Universiteit Leiden

Solving the shape ambiguity and domain gap problem using multiple-view classifiers and domain adaptation methods

Dominieq de Koster (s3803406)

Supervisors:
Prof. Dr. Joost Batenburg & Dr. Daan Pelt

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

June 17, 2026

Abstract

When classifying shapes, some viewpoints of shapes are uninformative, meaning that they are unable to yield sufficient information to uniquely determine their shape. To investigate this problem, we constructed a synthetic dataset containing both informative and uninformative views of ambiguous and unambiguous shapes. We demonstrated that multiple-view classifiers are able to integrate information from these informative and uninformative views. We also demonstrated that a multiple-view classifier pretrained on synthetic data performs poorly when performing inference on real-world objects. Finetuning on a small real-world dataset and applying traditional computer vision filters for pre-processing improved performance, showing that these methods could reduce domain gaps. The findings of this research may be relevant for industrial settings, where object poses may be unknown and real data may be sparse.

Contents

1 Introduction

Single-view image classifiers have become abundant in many settings over the past years, such as in retail, X-ray interpretation and surveillance [1]. In addition to this, the use of single-view image classifiers has grown in industrial settings, for example in anomaly detection, defect detection, and fault detection [16].The development of these classifiers has been ongoing for many years, and classification performance for arbitrary objects has reached very high scores: ConvNext-XL reached an accuracy of 87.8% on the ImageNet-22K dataset [8]. However, not every classification problem can be solved using these classifiers, since only having access to one view means that depth information is partially lost. Moreover, occluded objects or parts of them may cause classifiers to not even have access to parts of a scene, making it fundamentally impossible to perform certain classification tasks. For example, classifying certain three-dimensional shapes is difficult when presented in certain poses. Think of a cone, which bottom is oriented to the camera. A single-view classifier cannot determine whether the presented shape is a cone or a cylinder, as it is lacking access to crucial information. Specifically, given a set of three-dimensional objects, where each object contains a set of views, we investigate whether an object can be uniquely identified from a single view alone.

Furthermore, some industrial settings may produce a specific kind of product, which could be expensive or only produced in low quantities. This makes training a classifier that works on this product hard, as there may not be enough data available to train the classifier. To solve such a problem, a similar problem can be created synthetically using synthetic datasets. In this way, the classifier learns to solve the problem before it is deployed. However, there will always be a gap between the real-world domain and the synthetic domain, thus domain adaptation methods will always be needed. This problem, combined with the shape ambiguity problem, is what will be studied in this thesis.

1.1 Using multiple-view classifiers to solve the shape ambiguity and domain gap problem

To solve the problem of shape ambiguity in single-view classifiers, we will first train a single-view classifier on a synthetic dataset, and show that such a classifier is unable to correctly classify shapes for certain viewpoints. Next, we will train a multiple-view classifier on the same synthetic dataset, and show that this classifier is able to correctly classify shapes for any viewpoints. Finally, the trained multiple-view classifier will be tested on real-world objects. These objects will be retrieved from a scraped dataset of the internet, where after the pictures are stacked and their output is fed to the multiple-view classifier. In order to obtain a good performance, we will perform a multitude of methods to reduce the observed domain gap. In this way, we hope to answer the question as to what extent a randomly sampled additional view aids performance in the recognition of physical ambiguous three-dimensional shapes. The single-view classifier will serve as a baseline, in order to see a difference between classification performance using an additional view.

Earlier research has mostly focused on constructing new multiple-view classifiers that obtained a high performance on classifying 3D shapes, which were obtained from virtual 3D representations of real-life objects. These virtual 3D representations were usually stored as point clouds, and multiple-view classifiers would sample two-dimensional views from them. However, in this research we directly generate two-dimensional views based on virtual objects. This aids in computational

efficiency. Moreover, research on domain adaptation methods has mostly been focused on selecting useful data and learning the target distribution separately, while we will directly use traditional computer vision filters in our pipeline to extract useful features.

The results of this research can be used in industrial settings, where a tradeoff between accuracy and speed is necessary. For businesses, it is interesting to know whether obtaining an additional view of an object is necessary in order to aid classification performance of objects. This may differ per specific use case, though our pretrained model could help companies answering this question.

1.2 Thesis overview

The current chapter contains the introduction; Section 2 provides the background of this research, and identifies the current problems within the shape classification domain; Section 3 discusses the related works, where we will look into literature that aims to find a solution to the previously identified problems; Section 4 describes the experiments, their motivation and how these experiments were physically realized; Section 5 describes the outcome of the proposed experiments, and what the impact of the found results is. Finally section 6 summarizes the findings of all experiments, connects them and places them in a broader perspective, where after a proposal for further research is made.

This thesis was written for the LIACS BsC "Data Science and Artificial Intelligence", and was supervised by prof. dr. Joost Batenburg and Dr. Daan Pelt.

2 Background

In this chapter, we will explore the background that sets the stage for this thesis. We will first establish definitions that will be used throughout this paper, after which we examine the areas of multiple-view classification, data scarcity and pre-training, Blender, model shortcuts and qualitative evaluation methods.

2.1 Definitions

- **Shapes**: a shape is an object that exists in a three-dimensional space. Each shape is defined as a set of points in $\mathbb{R}^3$ adhering to a set of constraints. In this paper, whenever we refer to the term shapes, we limit ourselves to a set of six shapes: cubes, pyramids, cylinders, cones, spheres and rings.

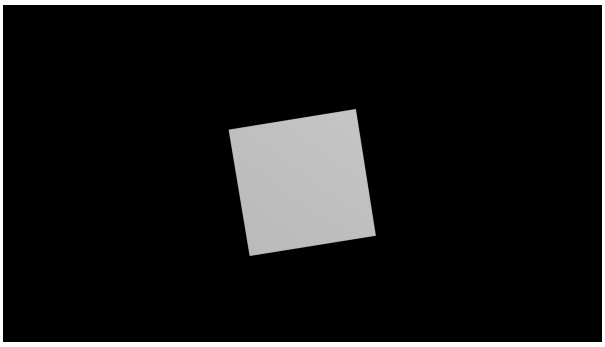
- **Views**: a view is a two-dimensional observation of a three-dimensional object consisting of one of the paper's shapes, which is made from a certain point in three-dimensional space.

- **Unambiguous shapes**: shapes where each viewpoint that exists is able to yield sufficient information to uniquely determine the 3D shape by one observer.

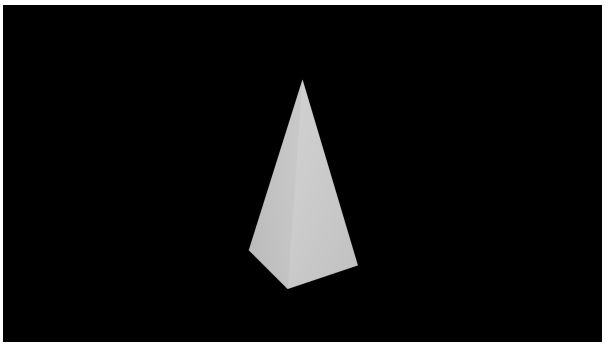
- **Ambiguous shapes**: shapes where at least one viewpoint exists that is unable to yield sufficient information to uniquely determine the 3D shape by one observer.

- **Uninformative views**: views of ambiguous shapes that are each individually unable to yield sufficient information to uniquely determine the 3D shape. An example is shown in figure 1a.

- **Informative views**: views of ambiguous shapes that are each individually able to yield sufficient information to uniquely determine the 3D shape. An example is shown in figure 1b.



(a) An uninformative view on a pyramid



(b) An informative view on a pyramid

Figure 1: Two examples that show the difference between an uninformative and an informative view on a pyramid

2.2 Classification of 3D shapes

Over the past years, there have been multiple proposed architectures that aim to accurately classify virtual 3D objects by sampling multiple views from this object. The datasets that are used for training these classifiers are usually made up of 3D models represented as point clouds, such as Modelnet-40 [17]. 3D models cannot only be stored using point clouds, but also using voxel grids, or a mesh structure. Before 2015, this variety in structures meant that architectures operating directly on three-dimensional representations of objects were usually based on hand-crafted features [15]. An exception to this rule is the 3D ShapeNet proposed by Wu, that directly operates on voxel grids. [17].

The paradigm of 3D-model recognition using sampled views started in 2015, when a paper by Su was published. In this paper, they compared classifiers which were trained under two conditions. In the first condition, classifiers were trained that operated directly on 3D representations of objects. In the second condition, classifiers were trained using multiple sampled 2-dimensional views. They showed that classifiers from condition 2, when tested with a single sampled view yielded the highest performance. For multiple views, the performance increased even more. Specifically, the CNN architecture gave high results. What must be noted here is that this research is over 10 years old, and the classifiers that the CNN architecture was compared with can be seen as 'traditional' machine learning methods, meaning they do not make use of deep learning [15].

2.3 Data scarcity and pre-training

In order to train the mentioned multiple-view classifiers, great amounts of data are necessary. Since these types of classifiers make use of labeled datasets containing 3D models, we need datasets that contain these. However, these are less abundant than labeled 2D datasets, which contain labeled images. There are two reasons for this lack of data: first, 'traditional' classifiers working on a single image have been around for longer, which means that datasets have already been created for

training these classifiers. Second, creating labeled 3D datasets requires more labour than creating labeled 2D datasets. In 2D datasets, images have usually been scraped from the web, and only need to be labeled by a human. However, 3D datasets require three-dimensional representations of objects, which have to be created manually. This is more labour-intensive than labeling pictures.

Fortunately, the internet is full of open-source 3D models. These models differ in quality of both the model itself, and the labeling. This was discovered by Yang, who found out that there were predominantly two types of noise in these scraped datasets: label noise and background noise. Label noise means that some models are labeled incorrectly, while background noise means that some objects also contain other modeled objects within a scene, diverting attention away from the object of interest [18].

2.3.1 Pre-training using synthetic datasets

Another solution that is used to combat data scarcity is creating synthetic datasets. These datasets contain scenes that are rendered by a computer, for example using Blender. Because a virtual world is used, creating an infinite amount of samples is theoretically possible. The idea is to first pre-train a classifier on this synthetic dataset, and later finetune it on real-world data. During the pre-training phase, the classifier will have learned the mapping between features and a class label, while during finetuning the classifier generalizes to the characteristics of the test distribution. This method was used by Richter for pre-training semantic segmentation in traffic and city settings. To do this, information from video games was extracted by placing a wrapper between the program and the OS, which extracted the rendered objects and their position in the scene. Because of this, scene segmentation was performed automatically. A human labeler only had to assign labels to the segments, and the label program also learned to predict labels during the labeling process, leading to the annotation task becoming easier for the annotators. Then, they pretrained a semantic segmentation model on this dataset, and later finetuned on a 'real' dataset that contained similar data. The classifier that was first pretrained on the synthetic dataset, and later finetuned on the real dataset reached a higher score than the classifier that was only trained on the real dataset [13]

2.3.2 Domain transfer and domain gap

Although synthetic datasets may seem like the perfect solution for data scarcity, they introduce new problems. The most important one is called domain gap. This means that models trained on a synthetic dataset will have to perform inference on real-life data. The distribution that the model is originally trained on is called the *Source distribution*, and the distribution that the model is ought to perform inference on is called *Target distribution.* The gap between the source distribution of the synthetic and the target distribution of real-life data could lead to structurally inferior inference performance. This is especially a problem in visual tasks, when the style of two distributions does not match. For example, there could be one distribution consisting of paintings, and one distribution consisting of images. There are multiple types of domain transfer, but the one that will be encountered in the thesis is *Transductive Transfer Learning*, where the source and target task of the model are the same, though the data distribution is different. For this kind of transfer learning, domain adaptation methods can be used. We will talk more about these methods in the related works section [4].

2.4 Blender

In order to create synthetic datasets of high quality, we need a tool that allows us to easily generate shapes based on parameters. Moreover, since we need a big dataset, an apt tool should be scalable. A tool that possesses these capabilities is *Blender*. Blender is an open-source tool that can be used for 3D-modeling and creating animations. It also includes a Python API, which can be used to automate these features. Blender has been used in multiple Machine Learning papers, indicating its usability. For example, Zaman used Blender to generate an extensive synthetic dataset, which consisted of six classes of 3D shapes with different lighting and materials. This dataset can be used to train models on working with different materials. However, since Blender generates synthetic datasets, domain gaps are a problem. A key aspect that was identified in this research was the fact that using different rendering engines produced different images. The two rendering engines that are used in Blender are Eevee and Cycles. Eevee is more performance-optimized, while Cycles produces images of better quality [20]. This signifies that synthetic datasets may have a different virtual representation of materials in comparison to their real-life counterparts. Domain gaps are further elaborated in the corresponding section.

2.5 Model shortcuts

Model shortcuts occur when a spurious relationship is present in labeled training data, causing a classifier to learn the spurious relationship between the set of exemplars and its label rather than the real relationship. Deep learning models are highly susceptible to this, since these features are usually more heavily correlated with the label than the explanatory features. In optimization problems, the gradient will then be mostly determined by this spurious feature, impacting the training loop. Also, the type of model matters: CNNs are more susceptible to space invariant features. If a spurious feature is thus space invariant, it will impact the gradient even more extremely.

When a spurious relationship is not present during inference, the test performance of the model heavily degrades. This can have severe consequences when this is discovered too late. [19]

To prevent model shortcuts, data should be highly representable of the class, and have diverse non-related features. However, this is not always possible: most pictures of cows are in fact taken on the countryside. This is known as selection bias. Other biases that can cause spurious relations are class imbalance and sampling noise. Sometimes, spurious relations may not even be obvious. In medical imaging, cases have been reported of models learning correlations based on the output format of X-Ray photos. Due to this, model shortcuts should be mitigated as much as possible. Fortunately, a range of difference methods exist. These will be discussed in the related works section

2.6 Grad-CAM

Grad-CAM is a tool that can be used in order to gain more insight into the predictions that are made by convolutional neural networks. When introduced, the tool was revolutionary as it allowed to peek into the inner workings of CNNs. The tool works by computing the gradient of the activations in an arbitrary layer with respect to an arbitrary class output. The gradients are then propagated backwards (without updating the weights), and an average of each gradient for each channel is taken. Next, the activations in each layer are weighed by this average gradient. Finally,

the resulting heatmap is upsampled to the original image size.

This tool has different use cases that all aid to the paradigm of explainable AI. Most notably, Grad-CAM can be used to investigate misclassifications of a model, or to identify model bias. For the first use, the Grad-CAM heatmap of misclassified exemplars is manually inspected, to see which part of the test instance the model (wrongly) attended to. For the second case, one can test for multiple types of bias by looking for spurious correlations. For example, in the original paper, a known biased model was tested that was trained on classifying images as either showing a doctor or a nurse. By manually inspecting misclassifications, the researchers discovered that the model mostly attended to gender features, which were spurious features rather than explanatory features [14].

3 Related Work

Having discussed recent architectures that are able to classify three-dimensional objects and the problems regarding the availability of suitable training data, we will now move on to possible solutions to these problems. In this section, we will look into conducted research that aimed to fix the problems that were posed in the last section. These solutions are mostly related to improving multiple-view classifiers, obtaining more data, reducing the domain gap when pretraining with synthetic datasets, and the prevention of model shortcuts. After having discussed the current state of research in these fields, we will formulate the research question that will be answered in this thesis.

3.1 Multiple-view classifiers

As already discussed in the background, multiple-view classifiers that operate on a multitude of sampled two-dimensional views on 3D objects are generally more accurate compared to classifiers that directly work on the three-dimensional representation of the 3D object [15]. Following this paper, other architectures were developed that also incorporated multiple sampled views from 3D objects.

For example, Qi took several views around an axis of an object, and fed these into a CNN as a well. However, this CNN was only used for feature extraction, as the views were fed in their geometric order into a bilateral LSTM. In this way, relations between the views making use of the order were learned in both directions. Next, the output features of the LSTM were aggegrated and pooled, where after they were fed into another CNN for classification. This method was highly succesful when trained and tested on the ModelNet-40 dataset[12]. Another way that relations between these views can be learned is by making use of an attention mechanism. This was done by Ma. For each view on an object, a view token is created. Next, for each patch of a view, a token is learned. For each hierarchy, the tokens are fed through a transformer, which processes a final output. In this way, the most informative patches per class are learned. To create a shape descriptor, the learned patch features are merged with the view tokens. This approach yielded very high performance as well [9]. Finally, another architecture called Swin-MGNet was also developed, which is based on the Swin transformer. A Swin transformer is a transformer architecture that contains differently sized windows per channel, allowing for more relations between windows to be learned. The Swin transformer outputs a view-level descriptor, which is then fed into a bilateral

LSTM. This approach is almost analogous to the approach taken by Qi, though feature extraction is now performed by a SWIN transformer as opposed to a CNN. After the feature extraction has been performed and the relations between views, the views are aggregated, and a class prediction is made by feeding the view aggregation into a fully connected linear layer. In Qi's paper, the aggregated views were passed through a CNN rather than solely a fully connected layer. The multiple-view SWIN-transformer was also able to obtain a high accuracy on the Modelnet-40 dataset [11].

As already stated in the introduction, past papers have mostly focused on constructing new architectures for classifying 3D objects, while this paper will focus on how much adding one extra view aids in the classification performance of 3D shapes.

3.2 Solutions for data scarcity

Data scarcity can be solved by scraping data from the internet. For example, Yang performed a research where labeled exemplars were scraped from the internet and were divided into groups based on possible label noise. This probability was calculated using feature extraction with an attention mechanism that aimed to extract common features. For different amounts of this probability, three groups were constructed: a 'good' dataset, a 'questionable' dataset and a 'poor' dataset. The poor dataset was immediately discarded, and the 'questionable' dataset was used differently in training in order to have a lower impact on the weights. In this way, existing models could reach a higher performance, as they could learn from more data. This noise could also contribute to model shortcuts, making a model trained on this data less robust. [18].

In order to fully overcome the problem of data scarcity, synthetic datasets can be used. These datasets are created by sampling from a number of parameters, leading to possibly infinite samples. For example, this technique was used by Zheng in order to create a 6-DoF object position estimation dataset. They used BlenderProc in combination with a physics simulation in order to create realistic samples for their dataset [21]. In this thesis, the earlier mentioned Blender application will be used to generate a synthetic dataset.

While synthetic datasets are a solution for data scarcity, they suffer from a phenomenon called domain gap. Solutions for this domain gap will be discussed in the next paragraph.

3.3 Reducing the domain gap

In order to reduce the domain gap, many ways exist. They can roughly be classified in three kinds of methods [4]:

- Shallow methods operating on deep features: these are traditional machine learning methods that regard the extracted features by the convolutional layers as vector. Although they are effective in some cases, the performance increase in models that use these methods is marginal.
- Fine-tuning CNN architectures: a method where parameters of a pre-trained model are unfrozen, and new labeled examples of the target distribution are used as learning exemplars. In this way, the full architecture is adapted to the new domain, while the target distribution may exist of smaller labeled datasets.
- Deep domain adaptation methods: these are architectures that are built on the concept of domain adaptation. These are deep models that work directly on the source and target

distributions. For example, a Stacked Denoising Autoencoders finds common features between source and target distributions for classification tasks.

Additionally, one can bring the source distribution and target distribution closer to each other by operating directly on the target distribution during inference. For example, Carlson created an algorithm that learned the visual effects of a sensor that was used during inference in order to add these effects to synthetic images. This method was more efficient in reducing the domain gap in comparison to methods such as domain randomization [2].

For this paper, we will try to finetune the model during inference on a small real, scraped dataset. Furthermore, traditional computer vision methods will be used for pre-processing exemplars, both during training and inference, in order to manually extract useful features and prevene the classifier from overfitting.

3.4 Mitigating model shortcuts

To mitigate model shortcuts, several methods exist. They can roughly be divided into three classes:

- Data augmentation. Augmenting exemplars can change certain spurious features, causing them to not be correlated anymore.
- Representation learning. Instead of learning a mapping from an exemplar to a label, a domain per class is learned. During inference, the model tries to assign the input to a learned class.
- Post-hoc methods: Models can be finetuned, for example on smaller datasets that are evenly distributed between classes.

In order to identify model shortcuts, several metrics can be used. Most of these metrics split up the exemplars in groups based on general features, and then calculate the accuracy of these groups. This is especially useful in classification tasks with many classes, where some classes may be misclassified more often than in general [19].

In addition to the earlier mentioned methods, other methods to identify model shortcuts have been researched. Müller proposed a method where a Variational Auto-Encoder (VAE) was trained that encodes latent factors from a dataset into latent dimensions. The feature-target correlations are examined, where each feature is tested for its predictiveness of a class. Classes that result in two highly disparate distribution estimates are likely to be predictive features. The most predictive dimensions were consecutively upsampled, which was later evaluated by a human judge for shortcuts. In practice, it turned out that only the top-3 predictive features needed examination in order to mostly eliminate model shortcuts. What was mostly interesting about this research is the fact that the authors mentioned that a human would always be needed for recognizing shortcuts, as machines are solely able to operate on statistics in data [10]. Furthermore, there was a research conducted by Cooke that specifically focused on medical imaging, as explainability is important in this specific sector. They used a technique called *Counterfactual Editing*, where using text prompts medical artifacts could be removed from medical images. In this way, the researchers could investigate whether the model added or removed spurious features. Moreover, incorporating these counterfactual images in the training loop improved model robustness [3].

In this research, we will solely use heatmap-based methods to investigate the model on model shortcuts. Although this technique has been proven to be less rigid than Müller's method, it is still

deemed to be reliable enough, since we are working with a dataset that contains few classes, and the main focus of this research is not placed on detecting model shortcuts.

3.5 Research gap

Based on the problems that were sketched in the previous paragraphs, the following research questions were proposed:

Main question: To what extent does a randomly sampled additional view aid performance in the recognition of physical ambiguous 3D shapes?
Sub question 1: How does the classification performance differ between a single-view and a multiple-view classifier classifying unambiguous and ambiguous synthetic shapes?
Sub question 2: What is the classification performance of a multiple-view classifier that is pretrained on synthetic shapes when presented with physical 3D objects recorded by an arbitrary RGB camera?

As already touched upon in the previous paragraphs, the research question that we are trying to answer has not been explicitly researched before. Although many aspects that make up this research have already been researched, answering the posed research question by pretraining a multiple-view classifier on a synthetic dataset, and later performing inference on real-life objects has not been done in one research. By designing the full pipeline in this thesis, new insights may be gathered that would not have shown up while researching the parts of this pipeline in isolation.

4 Approach

In this section, we will describe our approach which we will use to aid the research gap. In short, we will first develop a synthetic dataset, construct and train a 2D-CNN baseline, construct and train a 3D-CNN model, and finally evaluate the inference performance of the latter model in the real world.

4.1 Generating a synthetic dataset

4.1.1 Motivation behind generating a synthetic dataset

The first step of this research was to create a synthetic dataset. This synthetic dataset will be used to train both the single-view baseline and the multiple-view classifier. We chose this approach because there were no existing datasets suited for this research. Pose estimation datasets containing multiple sampled viewpoints on virtual three-dimensional objects did exist, but determining the uninformative views in this dataset was hard. Furthermore, the amount of uninformative views was not sufficient enough to use for research. There was also another dataset by Google, but this dataset did not contain any ambiguous shapes. To the best of our knowledge, no other datasets existed containing a sufficient number of ambiguous shapes and uninformative views.

4.1.2 Constructing the synthetic dataset

The tool of choice for generating this dataset is Blender. This tool allows both flexibility in creating the dataset contents, but also scalability for generating a sufficient amount of samples, as described in the background. To generate the dataset, the amount of classes had to be determined first, together with the sample size per class. For this research, we needed both ambiguous and unambiguous shapes. Four ambiguous shapes were chosen, each being two pairs which are indiscriminable from certain viewpoints: a cone and a cylinder, and a cuboid and a pyramid. The pyramids in this dataset will always have a flat base, and the cuboid will have the same width and height. In this way, the amount of variable parameters for each class will stay the same. For unambiguous shapes, a sphere and a ring (torus) were chosen.

Next, we had to determine how to create the required views. Ambiguous shapes each needed to have two kinds of views, namely informative and uninformative views. Informative views were created by placing the camera around the object, with an exception of placing it under the object. The uninformative views were created by placing the camera under the object. The camera was set to always face the object, which aided in programming the viewpoints. The parameter range of the camera for informative views and uninformative is shown in table 1. We decided to create a bounding box in the middle of the coordinate grid, wherein the object would be rendered. In this way, the camera would never end up in the object, causing unusable views. The same method was used for placing the light source.

	x	**y**	**z**
Informative	2.0 – 10.0	2.0 – 10.0	0.0 – 10.0
Uninformative	0.0 – 0.5	0.0 – 0.5	-2.5 – -3.0

Table 1: Parameter ranges for the camera

Next, the parameter space for each class had to be chosen. It was important that the parameter space of the unambiguous views of paired classes was always identical. Otherwise, a model shortcut could appear. For each class, the color was sampled from a grayscale, along with the amount of metallicness. We chose to sample from a grayscale as adding colors would add too much variety, causing the model to need more samples in order to generalize. Instead, during inference, the images that are captured will be converted to grayscale. Furthermore, a light location was sampled from coordinates that were outside the bounding box that the objects could take. This bounding box differs for informative and uninformative views. In uninformative views, the camera is pointed under the object, which means the light source must also be placed under the object. Thus, the parameter space where the light can be placed differs with respect to informative views. In addition to this, the intensity of this light source is also sampled. The values of these parameters are sampled from a continuous uniform distribution in a predetermined range. In table 2, the parameter range of the color, metalicness and light intensity are shown. In table 3, the parameter range for the x, y and z-coordinates of the light are shown for both informative and uninformative views.

Finally, the lengths and radii of all objects had to be constructed. Especially here, keeping the parameter spaces for uninformative views identical proved to be a challenge due to the way Blender generates each object. First, the linked classes of the pyramid and cube were constructed. This proved to be the greatest challenge. In Blender, a cube is constructed by either setting the scale of each side, or by setting its size. Constructing a pyramid was even harder, since there is no built-in

Parameter	Range
Color	0.15 – 0.6
Metallicness	0.0 – 1.0
Light Intensity	1000 – 10000

Table 2: Parameter ranges used by each object for sampling

	x	y	z
Informative	2.5 – 5.0	2.5 – 5.0	1.0 – 5.0
Uninformative	2.5 – 5.0	2.5 – 5.0	-2.0 – -5.0

Table 3: Parameter ranges for the light source

pyramid class. Instead, a cone with four vertices was used. The problem here was that the cone made use of a radius parameter. To make sure that the parameter spaces aligned, a parameter was sampled for the scale of each object. For the cone, this scale was converted to a radius by dividing by $\sqrt{2}$. In this way, the parameter spaces seemed to be equal. However, only after pre-training the 2D CNN multiple times, it was discovered that there was a model shortcut in this linked class. It turned out that cones using four vertices were generated with the sides along the axis, while cubes were generated perpendicular to the axes. Rotating each pyramid by $\frac{1}{4}\pi$ solved this issue. See figure 2 for an illustration of the identified shortcut.

(a) An uninformative view on a cube

(b) An uninformative view on a pyramid

Figure 2: Two uninformative views. When the rotation of the pyramids is kept at its default value, the orientation differs by 45 degrees from the cubes, causing a model shortcut

The other classes were easier to model. The linked classes of cone and cylinder both had a height and a radius parameter, thus they were easy to keep identical. In addition, these shape are made generated discretely by an amount of vertices for the circular part, but this parameter was also easy to keep the same. The two unambiguous shapes, the sphere and the ring, required no additional care. The sphere is sampled according to its radius, and the ring is sampled according to its outer radius and inner radius. The parameters that the shape sizes are sampled from can be seen in table 4. A visual representation of the dataset generation process can be seen in figure 3.

From these chosen parameter ranges, we can deduct a formal definition for each shape class. These will be as follows:

Shape	Scale	Radius 1	Radius 2	Height
Cube	0.3 – 1.0	–	–	–
Pyramid	0.3 – 1.0	–	–	0.3 – 3.0
Cylinder	–	0.3 – 0.5	–	0.3 – 3.0
Cone	–	0.3 – 0.5	–	0.3 – 3.0
Sphere	–	0.3 – 2.0	–	–
Ring	–	0.3 – 1.0	1.2 – 2.0	–

Table 4: Parameter ranges for size

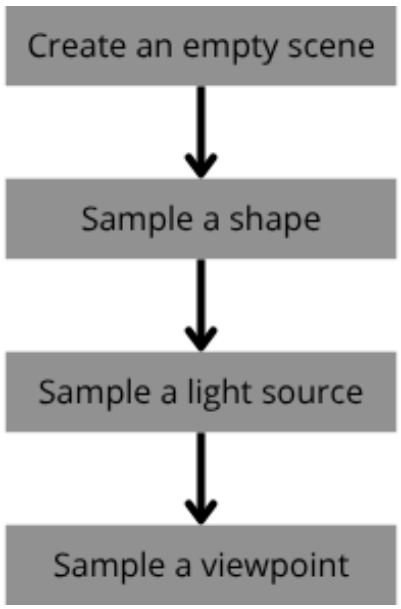


Figure 3: A summary of the dataset generation process

- A cube with scale $s \in [0.3, 1.0]$ is defined as: $CU(s) = (x, y, z) \in \mathbb{R}^3 || x | \leq s/2, |y| \leq s/2, |z| \leq s/2$.
- A pyramid with scale $s \in [0.3, 1.0]$ and height $h \in [0.3, 3.0]$ is defined as:

$$P(s,h) = \left\{ (x,y,z) \in \mathbb{R}^3 \;\middle|\; \begin{array}{l} 0 \leq z \leq h, \\ |x| \leq \frac{s}{2}\left(1 - \frac{z}{h}\right), \\ |y| \leq \frac{s}{2}\left(1 - \frac{z}{h}\right) \end{array} \right\}.$$

.

- A cylinder with radius $r \in [0.3 - 0.5]$ and height $h \in [0.3, 3.0]$ is defined as: $CY(r,h) = \{x, y, z\} \in \mathbb{R}^3 | x^2 + y^2 \leq r^2, |z| < h/2$.
- A cone with radius $r \in [0.3 - 0.5]$ and height $h \in [0.3, 3.0]$ is defined as:

$$K(r,h) = \left\{ (x,y,z) \in \mathbb{R}^3 \;\middle|\; \begin{array}{l} 0 \leq z \leq h, \\ x^2 + y^2 \leq \left(r\left(1 - \frac{z}{h}\right)\right)^2 \end{array} \right\}.$$

.

- A sphere with radius $r \in [0.3 - 2.0]$ is defined as: $S(r) = \{(x, y, z) \in \mathbb{R}^3 | x^2 + y^2 + z^2 \leq r^2\}$.
- A torus with inner radius $r_i \in [0.3, 1.0]$ and outer radius $r_o \in [1.2, 2.0]$ is defined as: $T(r_i, r_o) = \{(x, y, z) \in \mathbb{R}^3 | (\sqrt{x^2 + y^2} - r_o)^2 + z^2 \leq r_i^2\}$.

Each class consists of 1000 virtual samples. For unambiguous shapes, these are 1000 informative views, since unambiguous shapes only have informative views. Ambiguous shapes have 500 informative samples, and 500 uninformative samples. The dataset was split in a training, validation and test set, each having a proportion of 0.8, 0.1 and 0.1 respectively. The choice for these numbers was arbitrary, since sample size was not a problem due to the abundance of virtual data. The resolution of the images is 1920x1080. We chose to render the images using the Eevee engine, which is the standard engine for Blender 5.0.1 in Python. We chose to use Eevee as generating a six-class dataset using Eevee already takes half an hour. Moreover, using Cycles causes the dataset generation speed to slow down, while there was no visible increase in image quality. A longer running time was thus undesirable. Since the image quality did not improve, the choice of rendering engine is not deemed to be related to the size of the domain gap. Some examples of the samples that were created can be seen in figure 4.

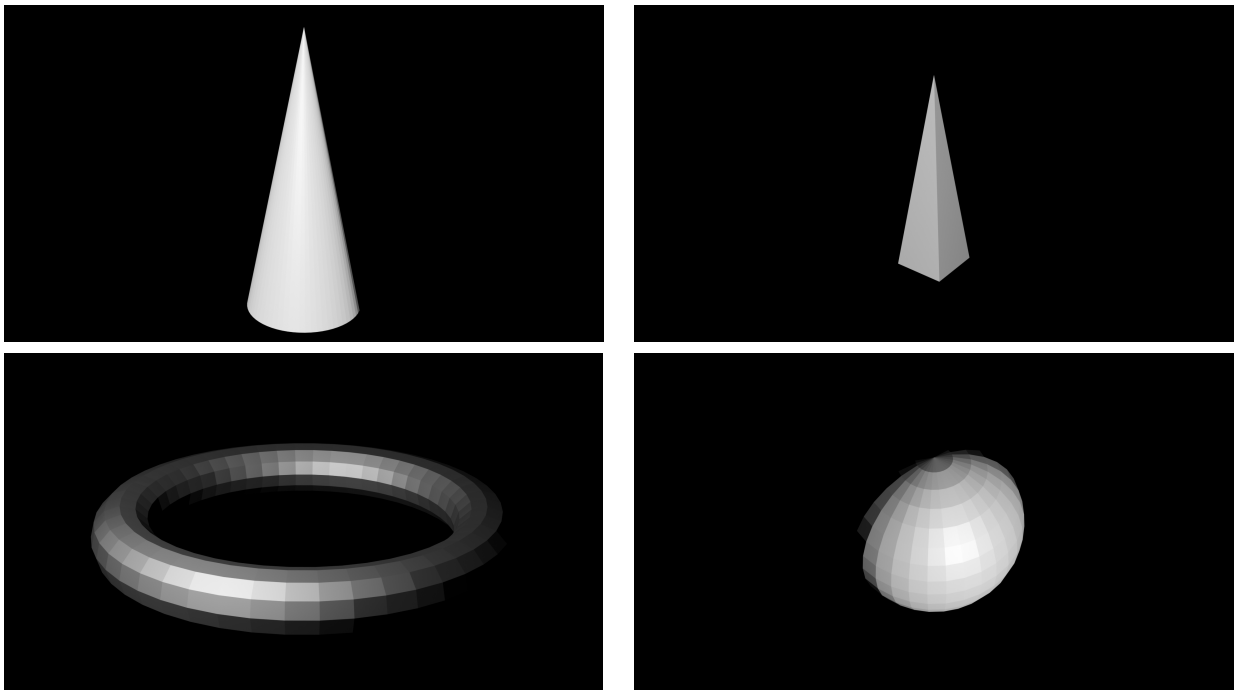


Figure 4: Some examples from the dataset

4.2 Developing a baseline

To test if uninformative views are indeed unable to yield enough information about a geometric shape on their own, a single-view classifier will be trained as a baseline. This classifier will be trained on the full dataset, thus on a mix of informative and uninformative views. After training has been completed, inference will be performed for informative and uninformative views individually.

We hypothesize that inference performance for informative views will be very high (90%+), while inference performance for uninformative views will be around 50%, as we suspect that the model has to choose between a cone and a cylinder or a pyramid and a cube. To check whether the accuracy of the uninformative views can indeed be attributed to random guessing, we will compare the accuracy with the expected accuracy of 50%, and check whether this difference is statistically significant using a one-sided Z-test.

4.2.1 Checking for model shortcuts

After performing a manual inspection of the rendered samples, no obvious model shortcuts were discovered. However, model shortcuts can be more subtle. To check for this, we have to evaluate our trained models both quantitatively and qualitatively. To do this, we will train a baseline.

Even though all rendered samples were inspected manually, model shortcuts can still appear. To check for this, we will adopt the same paradigm that was also used to check whether the uninformative accuracy can indeed be attributed to random guessing. If the reported accuracy is higher than 0.5 and statistically significant, there can be two possible explanations:

- Uninformative views are informative because they are inherently so. There is some factor that is also present in the real world that allows to determine the shape.
- Uninformative views are informative because there is a model shortcut. In this case, there is a factor that is present in the synthetic data, though not in the real world, that allows the model to determine the shape.

Not only will the performance be evaluated quantitatively, but also qualitatively. We plan to use Grad-CAM in order to see to which parts of uninformative views the model attends to. We will check the final three convolutional layers, since later layers usually attend to specific explanatory parts of images. If a model shortcut was indeed present, we would expect to see an extreme amount of attention to certain parts of the test images. This attention can then be expected in order to see whether a real model shortcut is present, or if the model is using a feature that can also be found in the real-world test distribution.

4.2.2 Architecture of choice

The single-view classifier of choice in this research is AlexNet [6]. AlexNet was chosen, since it has an acceptable performance, while being lightweight on modern GPUs. At the time of the paper publication, AlexNet reached a top-1 accuracy of 62.5% on the ImageNet dataset. This dataset contains a big amount of samples distributed over 22000 categories. Since we are working with a simple 6-class classification task with sufficient training data, AlexNet was deemed capable enough to learn this task. To aid training, a pretrained AlexNet model from the PyTorch library was used. In order to fit the AlexNet model on a six-class classification task, *linear probing* was used. This means that the final fully connected layer was replaced with a layer that mapped to six nodes.

The model was trained using PyTorch too, using SGD as optimizer, with a learning rate of 0.01, a momentum of 0.9, and a weight decay of 0.0001. The training loop ran for 10 epochs. This was deemed to be enough, since the initial steep decrease in loss flattened after epoch 5. Since the accuracy scores on the test set were very high, we decided not to try out other parameters values for the learning rate, momentum and weight decay. Moreover, using a more complicated model was

also not tested since this model yielded a high performance, in addition to the fact that using a more complicated model requires more computational resources, slowing down the training process.

4.3 Determining parameter spaces for uninformative views

If the posed hypothesis in the previous paragraph is correct, it means that we should be able to determine the parameter space for uninformative views.

To do this, we will train the same classifier using the same hyperparameters on different versions of the dataset. Instead of sampling the camera parameters, they will now be fixed. For each tested combination of camera parameters, a new dataset will be generated. First, we will qualitatively evaluate whether the used camera parameters do not yield inherent uninformative views, for example when the background is fully occluded by the object. In this case, also a human would not be able to determine whether we are dealing with a square or a round base. Next, we will train the classifier on the dataset, and note down the accuracy for uninformative views, and check whether this accuracy is higher than 0.5 and statistically significant. If so, it means that the chosen camera parameters do not yield uninformative views. If not, informative views are created. By systematically testing combinations of parameters, we can determine the parameter space that leads to uninformative views. This validates that the chosen parameter ranges for uninformative views only allows combinations of parameters that yield uninformative views. Furthermore, by knowing this about the dataset, other researchers can adapt the dataset more easily for their own research.

4.4 Multiple-view CNNs

After interpreting the performance of the single-view classifier and approximating the parameter space that yields uninformative views, a multiple-view classifier can be trained.

4.4.1 Creating and analysing new types of exemplars

For this multiple-view classifier, new exemplars have to be created. Three sets of new exemplars were created, where each exemplar consisted of two views. For each set, two random images from the same class were combined. In the first set, two informative views were combined. In the second set, an uninformative and an informative view are combined. In the third set, two uninformative views are combined. The first set serves as a baseline, to see whether the model works as good as the single-view classifier. The inference performance on the second set will learn us more about whether the model is able to use the informative view for classification. The third set will learn us about whether two uninformative views yield more information than one view. If so, the model may be able to exploit shortcuts, such as lighting information. This is summarized in table 5.

Also for this model, qualitative analysis will be performed using Grad-CAM. Here, Grad-CAM will be used to see to which image, and which image regions most attention is paid to. We are especially interested in using Grad-CAM on the second and third set. On the second set, we can test whether the model is indeed using the informative view. On the third set, we can test for possible model shortcuts by looking for areas that the model attends to strongly. However, since three-dimensional activations are very hard to understand, we will only look at the later layer, since the temporal structure has already collapsed in this layer, aiding interpretation.

View type	Purpose
informative + informative	baseline
uninformative + informative	view selection
uninformative + uninformative	dataset correctness

Table 5: All types of exemplars that are used to train and test the multiple-view CNN, and their purpose

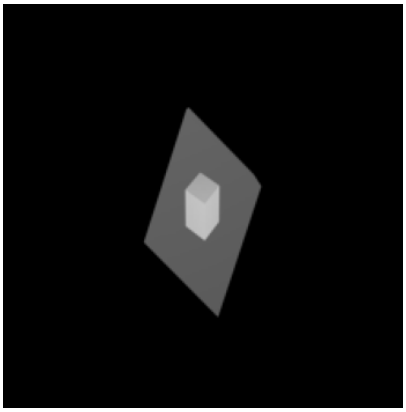


Figure 5: A visual representation of a 3-dimensional input. An uninformative view and an informative view have been stacked on top of each other. By pooling in the vertical (view) dimension, relations between views can be learned

4.4.2 Requirements for a suitable architecture

In order to perform our proposed research, we need to choose a CNN architecture that can work with multiple views. Naturally, we would use a 3D CNN for this task. A 3D CNN is a CNN that takes an extra dimension as its output. Usually, this dimension is time: these models are mostly used for video classification, where 2-dimensional frames are stacked according to their temporal order. In our research, we will adopt this architecture to work with multiple views rather than with multiple frames of a video. This should also work, since 3D CNNs for video classification learn relations between frames by pooling along the temporal dimension. In our proposed application, pooling would happen across views, allowing the model to learn relations between multiple views of one object class. An example input is shown in figure 5.

4.4.3 Architecture of choice

The architecture of choice for this task is Resnet-18 [5]. It consists of 18 layers, which are divided in 5 convolutional blocks. The first convolutional block downsamples the spatial dimension by halving it. This is useful for extracting coarse details from input frames, such as edges and textures. The second block keeps both the existing spatial and temporal dimensions intact, which is beneficial for extracting finer features among the channels. The third, fourth and fifth block downsample the input both spatially and temporally, allowing the features of all dimensions to be integrated, allowing for a rich representation.

After each convolutional block, residuals are added. These are the gradients of the network

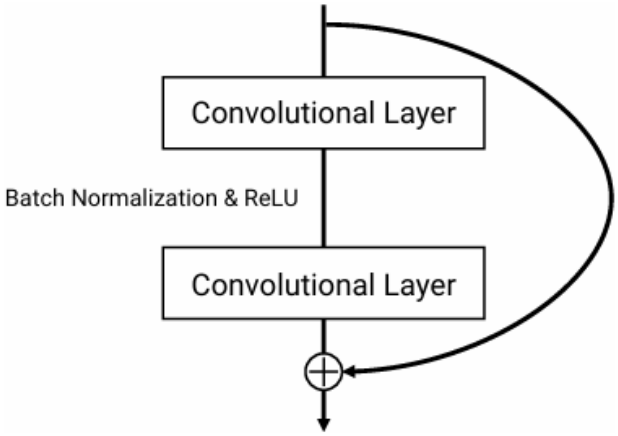


Figure 6: The parts of a convolutional block in a Resnet-18 network. Each block consists of two convolutional layers, with a Batch Normalization layer and a ReLU activation in between. The curved arrow is the flow of the residual gradients, which are added to the processed gradients of the convolutional layer. In this way, vanishing gradients are prevented. This figure is copied from Hara et al. [5].

before they were changed in the convolutional block. By adding these gradients to the new gradients, the problem of vanishing gradients is solved, allowing deeper networks to effectively keep extracting features. This is illustrated in figure 6, adapted from the Resnet-18 paper [5].

Finally, a global average pooling is applied over the output after the fifth layer. This means that the spatial and temporal information are destroyed. This is not a problem, since these features have already been captured in each layer. After this global average pooling has been applied, the features can be fed into a fully connected neural network, that learns to map the features of each input to their respective class.

We chose this architecture since it is known to capture features along the temporal dimension very well. Second, its general classification performance is high, and third, the model can be run on our test setup using the same batch size as the baseline. At the time of publication, the top-1 classification accuracy on the Kinetics dataset was 68%. This dataset consists of 400 types of actions. To adapt this classifier for our research, we used *linear probing* in the same way as we did for the baseline: the last fully connected layer was replaced with a layer that had an output size of six. To speed up training, we also used a pretrained model here.

4.4.4 Training the model

The model was trained with PyTorch, using the same parameters that were also used in the baseline. We also chose to train this model for 10 epochs, since initial training loss decreased fast, but plateaud in later epochs. Again, the accuracy of this model was very high, and thus we decided not to try out other models. Even more complex models would probably not have been possible, since the VRAM use while training this model was already reaching the maximum VRAM that the GPU had.

4.5 Domain transfer to real-world

In the final section of this research, we will evaluate the performance of classifiers that are pretrained on a synthetic dataset when used for inference in the real world. Both the single-view classifier and multiple-view classifier will be evaluated on a set of real-world objects which have shapes that resemble the classes used for pretraining.

We expect that inference in the real-world will be worse, since there is a domain gap. Moreover, we have pretrained our classifier using only a black background, while in the real world there could be different types of background. In addition to this, different backgrounds could also contain highly noticeable lines, which the model could misinterpret as belonging to the shape of interest, leading to wrong predictions. In order to reduce the domain gap in these cases, domain adaptation methods could be used. We will first try to finetune our classifier on the scraped dataset, to see whether a small dataset is sufficient in order to learn the classifier the new distribution it will perform inference on. Next to this, traditional computer vision methods and filters can be used to make closer objects stand out more, and reduce the influence of the background and other non-important lines. Finally, we will attempt to measure the domain gap by blending a synthetic image with a real image, such that the shapes in both pictures align. By performing inference on pictures created with different blending factors, the domain gap becomes measurable.

4.5.1 Methods of evaluation

To meaningfully compare the performance of models, we will evaluate the model accuracy on a small scraped, manually annotated dataset containing real-world images of the shapes we wish to classify. In addition to this, we will also record the cross-entropy loss of each used configuration, since this measure is more informative. This is caused by the fact that cross-entropy loss also incorporates the confidence of the predictions made by the model: confident incorrect predictions are penalized heavier than unconfident incorrect predictions. Since the scraped dataset has a small size, cross-entropy loss is deemed to be more informative in this case when it comes to comparing models. The formula of cross-entropy loss is defined as:

$$L = -\sum_{k=1}^{K} y_k log(p_k) \tag{1}$$

In this formula, K is the amount of classes, y_k is 1 if the prediction is correct and 0 otherwise, and p_k is the predicted probability of class k. From this formula, we can see that indeed both correctness and confidence are awarded with a lower loss.

4.5.2 Creating and testing a scraped dataset

To create a scraped dataset, a Google Images search was performed, using the class category as query. From the results, images were picked that were deemed to be representative. The conditions for each image were that they represented some 'real object', which one can encounter in their daily life. Furthermore, the background could not be too cluttered. An even background was acceptable, but backgrounds that contained other shapes were not used, for the reason that our classifier was not trained to classify the most prominent shape, but rather a single shape from an input. Another challenge that was encountered were the found images on Google Images when searching solely

using the class name as input. Since some shapes are only found in a limited number of classes, these items have an identical name to the shape name. This was mostly present in the pyramid class, were looking up "pyramid" yielded pictures of the Egyptian pyramids. A similar phenomenon was observed for the words "cone" and "ring", which yielded mostly traffic cones and jewelry respectively. On the other side, some class names are mostly used in a geometrical context, such as the words "cube" and "cylinder". The results mostly showed perfect renderings of these shapes, which can probably be attributed to the fact that they are used to teach children the naming of shapes. After performing a manual internet search, sufficient representative samples were obtained.

For each class, five samples were chosen. This was done with the intention that an 80%/20% training split would mean four training exemplars, and one test exemplar. For each exemplar, the source was recorded. In the end, this meant that a smaller dataset with 30 instances was created. Note that all views used in this part of the research are informative, since we are researching domain transfer instead of shape ambiguity. This also means that evaluation will only be performed on the multiple-view classifier trained in the II-configuration, meaning that it has been presented two informative views during training. The reason that only this classifier will be tested in this part of the research, is that the results will likely apply to the other classifiers as well, since we are working with an almost identical parameter space, apart from the camera range. Some examples of the collected data are show in figure 7.

(a) An image belonging to the sphere class of the scraped dataset

(b) An image belonging to the ring class of the scraped dataset

Figure 7: Two samples from the scraped dataset. As can be expected, this dataset contains more variation in background, materials and lighting compared to the original dataset

4.5.3 Reducing the domain gap: finetuning

After performing inference on the scraped dataset, the initial results were very poor. Due to this result, a number of methods were tried with the aim of reducing the domain gap.

First, finetuning was tried with the scraped dataset. Due to the small size of the dataset, we did not expect a great result, since the sample size is likely too small to allow the model to generalize to the new inference distribution. However, since this method has a high performance and has been mentioned throughout multiple papers ([13], [4], [19]), we did decide to try it. Finetuning was performed in Python using the PyTorch library. We decided to decrease the learning rate by a

factor of 10, since we were trying to capture finer features of the inference dataset, rather than the concept of shapes, which was done during pretraining. The final learning rate was set to 0.001. Also, the batch size was reduced from 256 to 5, since the dataset was way smaller. It was decided to use the full scraped dataset as the training dataset, since it was already very small. This means that the training loss will be used for evaluation. Only if the training loss after finetuning would have decreased significantly, more scraped exemplars could be retrieved in order to serve as a test set. As mentioned earlier, the training loss is measured using cross-entropy loss.

4.5.4 Reducing the domain gap: pre-processing

Second, pre-processing each image in both the training and inference process by means of traditional vision filters was tried. First, each image was converted to grayscale, where after a Gaussian Blur was applied. This was done in order to smooth high-frequency features, which are likely to be uninformative for the shape classification task. After that, filters for edge extraction were used, since we deemed these to be the most prominent feature in a shape classification task. The filters that were researched were the Sobel filter, the Canny filter, and the Laplacian filter.

The Sobel filter is a kernel that is applied to an input image. The kernel is defined for the x and y directions separately:

$$Sobel_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

$$Sobel_y = \begin{bmatrix} -1 & -2 & -1 \\ -0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3}$$

To find the total magnitude of the edges, we calculate the magnitude per pixel using $\sqrt{M_x^2 + M_y^2}$ [7].

The Canny filter is not a real filter, but rather a small algorithm that applies multiple filters for edge extraction. It first converts the image to grayscale, applies a Gaussian blur, and then applies the Sobel filter. This seems analogous to what we did in the previous paragraph, though the Canny algorithm not only calculates the magnitude per pixel, but also the gradient orientation. Using this gradient orientation, non-maximum edge suppression is performed. In non-maximum edge suppression, the positive and negative gradient of each pixel are compared with the positive and negative gradients of its neighbors. If the magnitudes are smaller than those of its neighbors, the magnitude is set to zero. In this way, false edges are removed. Finally, a process called hysteresis thresholding is performed. This means that magnitudes above a higher threshold are kept, magnitudes below a lower threshold are discarded, and magnitudes that fall in-between the thresholds are only kept if they are connected to an edge above the higher threshold [7].

The last filter that was tried is the Laplacian filter. The Laplacian is the second derivative of the multi-valued function that describes the edge intensity of an image in both x and y directions. Formally, this can be noted as:

$$L(x, y) = \nabla^2 I(x, y) = \frac{\delta^2 I}{\delta x} + \frac{\delta^2 I}{\delta y} \tag{4}$$

In order to make this more efficient, this function can be discretized in the form of a kernel. This kernel is defined as follows:

$$Laplacian = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{5}$$

After applying this kernel, there is no need to calculate magnitudes like was needed with the Sobel filter. This filter finds edges in any orientation. However, it is sensitive to noise.

All filters above are implemented in Python using the OpenCV library. This allows for an efficient image pre-processing, which saves time in the training loop. We expect the performance of these models to be better than the default model, since it allows the training and inference distribution to be more similar to each other. However, predicting the quantity of performance increase is hard, since we cannot exactly deduce the features that the CNN has learned. If the CNN also makes use of edges to determine the shape present in the input, the increase could be significant, as edges are more prominent in both the training and inference exemplars.

4.5.5 Measuring the domain gap: inspection using blended images

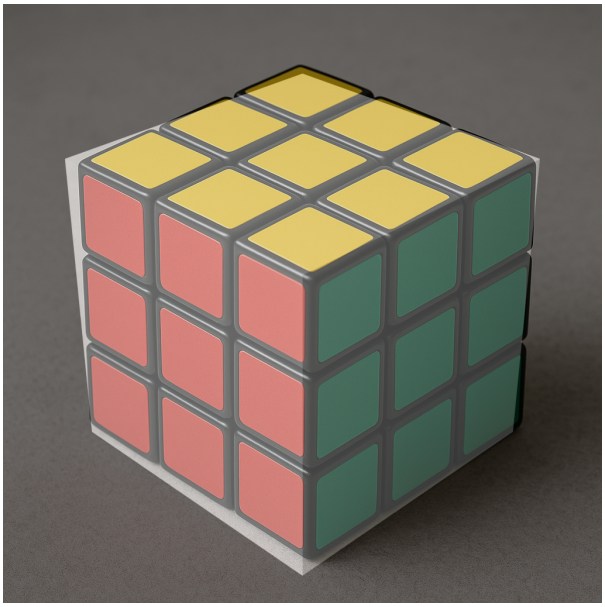
Third, we researched to what extent images can deviate from the dataset style before the classifier cannot classify them anymore. To do this, we chose two images from the scraped dataset, and tried to align a synthetic shape such that its edges aligned with the edges of the real shape. In this way, we could specify a certain blending factor, and research the extend of the domain gap: until what value of the blending factor would the classifier give the correct result? The blending factor lies within zero and one, with zero showing the real image fully, and one showing the synthetic image fully.

To create such exemplars, we had to create a specific synthetic shape that lied within our parameter space. This turned out to be challenging, since exactly aligning the images was difficult. In the end, we found an extra parameter that could be used in Blender to ensure that the edges aligned, which was focal length. Although adding this parameter meant that we would leave our parameter space of the synthetic dataset, it was chosen to do so since it significantly improved the quality of the alignment. If this factor caused the model prediction to be incorrect at a blending factor of zero, it would mean that not accounting for camera differences in the synthetic dataset causes poor predictions in the real world.

In the end, two instances were created. Both instances belonged to the cube class, and can be seen in figure 8 What can be observed is that in both figures, the shapes mostly overlap. However, the alignment is not perfect in both instances, with the ice cube having a worse alignment. This shows the challenge that was encountered when trying to align synthetic images with real images. This challenge can likely be attributed to the high number of parameters that made up a synthetic shape.

5 Experiments

In this section, we will present the results found in the experiments that were described above. The results will be analysed in order to deduce where they originate from, and what they mean with respect to the posed research question.



(a) A synthetic cube blended with a Rubik's cube. What can be seen is that most edges line up almost perfectly.

(b) A synthetic cube blended with an ice cube. What can be seen is that the edges do not line up perfectly, especially at the bottom. Still, the shapes mostly overlap

Figure 8: Blended images generated with a blending factor of 0.5.

5.1 Evaluating baseline performance

5.1.1 Quantitative experiments

The inference accuracy of uninformative views is compared using a one-sided Z-test for a proportion to determine whether the model is classifying uninformative views randomly, or whether its accuracy is too high in order to be attributed to guessing randomly. The reported test accuracies for each view type, along with its sample size, can be seen in table 6.

	accuracy	**sample size**
Informative	0.98	409
Uninformative	0.55	191

Table 6: The reported test accuracy for each test instance, split among informative and uninformative views. The informative accuracy is significantly higher than the uninformative accuracy

We make two hypotheses, which we will test using the one-sided Z-test:

- $H_0 : \pi_{uninformative} = 0.5$
- $H_a : \pi_{uninformative} > 0.5$

The chosen value of α is set to 0.05. Performing this test gives a test statistic of $p = 0.17$. Since $p > \alpha$, we fail to reject the null hypothesis. From this we conclude that there is not enough evidence that the reported accuracy score for uninformative view is not the result of random guessing.

Since the observed informative accuracy is high, and the observed uninformative accuracy is low, we can conclude that the chosen AlexNet model is able to capture the structure of the dataset, but fails to accurately classify uninformative views.

5.1.2 Qualitative experiments: using Grad-CAM

From our quantitative experiments, we know that uninformative views do not give enough information to the model. In order to be certain that the baseline model extracted correct features, we qualitatively evaluated a sample from the dataset, and inspected which areas the model attends to. This was done for both informative and uninformative views. The tool of choice was Grad-CAM.

After inspecting both uninformative and informative exemplars in multiple layers, we were able to attribute a coarse function for convolutional layer 3 and 4. Note that Grad-CAM only highlights gradients that mostly attributed to the model's decision. By inspecting multiple exemplars, we hope to generalize the activations per layer, but it must be noted that the sample is very small in relation to the dataset, and thus the generalizations may not fully be correct. This is the consequence of using a deep-learning model, where we may not even be able to attribute a specific function to a layer.

5.1.3 Qualitative experiments: Heatmaps in layer 3

In layer 3, it seems like edge detection is mostly taking place. This can be seen in figure 9. We can see that the gradient with the respect to the prediction is mostly around the edges of the object. It makes sense that the model would extract this feature, since the edges can give information to distinguish between shapes. This can especially be seen in the two lower pictures, where we see two shapes that mostly consist of round edges. For the ring, the model attends strongly to the curves in the middle of the object, which is what makes it distinct from the sphere. For the sphere, the edges are mostly attended to, especially the right-bottom part or the left-top part. We don't know why the models is doing this.

For uninformative views, we can also see that the model attends to the edges, but less strongly when compared to informative views. Examples are shown in figure 10. This may have to do with the fact that the model does not know where to attend to, as the uninformative views inherently do not have information in them. This may cause the model to instead focus on the informative views and learn their features, as this minimizes the loss the most, which is the learning objective of the classifier.

5.1.4 Qualitative experiments: Heatmaps in layer 4

In layer 4, another pattern was spotted in the heatmaps of the gradients. Here, the model seems to attend more to the object as a whole, and less to the background. However, there are some exemplars where there is still more attention to the edges rather than the whole object. This is a pattern that is seen across both informative and uninformative view. For examples, see figure 11 and figure 12. It seems like layer 4 is responsible for detecting the object as a whole with respect to the background.

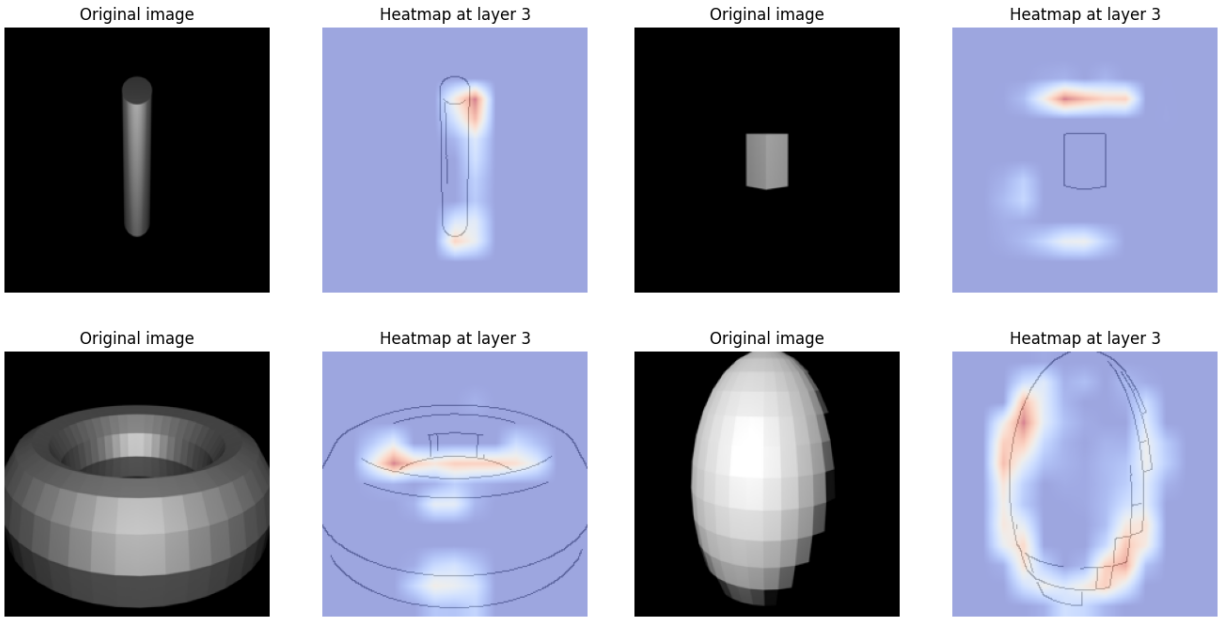
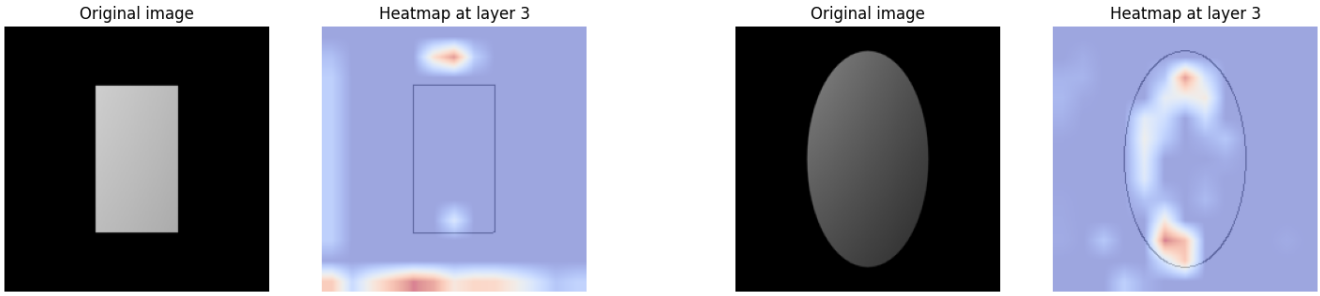


Figure 9: Heatmaps for different informative views at layer 3. We can observe that the high gradients are mostly near the edges of the objects.



(a) An uninformative view on a cube

(b) An uninformative view on a cylinder

Figure 10: The heatmaps of two uninformative views. Although we can see that edges are mostly attended to, there is also attention to other unrelated parts.

5.2 Determining the parameter space for uninformative views

To find the parameter space for uninformative views, we generated the dataset using different fixed values for the uninformative view points. For each tested combination of values, we tested the dataset using the corresponding test dataset and reported the classification accuracy for uninformative views. We tested this accuracy for statistical significance from random guessing using the same test as in section 5.1.1. If we failed to reject the null hypothesis, we deemed the views to be uninformative, otherwise we deemed the views to be informative. The values that were tried out and their corresponding uninformative view accuracy are shown in table 7. Empty entries for the uninformative accuracy column means that the values were not uninformative, because manual inspection of the dataset samples showed that edges of the shape were shown, making the

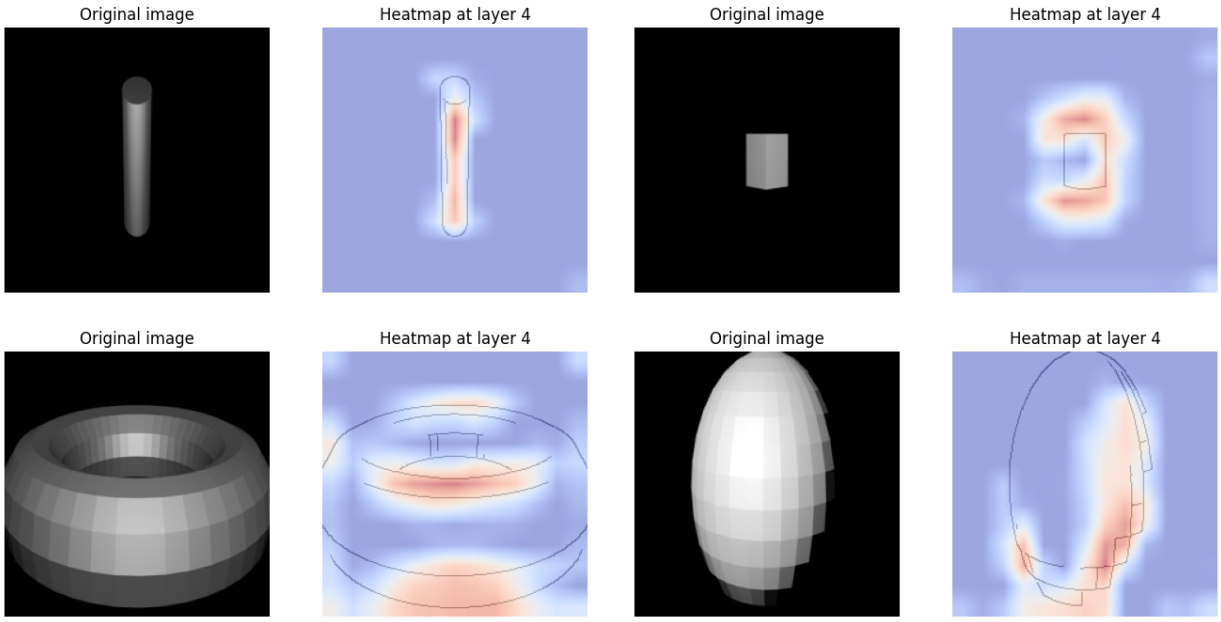
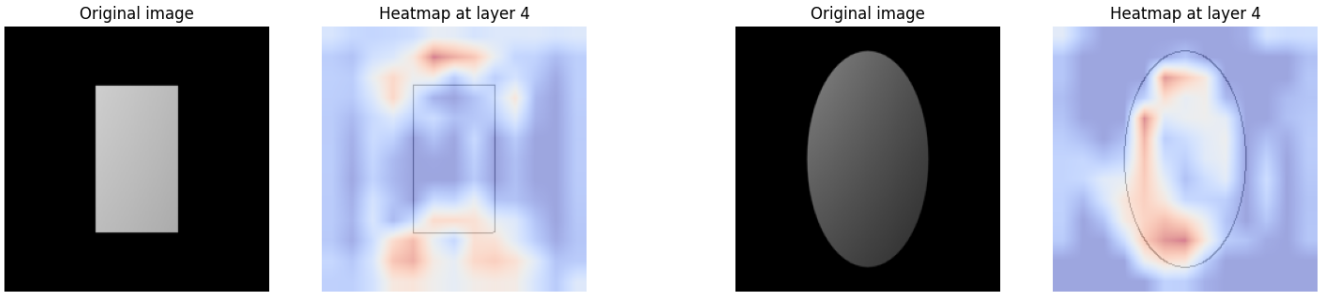


Figure 11: Heatmaps for different informative views at layer 4. We can observe that the high gradients are mostly at the objects as a whole, but some edge detection is still taking place.



(a) An uninformative view on a cube

(b) An uninformative view on a cylinder

Figure 12: The heatmaps of two uninformative views. In the cube example, there is mostly attention to the edges, but in the cylinder example there is mostly attention to the whole object

views informative by definition. In this case, significance is indicated by yes. It could also be the case that manual inspection showed that there were exemplars where the camera was too close to the object, making the views unusable as only surface was shown.

From this table, we can deduct a rough estimate of the space for uninformative views. Whenever the X and Y coordinate are 0, the uninformative accuracy does not change when Z is varied. This means that keeping the camera in the middle of the object means that a view is always uninformative. However, when we start to slightly shift to the left or to the right by setting the X and Y coordinate to 0.1, we immediately lose all ambiguity in the views. This means that there are probably some samples where the edges are revealed, as the uninformative accuracy suddenly increases.

X	Y	Z	Uninformative Accuracy	Significant
0	0	-2.5	–	No
0	0	-3	0.55	No
0	0	-4	0.53	No
0	0	-5	0.54	No
0.1	0.1	-2	0.58	Yes
0.2	0.2	-1	–	No
0.2	0.2	-2	0.64	Yes
0.5	0.5	-1	–	Yes

Table 7: Camera viewpoints that were tested out and their associated uninformative accuracies and significances.

5.3 Evaluating multiple-view classifier performance

5.3.1 Quantitative experiments

To evaluate the multiple-view classifier, we will evaluate the accuracy in an analogous way to the single-view classifier. However, since we are training and testing on three different configurations, we will obtain three different accuracy scores. Rather than evaluating by splitting between informative and uninformative views, we will evaluate per class for each configuration. The results of this evaluation can be seen in table 8.

Classes	**II**	**IU**	**UU**
Cube	1.00	1.00	0.74
Pyramid	1.00	0.98	0.82
Cylinder	0.98	1.00	0.30
Cone	0.98	1.00	0.54
Sphere	1.00	1.00	1.0
Ring	1.00	1.00	1.0
Average	1.00	1.00	0.60

Table 8: The reported test accuracy for each test configuration, split among classes. The accuracy for case II and IU is significantly higher than for case UU.
II: Informative + Informative, IU: Informative + Uninformative, UU: Uninformative + Uninformative.

When we analyze these results, we immediately notice that the UU configuration performs significantly worse than the other two configurations. This was in line with our expectations. Since II and IU perform well, we know that the model is both able to capture the relevant features, but also extract relevant features from exemplars that contain both an informative and an uninformative view. What can be noticed when examining the UU accuracy, is that the unambiguous shapes score similarly to the unambiguous shapes in the other configurations. Furthermore, we also see that the classification accuracy for uninformative views in ambiguous objects is not evenly spread. For example, we observe that cubes and pyramids are classified correctly quite a few times, while the prediction accuracy of cylinders and cones is poor and as expected. The classification accuracy of cubes and pyramids is higher than expected, which may indicate a possible model shortcut,

or the model's ability to integrate two views uninformative views of this class in order to get a sensible conclusion about the class more often than average. However, since the accuracy is not as high as these classes' performance in the other configurations, such a phenomenon may not always be possible. In order to find out where this interesting result comes from, we will inspect it more closely in the next section.

5.3.2 Qualitative experiments

To inspect how the model comes to its decisions, we will qualitatively inspect its predictions using an adapted 3D-version of Grad-CAM. In this section, we will only look at the predictions made in layer 4, after the second convolutional block. The reason for this is that later layers give more information, and in the multiple view case it means that information of both views has been integrated.

When performing an initial examination of the test dataset, we see that the learned filters are similar to the filters learned in the baseline. In the last layer, we mostly observe edge detection, but also a modeling of distinctive features, such as the tip of a cone. For the unambiguous shapes, two examples are presented in figure 13. For ambiguous shapes, the combinations of informative views are shown in 14, the combinations of an informative and uninformative view in 15, and finally the combinations of two uninformative views in 16.

(a) (b)

Figure 13: The activations of two unambiguous shapes. We can see that the model attends mostly to one shape as a whole.

What can be noticed is that for each type of view, the model learns to attend mostly to edges and corners, but usually only of one shape. The hypothesis is that even within informative views, some views yield more useful information than others, for example because they show the distinctive feature of a class more prominently. The model is learning to maximize the confidence of its

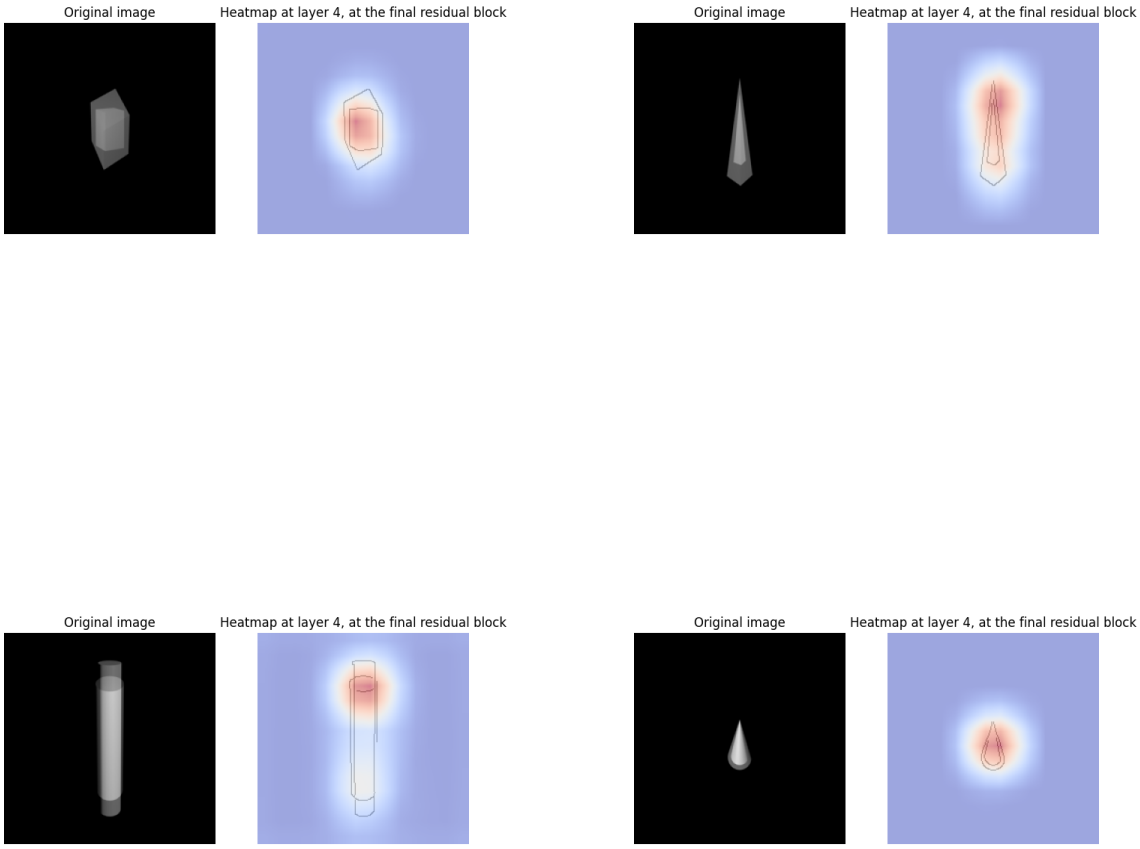


Figure 14: The activations of two uninformative views. We can see that features of mostly one object are attended to. For example, the top of the smaller cylinder is mostly attended to. Also distince features can be seen, as the tip fo the pyramid is attended to.

predictions, which means that the model learns more from views with many distinctive features, as these exemplars aid the model the most in learning.

5.4 Evaluating real-life inference performance

5.4.1 Testing real-life examples

After having obtained a scraped dataset, instances from this dataset were presented to both the single-view and multiple-view classifier. The images were first presented in grayscale, and

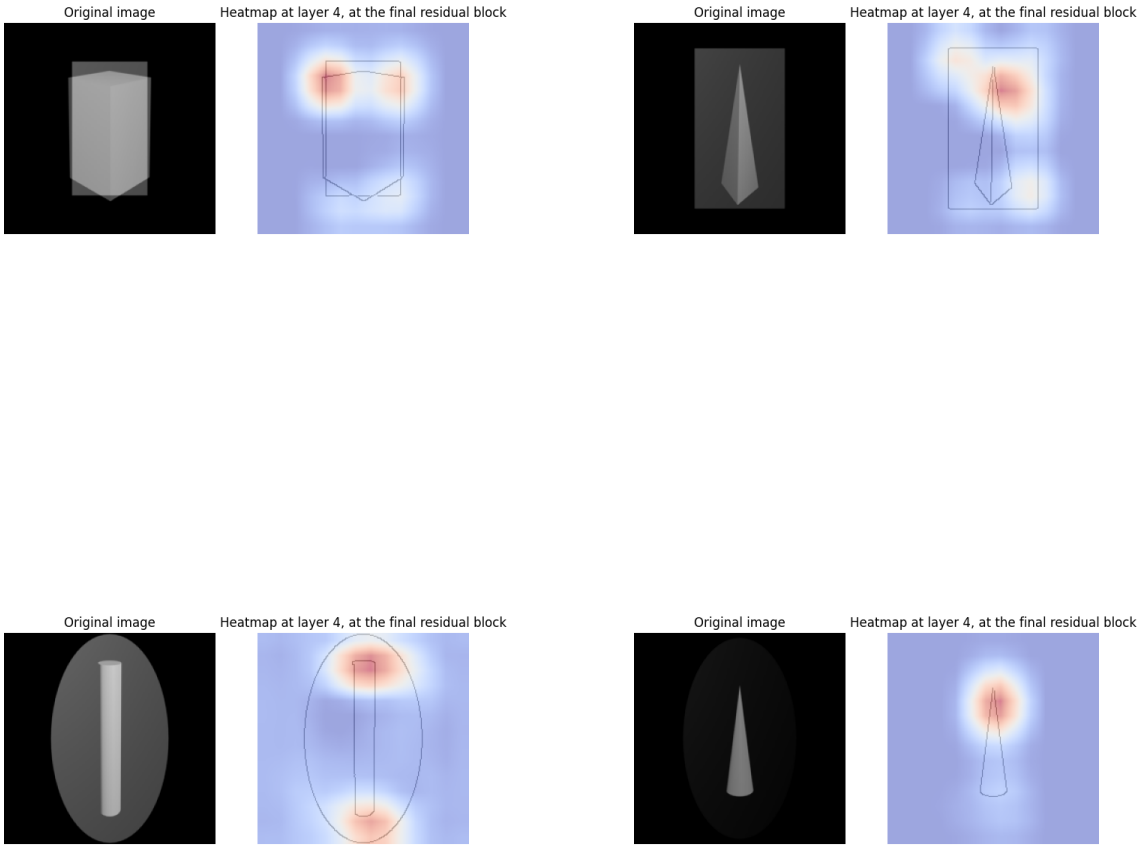


Figure 15: The activations of an informative and an uninformative view. We can see that the model attends to solely the informative view. For example, the upper and lower parts of the cylinder is mostly attended to, but also the top of the pyramid and the cone, and the corners of the cube.

cropped to the input image size that the model required. This means that only the absolutely necessary transformations had been conducted. Also the single-view baseline was tested, in order to see if any significant differences between both models could be observed. If this is the case, it means that one model has generalized significantly better to the features of the data rather than memorizing the data, since both models scored perfect scores when evaluated on the synthetic dataset. The inference accuracy and the total loss of the scraped dataset for both the single-view and multiple-view classifier are shown in figure 9.

It is noticeable that the 3D classifier obtains a worse performance than the single-view classifier

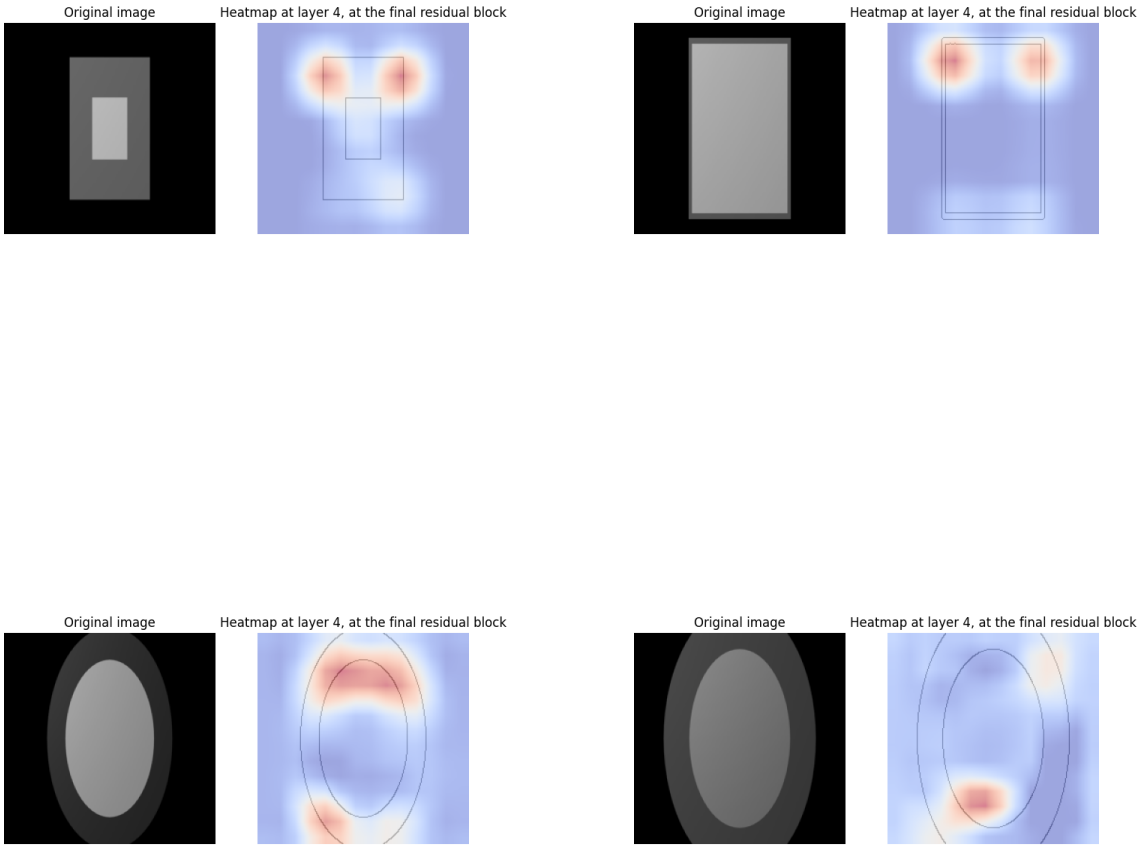


Figure 16: The activations of two uninformative views. We can see that the model is trying to attend to the edges and corners, but these are in this case not informative enough to come to the correct decision

on both the accuracy metric and the loss metric. This is an unexpected result, as we hypothesized that the accuracy and loss would be the same for both classifiers, since we are performing the same task using both classifier, though the multiple-view classifier actually has more information available. In addition to this, inference is performed on the same dataset, while pretraining was also done on the same dataset.

A possible explanation for this result is the fact that the AlexNet architecture is less deep than the Resnet-18 network. This could cause the AlexNet architecture to learn more general features, which are of more use when performing inference on a different target distribution in comparison to

Classes	single-view classifier	multiple-view classifier
Cube	0.40	0.00
Pyramid	1.00	0.00
Cylinder	0.80	0.00
Cone	0.60	0.00
Sphere	0.00	0.00
Ring	0.20	1.00
Average accuracy	0.50	0.17
Total loss	74	323

Table 9: Classifier accuracy and total cross-entropy loss for both the single view classifier and the multiple-view classifier, presented with real images. What can be immediately observed is the difference in classification accuracy and total loss. The single-view classifier seems to have learned representative features to some extent, while the multiple-view classifier seems not. In addition to this, a class accuracy of 1.00 for one class could mean that there is some feature in the real images that causes these predictions.

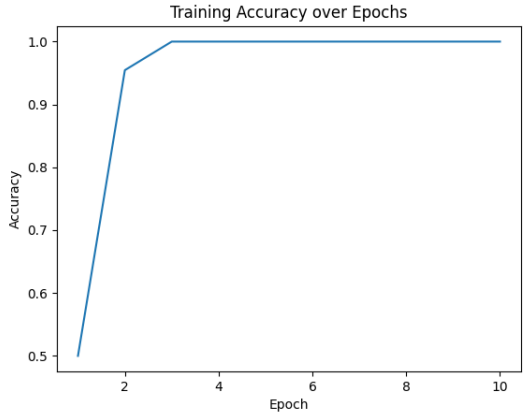
the possibly more detailed features learned by the Resnet-18 network. To find this out, a shallower model could be used, or an inflated version of the AlexNet.

Another possibility is that the multiple-view classifier did not learn to integrate features correctly, which could be catastrophic when the integration of features is needed when performing inference on a difference distribution. This could be tested by passing two images to the single-view classifier, and taking an average of the two softmax outputs. From this average, a new label could be deduced by choosing the class that has the highest average. However, this would only be a sensible method when working with two informative views, and for finding out the cause of the problem. When a combination of an informative and an uninformative view is used, a two-dimensional classifier can by definition not give a useful prediction about un informative view. In a possible industrial process, there is also no information available about whether the presented view is informative or uninformative, thus this method is not suitable for implementation.

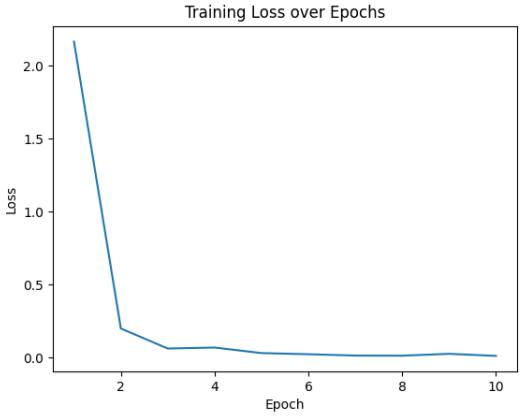
5.4.2 Reducing the domain gap: finetuning

Since the results of the 'naive' method were very poor, we tried to finetune the model on the scraped dataset. This method gave indications of success, but the actual effectiveness was not verifiable due to the very small size of the dataset. In figure 17 you can see the training accuracy and training loss per epoch.

What can be observed is that the training loss decreases drastically during finetuning, while the accuracy is fluctuating, though it increases in general. The fluctuating accuracy can be attributed to the size of the dataset. Since the dataset is very small, one extra misclassification can lead to a fluctuating accuracy. Still, this data looks promising, as it gives indications that the model is learning the features of the target distribution. However, we should remain critical towards the result, as there is no test set which we can use to research how well the model generalized to the new distribution. The training loss graph also gives us reason for initial optimism as the loss during training decreases rapidly, but later decreases steadily. The latter fact could mean that the model is using later epochs to memorize the presented data, rather than extracting features. In conclusion,



(a) The training accuracy during the finetuning process, plotted for each epoch. Training accuracy rises until it reaches a value of 1.0 at epoch 4.0. This gives an indication that the finetuning process works, but also raises questions whether the classifier learns the features of the data or if the classifier memorizes the data as a whole.



(b) The training loss during the finetuning process, plotted for each epoch. Training loss decreases rapidly until epoch 3, where after it decreases steadily. This indicates that the model has captured most features at epoch 4, and that later epochs are mostly memorizing the data itself.

Figure 17: Data about the finetuning process.

finetuning could be used on such classifier, though a bigger secondary dataset is needed. This may not be viable for all use cases, since it could be very expensive to create extra samples, as some real-life objects may be costly to produce.

5.4.3 Reducing the domain gap: using filters

Another approach that was conducted is pre-processing images using traditional computer vision filters, during both training and inference. The three used filters (Sobel, Canny, Laplacian) were explained in the methodology. To test these filters, we first crop the two input images to the 100x100 input size, where after they are converted to grayscale and a Gaussian blur is applied. Only then, the edge filter is applied. This is done for both the training loop and the given input for inference. With this method, we tried to bring the source and target distributions closer to each other. After pretraining on the synthetic dataset using these filters, a similar near-perfect accuracy was achieved for the Sobel and Laplacian filter (0.98 and 0.99 overal accuracy respectively). However, the Canny filter only obtained an overal accuracy of 0.90. This indicates that this particular filter may not be suitable for this task. A possible reason for this is that the Canny filter is very good in extracting many fine lines, while our task only requires the extraction of a limited amount of clear lines. After the pretraining, inference was performed on the scraped dataset. The scraped data was first pre-processed in the way mentioned in this paragraph. The results of performing inference on the scraped dataset with these new classifiers is shown in table 10.

From the average accuracy and total loss, we see an interesting result. The highest accuracy is obtained by the Laplacian filter, but the lowest loss is obtained by the Canny filter. Obviously, the Sobel filter does not seem to be the right choice for this task, since it has a very low accuracy and

Classes	Sobel	Canny	Laplacian
Cube	0.50	0.00	0.00
Pyramid	0.00	0.50	0.00
Cylinder	0.00	0.00	0.50
Cone	0.00	0.00	0.25
Sphere	0.00	0.00	0.00
Ring	0.00	0.67	0.67
Average accuracy	0.09	0.18	0.23
Total loss	809	200	403

Table 10: The reported test accuracy and total loss for the classifier pretrained using filters. Is is noticeable that the Canny filter, which performed worse during pretraining, is actually performing better than the Sobel filter during the inference phase on scraped images. It is possible that the Canny filter is able to handle noise found in real images better. This is also captured by the fact that it has the lowest total loss.

a high loss in general. Since we are working with a small dataset, the loss measure is the best way to compare models. It seems like using the Canny filter allows the model to better generalize to the useful features of the model. This can also be explained by the fact that the Canny filter had the lowest accuracy on the synthetic dataset. The Laplacian filter also seems like a good candidate, since there are more classes where there exists at least one correct prediction. The use of these filters could be further researched in an experiment where a real dataset with a bigger size would be obtained.

5.4.4 Reducing the domain gap: Improving the dataset

Since we observed in the previous results that the classifier tended to classify presented inputs as rings in a high amount of cases, we suspected that there was some kind of attribute of the ring shape that was present in the target inference distribution. What was especially noticeable in the ring samples was the fact that the ring took up a big portion of the image. To find out whether this caused problems in the real-life inference task, a perturbated dataset was created where the ring size was reduced from having a maximum outer radius of 2 units to a maximum outer radius of 1 unit. After generating this dataset, the multiple-view classifier was pretrained on this dataset, and inference was performed on the real-life dataset using the classifier in the II model. The inference performance is shown in table 11.

From this table, we observe that the perturbed dataset did not solve the problem of the classifier classifying each real image as belonging to the ring class. This means that the problem is not caused by a possible model shortcut in the form of the ring size giving away the class, but rather in the shape of the ring itself. What is suspected is that the shape of a ring consists of many edges in all directions, and many planes that each have a different luminosity due to the shadows that are rendered in Blender. When performing inference on real images, there is more noise in the image. Since the ring class is richer in features, it could mean that this noise is picked up by the classifier, and that the features of this noise are mapped to the ring class due to the latter being feature-rich. To further study this, a new dataset could be constructed where the discretization of the ring class is made more accurate, thus reducing the amount of lines and planes in the output.

Classes	multiple-view classifier
Cube	0.00
Pyramid	0.00
Cylinder	0.00
Cone	0.00
Sphere	0.00
Ring	1.00
Average accuracy	0.17
Total loss	497

Table 11: The inference performance on the real-dataset, given by an informative-informative multiple-view classifier pretrained on the perturbed dataset, where the physical size of the generated rings was limited. What can be observed is that this perturbation does not solve the problem of the classifier classifying each real image as a ring

This may lead to the ring features, especially the round edges, being learned better. Although a smoothing operator is present in Blender, it is not entirely clear what its effects would be on the overall dataset.

This reasoning may also explain why the per-class accuracy varies when working with traditional computer vision filters, as the main goal of these filters is to reduce noise and accentuate meaningful features of images. Since noise is reduced by using these filters, the risk is lower that the classifier learns to map the noise to one of the many features of the ring class.

Another perturbation of the dataset focused on making the dataset more complex. This was done by varying the roughness of the object's material, adding possible bevels to the edges. The same was done for the warmth of the light, the color of the background, and focal length was added as an extra parameter. In addition to this, a plane was placed under each object. The parameter ranges of these extra factors are shown in table 12. Again, the parameters are uniformly sampled from the shown ranges. After generating this dataset, we pretrained using this dataset and later performed inference using the scraped dataset. The test performance after the pretraining and the inference performance of the scraped dataset are shown in table 13.

Parameter	Range
Roughness	0.2 – 0.9
Bevels	0.001 – 0.05
Light Temperature	0.7 – 1.0
Background color	0.0 – 0.2
Focal length	18 – 85

Table 12: Additional parameters and their ranges used by each object in the complex dataset for sampling

Observing the difference in accuracy when performing inference on the synthetic dataset and the real-life dataset using the pre-trained multiple-view classifier, we see a similar score compared to the original, simpler dataset. This indicates that in this case, making the dataset more complex. Since we made the background color a free parameter, it also means that the black background is

Classes	Test performance after pretraining	Real-life inference performance
Cube	1.00	0.25
Pyramid	0.90	0.00
Cylinder	1.00	0.00
Cone	0.96	0.00
Sphere	1.00	0.00
Ring	1.00	1.00
Average accuracy	0.98	0.18
Total loss	0.19	158

Table 13: Test accuracy after pretraining and performing inference on the more complex synthetic dataset, compared with the real-life inference performance. What can be observed is that the performance is identical to the simpler dataset, indicating that making the dataset more complex did not work.

not the sole cause of the poor predictions. This is logical, since the background in a real image will often not contain one even color, but rather a collection of objects of different color, or even a horizon.

5.4.5 Reducing the domain gap: discussion

In the previous sections, we saw different methods that were used to improve the informative-informative multiple-view classifier's performance: finetuning, filters, and creating an improved dataset. Finetuning seemed to be the most promising method, but it was questioned whether the high training accuracy would extend to a high test accuracy, if a test set were to exist. This also brings us to the main problem of this method, which is the fact that for finetuning a dataset of considerable size is still needed.

Using filters also improved the real-life inference performance of the classifier. Notably, only the Canny and Laplacian filter yielded a good accuracy. Again, testing to see whether this is useful can only be done with a greater-sized real-life dataset. Still, this method was deemed to be more robust in comparison with finetuning.

Creating an improved dataset did not help at all. This was a rather interesting result, as adding more variables that can also be seen in the real-world was expected to bring the source distribution closer to the target distribution. This means that there may be another underlying factor that separates these two distributions. In an earlier research by Carlson, a fully connected network was used to learn the sensor style of the real data [2]. An adaptation of this method could be made for learning the Blender parameters of interest.

5.4.6 Measuring the domain gap using blended images

After having performed experiments which aimed to improve the overall classification accuracy, we will now try to determine the extent of the domain gap. To do this, two blended images were created with a certain blending factor alpha. These two blended images were stacked, so they could be used as a multiple-view exemplar for the II multiple-view classifier. Both images belonged to the cube class. The results of this for different blending factors is shown in table 14.

Blending factor	Prediction	Confidence
0	Ring	0.56
0.05	Ring	0.77
0.1	Ring	1.0

Table 14: Different blending factors, and the given prediction with its confidence. Since the ring prediction is wrong, a higher confidence means that the total loss starts increasing when the blending factor increases.

It is noticeable that at a blending factor of 0, the classification of the classifier is not correct. This is a highly interesting result. We suspect that this has to do with the fact that we changed the camera lens focal length in order to align the synthetic shapes with the real shapes. Still, this is interesting, as in the more complex dataset we added different focal length parameters, in order to capture differences in different cameras. This approach did not yield a better performance, while here it seems that a difference in focal length destroys the robustness of the model. However, we can still see a deterioration in classification performance when the blending factor increases. This is logical, as increasing blending factors mean that the exemplar starts to deviate from the source distribution to the target distribution.

6 Conclusions and Further Research

In this research, we aimed to answer the question to what extent a randomly sampled additional view aids performance in the recognition of physical ambiguous three-dimensional shapes. To answer this question, we looked at two main aspects: the difference in accuracy between single-view and multiple-view classifiers, and the difference in accuracy between performing inference on a synthetic and a real dataset. The focus of this research was to find out how multiple-view classifiers could be used for classification tasks in an industrial setting, where data sparsity could be an issue, and the orientation of objects may not always lead to an informative view when an image of it is captured.

The main findings were that adding an additional informative view to an uninformative view allowed a multiple-view classifier to correctly classify the presented synthetic shape, where as a single-view classifier would have failed when only presented with an uninformative view. This finding was strengthened by the use of qualitative methods, which we used to demonstrate that these models actually attend to useful features. However, pretraining on a synthetic dataset and performing inference on a real dataset without any domain adaptation methods proved to be highly unsuccessful. Several methods were used to reduce the domain gap, with varying success and reliability. We identified that finetuning, the Canny filter and the Laplacian filter are methods that show signs of improving classification accuracy and reducing total loss, but the reliability of the evaluation metrics proved to be questionable. In the next section, we will discuss the posed conclusions in more detail.

6.1 Single-view classifiers vs. multiple-view classifiers

First, we created a synthetic dataset containing six classes of objects, which consisted of four ambiguous shapes and two unambiguous shapes. The samples of the four ambiguous shapes were split into informative and uninformative views. By showing that the test accuracy of informative views was high, while the test accuracy of uninformative views was low, we found evidence that the uninformative views used in this research did indeed not contain sufficient information. Using a statistic test, we showed that the classifier did not behave better than randomly guessing the class that the uninformative view belonged to. Next, using Grad-CAM, we tried to interpret the model and find out which convolutional layers were used for what purpose. Layer 3 activations suggested that this layer was mostly used for edge detection, while layer 4 activations seemed to attend to objects in general. This pattern seemed to occur most in informative views, as some uninformative views showed random activations. By means of this, we claim that we have sufficiently proven the fact that the single-view classifier is *not able* to classify uninformative views.

Next, we switched to evaluating the proposed multiple-view classifier. This was done in multiple configurations: training and performing inference on two informative views, one informative and one uninformative view, and two uninformative views. We showed that the classification accuracy of configurations consisting of at least one informative view were high, while the configuration without any informative view yielded a low accuracy. Using Grad-CAM, it was observed that the learned filters by this classifier were similar, as there also seemed to be an edge detection layer, and instances where the model attended to the shape as a whole. In addition to this, there were also exemplars identified where the model specifically attended to features that were distinctive of a shape, such as the tip of a cone. Using this data, we claim that we have sufficiently proven that a multiple-view classifier presented with at least one informative view has enough information to uniquely determine ambiguous shapes, and a classifier that is presented with no informative views does not have enough information to uniquely determine ambiguous shapes.

6.2 Domain adaptation

After having obtained a working multiple-view classifier, we decided to test its inference performance on real-world physical shapes. The dataset that was used to do this was a dataset consisting of scraped images of the internet. It must be noted that for this part of the research, we only used the multiple-view classifier that worked with two informative views, as these objects were easier to find on the internet. Furthermore, we expected real-life inference performance to be the same for uninformative views, as the same variations in light, camera, and background are possible. After testing the inference performance of the original classifier on this dataset, results were very poor. To solve this problem, we tried to move the source and target distribution closer to each other by finetuning on the scraped dataset. This improved the training accuracy, which gives an indication that the classifier is able to capture the features of the new distribution, but since we did not have enough samples for a proper test set, we could not test this. We deem the chance that the classifier over-generalized to the new data to be highly probable. Next, pre-processing each exemplar using a traditional computer vision filter for both pretraining and inference was done. The Sobel filter turned out to do an even poorer job, while the Canny and Laplacian filters actually improved performance. We think that this result is reliable, especially since the scraped dataset was used as a test dataset. Finally, making the dataset more complex was tried, but this proved to not work.

From this, we conclude that just making datasets more complex is not the solution for the domain gap problem, since it does not aid the problem of an overfitting classifier. Rather, we believe that focus should be on methods that allow classifiers to extract useful features for classes, such as was done by pre-processing all inputs using a traditional computer vision filter.

Finally, we tried to measure the domain gap by creating a blended image between a synthetic and a real exemplar, and measuring the cross-entropy loss for different blending factors, in order to make the domain gap measurable. This proved to be unsuccessful, as at a blending factor of 0 the prediction was already incorrect. This could be attributed to the fact that the focal length of the virtual camera capturing the synthetic object was changed to better overlap the physical image, but this meant that we had left our parameter space. This is an indication that the pretrained classifier overfit on the synthetic dataset. However, the more complex dataset did feature a free focal length parameter to capture differences in lenses, but this dataset performed worse in general.

6.3 Future work

From the presented findings, new research subjects can be identified. For example, we used a complex three-dimensional classifier as multiple-view classifier, but a simpler architecture or an adapted architecture may work better, especially when it comes to generalisability, as most three-dimensional classifiers are originally designed for video analysis. A simpler architecture could also mean lower inference times, which is a desirable property in an industrial settings.

When it comes to the area of domain gap reduction, the finetuning experiment of this research could be repeated with a bigger dataset. The scraping of this dataset could be done automatically, which has been performed in earlier research [18]. In this way, we can also find out how many samples are needed until the classifier reaches the desired accuracy. This can also be compared for multiple types of classifiers.

Finally, we identified that the Canny and Laplacian filter gave indications that they are able to reduce the domain gap, when used for pre-processing samples in both the training and inference pipeline. The use of these filters, other filters or combinations of filters could be researched, as they may be able to reduce variance in both the source and target distribution, helping the generalisability of deep models, which is a problem when being pretrained on a simple synthetic dataset.

References

[1] L. Kumar Boran and A. Husain Joya. From pixels to predictions: A comprehensive survey of image classification. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 2023.

[2] Alexandra Carlson, Katherine A. Skinner, Ram Vasudevan, and Matthew Johnson-Roberson. Sensor transfer: Learning optimal sensor effect image augmentation for sim-to-real domain adaptation. *IEEE robotics and automation letters*, 4(3):2431–2438, 2019.

[3] Lauren H. Cooke, Matthias Jung, Jan M. Brendel, Nora M. Kerkovits, Borek Foldyna, Michael T. Lu, and Vineet K. Raghu. Roentmod: a synthetic chest x-ray modification model to identify and correct image interpretation model shortcuts. *NPJ digital medicine*, 2026.

[4] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. 2017.

[5] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. 2017.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks, 2017.

[7] LearnOpenCV Team. Edge detection using opencv, June 2021. Accessed: 2026-05-25.

[8] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s, 2022.

[9] Xiangyu Ma, Jing Bai, Zenghui Su, and Yubin Wang. Pvstrans: Patch-view-shape progressive interaction transformer for 3d shape recognition. *Information processing & management*, 63(1):104279–, 2026.

[10] Nicolas M Müller, Simon Roschmann, Shahbaz Khan, Philip Sperl, and Konstantin Böttinger. Shortcut detection with variational autoencoders. 2023.

[11] Xin Ning, Limin Jiang, Weijun Li, Zaiyang Yu, Jinlong Xie, Lusi Li, Prayag Tiwari, and Fernando Alonso-Fernandez. Swin-mgnet: Swin transformer based multiview grouping network for 3-d object recognition. *IEEE transactions on artificial intelligence*, 6:747–758, 2025.

[12] Shaohua Qi, Weijun Li, and Guowei Yang. Double weighting convolutional neural net-works for multi-view 3d shape recognition. *IET computer vision*, 19(1), 2025.

[13] Stephan R. Richter, Vibhav Vineet, Stefan Roth, Vladlen Koltun, Bastian Leibe, Max Welling, Nicu Sebe, and Jiri Matas. Playing for data: Ground truth from computer games. In *Computer Vision – ECCV 2016*, pages 102–118, Cham, 2016. Springer International Publishing.

[14] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, 2020.

[15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. 2015.

[16] Xinyue Sun, Shusen Yang, and Cong Zhao. Lightweight industrial image classifier based on federated few-shot learning. *IEEE transactions on industrial informatics*, 19(6):1–10, 2023.

[17] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. 2014.

[18] Xizhong Yang, Qi Guo, Wenbin Chen, and Mofei Song. Webly supervised 3d shape recognition. *Pattern recognition*, 158:110982–, 2025.

[19] Wenqian Ye, Luyang Jiang, Eric Xie, Guangtao Zheng, Yunsheng Ma, Xu Cao, Dongliang Guo, Daiqing Qi, Zeyu He, Yijun Tian, Megan Coffee, Zhe Zeng, Sheng Li, Ting-hao, Huang, Ziran Wang, James M Rehg, Henry Kautz, and Aidong Zhang. The clever hans mirage: A comprehensive survey on spurious correlations in machine learning. 2025.

[20] Khandoker Ashik Uz Zaman, Ashraful Islam, and Md Abu Sayed. Render lighting dataset: A collection of rendered images with varied lighting conditions using blender render engines. *Data in brief*, 54:110331–, 2024.

[21] Tianyu Zheng, Chunyan Zhang, Shengwen Zhang, and Yanyan Wang. Deep learning-based 6-dof object pose estimation considering synthetic dataset. *Sensors (Basel, Switzerland)*, 23(24):9854–, 2023.