionally, pixel intensities are scaled to the unit interval to increase model stability during both training and inference. No further pre-processing or data augmentation is performed. Following [Red+15], we choose $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$ to mediate between detection and localization errors. The resulting loss function is optimized using the Adam algorithm [KB14] and an open-source, memory-saving implementation of backpropagation[1].

To determine optimal hyperparameters for the training procedure, we let both models overfit on small samples of training images and monitor convergence speed. Results suggested a learning rate of 10^{-3} at an exponential decay rates of $\beta_1 = 0.9$ and $\beta_2 = 0.999$ for Adam's first and second moment estimates, respectively. We optimize both models with these settings for 30 epochs and then reduce the learning rate to 10^{-4} for the remaining 20 epochs in order to potentially fine-tune the model.

[1]`https://github.com/openai/gradient-checkpointing`

4.2 Results

To monitor progress, we track the average loss on both the validation and the training partition during learning. The resulting loss curves are depicted in Figure 4.1. The validation loss is computed after each completed epoch of training. For the training set, we retain the momentary loss of each batch during the training pass and average the resulting records afterwards. While this approach is not fully sound from a mathematical standpoint (learning steps on earlier batches affect the loss values of later batches), it is less time-consuming than re-evaluating the objective function on the entire training set after the learning phase. Since a full epoch on the training and validation sets takes around one hour on our machine, we deem the resulting discrepancy as tolerable.

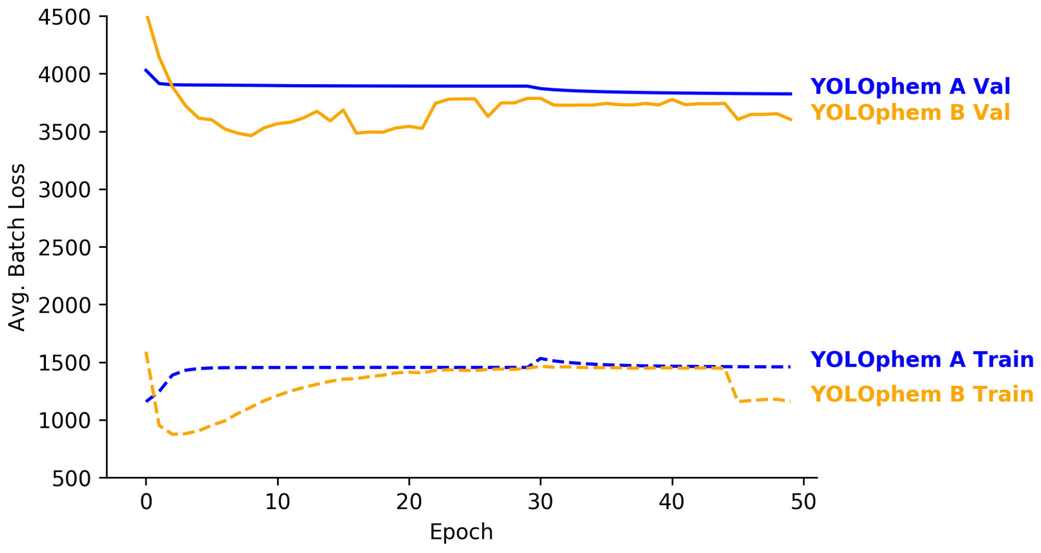

Figure 4.1: Development of the averaged batch loss during training. Model variants are discerned by color (blue = YOLOphem A, orange = YOLOphem B), while line styles denote the dataset partition on which the loss was measured (solid = validation, dashed = train).

As can be seen in Figure 4.1, both models show convergent behaviour after a comparatively small number of epochs. Note that, while the plot suggests a low initial error on the training set, the actual train loss started at a higher level and decreased rapidly during the first few iterations of the first epoch. This effect is however not visible in the graph, as it is absorbed by our approach of averaging batch losses to determine epoch loss. Both YOLOphem A and B yield similar loss values of approximately 1,500 for training and 4,000 for validation, although model B scores marginally better than model A.

Overall, the final errors of both models are comparatively high, as a validation loss of 4,000 relates to an average squared-loss contribution of $4,000 \div (7 \times 7 \times 3) \approx 27$ per cell. Further, manual inspection of predictions on the validation and test partitions showed that both models output very similar bounding boxes and confidence scores for widely varying inputs. Since the architecture was able to overfit on a small set of samples in prior experiments, we rule out the possibility of a programming error and interpret these observations as inability of the model to learn the Face Detection task (a detailed discussion of this assessment is given in the following section). We therefore refrain from evaluating our results in terms of precision-recall curves as they do not contribute relevant further information.

4.3 Analysis

As described in the previous section, our model was not able to generalize to the task of Face Detection. While reasons for such a behaviour can be manifold (e.g. inadequate hyperparameter settings, an inadequate loss formulation or issues with gradient stability), we are fairly certain that the bad performance of YOLOphem must be attributed to the low input resolution of 109×109 pixel. This makes it practically impossible for the model to detect small faces in images captured at a higher original resolution. For comparison, YOLO and Faster R-CNN work on input images of at least four times this size.

Recall that the dimensioning of the input was not a result of choice but of necessity, as the training of larger models would have exceeded the 16 gigabytes of GPU memory available at our system (see section 4.1). We however find this circumstance intriguing as our models contain only a few million parameters each, whereas much larger CNN architectures have been trained with less video memory. For example, AlexNet [KSH12] has reportedly been trained using a third of this amount of memory while its parameter count exceeds that of YOLOphem A by a factor of 19.

Upon further investigation, we found that the high memory demand for training YOLOphem originates from two suboptimal characteristics of Capsule Networks:

(1) Constancy of tensor size For both convolutional layers and convolutional capsule layers, the spatial extent of the output tensor equals roughly that of the input divided by the layer's stride. This is not particularly problematic in CNNs, as intermediate tensors can always be down-sampled by pooling layers without paying the cost of introducing new parameters to the model. In Capsule Networks however, this leads to a certain duality of network depth and layer size since the introduction of additional convolutional capsule layers is the only mean by which tensor sizes can be reduced. In fact, the design of YOLOphem's Capsule Network was largely guided by the constraint of starting at an input size of 448 pixel per side (for comparability to YOLO) and eventually arriving at output filter maps of size 7×7.

(2) Memory requirements of vectorized capsule routing The training of deep CNNs almost invariably requires the parallel computing capabilities of modern GPUs in order to keep training times reasonable. As Capsule Networks perform considerably more complex computations than CNNs, we follow a similar approach and implement the convolutional capsule layers in YOLOphem in a vectorized manner. To this end, we exploit the dynamics of the matrix-vector product in TensorFlow which interprets higher-order input tensors as multi-dimensional arrays of matrices and vectors and performs the matrix-vector product in an element-wise fashion. Using this property, we compute a $batch\_size \times H_{out} \times W_{out} \times C_{out} \times K \cdot K \cdot C_{in} \times D_{out}$ tensor of prediction vectors by multiplying a $batch\_size \times H_{out} \times W_{out} \times C_{out} \times K \cdot K \cdot C_{in} \times D_{out} \times D_{in}$ tensor of transformation matrices with a $batch\_size \times H_{out} \times W_{out} \times C_{out} \times K \cdot K \cdot C_{in} \times D_{in}$ tensor of input state vectors (where H_{out}, W_{out} denote the spatial extent of an output filter, C_{in}, C_{out} denote the number of input and output filters, D_{in}, D_{out} denote the dimensionality of the input and output capsules and K denotes the kernel size). While each computed prediction is strictly necessary for routing, this results in vast intermediary tensors that depend on the size of the output layer (eg. ≈ 4.6 gigabytes for the tensor of transformation matrices from layer 3 to layer 4 for one input sample). This is aggravated by the fact that gradients have to be computed for each transformation matrix during the training. Reverting to an iterative implementation instead of a vectorized algorithm is equally unfeasible, as network evaluation already requires a large amount of time.

In conjunction, these issues make it impossible to process high-resolution input images with current Capsule Network architectures on contemporary commodity hardware. This, in return, renders Face Detection with a capsule-based variant of the YOLO object detector unfeasible.

5 Conclusion

In this thesis, we applied Capsule Networks to the task of Face Detection. To this end, we devised a custom architecture (YOLOphem) by adapting a state-of-the-art object detector (YOLO) to the domain of capsules. We implemented and trained two variants of YOLOphem and discussed the behaviour of the resulting models during training. Further, we provided an in-depth discussion of current approaches to Computer Vision and Face Detection, as well as the theoretical underpinnings of Capsule Networks.

Unfortunately, the problems outlined in section 4.3 make it difficult to answer our initial research questions (see section 1.2) decisively. We nevertheless align our results with them in order to put our findings into perspective:

1. **Are Capsule Networks suitable as a drop-in replacement for Convolutional Neural Networks in existing architectures?**
   While Capsule Networks should – from a theoretical standpoint – be able to replicate the functionality of Convolutional Neural Networks in larger architectures, the resulting computational implications limit their usefulness in practice (see item 2.). Apart from this, existing architectures may require minor modifications to account for the inherent properties of capsules. This might include adapting the loss function to encourage capsules to encode the presence of their encoded entities as capsule activations or introducing additional "decoder"-layers to extract entity-related information from a capsule's pose vector.

2. **Can larger Capsule Networks be trained efficiently?**
   No, since the memory demand of the (vectorized) pose prediction and routing process renders the training of Capsule Networks with many layers or large input resolutions on commodity hardware unfeasible. Note that this issue should also be present in a newer version of Capsule Networks recently proposed by Hinton *et al.*, although we have not conducted any experiments in that regard. We present ideas on how this issue might be alleviated in chapter 6.

3. **Are Capsule Networks a suitable model (component) for the detection task?**
   We cannot give a precise answer to this question, as the insufficiency of our model has to be attributed primarily to its poor input resolution and only indirectly to the nature of capsules.

6 Future Work

We want to conclude this thesis by providing a brief list of suggestions for potential future research regarding Capsule Networks:

The space of Capsule Networks is still mostly unexplored; in particular their embedding into larger architectures and the adaption of loss functions remain open issues without established rules or guiding heuristics. Judging from our experience with deep neural networks, we also suppose that the gradient behaviour of capsules during backpropagation could be worth investigating.

With respect to this work, we would find it interesting to test whether the YOLOphem could be trained (in full) using more powerful hardware. We hope that such an experiment would provide insight into the suitability of Capsule Networks for detection tasks and the training of deeper Capsule Network architectures.

Simply training on larger machines does of course not solve the computational issues that prevent Capsule Networks with high-resolution inputs to be trained on commodity hardware. A potential solution to this problem might be to prune inactive capsules by computing predictions only from capsules whose activations exceed a certain threshold. Another idea would be to once more draw inspiration from the brain: as pointed out in the outset of this thesis, the brain assembles a "mental picture" of the scene through a series of coordinated eye movements. For each of the resulting *fixation points*, only a small part at the center of the visual field is processed at a high resolution. Following this observation, it may be sufficient to let a Capsule Network with a small input size process individual regions of an image and combine the results in a later step. Perhaps it is such a second-order routing algorithm, which "tells the Capsule Network where to look" that is missing from the equation.

In summary, one can say that while Capsule Networks exhibit favourable theoretic properties, their computational requirements impede application in practice. Whether this issue can be resolved via algorithmic improvements remains an open question.

List of Figures

List of Tables

Bibliography

[ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein GAN." In: *CoRR* abs/1701.07875 (2017).

[AHS88] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. "Connectionist Models and Their Implications: Readings from Cognitive Science." In: ed. by D. Waltz and J. A. Feldman. Norwood, NJ, USA: Ablex Publishing Corp., 1988. Chap. A Learning Algorithm for Boltzmann Machines, pp. 285–307. ISBN: 0-89391-456-8.

[BSM17] D. Berthelot, T. Schumm, and L. Metz. "BEGAN: Boundary Equilibrium Generative Adversarial Networks." In: *CoRR* abs/1703.10717 (2017).

[Can86] J. Canny. "A Computational Approach to Edge Detection." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (June 1986), pp. 679–698. ISSN: 0162-8828. DOI: `10.1109/TPAMI.1986.4767851`.

[CUH15] D. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)." In: *CoRR* abs/1511.07289 (2015).

[DHS10] J. Duchi, E. Hazan, and Y. Singer. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*. Tech. rep. UCB/EECS-2010-24. EECS Department, University of California, Berkeley, Mar. 2010.

[DT05] N. Dalal and B. Triggs. "Histograms of Oriented Gradients for Human Detection." In: *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*. CVPR '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 886–893. ISBN: 0-7695-2372-2. DOI: `10.1109/CVPR.2005.177`.

[Eve+10] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. "The Pascal Visual Object Classes (VOC) Challenge." In: *Int. J. Comput. Vision* 88.2 (June 2010), pp. 303–338. ISSN: 0920-5691. DOI: `10.1007/s11263-009-0275-4`.

[FMR08] P. Felzenszwalb, D. Mcallester, and D. Ramanan. "A Discriminatively Trained, Multiscale, Deformable Part Model." In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on* (June 2008).

[Fuk80] K. Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.” In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. ISSN: 1432-0770. DOI: `10.1007/BF00344251`.

[Fur+14] S. Furber, F. Galluppi, S. Temple, and L. A. Plana. “The SpiNNaker project.” In: 102 (May 2014), pp. 652–665.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[Gir+13] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. “Rich feature hierarchies for accurate object detection and semantic segmentation.” In: *CoRR* abs/1311.2524 (2013).

[Gir15] R. B. Girshick. “Fast R-CNN.” In: *CoRR* abs/1504.08083 (2015).

[GK02] W. Gerstner and W. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002. ISBN: 9780521890793.

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative Adversarial Nets.” In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680.

[Gul+17] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. “Improved Training of Wasserstein GANs.” In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Curran Associates, Inc., 2017, pp. 5767–5777.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun. “Deep Residual Learning for Image Recognition.” In: *CoRR* abs/1512.03385 (2015).

[He+17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. “Mask R-CNN.” In: *CoRR* abs/1703.06870 (2017).

[Hin14] G. E. Hinton. *What is wrong with Convolutional Neural Nets?* Youtube. 2014. URL: `https://www.youtube.com/watch?v=rTawFwUvnLE`.

[HM19] Homer. and A. T. Murray. *The Odyssey / Homer ; with an English translation by A.T. Murray*. Greek, Modern (1453-). Heinemann London, 1919.

[HSF18] G. Hinton, S. Sabour, and N. Frosst. “Matrix capsules with EM routing.” In: 2018.

[IS15] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *CoRR* abs/1502.03167 (2015).

[Iso+16] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks." In: *arxiv* (2016).

[JL10] V. Jain and E. Learned-Miller. *FDDB: A Benchmark for Face Detection in Unconstrained Settings*. Tech. rep. UM-CS-2010-009. University of Massachusetts, Amherst, 2010.

[KB14] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980 (2014).

[Kla+15] B. Klare, B. Klein, E. Taborsky, A. Blanton, J. Cheney, K. Allen, P. Grother, A. Mah, and A. K. Jain. *Pushing the Frontiers of Unconstrained Face Detection and Recognition: IARPA Janus Benchmark A*. Tech. rep. June 2015.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105.

[KW13] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes*. cite arxiv:1312.6114. 2013.

[LeC+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition." In: *Neural Comput.* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: `10.1162/neco.1989.1.4.541`.

[Lin+14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and L. Zitnick. "Microsoft COCO: Common Objects in Context." In: *ECCV*. European Conference on Computer Vision, Sept. 2014.

[Lin+16] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. "Feature Pyramid Networks for Object Detection." In: *CoRR* abs/1612.03144 (2016).

[Lin+17] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. "Focal Loss for Dense Object Detection." In: *CoRR* abs/1708.02002 (2017).

[Liu+16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. "SSD: Single Shot MultiBox Detector." In: To appear. 2016.

[Liu+18] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. "Path Aggregation Network for Instance Segmentation." In: *CoRR* abs/1803.01534 (2018).

[LSD14] J. Long, E. Shelhamer, and T. Darrell. "Fully Convolutional Networks for Semantic Segmentation." In: *CoRR* abs/1411.4038 (2014).

[LWT12] P. Luo, X. Wang, and X. Tang. "Hierarchical face parsing via deep learning." In: *CVPR*. IEEE Computer Society, 2012, pp. 2480–2487.

[Mar+15] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[NH10] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines." In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7.

[OKK16] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. "Pixel Recurrent Neural Networks." In: *CoRR* abs/1601.06759 (2016).

[Oor+16] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders." In: *CoRR* abs/1606.05328 (2016).

[Pap66] S. Papert. *The Summer Vision Project*. AI memo. Massachusetts Institute of Technology, Project MAC, 1966.

[Ram12] D. Ramanan. "Face Detection, Pose Estimation, and Landmark Localization in the Wild." In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 2879–2886. ISBN: 978-1-4673-1226-4.

[Red+15] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi. "You Only Look Once: Unified, Real-Time Object Detection." In: *CoRR* abs/1506.02640 (2015).

[Rei+17] V. M. Reid, K. Dunn, R. J. Young, J. Amu, T. Donovan, and N. Reissland. "The Human Fetus Preferentially Engages with Face-like Visual Stimuli." In: *Current Biology* 27.12 (2017), 1825–1828.e3. ISSN: 0960-9822. DOI: `https://doi.org/10.1016/j.cub.2017.05.044`.

[Ren+15] S. Ren, K. He, R. Girshick, and J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett. Curran Associates, Inc., 2015, pp. 91–99.

[RF16] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger." In: *CoRR* abs/1612.08242 (2016).

[RF18] J. Redmon and A. Farhadi. "YOLOv3: An Incremental Improvement." In: *arXiv* (2018).

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. "Neurocomputing: Foundations of Research." In: ed. by J. A. Anderson and E. Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[Rus+15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: `10.1007/s11263-015-0816-y`.

[Sch+10] J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. "A wafer-scale neuromorphic hardware system for large-scale neural modeling." In: *International Symposium on Circuits and Systems (ISCAS 2010), May 30 - June 2, 2010, Paris, France*. 2010, pp. 1947–1950. DOI: `10.1109/ISCAS.2010.5536970`.

[Sen+18] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy. "Going Deeper in Spiking Neural Networks: VGG and Residual Architectures." In: *CoRR* abs/1802.02627 (2018).

[SFH17] S. Sabour, N. Frosst, and G. E. Hinton. "Dynamic Routing Between Capsules." In: *CoRR* abs/1710.09829 (2017).

[SKP15] F. Schroff, D. Kalenichenko, and J. Philbin. "FaceNet: A Unified Embedding for Face Recognition and Clustering." In: *CoRR* abs/1503.03832 (2015). arXiv: `1503.03832`.

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[Sut+13] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the Importance of Initialization and Momentum in Deep Learning." In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, 2013, pp. III-1139–III-1147.

[SZ14] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2014).

[Sze+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions." In: *Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. 1st. Berlin, Heidelberg: Springer-Verlag, 2010. isbn: 1848829345, 9781848829343.

[Tan+18] X. Tang, D. K. Du, Z. He, and J. Liu. "PyramidBox: A Context-assisted Single Shot Face Detector." In: *CoRR* abs/1803.07737 (2018).

[Uij+13] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. "Selective Search for Object Recognition." In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171.

[VJ01] P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features*. 2001.

[Yan+15a] B. Yang, J. Yan, Z. Lei, and S. Z. Li. "Fine-grained Evaluation on Face Detection in the Wild." In: *Automatic Face and Gesture Recognition (FG), 11th IEEE International Conference on*. IEEE. 2015.

[Yan+15b] S. Yang, P. Luo, C. C. Loy, and X. Tang. "From Facial Parts Responses to Face Detection: A Deep Learning Approach." In: *ICCV*. IEEE Computer Society, 2015, pp. 3676–3684. isbn: 978-1-4673-8391-2.

[Yan+16] S. Yang, P. Luo, C. C. Loy, and X. Tang. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Zha+14] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. "Learning and Transferring Multitask Deep Representation for Face Alignment." In: *CoRR* abs/1408.3967 (2014). arXiv: `1408.3967`.