Contrastive Capsule Networks

Dario E. Shehni Abbaszadeh

Layout: typeset by the author using LaTeX.
Cover illustration: Unknown artist

Contrastive Capsule Networks

Dario E. Shehni Abbaszadeh
10791655

Bachelor thesis
Credits: 18 EC

Bachelor *Kunstmatige Intelligentie*

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisor
I. Sosnovik MSc

UvA-Bosch
Delta Lab
University of Amsterdam
Science Park 904, Room C3.201
1098 XH Amsterdam

25 Jun, 2021

Abstract

Convolutional neural networks (CNNs) demonstrate state-of-the-art results in many image and video analysis tasks. They gradually transform raw pixels into a meaningful representation layer by layer. While CNNs process all images in the same way, Capsule Networks use an adaptive algorithm. Raw pixels are first transformed into capsules that encode different features, their poses, and probabilities. Capsules that find an agreement are then transformed into the output of the network. Such an approach demonstrates significant improvement in tasks where the model needs to adapt to significant pose variations of the object, such as rotations or scale changes. In this project, we aim to demonstrate that by using capsule networks it is possible to build an extractor of the pose from an image without any supervision.

Contents

1 Introduction

Artificial intelligence, or AI, can be defined as the pursuit of using machines to create intelligence. However, in practice this is not what most AI research is focused on. Current day AI is used to perform much more specific tasks ranging from image detection, to finding trends in data. One method of achieving this is by providing large amounts of data and letting the machine learn for itself. This is called machine learning, an umbrella term for various techniques that allow computers to solve problems using data. Machine learning is applied across various industries for different tasks. Common problems where machine learning is applied include fraud detection, image recognition, spam detection and product recommendations.

Another prominent field in AI is computer vision. Unlike machine learning, computer vision is not defined by the methods used, but by the problems that it tries to solve. Computer vision deals with digital images and videos. This often involves the use of machine learning. Such as using data to train neural networks for image recognition. A prominent example of computer vision is computational photography. This involves the use of digital computation instead of optical processes to process pohotographs. This allows smaller devices such as smartphones to have improved camera capabilities despite hardware limitations.

The problems that AI tries to solve tend to be problems that the human brain is already capable of solving. Therefore, when attempting to find solutions to these problems it can be valuable to understand how the human brain approaches these problems. The brain consists of billions of interconnected neurons. Each neuron is a cell that receives, processes and transmits information. Each neuron is connected to other neurons. Through these connections a neuron outputs an electrical signal. If the strength of this input signal reaches a certain threshold the neuron is 'activated' and sends out an output signal (Abraham, 2005).

Artificial neural networks are modeled after this same architecture. They consist of layers of neurons. The neurons in a layer receive various inputs from the previous layer. The signal sent by a neuron is the result of the weighted sum of these inputs. The network improves its performance by adjusting these weights. The input is received in the input layer and moves through the network in one direction. The output layer then outputs a final result. Based on this output the weights of the model are adjusted using the backpropagation algorithm. For backpropagation to work it is necessary to have a numerical representation of this accuracy. This is where loss functions are used. A loss function takes the target output and the final output of the model. It then gives a value representing the discrepancy between the two. Backpropagation works by taking the partial derivative of the loss, with respect to each weight. This partial derivative is then used to increase or decrease the respective weight accordingly (Abraham, 2005).

Traditional neural networks are built around the neuron, which produces an output based on the weighted sum of all neurons in the previous layers. However, this is not suitable for all tasks. Image processing tasks tend to favor convolutional neural networks (CNNs). Unlike fully connected layers, convolutional layers do not receive the input from all neurons in the previous layer. In stead, they rely on convolutions taking the weighted sum of a local area. This reduces computation time and leverages the fact that pixels that are close to each other are more likely to be related, or to be part of the same structure. This is why CNNs are favored in computer vision tasks, as images tend to be 2 dimensional and therefore require more information to be processed. However, although CNNs have been successful in image recognition they also have limitations. One such limitation is the difficulty in understanding the spatial relationship between these features. Max-pooling is a technique which attempts to resolve this by downsampling the input, but it provides only limited spatial invariance as the pooling mechanism only samples pre-defined areas (Jaderberg et al., 2015). CNNs are therefore limited in their ability to recognize an object when the spatial relationships between its features change.

Sabour et al. (2017) has demonstrated that capsule networks can achieve higher classification accuracy than conventional convolutional models on the MNIST dataset of handwritten digits. They also propose a technique called dynamic routing that can significantly improve these results. Capsule networks struggles on more complex data that might be found on datasets such as CIFAR-10. This is because of the higher volume of information which increases the complexitiy of the problem. Capsule nets are still in a research and development phase and not reliable when compared to more traditional methods such as CNNs. However, the concept is promising and further research could lead to capsule networks being more commonly used in image recognition tasks

Current experiments focus on reconstructing the affine parameters of transformations applied to individual objects. While promising, this is not sufficient for dealing with complex scenes where parts of the scene can have inconsistent spatial positions relative to the canonical image. This study will evaluate whether a capsule network trained to estimate the parameters of an affine transformation of a small patch, can be applied patch-wise to a complex scene to estimate its geometry.

2 Neural Networks

Capsule networks are a type of neural network. In order to understand capsule networks it is neccessary to have have a general understanding of how neural networks function, and what their building blocks are. The following section will explain the following types of network architectures; fully connected neural networks, convolutional neural networks and autoencoders. We will also discuss the process through which these networks improve themselves and how their components work.

2.1 Types of Neural Networks

In fully connected neural networks a neuron receives input from all neurons in the previous layer, and sends its ouput to all neurons in the next layer (Figure 1). Due to the high number of connections this architecture is computationally intense and prone to overfitting. An Fully connected architecture makes no assumptions about the structure of the data. This means that they can be used for a broad range of applications, but are typically outperformed by more specialized architectures. Fully connected networks are typically not used for image processing.

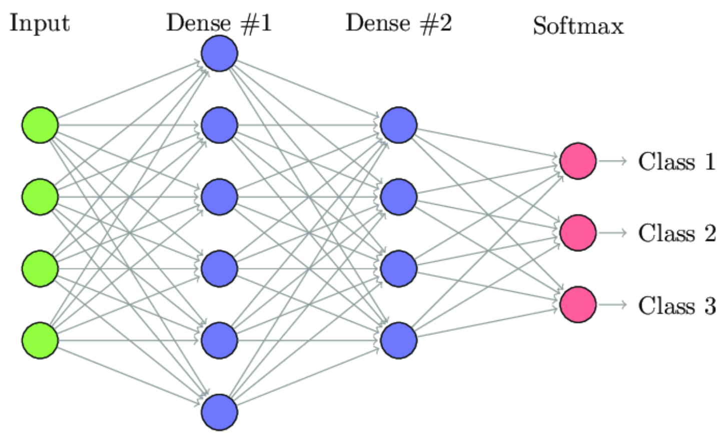


Figure 1: A fully connected neural network (Pelletier et al., 2019).

An architecture more suited for computer vision tasks is the convolutional neural network (CNN). CNNs are based on the assumption that the input is an image. Pixels in an image that are in close proximity to each other are more likely to be part of the same entity. Pixels on opposite sides of the image are less likely to be related. CNNs take advantage of this property of images by connecting each neuron only to a local region of the input. This local region is the recepetive field of the neuron. The receptive field is defined by the kernel (also called a filter) that is used to sample the input, as can be seen in figure 2.

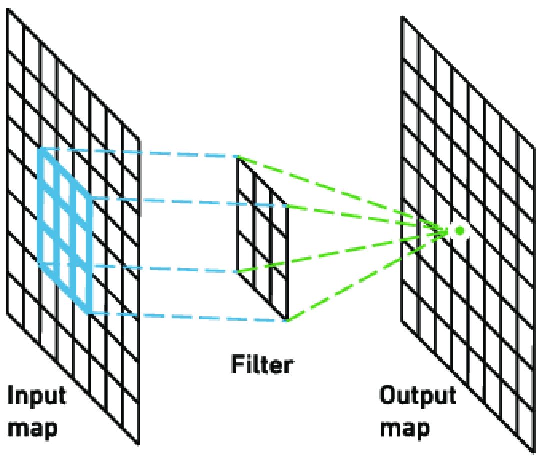


Figure 2: A representation of the kernel being applied to the input in a convolutional layer (Yakura et al., 2018).

Another difference between CNNs and regular neural networks is that the shape of the input of a CNN has three dimensions. Each pixel has a position along the width and height of the input, and also a value. The dimensionality of this value is the depth of the image. A kernel has a limited receptive field with respects to the width and the height. But will always extend through the entire depth of the input. The depth of the input is also called the number of channels. The kernel is applied to all channels and extracts a feature over the input, producing a 2-dimensional output map (Figure 3). As each kernel extracts a given feature, multiple kernels can be used to extract multiple features. These 2-dimensional outputs are then stacked, resulting in an output that has not only a width and height, but also a depth based on the number of kernels. A convolutional layer can have multiple channels where each channel can have its own kernel. Each kernel is trained to detect a certain feature and produces a 2-dimensional output map. These outputs are then stacked. The depth of the output is defined by the number of kernels that is used.

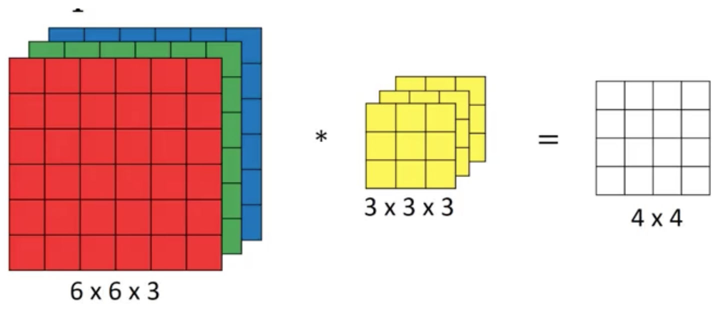


Figure 3: A kernel is applied to all channels and outputs a 2-dimensional map (Andrew Ng, 2020). The depth of the output is defined by the number of kernels that is used.

Another type of neural network architecture is an autoencoder. Autoencoders are trained to compress (encode) and decompress (decode) data. The encoder compresses the data to a latent space. This a lower dimensional representation of the input. The decoder then reconstructs the input from this latent space (Figure 4). Since much of the data is lost during this compression, the encoder is forced to learn how to select the features that contain the most information.

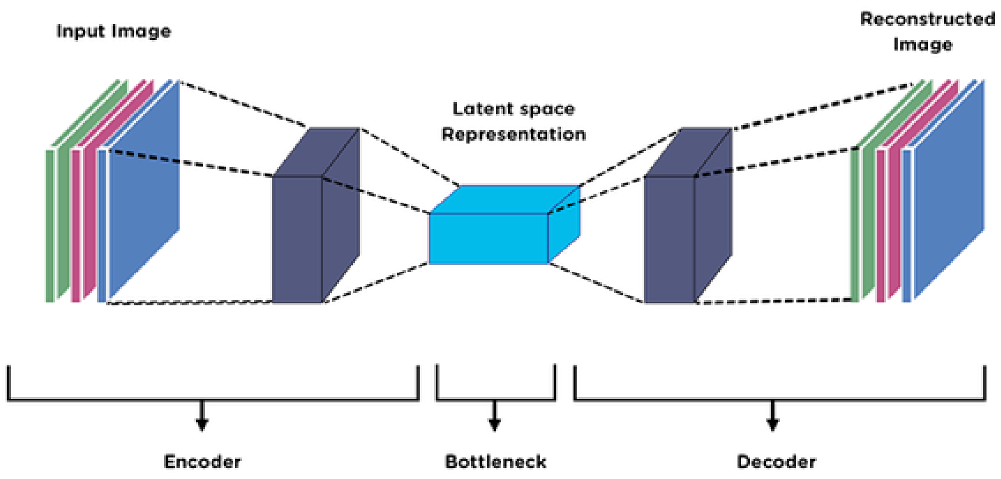


Figure 4: An autoencoder architecture using convolutional layers (Birla, 2019).

2.2 Building Blocks

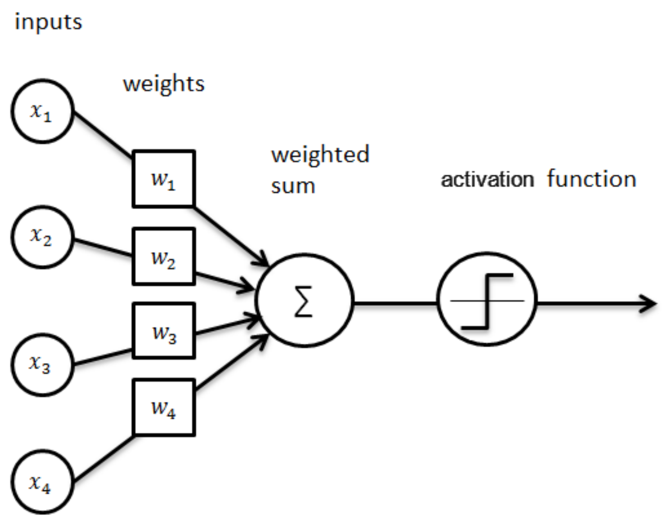


Figure 5: A single neuron in a neural net (Ognjanovski, 2019).

A neuron is the basic computational unit of a neural network. A neural net consists of a large number of processing neurons that can be fully or partially interconnected. They are organized into layers of neurons. The data then moves through the network in one direction. An individual neuron is connected to several neurons in the previous layer, from which it receives data, and several neurons in the following layer, to which it sends data (Figure 5). Each of these incoming inputs is multiplied by a weight, allowing the network to prioritize certain connections over others (Equation 1). The weights decide how strong these connections are.

$$z = b + \sum_i W_i x_i \tag{1}$$

CNNs have a specific structure called a convolutional layer. As shown in figure 2, convolutional layers perform a convolution on the input using an array of learnable weights called a kernel. The kernel is applied systematically across the input, and transforms each patch into a scalar value. Similar to fully connected layers, the result is a weighted sum over the input as seen in figure 6.

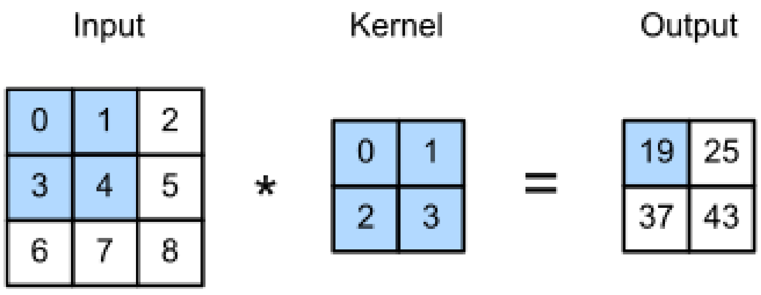


Figure 6: A kernel taking the weighted sum over a local patch of the input (Zhang et al., 2020).

This weighted sum is then transformed by a nonlinear activation function. An activation function in a neural network defines how the weighted sum of the input is transformed into the output of a neuron. One of the purposes of this activation function is to add nonlinearity to the network. Linear equations are less complex, and therefore less able to learn complex mappings. Without this nonlinearity the a neural network would behave as a single layer network regardless of how many layers there are. The output of this activation function decides whether the neuron gets activated or not. There are various types of activation functions. The most commonly used ones are the rectified linear unit (ReLU) function, the sigmoid function and the softmax function.

The ReLU function is commonly used in the hidden layers. It is calculated by taking MAX(0, x). This means that it will not change the input unless it is below 0, in which case ReLU will output 0 (Figure 7). One advantage of the ReLU function is that it is relatively fast to compute compared to other activation functions. A disadvantage of ReLU is that it has a gradient of zero for inputs below or equal to 0. This causes multiplication by zero which results in gradient descent not altering the weights. This is known as the dying ReLU problem (Lu et al., 2020).

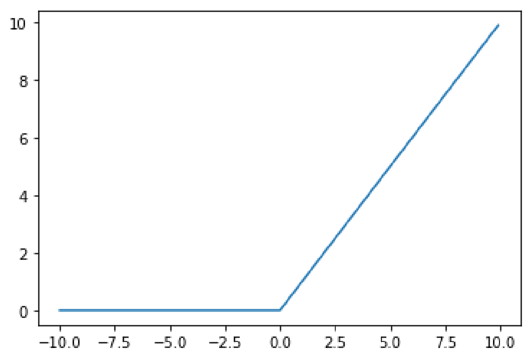


Figure 7: The ReLU function

The sigmoid function, also known as the logistic function, maps a number to a value between 0 and 1 (Equation 2). This makes the sigmoid function usefull for two-class classifcation, such as deciding whether a neuron should be activated or not. It can also be used for computing a probability. Like ReLU it is commonly used in the hidden layers. A disadvantage of the sigmoid is that an input near 0 or 1 will have a low gradient approaching zero. Consecutive multiplication of these gradients will therefore quickly result in a value approaching zero.

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

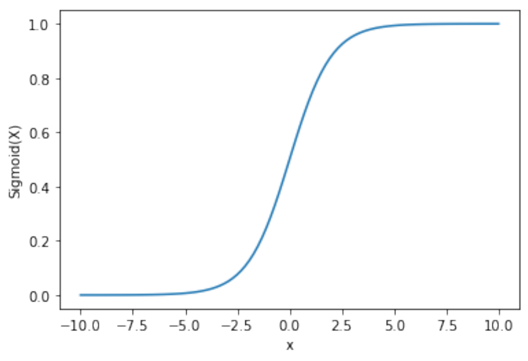


Figure 8: The sigmoid function

Softmax is used to compute prability distributions from a vector of scalars. It converts each of the scalars to a value between 0 and 1, with the sum of the vector being 1. This can be usefull when trying to predict multiple classes, as softmax will return the relative probabilities of all the classes. The difference with the sigmoid function is that softmax takes into account the probabilities of the other classes.

This is why it is used in multiclass classification where there are more than two classes. Softmax is most commonly used in the output layer of the network.

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3}$$

A common problem when training neural networks is overfitting. This occurs when a model fits too closeley to the training data. Larger neural networks are more likely to overfit. Especially when the training dataset is relatively small. Batch normalization is a technique that can be used to normalize the output of the previous layers. While batch normalization is effective in reducing overfitting, the exact reason for why it works is a topic of discussion. Another method that helps to reduce overfitting is the use of dropout layers. Dropout layers can be used to randomly discard some of the outputs during the training. By randomly disabling some percentage of neurons, the network learns not to rely too much on any specific connection. Dropout layers are typically used after fully connected layers in the network (Srivastava et al., 2014).

CNNs can use a pooling layer downsamples the input data. This helps to prevent overfitting. Pooling is applied seperately to each depth dimension. The most commonly used operation for pooling is the MAX operation, which takes the local maximum, using a 2x2 filter and a stride of 2. The stride parameters indicates the step size when sliding the kenrel across the input map. This reduces the number of parameters by 75%, thus also reducing the computational cost. Pooling decreases the width and height of the input, whilst retaining the depth. The kernel size for the pooling layer tends to be small, as to not lose too much of the information.

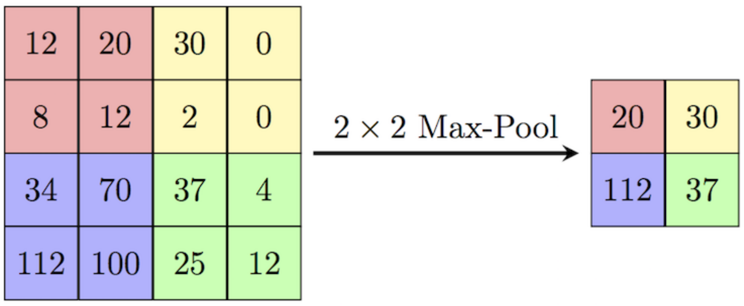


Figure 9: 2x2 maxpooling with a stride of 2 on a 4x4 input map (Andrew Ng, 2020).

2.3 Loss Functions

MSE, CSL, LOG

2.4 Backpropagation

Backpropagation adjusts the weights in the network to minimize the loss. All weights in a neural network are initiliazed randomly at first. When the network outputs a prediction, a loss function is used to calculate the loss. The loss for a given input is a function of the weights of the network. The relationship between the loss and the weights is visualized in figure 10. This means that in order to minimize the loss it is neccessary to find the global minimum of the loss function. Since there is no straightforward way to find the global minimum of a complex function such as a neural network, backpropagation uses stochastic gradient descent to find a local minimum by making incremental changes to the weights based on the gradient.

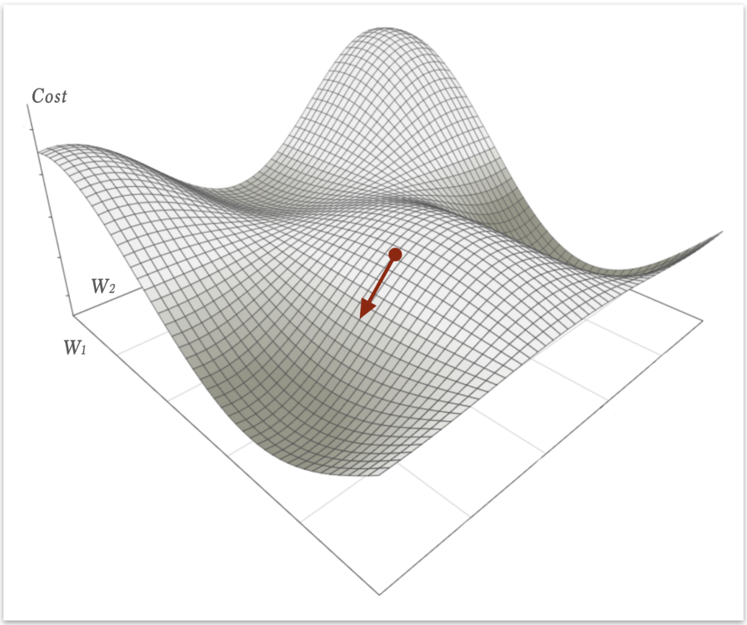


Figure 10: The loss function plotted for a network with two weights ().

The gradient of the loss returns a vector pointing in the direction where the loss increases most rapidly. By taking the negative gradient it is possible to calculate the direction where the loss decreases the most. The network can be seen as a complex function where each weight is a parameter in that function. Given a loss function E, the partial derivative can be computed for a single weight or bias as seen in equation 4. his allows the network to optimize each weight individually.

$$\frac{\partial \boldsymbol{E}}{\partial w} \quad \text{and} \quad \frac{\partial \boldsymbol{E}}{\partial b} \tag{4}$$

3 Capsule Networks

3.1 Capsules

A key feature of capsule networks is that they are equivariant. Information about the location and pose of an object is preserved throughout the network. A capsule network consists of layers of capsules. A capsule is a group of neurons encapsulating properties of a single entity. Each of these capsules is its own neural network. Capsules correspond to an entity or sub-entity in the input image. Capsules in lower layers will detect simple features such as edges or gradients. Higher level capsules will combine these simple features into more complex features.

Not only can a capsule detect the presence of such an entity, it is also able to detect properties of that entity. Each capsule in the layer outputs a vector that encodes this information and passes it on to the next layer. A squash function is used to ensure that the vector length is between 0 and 1. The capsules in this next layer also correspond to an entity. These entities are typically composite entities of those in the previous layer. Each of these capsules will then integrate the vectors of the previous layer and come up with their own output vector encoding the information for their corresponding entity.

In a typical neural network, the outputs of a layer would be multiplied with a weight matrix to determine the inputs for the subsequent layer. In a capsule network, however, the capsules can decide where to direct their output. A capsule will route its output to the appropriate capsule in the following layer depending on its output. This is accomplished through an iterative procedure via the routing mechanism.

3.2 Equivariance

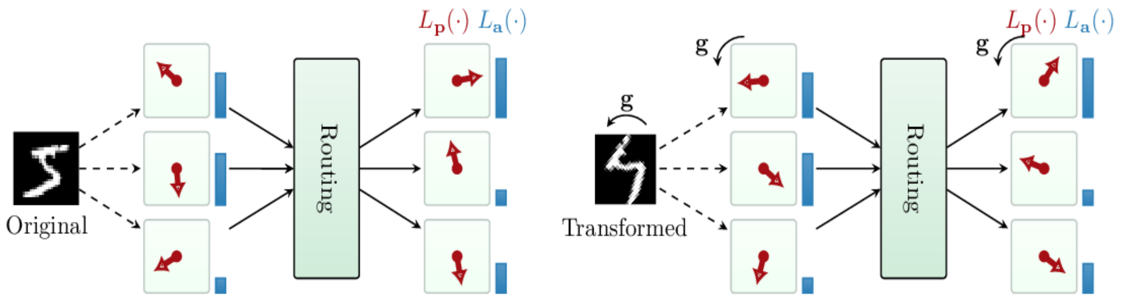


Figure 11: Equivariance of the pose vectors (Lenssen et al., 2018).

Equivariance is a form of symmetry. A function is said to be equivariant with respect to a given transformation if applying that transformation and then computing the function produces the same result as computing the function and then applying the transformation (Equation 5). This means that if you transform the input. The output will be transformed accordingly. Figure 11 visualizes how rotating the original images causes the pose vectors to be rotated in the exact same way. Thus showing that it is equivariant with respect to rotation.

$$f(L(x)) = L'(f(x)), \quad \forall x \tag{5}$$

part of the reason that capsule networks are intersting to use has to do with equivariance.

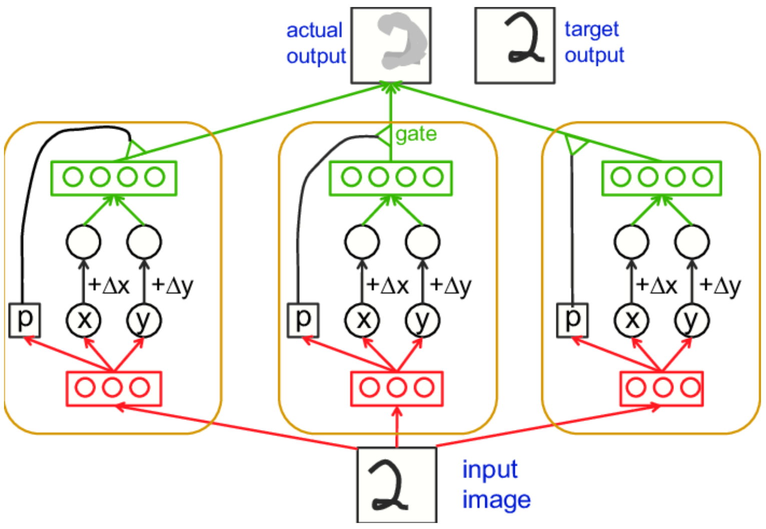


Figure 12: Three capsules that model translation (Hinton et al., 2011).

Figure 12

3.3 Dynamic Routing

Dynamic routing is done through a method called routing by agreement. The lower level capsules all output a prediction vector. This vector contains the pose of feature that the capsule is trained to detect, and also the probability that the feature is present in the input. This vector is only routed to the higher level capsule if the other low level capsules in that layer agree with each other. Agreement is measured by comparing the output vectors of the capsules in a given layer. If the capsules agree, then the output is routed to the next capsule. The agreement between the vectors is dependent on which capsule in the following layer the information is being routed to. Capsules can be in agreement with each other when routing to a given capsule in the next layer but disagree when being routed to another capsule. This allows for different combinations of capsules to be routed to each capsule in the next layer. Over each iteration of the routing algorithm, the network will increasingly settle on a routing path for the capsules.

3.4 Our Capsules

This experiment aims to demonstrate that by using capsule networks it is possible to build an extractor of the pose from an image without any supervision. This will be done by extracting the translation from an image. This will be simpler to implement compared to affine pose extraction. This is possible because an affine transformation is a composition of one-parameter groups. Each of those groups can then then be described as a translation by reparameterizing the coordinate system, e.g. a rotation in cartesian coordinates is the same as translating in polar coordinates (Hel-Or & Teo, 1998). In order to exract the pose, the model will have a primary capsule layer with multiple capsules that each detect a different feature. A randomly translated version of the input image will be created and passed to each of these capsules, together with the original input image. The capsules will output a probability p and a $\vec{xy}$ vector for each image. p is the probability that the feature corresponding to the capsule is present. The vector $\vec{xy}$ is the pose which represents the position of the feature with respects to its canonical view. The canonical view being the the feature, as learned by a specific capsule, where $\vec{xy}$ is equal to zero. Each capsule learns only one specific feature. Therefore, the learned feature can be reconstructed with any given pose from just the translation vector $\vec{xy}$, as the feature itself is inherent to the capsule. The reconstructed image is then multiplied by the probability that the feature is present. The output of each capsule is then summed up, resulting in the final reconstructed image.

Each capsule is also given a translated version of the input image, and the parameters of its translation $\Delta\vec{xy}$. The translation vector $\Delta\vec{xy}$ is applied added the pose $\vec{xy}$ extracted from the original image. The translated image is then reconstructed by a decoder from only the translation vector $\Delta\vec{xy}$ and the pose extracted from the orignal image $\vec{xy}$.

The loss is then calculated by summing up the mean squared error of both the reconstruction of the image and the reconstruction of the translated image.

The model will be built in Python 3.6 using the PyTorch library. The architecture will be based on a transforming autoencoder as described by (Hinton et al., 2011).

Asked to predict orientation. Evalue mode for 2 different iamges, images 2 is rotated version of 1.

Subtract outputs (pose diference)

4 Experiments

Capsule networks use capsule.

4.1 Fully Connected Neural Network

First a convolutional neural network was created to predict the class of handwritten digits. The network was built in Pytorch with 2 convolutional and 2 linear layers. This model was trained on the MNIST dataset and reached an accuracy of 99%. The loss functino used for the model was the negative log likelihood loss, which is usefull for classification problems.

4.2 Convolutional neural net

4.3 Autoencoder

For this experiment an autoencoder was made for prediction and reconstruction on MNIST. the model consisted of an encoder and a decoder. The encoder outputted a prediction of the class. The decoder then used this prediction to reconstruct the image. The model used 3 convolutional layers in the encoder, and a linear layer to transform the data to a 10 dimensional vector containing a probability for each class. One scalar for the probability of each class. Another linear layer then upsampled the data from those 10 values. The result was reshaped to 2 dimensions, and a decoder with 3 convolutional layers was used to reconstruct the input image. The model used two loss functions; Cross entropy loss was used on the output of the predictor. Mean squared error was used to evaluate the reconstructed image. The loss was then defined as the sum of the output of these functions.

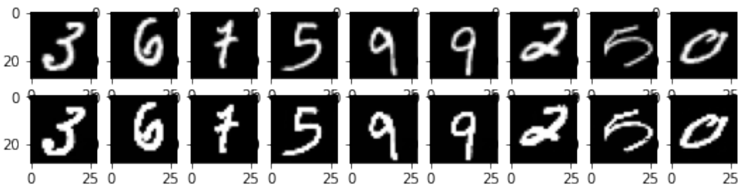


Figure 14: A plot showing the target images above and the reconstructions below.

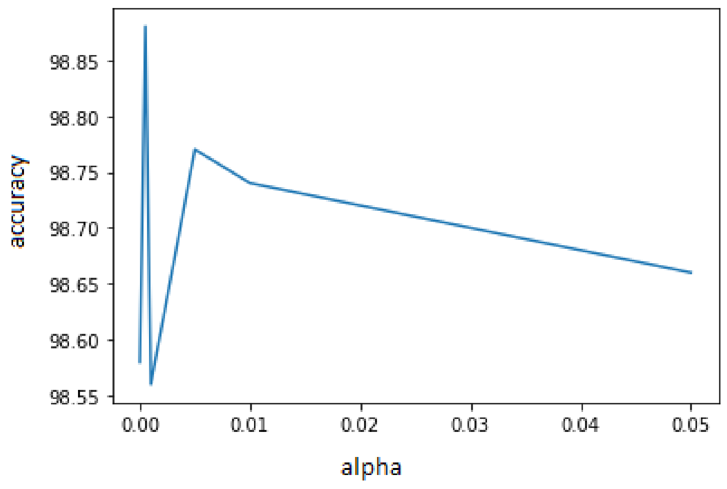


Figure 15: aaaaaa

loss = 0.750106 (single cap tae)

4.4 Transforming Autoencoder

A transforming autoencoder (TAE) was trained for reconstruction on the MNIST dataset. For this experiment the target reconstructions were randomly translated. The parameters for this random translation, x and y, were also passed on to the network. The TAE consists of an encoder, which outputs 3 values; A probability p and two parameters for the translation, x and y. The translation on te target output was then also applied to the x and y output of the encoder. The image was then reconstructed solely from these translated x and y values. Each capsule was not trained to reconstruct the whole image. In stead, each capsule was trained to reconstruct a specific feature with a specific translation. Each capsule also outputted a probability representing how likely it is that that given feature is present. The reconstruction of each capsule was then multiplied with its respective probability. The final output of the model was obtained by summing up the output of each capsule.

4.5 Equivariant Capsule Network (MNIST

4.6 Equivariant capsule network (CIFAR)

Current experiments do not provide enough flexibility.

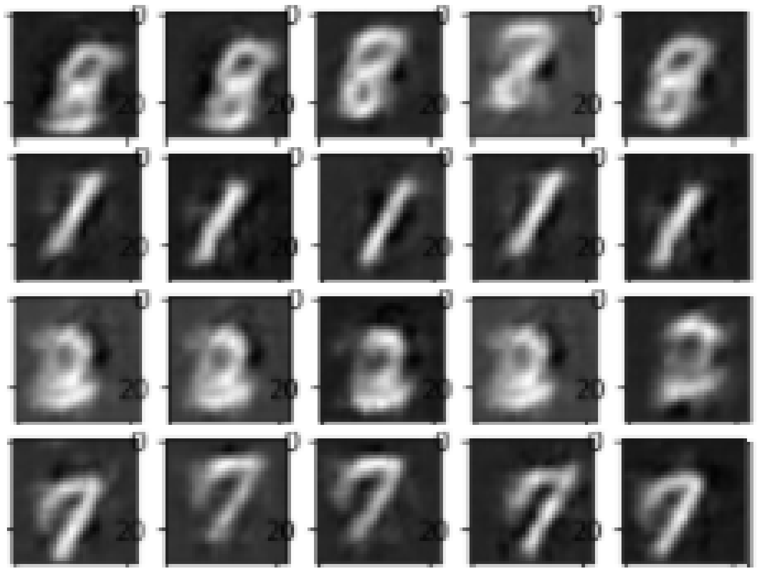


Figure 16: Reconstructed images for the classes 8, 1, 2 and 7 respectively from top to bottom.

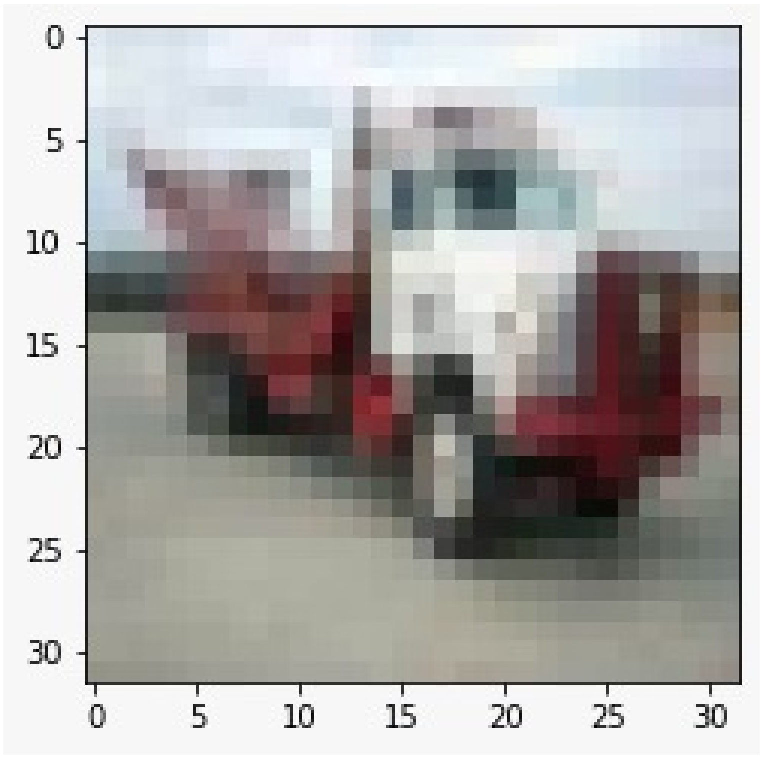
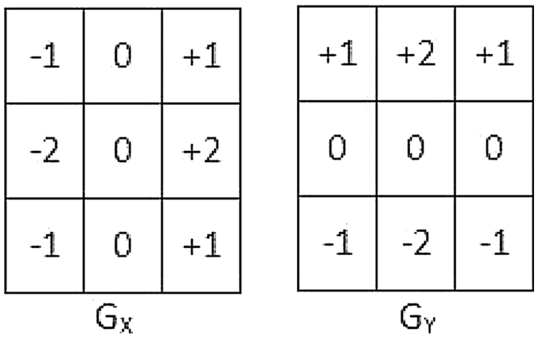


Figure 17: Original image in CIFAR-10.

Gradients are computed in the x and y directions by convolving the image with the kernels shown in figure 18. The resulting gradients, G_x and G_y, together form the vector $\vec{G_{xy}}$ of the direction of the gradient (Figure 19). By taking the norm of each vector we get a ???? as shown in figure 20



-1	0	+1
-2	0	+2
-1	0	+1

G_X

+1	+2	+1
0	0	0
-1	-2	-1

G_Y

Figure 18: The kernels used for computing the gradients in the x and y directions.

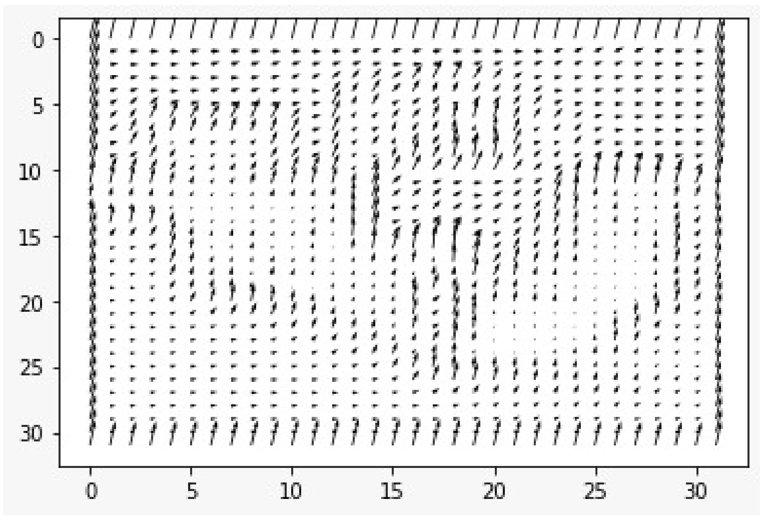


Figure 19: Vector view of the combined x and y gradients.

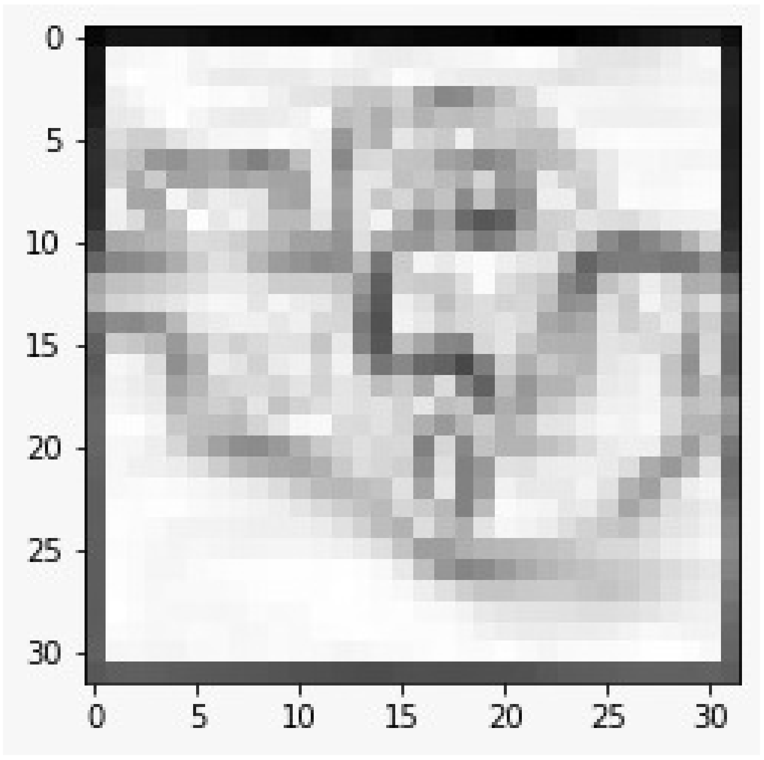


Figure 20: Visualized gradient length.

References

Abraham, A. (2005). Artificial neural networks. *Handbook of measuring system design.*

Andrew Ng, Y. B. M., Kian Katanforoosh. (2020). Deep learning specialization. https://www.coursera.org/specializations/deep-learning

Birla, D. (2019). *Basics of autoencoders.* Retrieved June 10, 2021, from https://medium.com/@birla.deepak26/autoencoders-76bb49ae6a8f

Hel-Or, Y., & Teo, P. C. (1998). Canonical decomposition of steerable functions. *Journal of Mathematical Imaging and Vision*, *9*(1), 83–95. https://doi.org/10.1023/A:1008274211102

Hinton, G. E., Krizhevsky, A., & Wang, S. D. (2011). Transforming auto-encoders. In T. Honkela, W. Duch, M. Girolami, & S. Kaski (Eds.), *Artificial neural networks and machine learning – icann 2011* (pp. 44–51). Springer Berlin Heidelberg.

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks. *CoRR*, *abs/1506.02025.* http://arxiv.org/abs/1506.02025

Lenssen, J. E., Fey, M., & Libuschewski, P. (2018). Group equivariant capsule networks. *CoRR*, *abs/1806.05086.* http://arxiv.org/abs/1806.05086

Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2020). Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, *28*(5), 1671–1706. https://doi.org/10.4208/cicp.oa-2020-0165

Ognjanovski, G. (2019). *Everything you need to know about neural networks and backpropagation — machine learning easy and fun.* Retrieved June 10, 2021, from https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

Pelletier, C., Webb, G., & Petitjean, F. (2019). Temporal convolutional neural network for the classification of satellite image time series. *Remote Sensing*, *11*, 523. https://doi.org/10.3390/rs11050523

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic routing between capsules. *CoRR*, *abs/1710.09829.* http://arxiv.org/abs/1710.09829

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, *15*(1), 1929–1958.

Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., & Sakuma, J. (2018). Malware analysis of imaged binary samples by convolutional neural network with attention mechanism, 127–134. https://doi.org/10.1145/3176258.3176335

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2020). *Dive into deep learning* [https://d2l.ai].