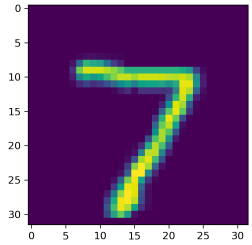
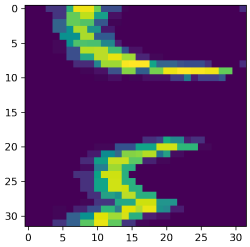
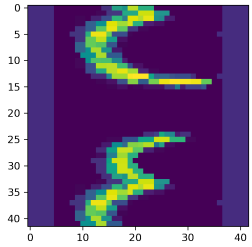
Cylindrical CNN for Beer Cap Classification

(A online version with animations can be found HERE (https://github.com/mrlomar/CVbyDL-Rotation-Invariant-CNN/blob/main/CyCNN.ipynb))

In this notebook, we try to develop a **Convolution Neural Network** that can classify randomly rotated beer caps while training on limited data. We do this by trying to make the network **invariant to rotation** to reduce the amount of samples needed for training. Our thought behind this is that a network that inherently comprehends rotation will need less training data to obtain reasonable results than a traditional CNN that needs to learn the rotations. To achieve this invariance, we implement the theory published in the paper CyCNN: A Rotation Invariant CNN using Polar Mapping and Cylindrical Convolution Layers (https://arxiv.org/pdf/2007.10588.pdf). First we apply the theory to the MNIST dataset to assess whether it works as intended and secondly we apply the theory to our own generated data set to assess the viability of our approach.

Creating CyCNN

Making the network invariant to rotations consists of two steps, namely: **(1)** Transforming the input to polar coordinates and **(2)** using Cylindrical Convolutional Layers. Both of these steps are discussed in their respective subsections.

Normal Coordinates	Polar Coordinates	Circular Padding
		

Polar Coordinates

The idea of polar coordinates is to express locations in the image as (r, ϕ) instead of (x, y), where r is the distance to the origin and ϕ is the angle. Because one of the coordinates includes the rotation in the original image, using polar coordinates transforms rotations in the original space to translations in polar space. Since CNN's *should* be translation invariant, this would in theory make the network invariant to rotations in the original space. But since there is no free lunch in computer science, using this does make the network less robust to translations in the original space.

A visualisation of the process discussed in this subsection can be found in the first and second row of the table above.

Cylindrical Convolutional Layers

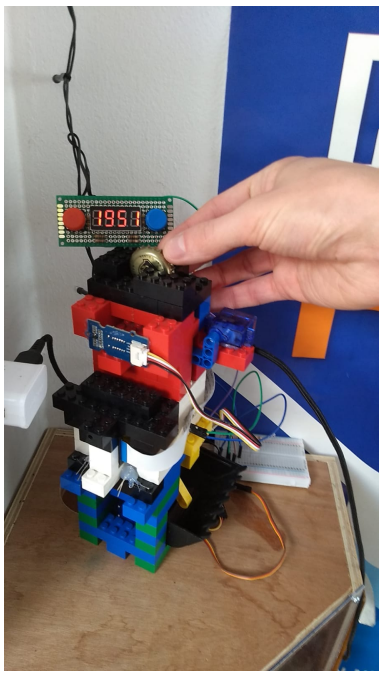
The next step to making the CNN invariant to rotations is to implement Cylindrical Convolutions Layers. While this might sound intimidating, it is actually rather simple. To make the convolutional layer cylindrical, one only has to use a special type of padding. When using polar coordinates the ϕ coordinate represents the rotation, this means that the top of the image connects to the bottom of the image. To communicate this information to the CNN the padding on the bottom should continue with the pixels found at the top of the image. For the r coordinate there is no such relation. Because of this on the x-axis *Zero padding* is used.

A visualisation of cylindrical padding can be found in the second and third row of the table above.

Beer Cap Dataset

Photo acquisition

To create our beer cap dataset, a setup with a Raspberry Pi with Pi Camera was used. In this setup, a beer cap enters on top, when detected by a sensor, the first servo will let it through, then the beer cap enters a dark space which is lid by LEDs for consistent lighting conditions. There it is photographed, after which a second servo will let it pass and then it drops down in a big container. The first servo prevents a second beer cap from entering while another beer cap is in process. The setup also includes a 7-segment display that shows the total counted beer caps. We had some struggles with getting good lighting conditions since we wanted good and consistent photos. Photos taken in the middle of the day looked way better than later on the day. Furthermore, the direction of the light mattered for shadows a lot. We solved this by using LEDs and preventing most other light in the picture. Because the plexiglass in front of the beer caps reflects the LEDs light, this was also a challenge which we solve by careful placement of the LEDs, specific camera settings and accepting some reflections.



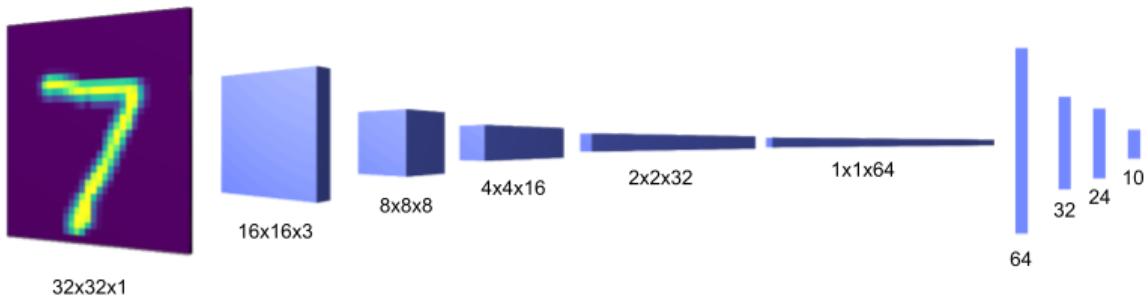
Data augmentation

We wanted good and consistent images so we could increase the size of the dataset by data augmentation. We first cropped the pictures, then resized them to 32 by 32 pixels. We then included those images and several randomly rotated variants of those images in the dataset. The idea being that bottle caps that would enter already have a random rotation, and by randomly rotating images, we simulated more entries of the same brand.

Comparison MNIST

Network Architecture

To evaluate our CyCNN on the MNIST dataset, the architecture seen below was used both for the CNN and CyCNN. Between each cube shown in the figure below there is a convolutional layer which applies the following steps to the input: **(1)** Apply cylindrical padding, **(2)** Apply a 3x3 convolution, **(3)** Apply the ReLu function and **(4)** Apply 2x2 max-pooling. The (Cy)CNN is then connected to a fully connected neural network to classify the input. When using the CyCNN model, there occurs a polar transformation between the input and the first convolution layer.



Experiment

To assess the effectiveness of using the CyCNN rather than the traditional CNN, we compare the results of 4 models on non-rotated and randomly rotated images. The models we evaluate are the following:

- **CyCNN^R** - This model uses polar coordinates and cylindrical convolutional layers, but also includes an extra step. When an input is given to the network, CyCNN^R first randomly rotates the image before feeding it to its learned network. This means that while training the model will see and learn rotations of the original data.
- **CyCNN** - This model uses polar coordinates and cylindrical convolutional layers, but does not randomly rotate the images. This means that when the model is evaluated on rotated images, it will never have seen these rotations before. Any improvements over the traditional CNN will be purely due to the use of polar coordinates and cylindrical convolutional layers.
- **CNN^R** - This model is a standard CNN, but it also randomly rotated the input images before feeding it into the network. This approach could theoretically allow the CNN to learn the rotations on its own.
- **CNN** - This model is a standard CNN that does not randomly rotate the images. Therefore, during testing it will never have seen these rotations before.

To compare the models, each model is trained for 2 epochs. In the first epoch the learning rate is set to 0.001 and in the second epoch the learning rate is set to 0.0001.

Results

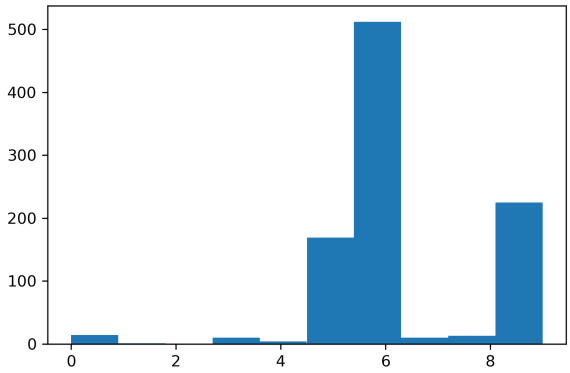
The results of the experiment discussed above are shown in the table below.

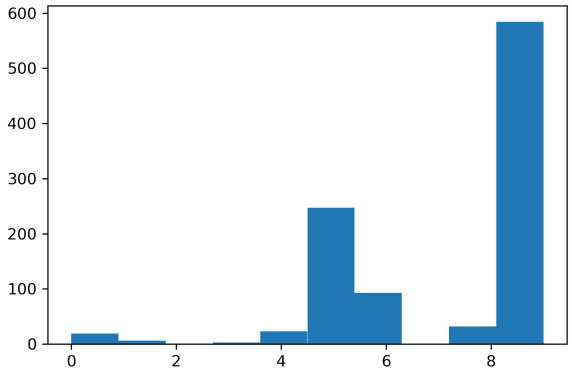
Model	Normal Accuracy	Rotated Accuracy
CyCNN^R	92%	**91%**
CyCNN	94%	72%
CNN^R	88%	88%
CNN	**97%**	39%

As you can see, **CyCNN^R** achieves the highest accuracy on the randomly rotated data. While the traditional CNN achieves the highest accuracy on the non-rotated data, its accuracy tanks when the images are randomly rotated (as expected). Comparing the CyCNN to the CNN, we can see that CyCNN inherently is more invariant to rotations, as it achieves an accuracy of 72% rather than 39% on the randomly rotated data. Overal, we can see that using a CyCNN^R seems to be the best approach for our use case.

Lastly, an interesting thing to note is the distinction between 6's and 9's when the data is randomly rotated. In theory, a network this is invariant to rotation should be able to distinguish a 6 from an upside-down 9 and vice-versa. A network that is not invariant to rotations would not be able to do this. To investigate this, we look at the prediction distribution for 6's and 9's for the CyCNN model. The results are shown below.

CyCNN 6 predictions	**CyCNN 9 predictions**



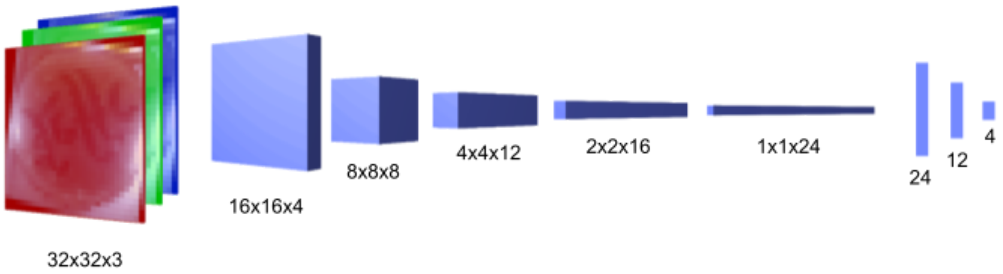


We can see that while the CyCNN is not able to distinguish between the two classes fully, it is able to correctly classify both classes the majority of the time. We can also see that strangely enough the network has more difficulty with distinguishing between a 9 and a 5 than between a 9 and a 6. This could be because when using polar coordinates a 5 might be more similar to a 9 than a 6 is to a 9.

Comparison Beer caps

Network Architecture

To evaluate our CyCNN on the CAP dataset, the architecture seen below was used both for the CNN and CyCNN. Between each cube shown in the figure below there is a convolutional layer which applies the following steps to the input: **(1)** Apply cylindrical padding, **(2)** Apply a 3x3 convolution, **(3)** Apply the ReLu function and **(4)** Apply 2x2 max-pooling. The (Cy)CNN is then connected to a fully connected neural network to classify the input. When using the CyCNN model, there occurs a polar transformation between the input and the first convolution layer.



Experiment

To assess the effectiveness of the CyCNN on the beer cap dataset, we first train on the full dataset where we use 80% for training and 20% for testing. The models, CyCNN^R, CyCNN, CNN^R and CNN, are each trained 10 epochs starting with a learning rate of 0.001 and after epoch 5 a learning rate of 0.0001 with a momentum of 0.9.

We trained for different (lesser) amounts of training data to see if the CyCNN requires less data than the CNN. We used 20% as test data, then decreasingly used less data (80%, 60%, 40%, 20%, 10%) of the total data as train data. The remaining data was not used.

Results

The results of the experiment discussed above are shown in the table below. In each cell the accuracy on the test set followed by the randomly rotated test set is shown.

Model \ % data used for training	80%	60%	40%	20%	10%
CyCNN^R	98%/**99%**	**99%**/**99%**	98%/98%	93%/94%	**94%**/94%
CyCNN	**99%**/**99%**	**99%**/98%	**99%**/**99%**	**97%**/**97%**	93%/**95%**
CNN^R	96%/96%	95%/95%	94%/94%	92%/92%	80%/82%
CNN	98%/93%	96%/90%	88%/82%	91%/87%	79%/81%

As you can see in the table, the CyCNN overall performs the best. On this dataset we expected (almost) no difference between the CyCNN^R and the CyCNN because the beer caps already have lots of different rotations. When there is less data to train on, the difference between the CyCNN and the CNN increased, which supports our hypothesis.

Discussion

During our experiment, we identified a three main issues. The first one being that the we did not use K-fold cross validation to make our experiments more robust. We made this choice mainly due to time and resource limitations.

Secondly, training epochs were limited due to time restrictions. This could influence the results if CyCNN is able to learn to a higher accuracy than a traditional CNN, but we can not confirm this.

Lastly, during training on the CAP dataset, the network sometimes had a sort of restart and would not recover from it. This meant that sometimes CyCNN or CNN would achieve terrible results. We haven't yet been able to explain this phenomenon.