TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Master's Thesis
"Robotics, Cognition, Intelligence"

Visual Similarity Detection Based on Latent Representations

Kanstantsin Tkachuk

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Master's Thesis
"Robotics, Cognition, Intelligence"

Visual Similarity Detection Based on Latent Representations

Visuelle Ähnlichkeitserkennung basiert auf latenten Darstellungen

Author:	Kanstantsin Tkachuk
Supervisor:	PD Dr. habil. Rudolph Triebel
Advisor:	Maximilian Durner
Submission Date:	15. November 2021

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

Munich, 15. November 2021 Kanstantsin Tkachuk

Abstract

In this thesis we develop and evaluate multiple scalable solutions for the task of image similarity detection as part of an automated testing system for the rendering pipeline of the game Space Engineers by GoodAI.

We implement image similarity detectors based on comparison of compact representations of the input images generated by three self-supervised representation learning architectures: Multi-Path Augmented AutoEncoder by Sundermeyer et al. [12] and two kinds of Siamese networks partially based on the first architecture.

We demonstrate the applicability of the aforementioned architectures in the given setting of synthetic training data and existing domain gap between the training and the application domains and evaluate their ability to produce latent representations which meet the requirements of robust generalization and invariance to variations in background, lighting, texturing and object's pose within the field of view.

We demonstrate the benefits of the multi-path architecture for the descriptiveness of latent representations with respect to the appearance features of the objects in the images as opposed to the geometric features examined in the original paper [12].

Acknowledgments

First and foremost I would like to thank my supervisor PD Dr. habil. Rudolph Triebel, my advisor Maximilian Durner and also Dr. Zoltán-Csaba Márton and my colleagues Dominik Meinzer, Dr. Steffen Rühl and Carsten Zumsande for making this work possible.

I would also like to thank Dr. Martin Poliak and Ondřej Nepožitek of GoodAI for providing the data along with documentation and support.

Last but not least, I would like to thank everyone who supported me during my work on this thesis and throughout my master's studies: my beloved partner Tatsiana, my family: my mother Alena, my father Aliaksandr, my grandmother Nataliya Fedorovna and my godfather Vladimir; my friends Darya and Sviataslau-Francesco and my colleagues Carsten Zumsande and Dominik Meinzer to whom I am especially grateful for their patience.

Contents

Acknowledgments

1 Introduction **1**

2 Literature review **5**

3 Methodology **11**
3.1 Datasets 12
3.1.1 Training datasets 12
3.1.2 Training data augmentation 15
3.1.3 Evaluation datasets 18
3.2 Multi-Path Augmented AutoEncoder 19
3.3 Siamese networks 22
3.4 Evaluation 26
3.4.1 Experiment evaluation 26
3.4.2 Final evaluation 28

4 Results **35**
4.1 Detector based on Multi-Path Augmented AutoEncoder 35
4.1.1 Number of decoders 35
4.1.2 Number of models in the training set 35
4.1.3 Other augmentation techniques 37
4.1.4 Final model and ablation study 39
4.2 Detector based on siamese networks 41
4.3 Final performance comparison 43

5 Discussion **53**

6 Conclusion **57**

Chapter 1

Introduction

The task of **visual object similarity detection** arises in an ever increasing variety of application domains from autonomous driving (person and vehicle re-identification and tracking) to robotics (object detection, navigation) to computer graphics and game development (automated testing and evaluation of rendering pipelines and game engines). Often the advancements made in one application field, even if they cannot be transfered from one domain to another right away, inspire ideas in other fields or raise the question of how such a transfer might be achieved.

In this thesis we examine a task from the domain of game development. The formulation of the task as well as the data necessary for the development of a scalable solution is provided to us by our partner GoodAI in cooperation with German Aerospace Center (DLR).

The task we solve in this thesis is the task of automated testing of the rendering pipeline in the game Space Engineers developed by GoodAI. Space Engineers is a sandbox game about engineering, construction, exploration and survival in space and on planets. Players build space ships, space stations, planetary outposts of various sizes and uses, pilot ships and travel through space to explore planets and gather resources to survive.

In the game there is a certain number of **constructable blocks** — parts of different mechanisms and structures that can be built by players. Each of these blocks has several **completeness stages** from stage zero (the construction has just started) to the fully built usable block. Each completeness stage has a different appearance and is represented by a different 3D model in the game. The models replace each other during the construction of the block in order to visualize the construction progress.

In order to test the rendering pipeline, GoodAI render the aforementioned 3D models in-game and manually check whether the correct 3D model has been rendered for the given block in the given completeness stage.

Manual testing requires manpower and time, so the task we solve in this thesis is automating the testing by developing a **similarity detector** capable of predicting whether the model rendered in the game is correct given the screenshot of the rendered model and the reference 3D model.

Our similarity detector first renders a reference image of the given 3D model using an internal renderer. Afterwards, the input screenshot and the reference image are passed as input to the same convolutional **encoder network** in order to obtain compact 1D representation vectors which can then be compared using simple vector similarity metrics such as L2 distance or cosine similarity.

The encoder network must be able to capture the relevant features of the models in both images (which may look fairly different) and ignore irrelevant features such as background and lighting. This task is challenging for a number of reasons.

Size and variability of the block set: new models are being added to the game and older models might be removed or replaced in the future. Generating and annotating new data for thousands of models and retraining the encoder network after each new update is infeasible, therefore, the network must be able to generalize well beyond the models seen during training.

Variable lighting and background: the models can be rendered in different lighting conditions on various, possibly cluttered backgrounds, so the encoder network must be invariant to lighting changes and be able to extract and ignore the background and the clutter.

Out-of-game rendering limitations: since the appearance of the models in the game depends on certain internal optimizations and custom materials, the models cannot be rendered outside of the game to look exactly like their in-game counterparts using any of the available renderering software such as Blender or OpenGL. This creates the need for a encoder network which would be at least to some extent invariant to texture changes while remaining sensitive to the overall shape as well as small details of the block's geometry.

This also implies that the data that can be generated for the training of the encoder networks (i.e. can be rendered by our external renderers) does not belong to the application domain of the similarity detector, since the actual in-game screenshots will look different. In order to make the encoder

applicable in both domains, we need to find a way to bridge a softer version of "sim-to-real" gap from the domain of untextured or partially textured isolated models to the domain of textured models in a rich game environment.

Variations in model pose: in the game, the 6 DoF pose of the model relative to the camera can only be estimated approximately, so the similarity detector must either render multiple reference images of the model from different views (none of which may coincide exactly with the screenshot) or operate using approximate poses provided by the game. In any case, the encoder network must be invariant to small variations in object's position and orientation within the field of view of the camera in order to ensure robust similarity detection.

In this thesis, we examine three architectures as candidates for the encoder network: **Multi-Path Augmented AutoEncoder** by Sundermeyer et al. [12] and two kinds of **Siamese networks** partially based on the first architecture.

We describe the key features of the architectures above which make them particularly suitable for our task. We then implement the respective models, train them on synthetic data generated from the 3D models provided by GoodAI and implement and evaluate similarity detectors incorporating the resulting encoder networks.

We evaluate the influence of individual key features on the overall similarity detection performance, combine the ones shown to be most beneficial into the final version of each model and perform an ablation study.

Apart from that, we explore the benefits of the novel multi-path autoencoder architecture by Sundermeyer et al. [12] applied for the first time to the task of binary classification based on latent representations. We explain how the multi-path architecture affects the descriptiveness of latent representations generated by autoencoders with respect to the visual features of the objects instead of geometric features such as 6 DoF poses.

Chapter 2

Literature review

The key feature of our similarity detection task is the **domain gap** between the available synthetic training data and the expected evalutation/application domain consisting of in-game screenshots (Figure 2.2).

Numerous research papers propose different solutions for visual domain adaptation problem in deep learning. Wang et al. [15] divide currently available approaches into three categories:

- *generative models*: use GANs to generate synthetic images similar to the images from the application domain using information from the source domain images and preserving the annotation information;

- *non-generative models*: train in the source domain and then map the data from the application domain to the source domain using a domain-confusion loss;

- *reconstruction-based approach*: use encoder-decoder or adversarial architectures to reconstruct images in the application domain from images in the source domain.

Among all these categories, the **encoder-decoder** reconstruction-based approach is most suitable for our setting since we are interested in generating compact and descriptive representations of the images from both domains rather than converting images from one domain to another. The latent bottleneck layer of the encoder-decoder architecture is a good candidate for such representation.

Figure 2.1: Style randomization applied to an image from the Office domain adaptaion benchmark (from [7]). Shape and semantics are preserved but the style (texture, color and contrast) is randomized.

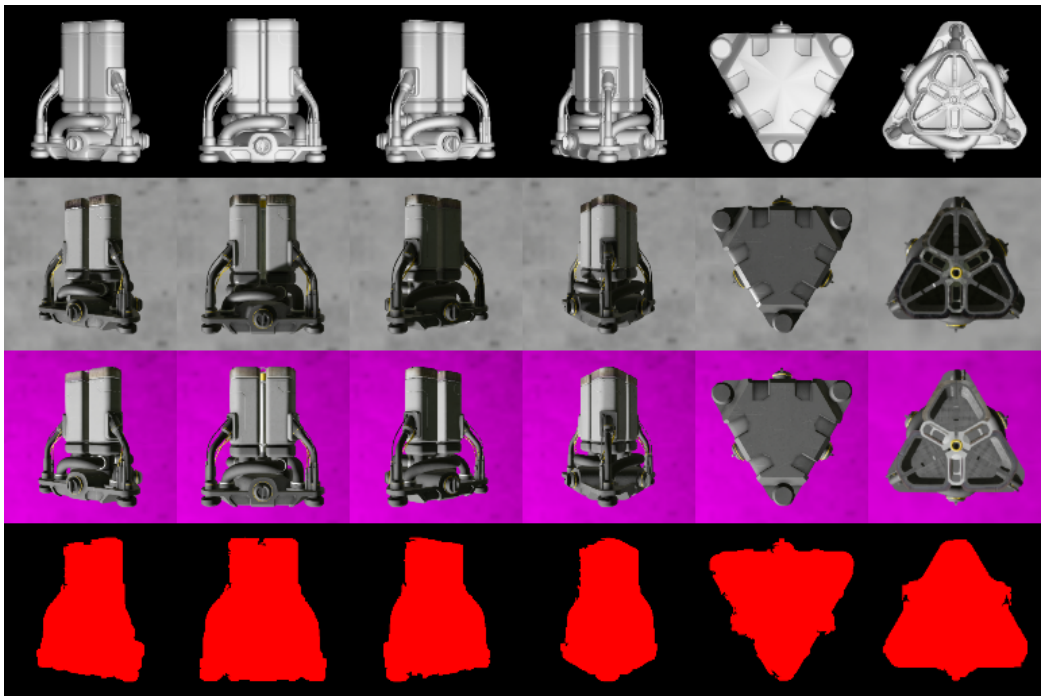


Figure 2.2: An illustration of the domain gap between the training data (top row) and evaluation data / application domain (rows 2 and 3).

However, since no images from the application domain are available to us at training time, we have to find a way to emulate them using the training images. This can be done using specific **data augmentation techniques**.

For this task, useful ideas can be adopted from the other domain adap-

tation approaches. For instance, Jackson et al. [7] propose an *image style augmentation technique* which uses style GANs with embeddings sampled from a multivariate normal disctribution in order to apply random style to real images from the STL-10 classification benchmark [4], the Office domain adaptation benchmark [10] and the KITTI depth estimation benchmark [14]. Application of random style preserves the shape and the semantic content of the images while randomizing **texture, contrast and color** (Figure 2.1). The same technique can of course be used for augmentation of synthetic images.

Another augmentation approach specifically designed for bridging the sim-to-real gap is proposed by Carlson et al. [2]. They argue that the domain gap can be reduced by modelling the *variation in the sensor domain* used to capture real-world images. Their augmentation method emulates the influence of different sensor models on the domain shift by introducing variations in **blur, exposure, noise, color temperature and chromatic aberration**.

Since the aforementioned augmentation techniques modify both the background and the appearance of the object, they simultaneously address two of our requirements: background invariance and texture invariance.

In the case of synthetic data the effect of style augmentations can be further enhanced by considering the appearance of the model and the background separately. For instance, Ma et al. [8] propose an augmentation technique which employs GANs only for generation of **random background** while keeping the appearance of the rendered object unchanged.

At this point we consider the **Multi-Path Augmented AutoEncoder** architecture by Sundermeyer et al. [12]. It is based on the Denoising Autoencoder arthitecture which inherently requires augmentation of the input data to generate descriptive latent representations. The augmented data is used as input to the autoencoder while the unaugmented original images are used as reconstruction goal (Figure 2.3).

The synthetic images are augmented using the following techniques, among others:

1. using random images as background

2. randomly translating and scaling the objects in the images

3. adding channel-wise variations in brightness and contrast of the images

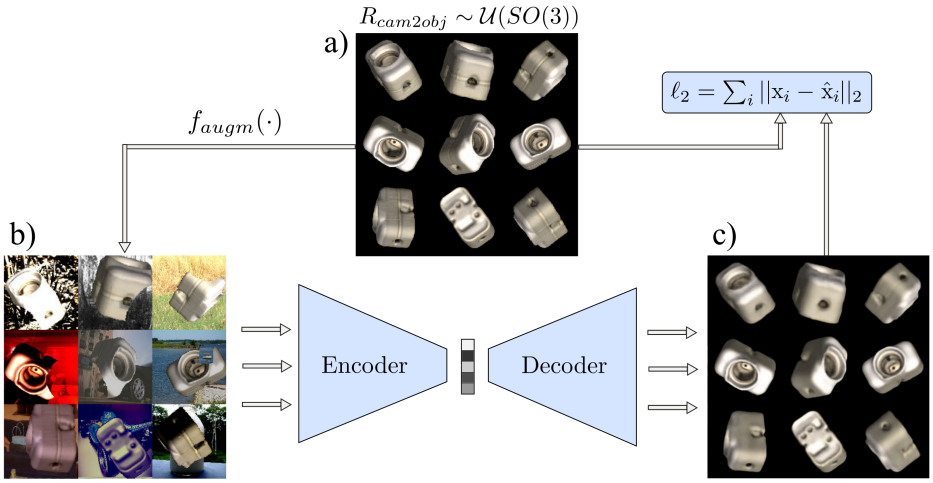


Figure 2.3: Diagram of a single-decoder Augmented AutoEncoder by Sundermeyer et al. (from [13]). The augmented images are used as input to the autoencoder and the unaugmented original images are used as reconstruction goal for the decoder.

4. using randomized lighting at rendering time.

The usage of the augmentation techniques above enforces invariance to the corresponding image features in the latent representations generated by the autoencoder. This is particularly useful for our task, since each of the used techniques addresses one or more of our requirements: 1 addresses the background invariance requirement, 2 addresses the invariance to perturbations in the translation and rotation of the model, 3 and 4 simultaneously address the texture and lighting invariance requirements. At the same time, these techniques in part correspond to the augmentations proposed by Jackson et al. [7] and Carlson et al. [2] which were proven to be able to reduce the sim-to-real gap in their respective settings.

Meanwhile, the ability of the autoencoder architecture to extract common within the training domain features which are relevant for generating descriptive latent representations addresses the requirement of strong generalization ability and unversal applicability of the encoder network on previsouly unseen data.

However, the Multi-Path Augmented AutoEncoder architecture is used by the authors for the task of 6 DoF pose estimation which is *per se* unrelated to

the visual similarity detection. They argue that the single-encoder-multiple-decoder architecture allows to better capture geometric information common for different objects and use it to enhance pose estimation.

In this thesis we examine the possible benefits of the multi-decoder architecture for generation of descriptive latent representations with respect to the **appearance features** of the object rather than geometric features.

As a baseline experiment we additionally consider another self-supervised learning architecture commonly used for image similarity detection which is the **Siamese network architecture**.

The core principle behind Siamese networks is using parallel encoder branches with shared weights to generate latent representation vectors of the input data and then using a simple vector similarity metric to compare these representations. The loss of a Siamese network is constructed in such a way as to increase the similarity of representations generated from images which belong to the same class and decrease the similarity of images of different classes.

Siamese networks have been successfully used for face recognition [6, 11], person re-identification [3, 16] and object tracking [1, 9].

We examine the ability of the Siamese networks to learn feature invariant representations using the same augmentation techniques as for the Multi-Path Augmented AutoEncoder. Apart from that, we examine a different approach to learning domain-invariant visual features by training the Siamese networks exclusively on the data from our synthetic image domain and showing that latent representations generated by the networks from the application domain images contain descriptive features and can be used for similarity detection.

Chapter 3

Methodology

In order to solve the task of robust block detection we implement a **similarity detector** that takes a screenshot, a 3D model and an optional 6 DoF pose of the model as input and produces a binary decision on whether the model depicted in the given screenshot is the same as the given 3D model.

It operates in the following way:

1. render one or more textureless reference images of the given 3D model in OpenGL
2. use a deep neural network as an encoder to generate 1D latent representation vectors of the screenshot and of the reference images
3. calculate the similarity values between the latent representations of the screenshot and each reference image based on a predefined similarity metric (e.g. L2-distance or cosine similarity)
4. reduce all similarity values to a single value describing the similarity between the model in the screenshot and the given 3D model
5. if the similarity exceeds a fixed decision threshold, return a positive prediction (the models match), otherwise return a negative prediction (the models do not match).

We examine three deep self-supervised learning architectures as candidates for the encoder network: the encoder part of a **Multi-Path Augmented AutoEncoder** examined in detail in Section 3.2 and the encoder branches of two kinds of **Siamese networks** (Section 3.3).

We train our self-supervised models on two datasets of the same size generated using two different sets of 3D models taken from the model set provided by GoodAI (all datasets are described in detail in Section 3.1).

To evaluate our detectors we use the screenshots of each corresponding 3D model taken in-game and provided as part of the dataset. The evaluation is performed accoring to two evaluation algorithms proposed by GoodAI described in Section 3.4 as Algorithm 1 and Algorithm 2.

We use the full dataset (models and screenshots) to evaluate the final performance of the detectors.

3.1 Datasets

The dataset provided by GoodAI contains 3D models of 610 constructable blocks from the game Space Engineers, each having from 2 to 7 completeness stages. Each stage of each block is represented by a unique textured 3D model which can be used to render images for training datasets. The dataset also contains in-game screenshots of each 3D model and some additional information used for evaluation.

3.1.1 Training datasets

The GoodAI dataset provides a total of 1825 3D models representing different completeness stages of different blocks. GoodAI allows usage of any subset of the models for training and evaluation of the detector, including using the whole dataset if needed, since GoodAI is evaluating the final version of the detector on an internal testing dataset unavailable to us during development.

We define two training datasets of the same size (in terms of number of images) generated from two different subsets of 3D models.

We first define **block subset A** (Figure 3.1) which consists of 4 blocks with distinct geometrical shapes each similar to one of the basic 3D shapes:

1. `BasicAssembler` (cube)
2. `GratedStairs` (plane/rectangular prism)
3. `LargeDecoy` (sphere)
4. `LargeExhaustPipe` (triangular prism).

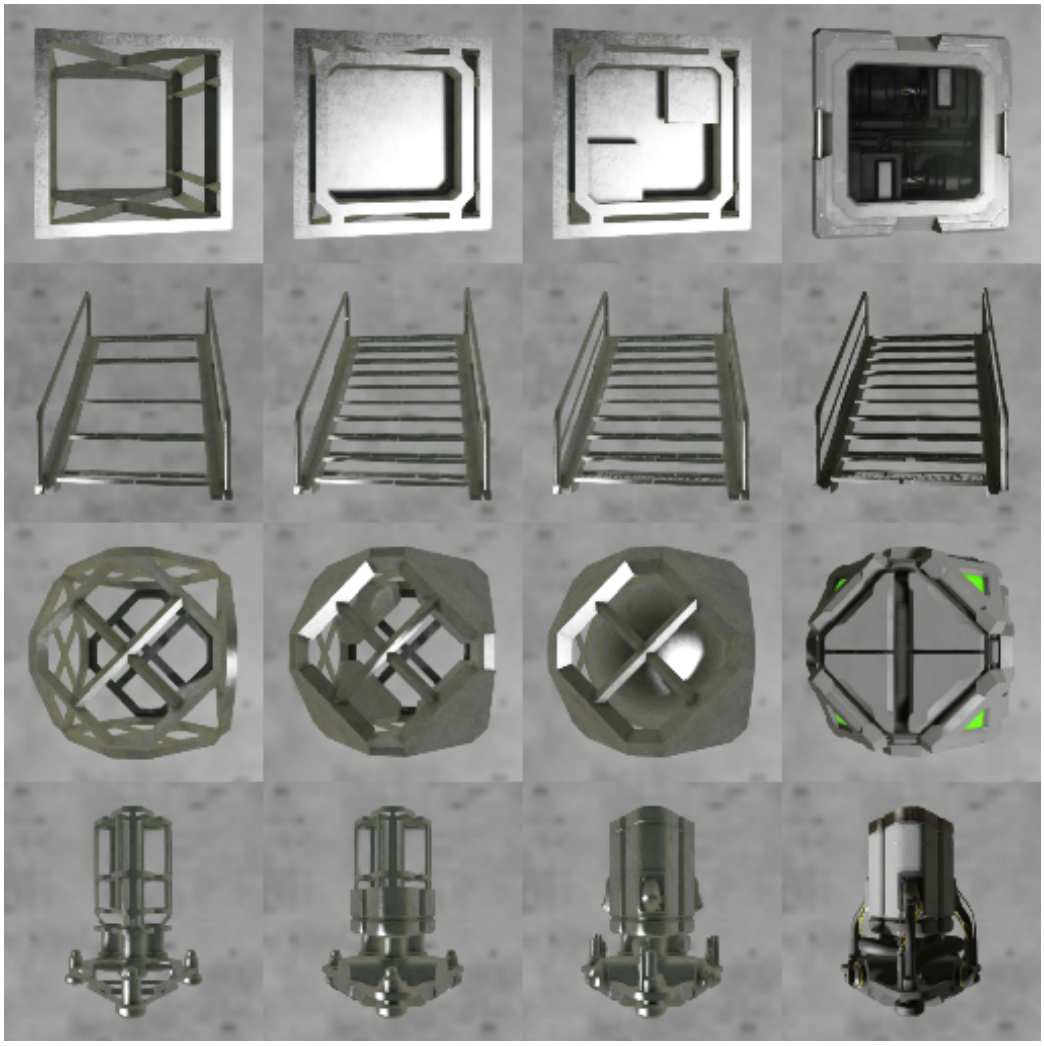


Figure 3.1: Block/model subset A (by rows): `BasicAssembler`, `GratedStairs`, `LargeDecoy`, `LargeExhaustPipe`

The blocks are chosen specifically for their shapes in order to ensure geometric diversity of the models and at the same time cover some of the basic 3D shapes which are likely to appear in one form or another as parts of unseen models in new test images.

Each block from the subset is presented in 4 completeness stages, each represented by its own 3D model for a total of 16 models that we define as **model subset A**. The mutual visual similarity of different stages varies be-

tween blocks from moderate (e.g. `LargeDecoy`) to high (e.g. `GratedStairs`).

For each model from model subset A we render 8000 images, each image depicting the model on uniform black background at the same position relative to the camera (with the model centered in the field of view and fully visible) but with a unique random orientation uniformly sampled from SO(3) (Figure 3.2).

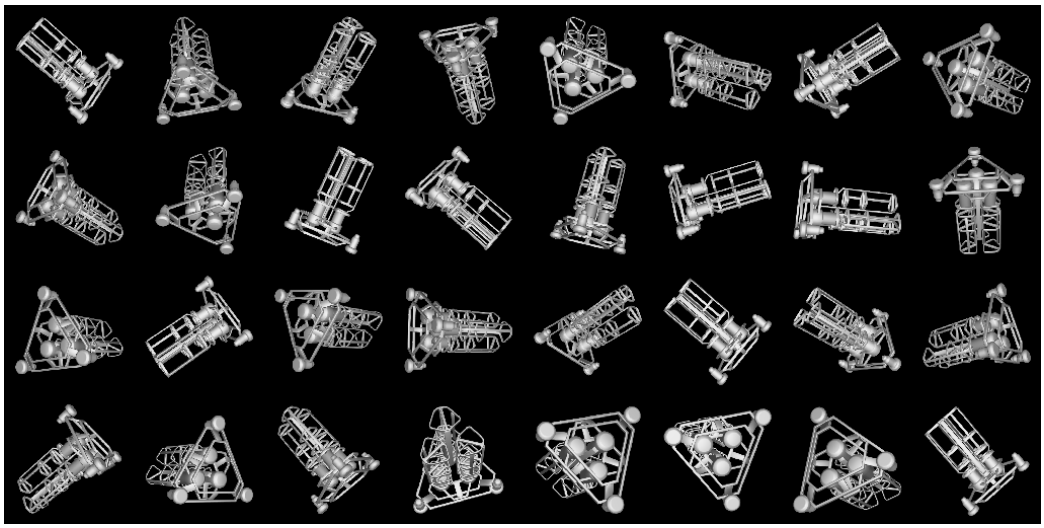


Figure 3.2: An example batch for block `LargeExhaustPipe`, stage 0

All training images rendered for all models from model subset A form **training dataset A** with a total size of 128000 images.

The second block subset — **block subset B** — comprises 20 different blocks of various shapes with 4 completeness stages each (Figure 3.3). The blocks are chosen as to maximize geometrical diversity and cover the widest range of stage similarity.

The 80 models corresponding to all stages of the 20 blocks from block subset B form **model subset B**.

The training images for model subset B are rendered in the same way as for model subset A except that the number of images per 3D model is reduced from 8000 to 1600 in order to generate the same total number of images of 128000. All images rendered for all models of model subset B form **training dataset B**.

Although all models in the provided dataset contain textures, they cannot be rendered accurately outside of the game due to their appearances being

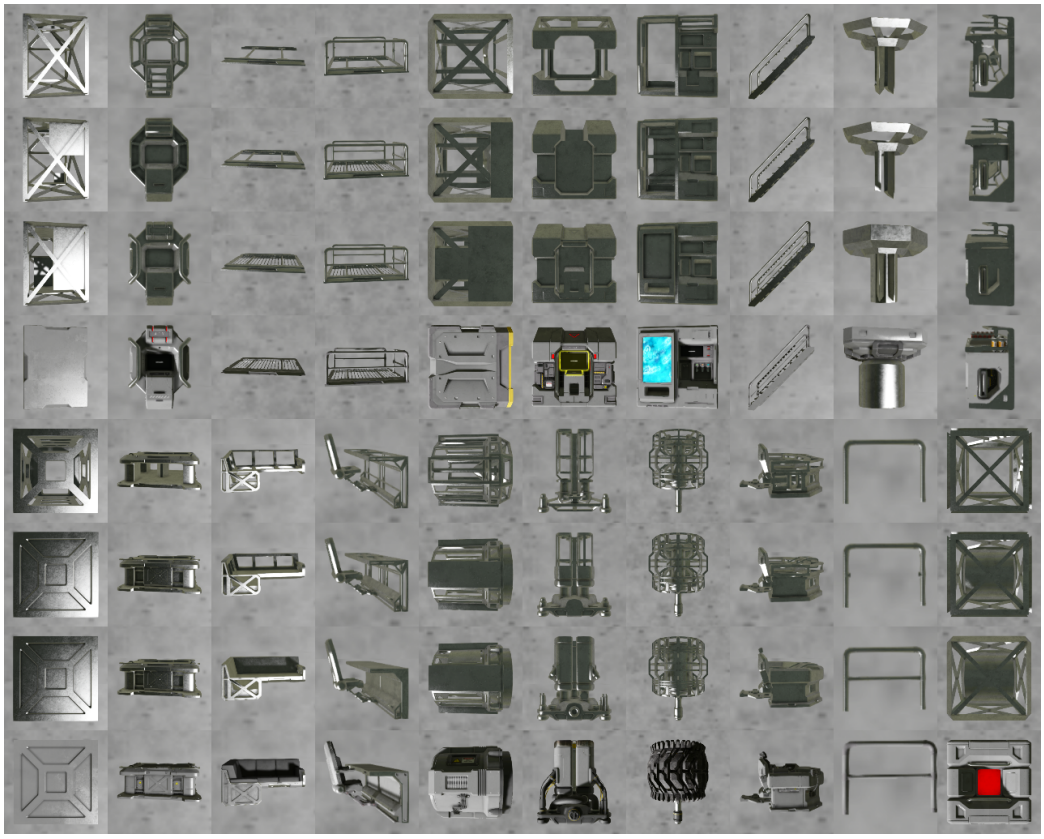


Figure 3.3: Block/model subset B

dependent on certain in-game optimizations and custom materials. This limitation makes rendering exact copies of the screenshots impossible both for training and for inference and creates a need for a certain level of texture invariance in latent representations.

For that reason the models are rendered in OpenGL in uniform gray color without textures and the encoders are trained to generate similar latent representations for textureless models as well as for textured models using data augmentation techniques described in Subsection 3.1.2.

3.1.2 Training data augmentation

The augmentation techniques we use to enforce invariances to certain features in our similarity detectors (Figure 3.4) are divided into the following

categories:

- background augmentation (random background)
- geometric augmentations:
  - random object scaling
  - random in-plane translation
- appearance augmentations:
  - random variations in brightness/contrast
  - random lighting
  - partial texturing.

Background augmentation involves replacing the black background of the training images with random images, in our case images from VOC2012 [5]. This might help enforce background invariance in latent representations by forcing the model to generate similar representations for images of the same object on different background.

Geometric augmentations, namely random scaling and translation of the object within the image, might be used to train the encoder to ignore the position of the object within the field of view of the camera.

In 2D images, scaling emulates translation of the object along the view axis of the camera and 2D translation corresponds to in-plane translation in the 3D world. Although 2D and 3D transformations are not equivalent and do not produce the exact same images, the differences become negligible as the images are converted to compact latent representations.

Introduction of random variations in image **contrast and brightness**, although more suitable for emulating different camera settings in images from real cameras, might also be useful for enforcing lighting invariance while training on synthetic images. We introduce brightness and contrast variations separately to each channel in order to diversify the color ranges of the resulting images and emulate different lighting tones.

The techniques described above are applied to rendered images as post-processing.

At the rendering step, we use **randomized lighting** for lighting invariance. The position of the light source and the parameters of the diffuse and

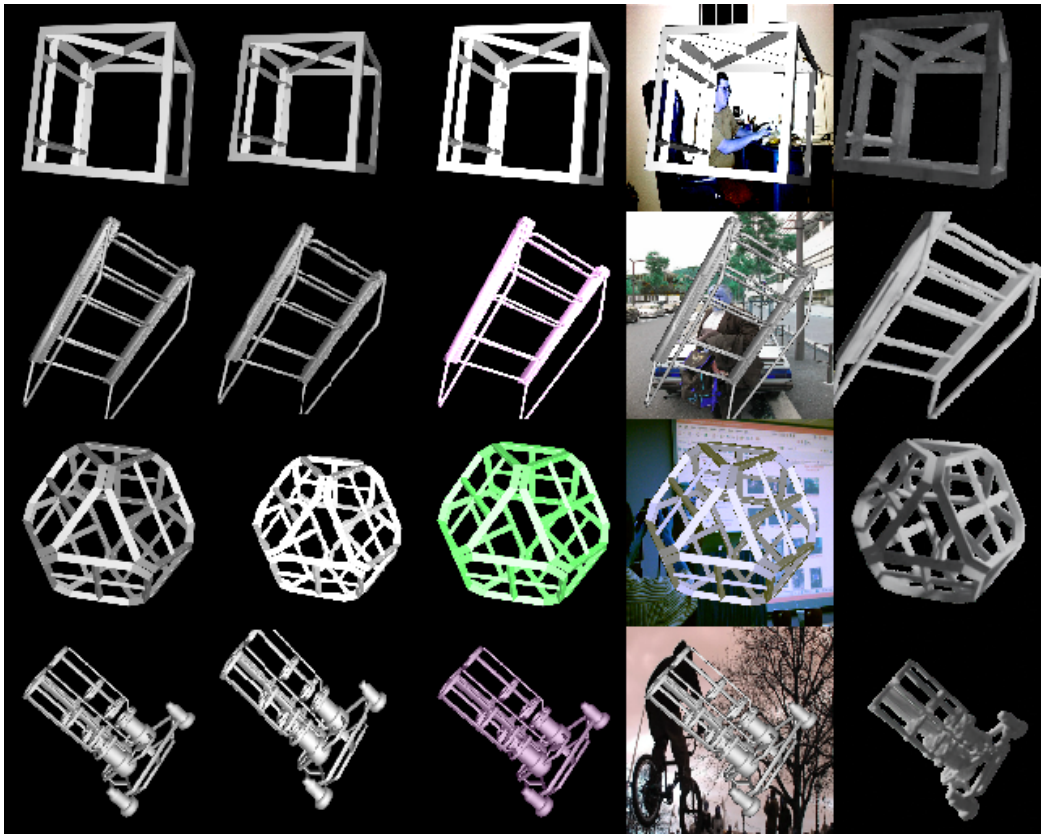


Figure 3.4: Augmentation techniques. Augmented features are, from left to right: no features, scale/translation, brightness/contrast, background, textures and lighting

specular light of our OpenGL Phong renderer are chosen randomly for each new image.

Another augmentation technique applied at rendering step is rendering the models with **partial texturing**. For the reasons explained in Section 3.1.1 we are unable to render the models the way they look in the game, but some of the textures and materials can still be rendered while the others are replaced with plain colors. By training the encoder to generate similar representations for partially textured and untextured models we enforce texture invariance.

3.1.3 Evaluation datasets

For each 3D model, the dataset contains the following information used for evaluation of the detection performance (Figure 3.5):

- 6 in-game screenshots of the model rendered on gray background
- for each screenshot:
  - an RGB mask encoding the pixels of this screenshot that belong to the object
  - a 6 DoF pose of the rendered model relative to the camera
  - an alternative screenshot of the model in the same pose but on purple background.

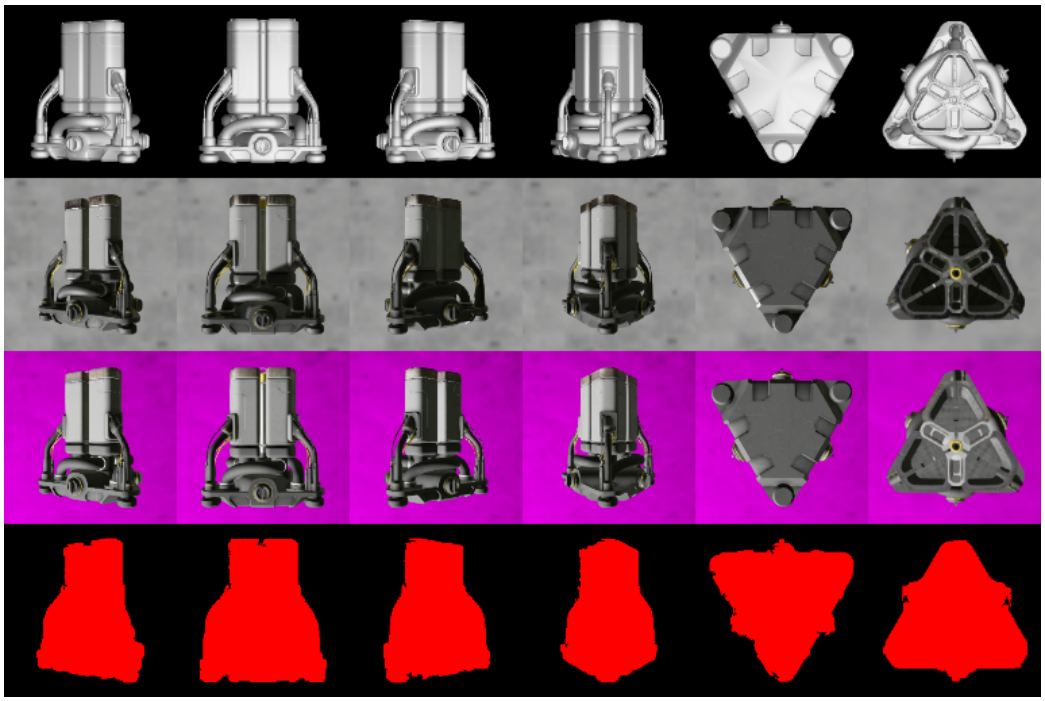


Figure 3.5: Evaluation data for each model, in this example `LargeExhaustPipe`: screenshots on gray background (row 2), screenshots on purple background (row 3), pixel masks (row 4). Row 1 contains the 3D model rendered in the poses specified for each screenshot.

Each of the 6 screenshots represents one of the principal views of the 3D model: front, back, left, right, bottom, and up. However, in some screenshots the objects are not facing the camera perfectly, instead they are slightly rotated around the vertical axis by a random amount. This is done to emulate the uncertainty in pose estimation which the detector is expected to be robust against during evaluation (more details in Section 3.4).

The 6 DoF pose data provided for each screenshot describes the complete transformation between the camera and the model, also including the random rotation around the vertical axis. The pose information can be used to render the model in the same poses as in the screenshots with minimum error in order to evaluate the detection performance provided that the geometric similarity of the models in screenshot and in the rendered image is near the possible maximum.

The purple background in the alternative screenshots allows the detector to better distinguish the 3D model from the background, further facilitating the detection.

We define the sets of evaluation data (models, screenshots, masks and poses) corresponding to the models from model subsets A and B as **evaluation dataset A** and **evaluation dataset B** respectively.

We define the complete set of evaluation data for all provided models as **evaluation dataset C**. This dataset is used for the final evaluation and comparison of the detectors' performance.

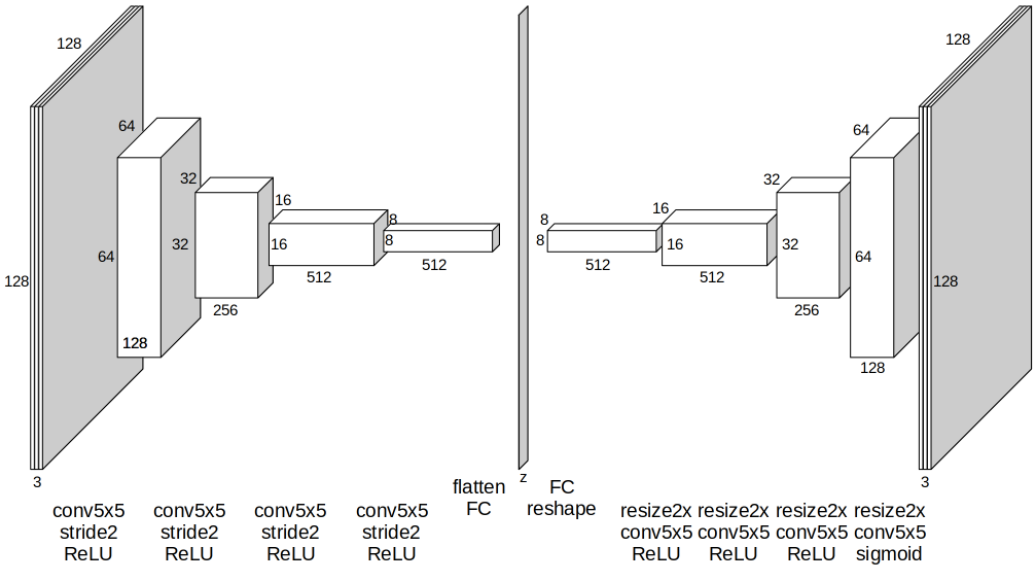
3.2 Multi-Path Augmented AutoEncoder

The complete structure of the Multi-Path Augmented AutoEncoder is presented in Figure 3.6. It consists of a single convolutional encoder and several identical but independent decoders (with no shared weights).

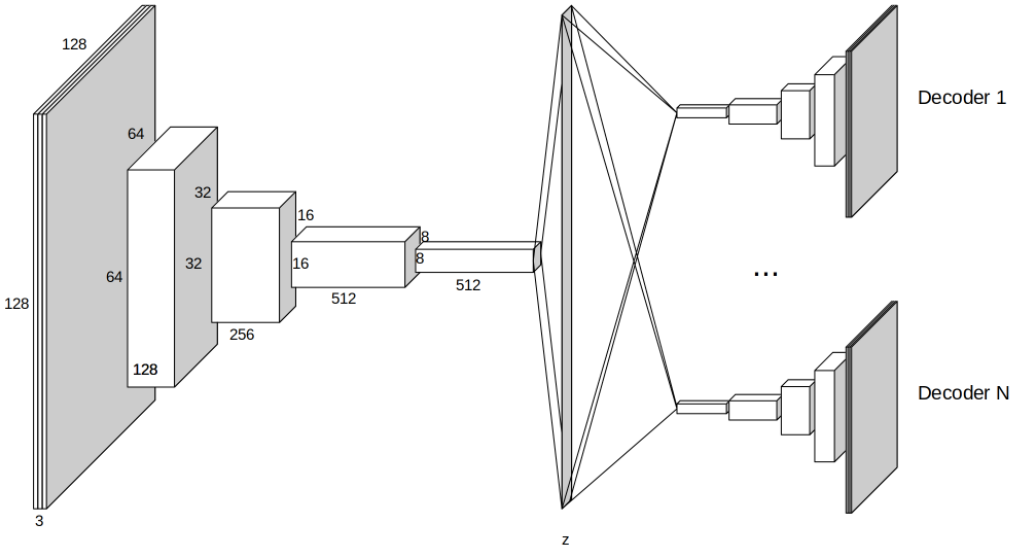
The decoders are discarded after training and the encoder part is used as latent representation generator within the similarity detector. The similarity metric for the latent representations is *cosine similarity* which is defined as follows:

$$S_C(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{3.1}$$

We mainly use training dataset A to train multiple variations of our autoencoder architecture.



(a) Single decoder



(b) Multiple decoders

Figure 3.6: Diagrams of the Multi-Path Augmented AutoEncoder (adapted from [13]), "resize2x" depicts nearest-neighbor upsampling.

In order to determine the influence of the **number of decoders** and the distribution of training data between them on the resulting detector's performance we examine three different arrangements of decoders:

1. one decoder per 3D model: 16 decoders in total, each decoder reconstructs images corresponding to one single stage of each block;
2. one decoder per block: 4 decoders in total, each decoder reconstructs images corresponding to all 4 stages of one single block;
3. one decoder for all models: a single decoder reconstructs all images from the dataset.

Along with the decoder part we examine the influence of the bottleneck part of the autoencoder by considering multiple values for the **latent space dimensionality**:

1. $\mathbb{R}^{128}$
2. $\mathbb{R}^{256}$
3. $\mathbb{R}^{512}$.

Apart from structural features we examine the influence of **data augmentation** on detection performance. The augmented images are provided as input to the autoencoder and the unaugmented versions of the same images are used as reconstruction goal for the decoders. This allows to enforce invariance to certain image features in the resulting latent representations constructed by the encoder.

Since our task requires the detector to be invariant to background and lighting and robust against small errors in objects' positions within the field of view, we use:

- background augmentation
- geometric augmentations (scaling and in-plane translation)
- rendering-time lighting augmentation (random lighting).

These augmentations techniques are evaluated by Sundermeyer et al. in their original paper [13] and are used in all our experiments, their influence on detection performance is not examined.

However, since our task additionally requires texture invariance, we do examine the influence of **partial texturing** and **brightness/contrast variations** on detection performance.

In addition to data augmentation techniques we examine the influence of the **diversity** of the dataset by additionally training an autoencoder on training dataset B.

In all experiments, a single-decoder autoencoder is used, unless explicitly stated otherwise.

3.3 Siamese networks

We use two different kinds of Siamese architectures: a **contrastive loss Siamese network** and a **triplet loss Siamese network** (Figure 3.7).

Both kinds consist of a certain number of identical parallel encoder branches with shared weights that generate latent representation vectors for the input images. These representations are then compared using the L2-distance metric and the losses are constructed in such a way as to increase the distance between the representations of dissimilar images and decrease the distance between the representations of similar ones.

After training, a single encoder branch of the Siamese network is used for latent representation generation in the similarity detector. All branches have the exact same structure as the encoder part of the Multi-Path Augmented AutoEncoder which means that the difference between the encoders extracted from different self-supervised architectures lies purely in the training method.

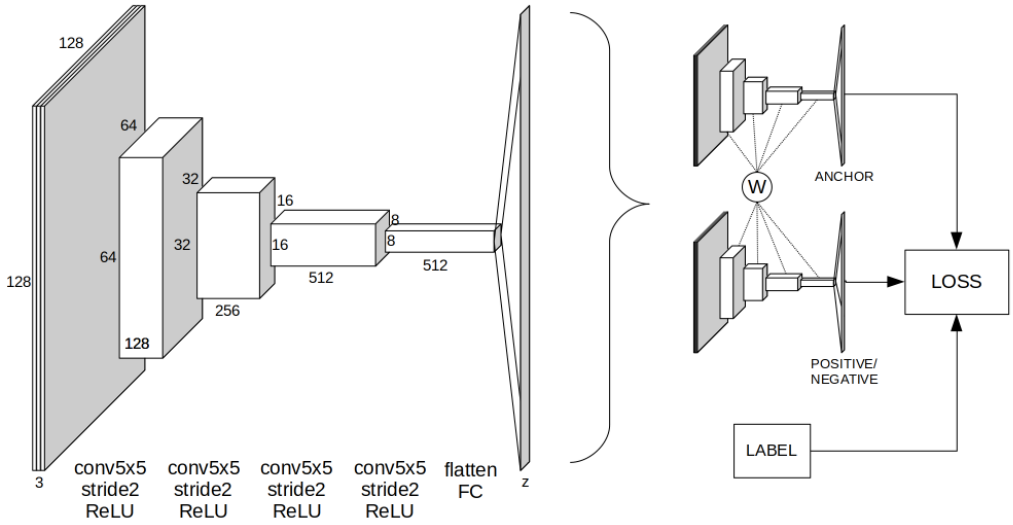
The similarity metric for the latent representations generated by Siamese networks is the *negative squared L2-distance* which is defined as follows:

$$S_{L2}(\mathbf{x}, \mathbf{y}) = -\|\mathbf{x} - \mathbf{y}\|^2 \tag{3.2}$$

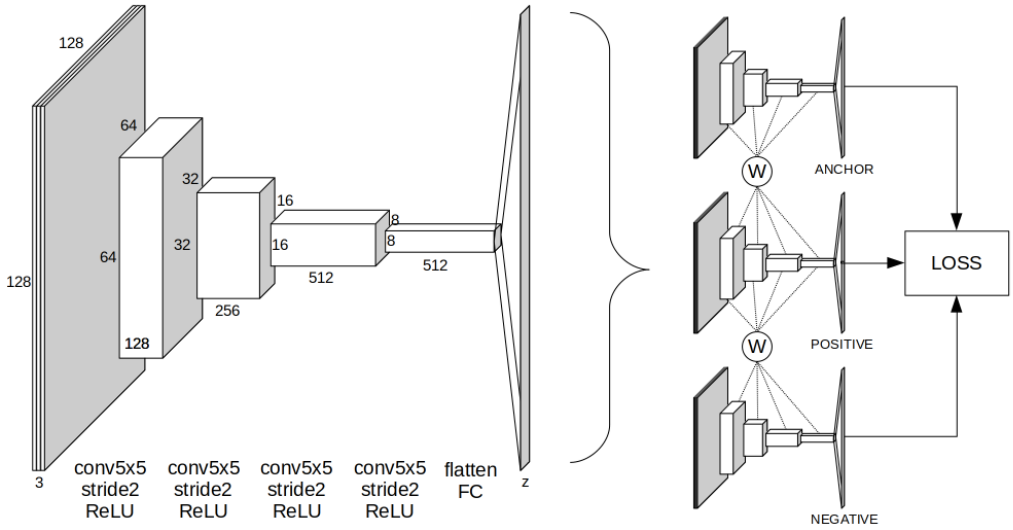
The **contrastive loss network** consists of two encoder branches with shared weights and one label branch (Figure 3.7a). The input for the network are pairs of images with a label which determines whether the pair is an example of similar images (label 1) or dissimilar images (label 0).

The images are fed into the encoders in order to obtain 1D latent representation vectors for each image. The loss is calculated based on the label and the L2-distance between the vectors as follows:

$$L_c = (1 - \hat{y}) \cdot \sigma(\|\mathbf{x} - \mathbf{y}\|)^2 + \hat{y} \cdot (1 - \sigma(\|\mathbf{x} - \mathbf{y}\|))^2 \tag{3.3}$$



(a) Network with contrastive loss



(b) Network with triplet loss

Figure 3.7: Diagrams of the Siamese networks

where $\mathbf{x}$ and $\mathbf{y}$ are the latent representation vectors, $\hat{y}$ is the ground truth label and $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

The **triplet loss network** consists of three encoder branches with shared weights (Figure 3.7b). The input for the network are triplets of images where the first image is an *anchor* or reference image, the second image is a *positive match*, i.e. an image which is similar to the anchor image, and the third image is a *negative match*, i.e. an image which is dissimilar to the anchor image.

The images are fed into the encoders in order to obtain 1D latent representation vectors for each image. The loss is calculated based on the L2-distances between the three vectors as follows:

$$L_t = \max\{\|\mathbf{a} - \mathbf{p}\|^2 + m - \|\mathbf{a} - \mathbf{n}\|^2, 0\} \tag{3.4}$$

where $\mathbf{a}$, $\mathbf{p}$, and $\mathbf{n}$ are the latent representations of the anchor, the positive match and the negative match respectively and m is the margin (10 in our case).

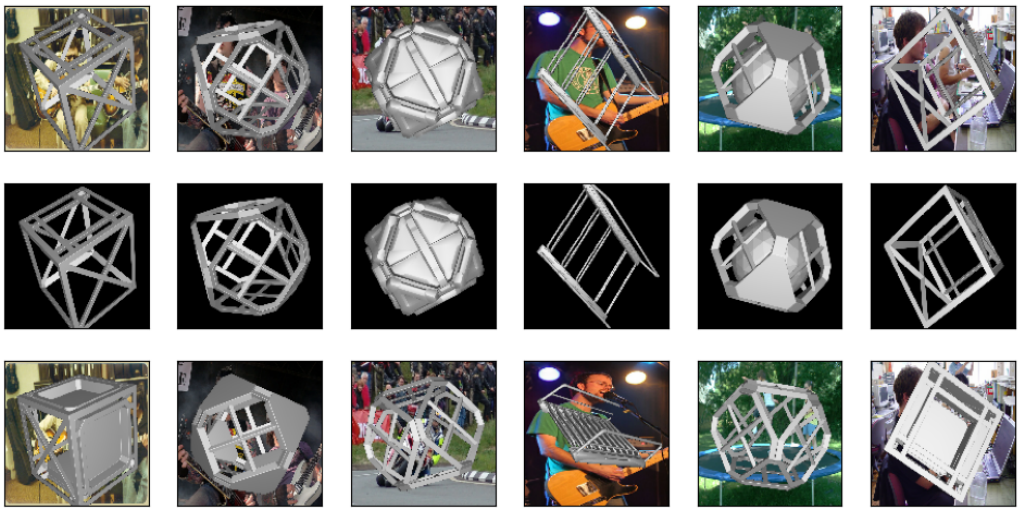
We form the training datasets for the networks by first forming a set of triplets for the triplet loss network, each consisting of an anchor, a positive and a negative match (Figure 3.8a), and then converting it to a set of pairs for the contrastive loss network by splitting each triplet into two labeled pairs: the anchor paired with the positive match labeled 1 and the anchor paired with the negative match labeled 0 (Figure 3.8b).

We mainly use training dataset A and augmentations described in Section 3.1.2 to create triplet datasets. For each image from the respective training dataset, we construct a triplet as follows (Figure 3.8a):

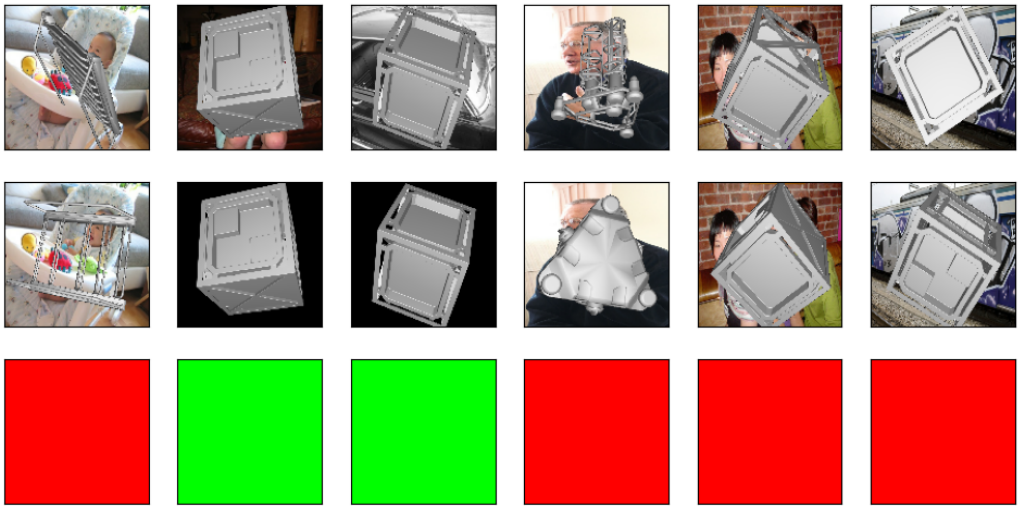
- set the augmented version of the image as *anchor*
- set the original image as *positive match*
- randomly pick an image of a different model, apply the same augmentations to it and set it as *negative match*.

In that way we keep the relevant features and remove the irrelevant features of the anchor image in the positive match and we keep the irrelevant features constant and provide different relevant features in the negative match.

In order to examine the ability of Siamese networks to learn feature-invariant latent representation for different features, we generate one triplet



(a) Batch from a triplet dataset for the triplet network (top row: anchors, middle row: positive matches, bottom row: negative matches).



(b) Batch from a pair dataset for the contrastive network (top row: anchors, middle row: positive or negative matches, bottom row: label, green for 1 (positive) and red for 0 (negative))

Figure 3.8: Batches of the training datasets for Siamese networks (here: with background augmentations)

dataset and one pair dataset for each of the following augmentation techniques:

1. background augmentation
2. geometric augmentations
3. random lighting
4. contrast/brightness variations
5. partial textures

and examine the influence of each separate technique on detection performance.

Apart from feature invariant learning we examine the ability of the Siamese networks to learn object similarity features that can be transfered between different domains. In our case the training domain is the space of textureless OpenGL images with black background and the evaluation domain is the space of textured in-game screenshots with variable background.

We train the models on an additional triplet dataset (and the respective pair dataset, Figures 3.9a and 3.9b) where the augmented images are replaced with random unaugmented images of the same model from a different view and the negative matches are replaced with random unaugmented images of a different model.

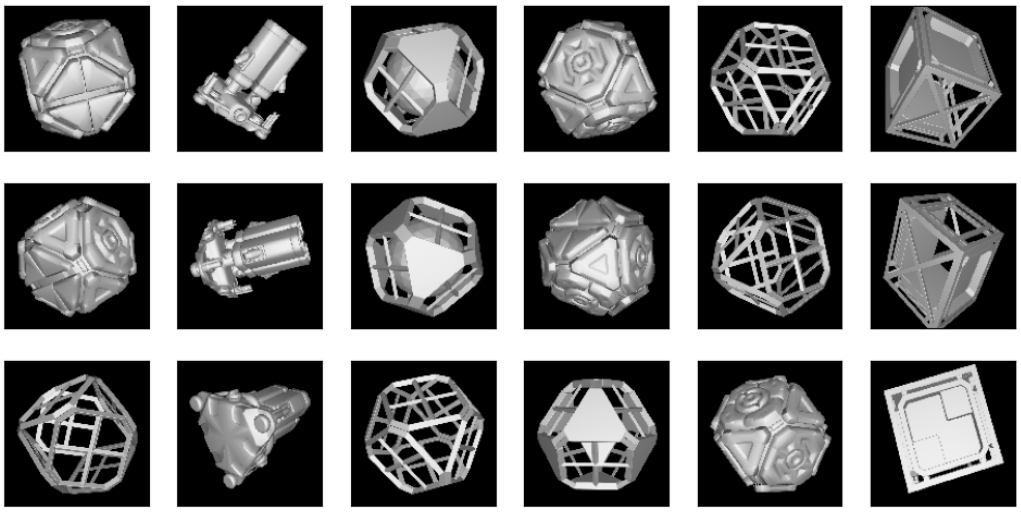
We also generate a similar pair of datasets from train dataset B in order to examine the influence of the dataset diversity on the performance of similarity feature learning.

3.4 Evaluation

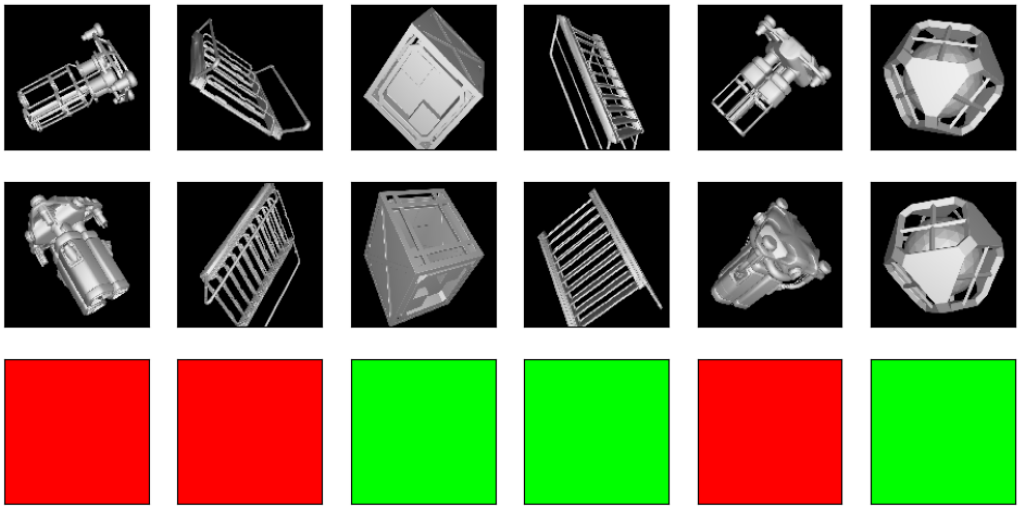
3.4.1 Experiment evaluation

In all our experiments evaluation is performed under the following conditions, unless explicitly stated otherwise:

- detectors are evaluated on evaluation dataset C (all models)
- purple background screenshots are used as input



(a) Batch from a triplet dataset for the triplet network (top row: anchors, middle row: positive matches, bottom row: negative matches).



(b) Batch from a pair dataset for the contrastive network (top row: anchors, middle row: positive or negative matches, bottom row: label, green for 1 (positive) and red for 0 (negative))

Figure 3.9: Batches of the training datasets for Siamese networks with augmented images replaced by images from different viewpoints

- for each screenshot, detectors render a single image of the model in the exact pose provided in the dataset on uniform purple background.

These conditions represent the most favorable situation in which the detector can be used during its real-world operation, so the obtained results in each case most likely represent the upper bound for the detector's performance.

Based on the results of all experiments for each architecture we combine the modifications and augmentations which are shown to have a positive effect on detection performance and apply them all at once to train the final models.

We also perform an ablation study to determine whether any of the improvements have a negligible or an opposite effect on detection performance when used in combination with other methods.

3.4.2 Final evaluation

For the final evaluation, one experiment is performed on dataset C for each combination of the following parameters:

- similar/dissimilar backgrounds:
  - the pink screenshot is used and the model is rendered on pink background
  - the gray screenshot is used and the model is rendered on black background
- use of the model pose by the detector:
  - render one reference image for each screenshot, use the exact pose
  - render one reference image for each screenshot, use a perturbed pose
  - do not use the pose; render all principal views of the model.

Using similar and dissimilar backgrounds allows to evaluate the background invariance property of the detector.

Providing the detector with exact model poses allows to evaluate its performance given the maximum geometric similarity of the models in the screenshot and in the reference image.

Using a perturbed pose allows to evaluate the robustness of the detector against errors in the model's pose given that the pose is provided but contains errors which do not exceed certain boundaries.

Providing no pose information to the detector and letting it render all principal views of the model as reference images allows to evaluate its ability to independently determine the correct orientation of the model and to produce a decision based on its estimation.

Since some blocks look identical from certain angles and have to be aligned in a certain way in order to be distinguished from one another, GoodAI also allows a **modification** of the detector which takes the screenshots of all **6 principal views** of the model at once as input. This modification is evaluated on the same combinations of parameters as the original detector.

Additionally, we define and evaluate another **modification** specifically for distinguishing between models which belong to the **same block** but to **different completeness stages**. Along with the expected 3D model, the modified version of the similarity detector receives the models which belong to each of the remaining completeness stages of the block in question and returns a positive prediction if the original 3D model has the largest similarity to the model in the screenshots among all given models.

The evaluation is performed accoring to two evaluation algorithms proposed by GoodAI. The first algorithm (see Algorithm 1) evaluates the ability of the detector to distinguish between different completeness stages of the same block, the second (see Algorithm 2) — between different blocks.

As input, both algorithms receive a dataset $D = \{d_{b,s}\}$, a prediction function $P_D(d, c)$ which is an abstraction of the similarity detector and the color c of the background on which the detector should render the models.

Each entry $d_{b,s} = (S_{b,s}, m_{b,s})$ of the dataset D corresponds to a single block b and stage s and contains the respective 3D model $m_{b,s}$ and a set $S_{b,s} = \{(s^{(i)}_{b,s}, p^{(i)}_{b,s})\}$ of one or more pairs screenshot/pose.

In our experiments, the set $S_{b,s}$ can either contain a single screenshot/pose pair corresponding to one single view of the model or 6 pairs corresponding to 6 principal views of the model (for the original/modified detector respectively). The pose in each pair is either the exact model pose for the respective screenshot or a perturbed pose or an invalid pose which tells the detector that it needs to render all principle views as reference images.

The prediction function $P_D(d, c)$ receives a data entry $d = (S, m)$ and a background color c as input and returns a binary prediction (True or False)

on whether the model in the screenshots of S is the same as model m. It is defined separately for each architecture using Algorithm 3 for the standard similarity detector and Algorithm 4 for the modification for different stages of the same block. The latter is only evaluated using Algorithm 1 while the former is evaluated using both evaluation algorithms.

In case of Augmented AutoEncoder architecture, the encoder $E(s)$ passed as input to Algorithm 3 and Algorithm 4 is the encoder branch of the autoencoder and the similarity metric $Sim(\mathbf{x}, \mathbf{y})$ is the cosine distance S_C defined in Equation 3.1. For Siamese networks the encoder $E(s)$ is a single encoding branch of the network and the similarity metric $Sim(\mathbf{x}, \mathbf{y})$ is the L2 similarity metric S_{L2} defined in Equation 3.2.

The similarity threshold Sim_{thresh} is calculated separately for each architecture as to maximize the detection performance on the evaluation dataset.

Input: dataset $D = \{(S_{b,s}, m_{b,s})\}$ of tuples (set of pairs screenshot/pose $S_{b,s}$, 3D model $m_{b,s}$) for each block b and stage s; background color $c \in \{\text{black, purple}\}$; prediction function $P_D(d, c)$ of the similarity detector

Output: accuracy of the detection, precision, and recall

`/* set of prediction/label pairs */`

$PL \leftarrow \emptyset$;

foreach $d_{b,s} = (S_{b,s}, m_{b,s}) \in D$ **do**

  prediction $\leftarrow P_D(d_{b,s}, c)$;

  ground truth label $\leftarrow$ True;

  $PL \leftarrow PL \cup \{(\text{prediction, ground truth label})\}$;

  foreach $(S_{b',s'}, m_{b',s'}) \in D : b' = b \wedge s' \neq s$ **do**

    prediction $\leftarrow P_D((S_{b,s}, m_{b',s'}), c)$;

    ground truth label $\leftarrow$ False;

    $PL \leftarrow PL \cup \{(\text{prediction, ground truth label})\}$;

  end

end

true positives $\leftarrow \{(p, l) \in PL : p = \text{True} \wedge l = \text{True}\}$;

false positives $\leftarrow \{(p, l) \in PL : p = \text{True} \wedge l = \text{False}\}$;

true negatives $\leftarrow \{(p, l) \in PL : p = \text{False} \wedge l = \text{False}\}$;

false negatives $\leftarrow \{(p, l) \in PL : p = \text{False} \wedge l = \text{True}\}$;

accuracy $\leftarrow \frac{|\text{true positives}|+|\text{true negatives}|}{|PL|}$;

precision $\leftarrow \frac{|\text{true positives}|}{|\text{true positives}|+|\text{false positives}|}$;

recall $\leftarrow \frac{|\text{true positives}|}{|\text{true positives}|+|\text{false negatives}|}$;

Algorithm 1: Evaluation 1: use different stages of the same block as negative examples

Input: dataset $D = \{(S_{b,s}, m_{b,s})\}$ of tuples (set of pairs screenshot/pose $S_{b,s}$, 3D model $m_{b,s}$) for each block b and stage s; background color $c \in \{\text{black, purple}\}$; prediction function $P_D(d, c)$ of the similarity detector
Output: accuracy of the detection, precision, and recall
`/* set of prediction/label pairs */`
$PL \leftarrow \emptyset$;
foreach $d_{b,s} = (S_{b,s}, m_{b,s}) \in D$ **do**
  prediction $\leftarrow P_D(d_{b,s}, c)$;
  ground truth label $\leftarrow$ True;
  $PL \leftarrow PL \cup \{(\text{prediction, ground truth label})\}$;
  $D' = \{(S_{b',s'}, m_{b',s'}) \in D : b' \neq b\}$;
  D'_{rand} = set of 10 random tuples from D';
  foreach $(S_{b',s'}, m_{b',s'}) \in D'_{rand}$ **do**
    prediction $\leftarrow P_D((S_{b,s}, m_{b',s'}), c)$;
    ground truth label $\leftarrow$ False;
    $PL \leftarrow PL \cup \{(\text{prediction, ground truth label})\}$;
  end
end
true positives $\leftarrow \{(p, l) \in PL : p = \text{True} \wedge l = \text{True}\}$;
false positives $\leftarrow \{(p, l) \in PL : p = \text{True} \wedge l = \text{False}\}$;
true negatives $\leftarrow \{(p, l) \in PL : p = \text{False} \wedge l = \text{False}\}$;
false negatives $\leftarrow \{(p, l) \in PL : p = \text{False} \wedge l = \text{True}\}$;
accuracy $\leftarrow \frac{|\text{true positives}|+|\text{true negatives}|}{|PL|}$;
precision $\leftarrow \frac{|\text{true positives}|}{|\text{true positives}|+|\text{false positives}|}$;
recall $\leftarrow \frac{|\text{true positives}|}{|\text{true positives}|+|\text{false negatives}|}$;

Algorithm 2: Evaluation 2: use random stages of different blocks as negative examples

Input: encoder $E(s)$; similarity metric $Sim(\mathbf{x}, \mathbf{y})$; similarity threshold Sim_{thresh}; tuple $d = (S, m)$ where $S = \{(s, p)\}$ is a set of pairs screenshot/pose, m is a 3D model; background color $c \in \{\text{black, purple}\}$
Output: prediction function P_D
using poses $\leftarrow (\forall (s, p) \in S : p$ is a valid 6 DoF pose);
if *using poses* **then**
  similarities $\leftarrow \emptyset$;
  foreach $(s, p) \in S$ **do**
    $r \leftarrow$ rendered image of m in pose p on background c;
    similarities $\leftarrow$ similarities $\cup\ Sim(E(s), E(r))$;
  end
  decision $\leftarrow$ (`mean`(similarities) $> Sim_{thresh}$);
else
  $R \leftarrow$ rendered principal views of model m on background c;
  similarities $\leftarrow \{Sim(E(s), E(r)) : \forall ((s, p), r) \in S \times R\}$;
  decision $\leftarrow$ (`min`(similarities) $> Sim_{thresh}$);
end
$P_D \leftarrow P_D(d, c) : (d, c) \mapsto$ decision;

Algorithm 3: Definition of similarity detector's prediction function

Input: encoder $E(s)$; similarity metric $Sim(\mathbf{x}, \mathbf{y})$; similarity threshold Sim_{thresh}; tuple $d = (S, \hat{m}, M)$ where $S = \{(s, p)\}$ is a set of pairs screenshot/pose, $\hat{m}$ is the expected 3D model, M is the set of models of all other stages of the same block; background color $c \in \{\text{black, purple}\}$
Output: prediction function P_D
using poses $\leftarrow (\forall (s, p) \in S : p$ is a valid 6 DoF pose);
`/* set of similarities for each model as pairs model/similarity */`
$P_{m/s} \leftarrow \emptyset$;
foreach $m \in \{\hat{m}\} \cup M$ **do**
  if *using poses* **then**
    similarities $\leftarrow \emptyset$;
    foreach $(s, p) \in S$ **do**
      $r \leftarrow$ rendered image of m in pose p on background c;
      similarities $\leftarrow$ similarities $\cup\ Sim(E(s), E(r))$;
    end
    $P_{m/s} \leftarrow P_{m/s} \cup (m,$ `mean`(similarities));
  else
    $R \leftarrow$ rendered principal views of model m on background c;
    similarities $\leftarrow \{Sim(E(s), E(r)) : \forall ((s, p), r) \in S \times R\}$;
    $P_{m/s} \leftarrow P_{m/s} \cup (m,$ `min`(similarities));
  end
end
`/* positive decision iff` $\hat{m}$ `has the (strictly) largest similarity value among all models */`
decision $\leftarrow (\forall (m, x), (m', x') \in P_{m/s} : x \geq x' \implies m' \neq \hat{m})$;
$P_D \leftarrow P_D(d, c) : (d, c) \mapsto$ decision

Algorithm 4: Definition of similarity detector's prediction function for distinguishing between stages of the same block

Chapter 4

Results

In this chapter we are referring to the task of distinguishing between different stages of the same block (with its performance evaluated using Algorithm 1) as **Task 1** and to the task of distinguishing between different blocks (with its performance evaluated using Algorithm 2) as **Task 2**.

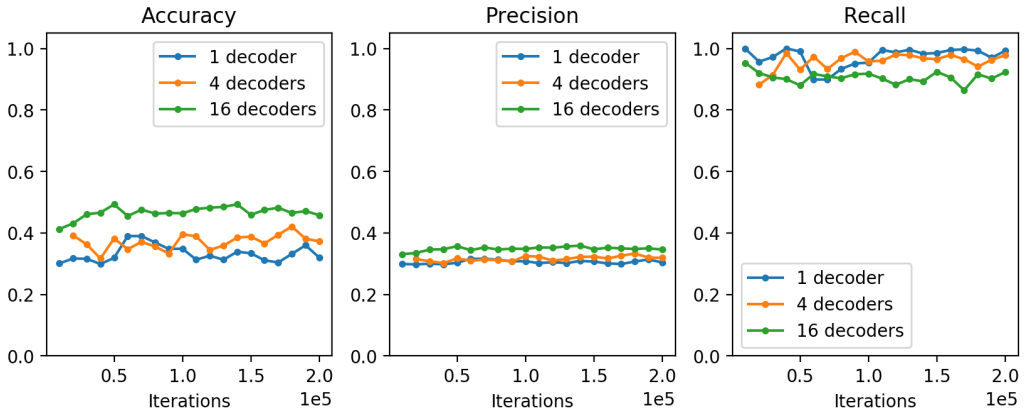
4.1 Detector based on Multi-Path Augmented AutoEncoder

4.1.1 Number of decoders

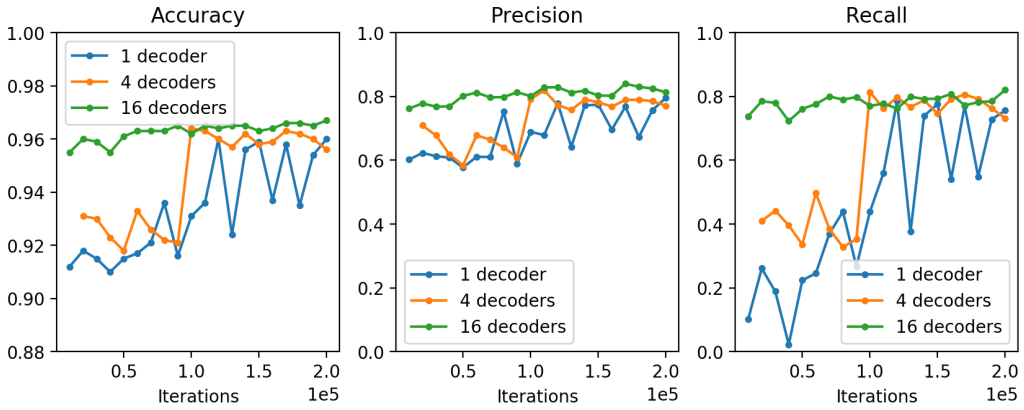
Figure 4.1 shows the performance of the detector based on Multi-Path Augmented AutoEncoder (MPAAE-detector in short) trained on dataset A with one decoder per model (16 in total), one decoder per block (4 in total) and one single decoder for all models. The results suggest that increasing the number of decoders leads to an improvement in classification accuracy for Task 2. The possible reasons for this are examined in Chapter 5. The results for Task 1 are inconclusive: there is a slight improvement in accuracy and precision but a decrease in recall.

4.1.2 Number of models in the training set

Figure 4.2 shows the performance of the MPAAE-decoder trained on dataset A versus dataset B. The results reveal that the performance of the latter is significantly lower on both tasks. This indicates that increasing the diversity



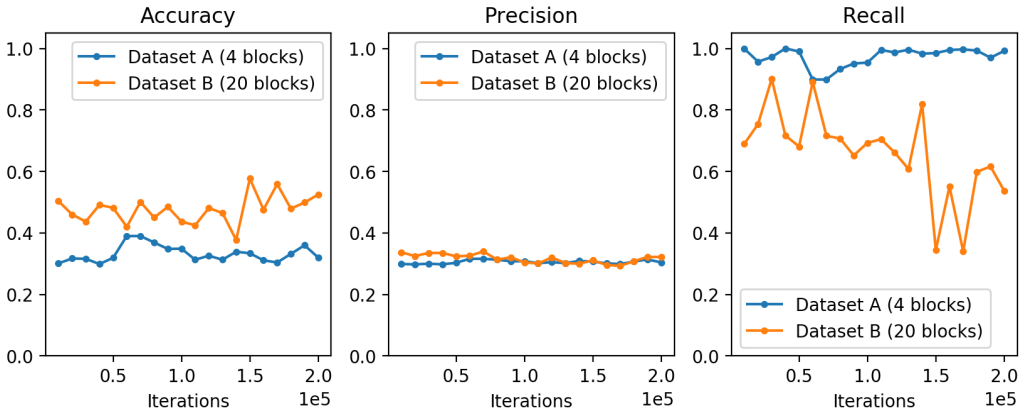
(a) Task 1



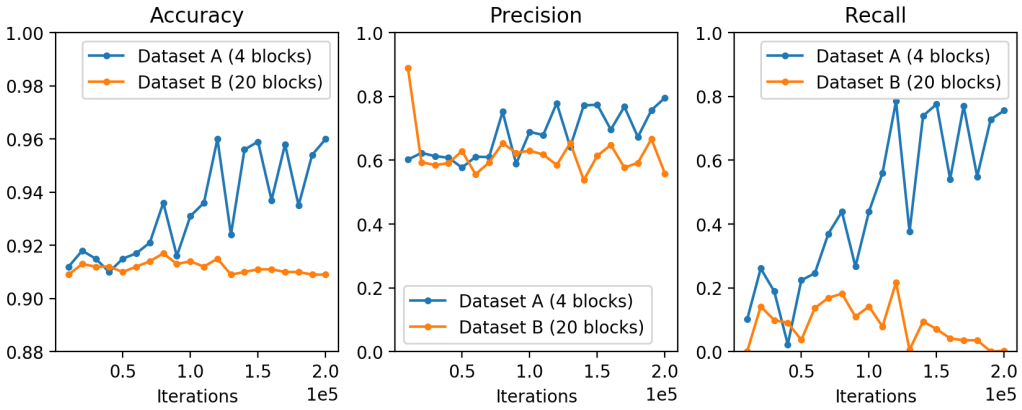
(b) Task 2

Figure 4.1: Performance of the MPAAE-detector trained on dataset A with 1 decoder per model (16 in total) / per block (4 in total) / for all models (1 in total)

of blocks in the dataset does not necessarily lead to an increase in accuracy. The possible reasons for this might be related to the reasons for the superior performance of the 16-decoder-MPAAE-decoder and are examined in Chapter 5.



(a) Task 1



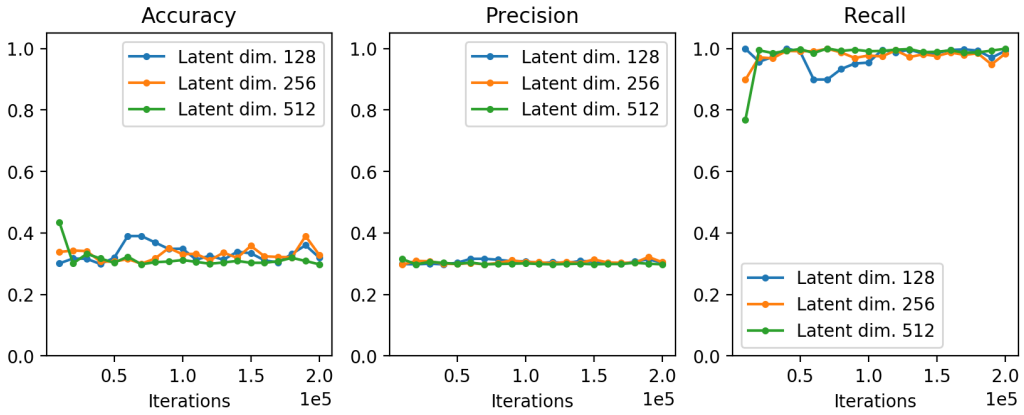
(b) Task 2

Figure 4.2: Performance of the MPAAE-detector trained on dataset A vs. on dataset B

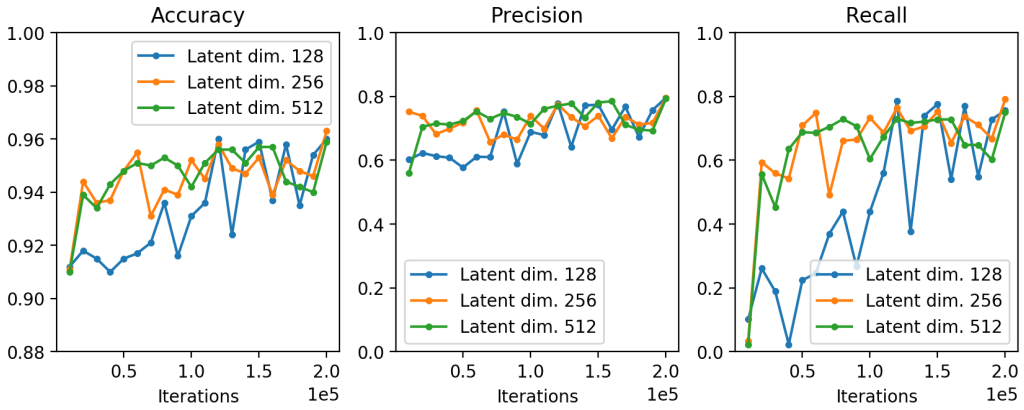
4.1.3 Other augmentation techniques

Figures 4.3, 4.4 and 4.5 show the influence of latent space dimensionality, brightness/contrast augmentations and partial texturing on the performance of MPAAE-detector respectively.

The results on Task 2 show that increasing latent space dimensionality as well as applying random brightness/contrast variations leads to a smoother loss curve and also increases the convergence speed. The effect of partial texturing appears to be less significant but still includes smoothing of the



(a) Task 1

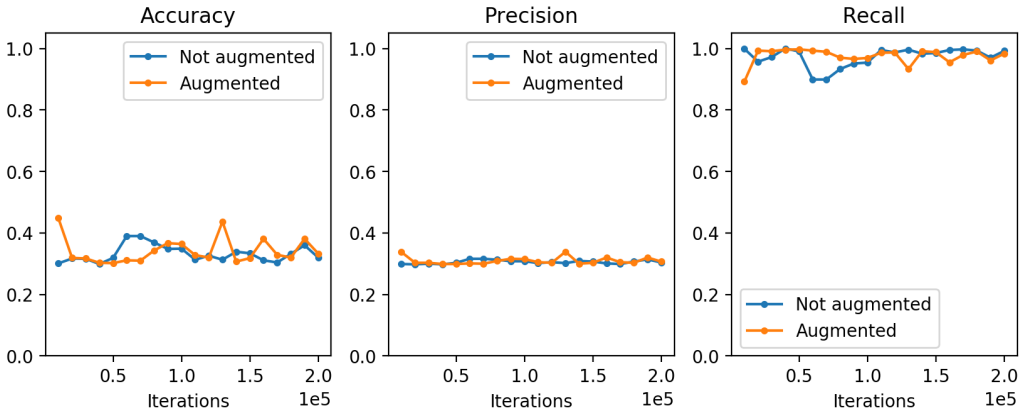


(b) Task 2

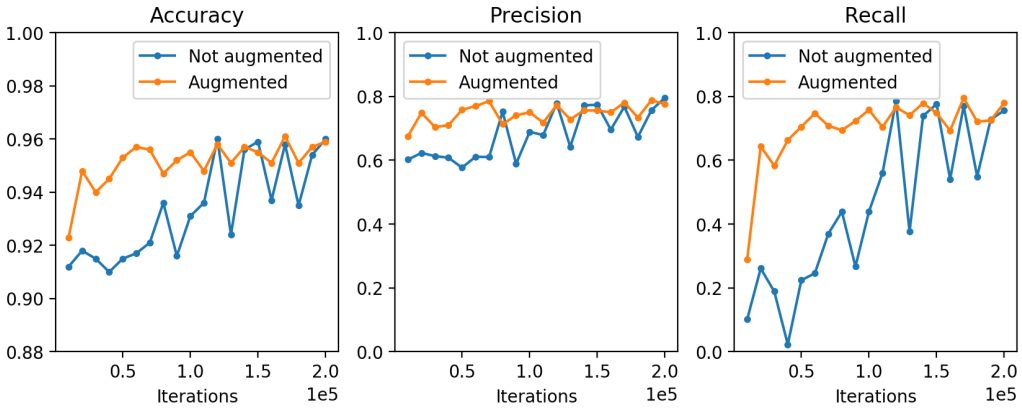
Figure 4.3: Performance of the MPAAE-detector trained with latent space dimensions of 128, 256 and 512

loss curve to some extent.

The results for Task 1 show similar loss curve smoothing for all augmentation techiniques above, however, the classification performance seems to be unaffected by any of the augmentations and does not improve with increasing number of training epochs. This suggests that the performance of the MPAAE-detector in the setting of Task 1, where the differences between the models lie in small details rather than in the overall shape, is near its peak for the given encoder architecture and decision algorithm. Therefore, an alternative decision algorithm is proposed and evaluated in Chapter 5.



(a) Task 1



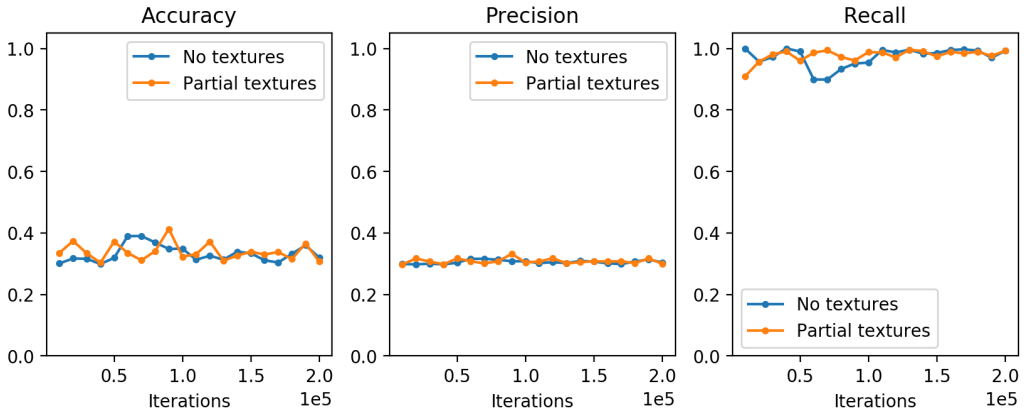
(b) Task 2

Figure 4.4: Performance of the MPAAE-detector trained on augmented images with brightness/contrast variations vs. normal images

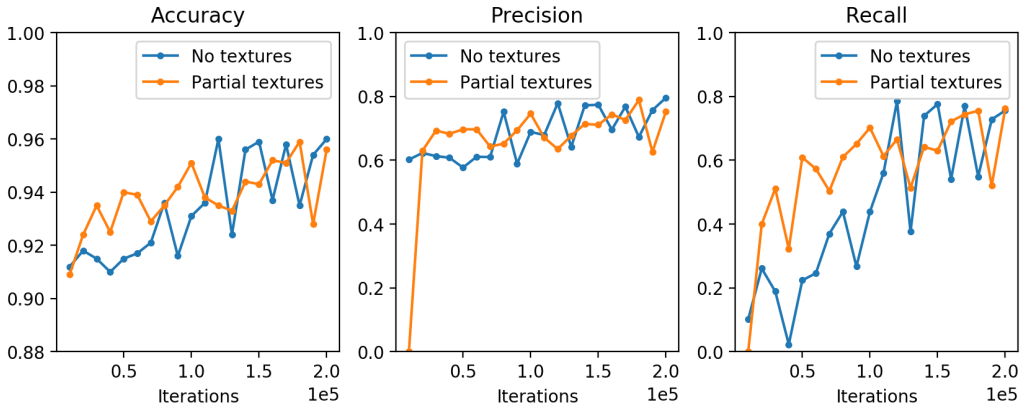
4.1.4 Final model and ablation study

In the final model we use all augmentation techniques which are shown to have a positive effect on classification performance:

- the number of decoders is set to 16
- the model is trained on training dataset A
- brightness/contrast augmentations are used



(a) Task 1



(b) Task 2

Figure 4.5: Performance of the MPAAE-detector trained on augmented images with partial textures vs. untextured images

- partial texturing is used
- the latent dimension is set to 1024 to account for the increased data diversity.

We perform an ablation study on the final model by modifying each the following parameters individually while keeping the rest unchanged:

- setting the number of decoders to 4 and 1
- removing brightness/contrast augmentations

- removing partial texturing
- setting the latent space dimension to 512 and 256.

Figure 4.6 shows the effect of **different numbers of decoders**. The results on Tasks 1 and 2 confirm that decreasing the number of decoders leads to an increase in loss variance and lower average classification accuracy. The results on Task 1 show the same trend of decreasing recall and increasing precision of the model with growing number of decoders as our original experiment in Section 4.1.1.

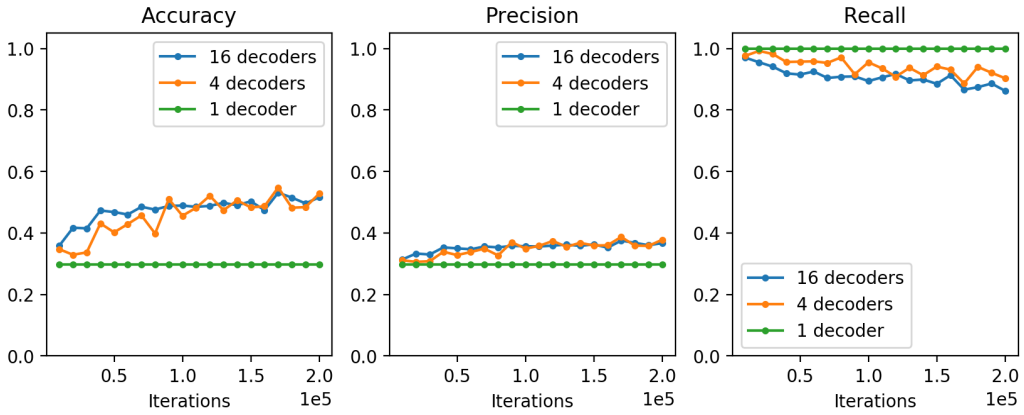
Figure 4.7 shows the effect of the **latent space dimensionality**. The results on Task 2 show that the latent space dimensions of 1024 and 512 provide approximately equal performance while setting the dimensionality to 256 leads to a slight performance decrease. The results on Task 1 show no significant difference in performane. Overall the effect of latent space dimensionality is less significant than those of the other parameters which means that it can be decreased without great performance loss if a smaller model size and faster runtimes are necessary.

Figure 4.8 shows the effect of **brightness/contrast augmentations** which is fairly strong both for Task 1 and Task 2. This suggests that this augmentation technique is of high importance for texture-invariant latent representation learning.

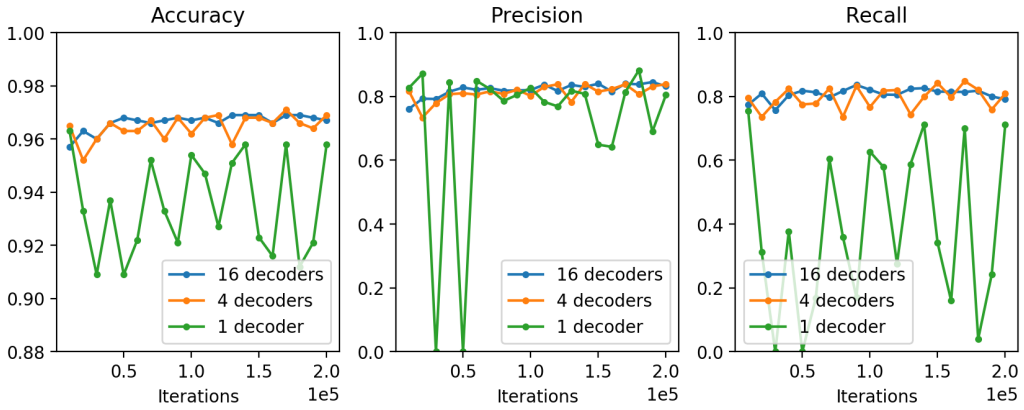
However, the largest impact on performance is produced by excluding **partial texturing** (Figure 4.9). The convergence becomes extremely slow with accuracy and recall showing improvement only after 100k iterations. This result is in strong contrast with the result from Section 4.1.3 and suggests a significant difference in the influence of texturing between the multi-decoder and single-decoder settings.

4.2 Detector based on siamese networks

All our Siamese models trained to provide feature invariance are shown unable to produce an encoder that would be applicable in the domain of in-game screenshots. In all cases, the maximum classification performance is reached when the recall is zero and corresponds to the strategy of generating negative predictions for all inputs. This is the case for all types of augmentations and both types of networks.



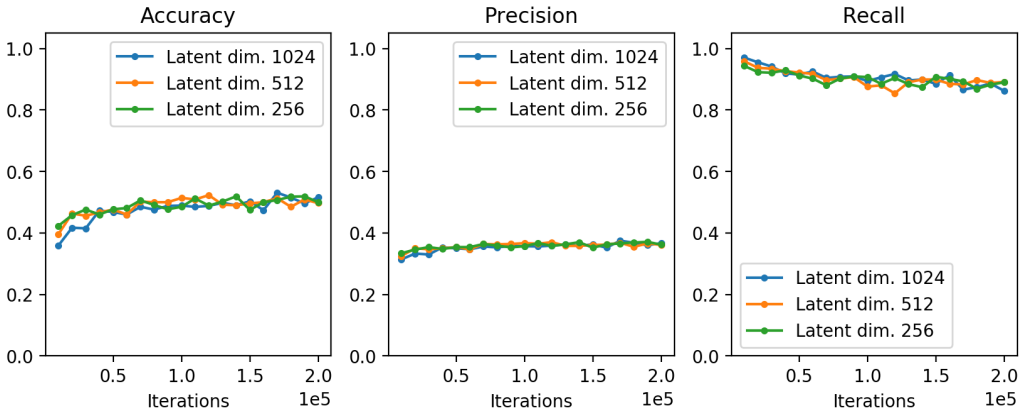
(a) Task 1



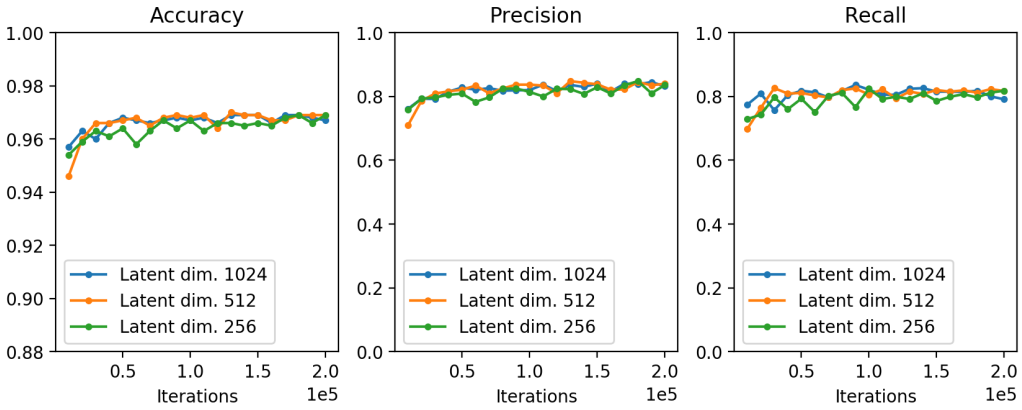
(b) Task 2

Figure 4.6: Performance of the final MPAAE-detector with 1 decoder per model (16 in total) / per block (4 in total) / for all models (1 in total)

However, training the networks on randomized unaugmented images (as shown in Figure 3.9) does produce encoders capable of capturing domain-invariant visual features of the models. Given that the screenshot and the rendered reference image have a similar background, the contrastive network encoder achieves the accuracy of 0.931 with precision/recall of 0.662/0.487 and the triplet network encoder achieves the accuracy of 0.925 with precision/recall of 0.724/0.291 evaluated on dataset C (Figures 4.11 and 4.10). Those values are improved by using screenshots of multiple views, as we show later in Section 4.3 and Figures 4.13 and 4.12.



(a) Task 1



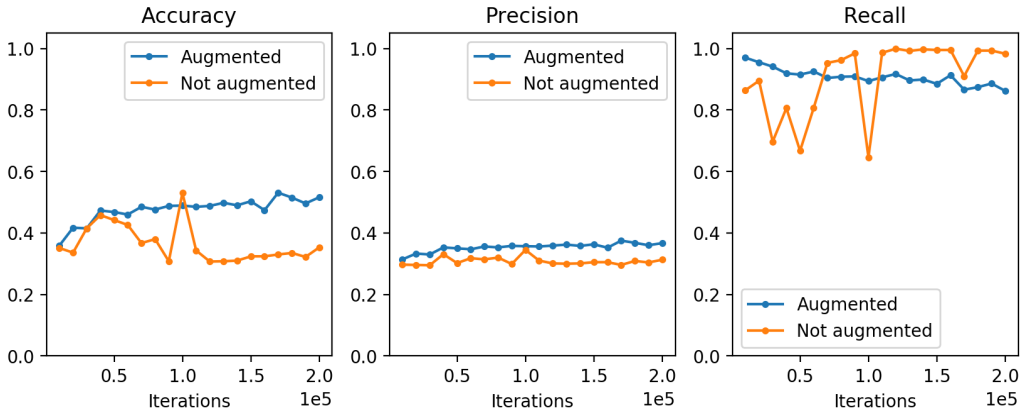
(b) Task 2

Figure 4.7: Performance of the final MPAAE-detector trained with latent space dimensions of 1024, 512 and 256

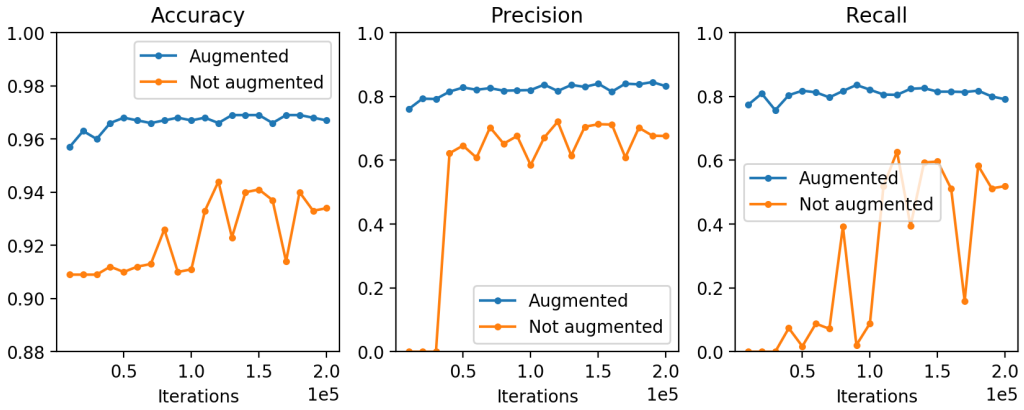
4.3 Final performance comparison

Tables 4.1, 4.2 and 4.3 show the results of the final evaluation of the similarity detectors according to the method described in Section 3.4.2. Each table represents one decision algorithm and one evaluation algorithm applied to all 3 models with all combinations of the remaining parameters.

Table 4.1 shows the results on Task 2 using the original decision algorithm (Algorithm 3). The MPAAE-detector shows consistently superior performance for all combinations of parameters.



(a) Task 1

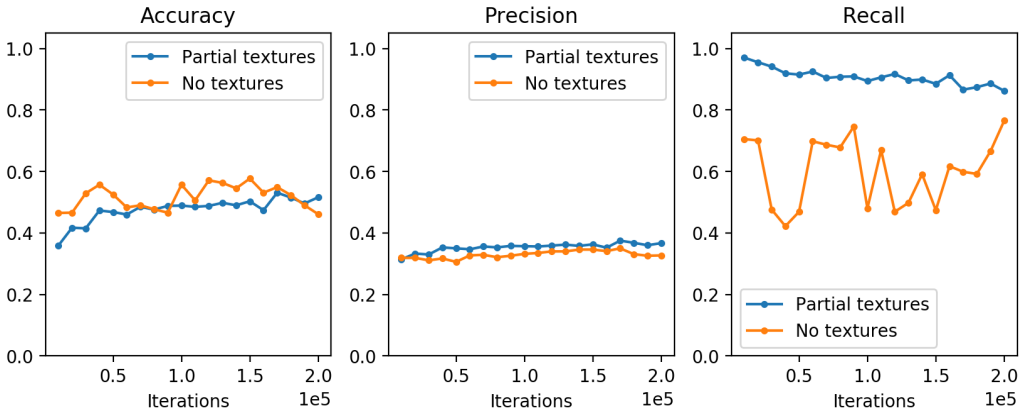


(b) Task 2

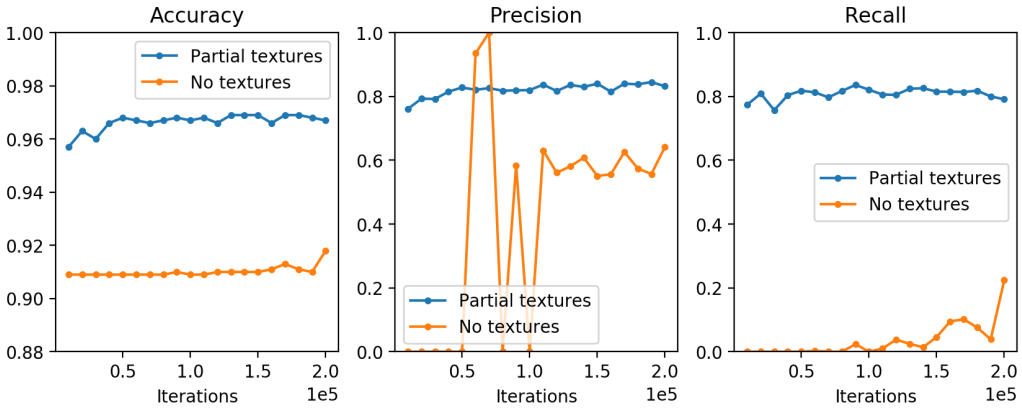
Figure 4.8: Performance of the final MPAAE-detector trained on augmented images with brightness/contrast variations vs. normal images

The comparison of the results of the MPAAE-detector for the settings with similar background ("Same BG") and different background ("Diff. BG") demonstrates the background invariance property of the encoder. The average difference in classification accuracy for the two settings is 0.23%. Meanwhile, the Siamese models are shown to only be applicable for the cases when the images have similar background for which they, however, show reasonable classification performance.

The results for single-view similarity detection (top part of the table) and multi-view detection (bottom part) reveal that classification accuracy



(a) Task 1



(b) Task 2

Figure 4.9: Performance of the final MPAAE-detector trained on augmented images with partial textures vs. untextured images

improves significantly (by an average of around 2%) when all 6 principal views of the model are available in the screenshots. This is true both for the setting when the poses are known (exactly or approximately) and when no pose information is available.

The comparison of the classification performance for exact and approximate pose information ("with pose" vs. "perturbed pose") reveals that all models are to some extent invariant to small pose changes. The difference in classification accuracy for exact and perturbed poses does not exceed 0.5% while in case of missing pose information ("without pose") the accuracy might

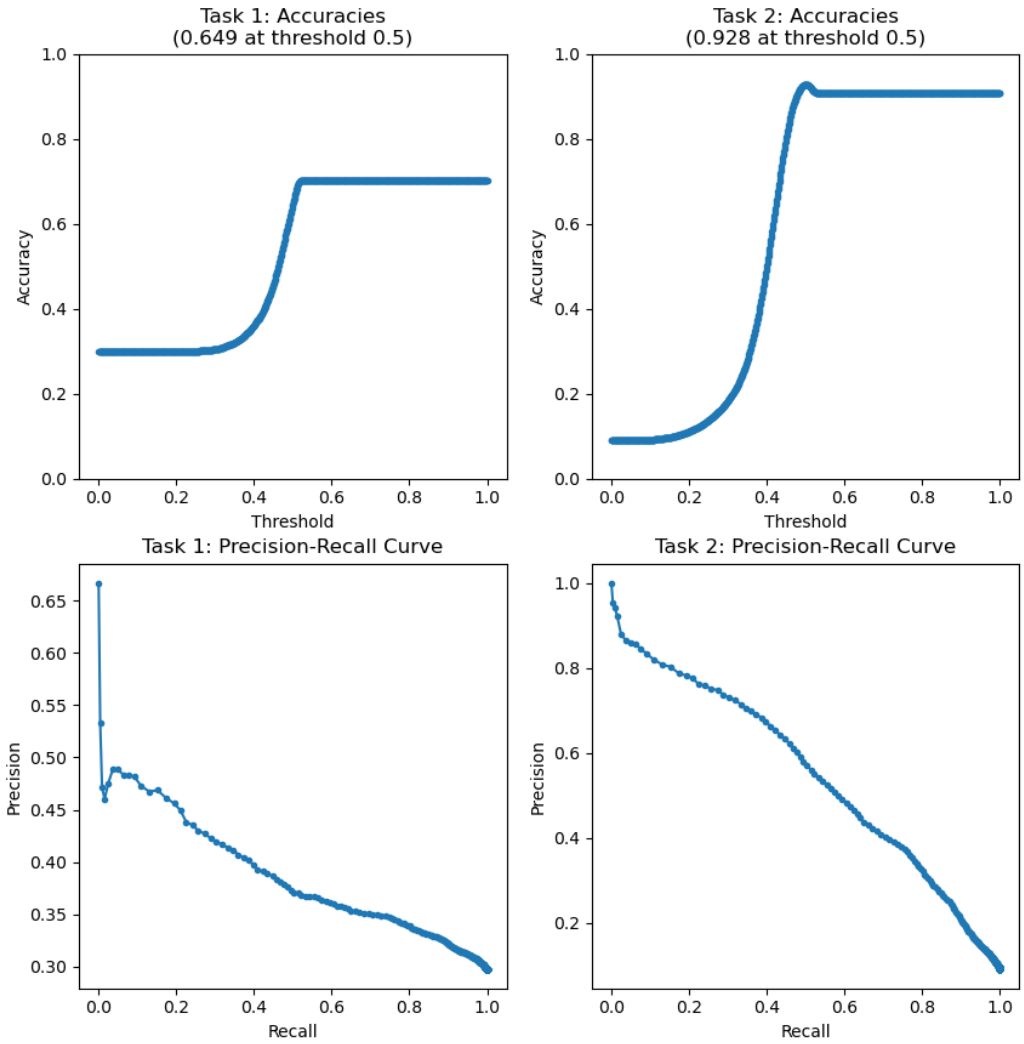


Figure 4.10: Performance of the final triplet Siamese similarity detector trained on randomized unaugmented images evaluated on dataset C using single-view detection

drop by as much as 5%.

Table 4.2 shows that on Task 1, none of the models can produce satisfactory results using the original decision algorithm (Algorithm 3). While the MPAAE-detector demonstrates low classification accuracy, the recall of both Siamese detectors is consistently zero which means that the maximum

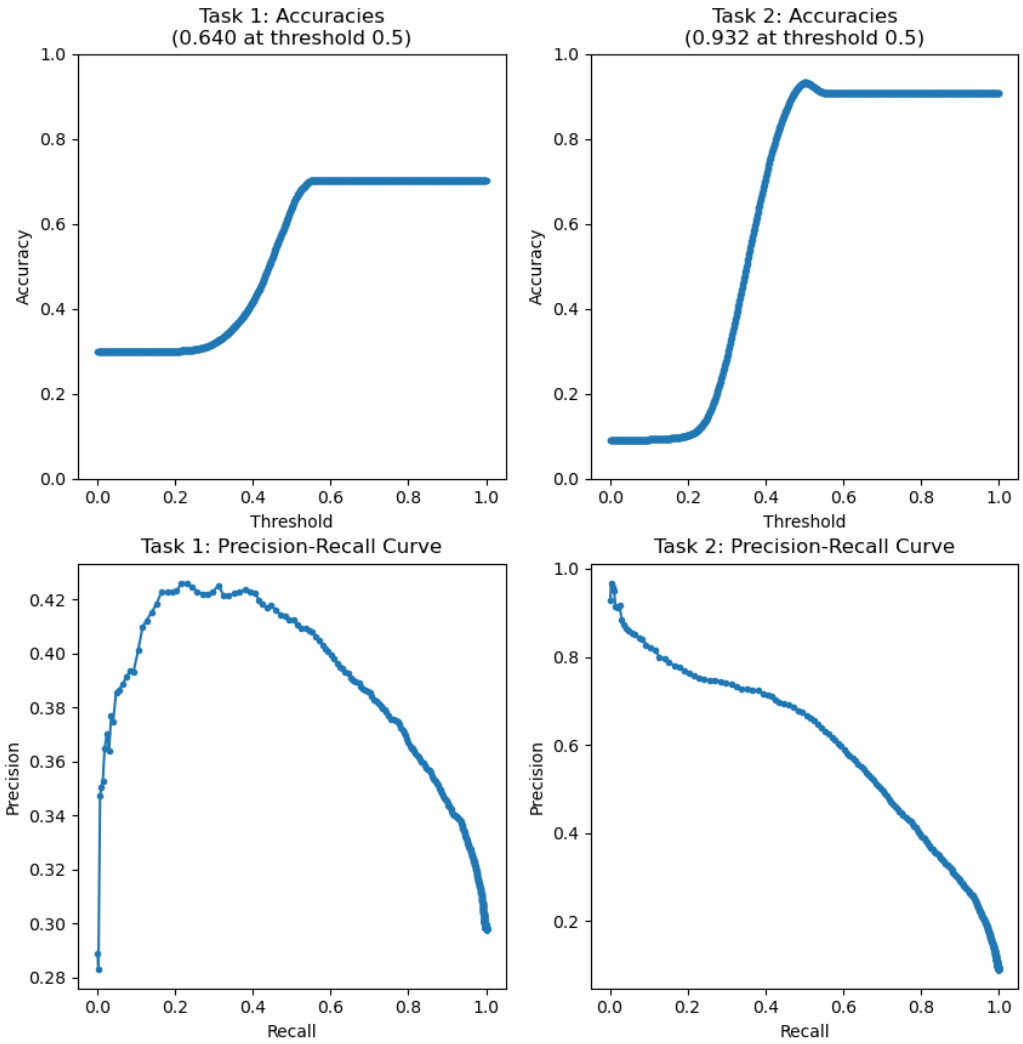


Figure 4.11: Performance of the final contrastive Siamese similarity detector trained on randomized unaugmented images evaluated on dataset C using single-view detection

accuracy of 0.702 is achieved as a result of producing negative predictions for all inputs and the model is not generating any information from the input.

The classification performance is improved significantly with introduction of the modified decision algorithm (Algorithm 4). The results in Table 4.3 show that the classification accuracy increases by as much as 30% in case of

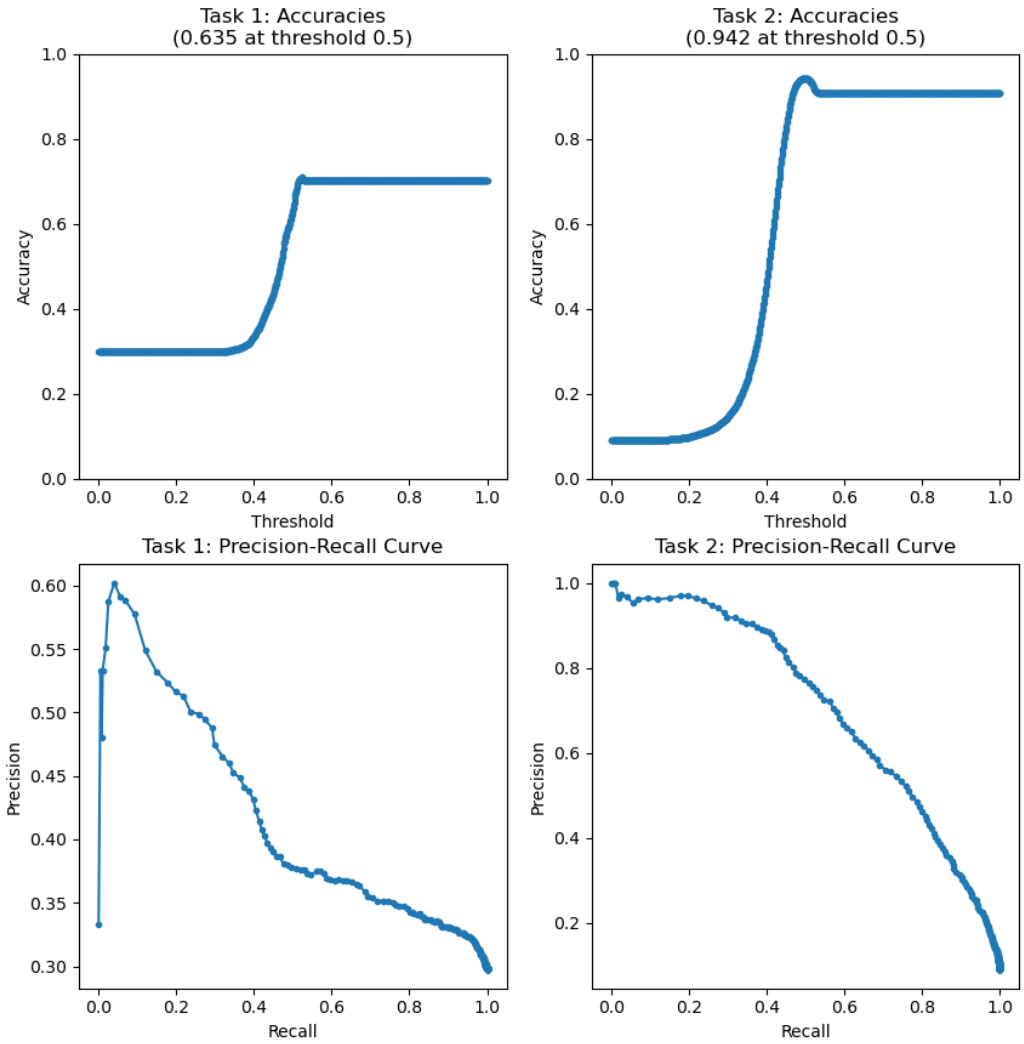


Figure 4.12: Performance of the final triplet Siamese similarity detector trained on randomized unaugmented images evaluated on dataset C using multi-view detection

the MPAAE-detector. Same as on Task 2 (Table 4.1), the MPAAE-detector shows superior performance in all experiments, however, the Siamese detectors, previously unable to produce any reasonable results on Task 1, still demonstrate accuracies of up to 85% with precision and recall of up to 75% in the multi-view setting.

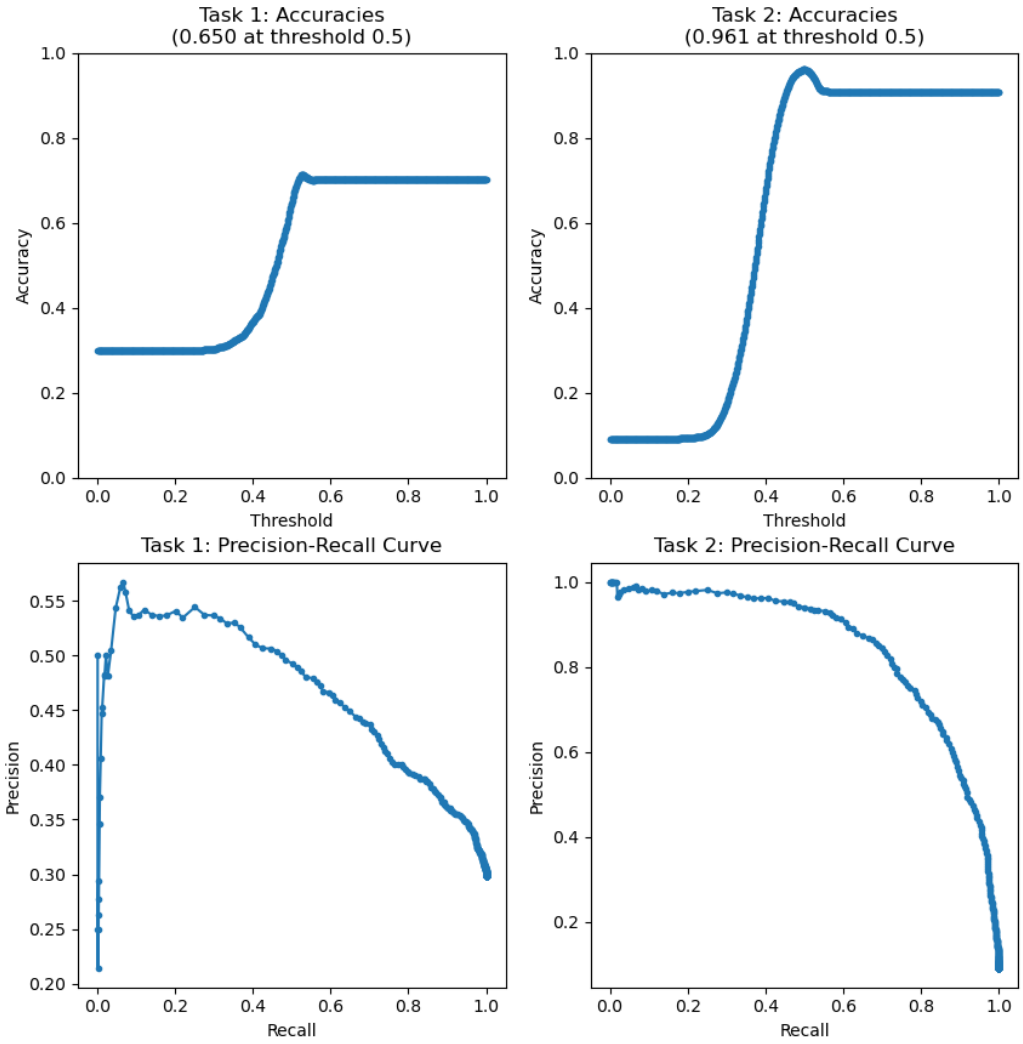


Figure 4.13: Performance of the final contrastive Siamese similarity detector trained on randomized unaugmented images evaluated on dataset C using multi-view detection

	MPAAE	Contrastive siamese	Triplet siamese
Same BG, w/ pose	**0.970**/0.849/0.821	0.931/0.662/0.487	0.925/0.724/0.291
Same BG, pert. pose	**0.965**/0.819/0.770	0.928/0.666/0.410	0.924/0.727/0.261
Same BG, w/o pose	**0.922**/0.681/0.258	0.910/0.662/0.011	0.909/ N/A /0.000
Diff. BG, w/ pose	**0.968**/0.825/0.824	0.909/ N/A /0.000	0.909/ N/A /0.000
Diff. BG, pert. pose	**0.962**/0.808/0.763	0.909/ N/A /0.000	0.909/ N/A /0.000
Diff. BG, w/o pose	**0.920**/0.643/0.265	0.909/ N/A /0.000	0.909/ N/A /0.000
Same BG, w/ pose	**0.989**/0.926/0.956	0.962/0.851/0.706	0.941/0.909/0.394
Same BG, pert. pose	**0.988**/0.914/0.958	0.959/0.841/0.679	0.943/0.829/0.465
Same BG, w/o pose	**0.939**/0.672/0.643	0.911/0.727/0.033	0.909/ N/A /0.000
Diff. BG, w/ pose	**0.989**/0.953/0.929	0.909/ N/A /0.000	0.909/ N/A /0.000
Diff. BG, pert. pose	**0.986**/0.908/0.946	0.909/ N/A /0.000	0.909/ N/A /0.000
Diff. BG, w/o pose	**0.934**/0.661/0.553	0.909/ N/A /0.000	0.909/ N/A /0.000

Table 4.1: Accuracy/precision/recall of different detectors on **Task 2** using Algorithm 2 for evaluation and **Algorithm 3** for decision; top: single-view similarity detection, bottom: multi-view similarity detection; best models highlighted in **bold**, failing models (recall = 0) highlighted in red.

	MPAAE	Contrastive siamese	Triplet siamese
Same BG, w/ pose	0.543/0.531/0.375	0.702/ N/A /0.000	0.702/ N/A /0.000
Same BG, pert. pose	0.487/0.593/0.402	0.702/ N/A /0.000	0.702/ N/A /0.000
Same BG, w/o pose	0.659/0.608/0.366	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, w/ pose	0.522/0.541/0.379	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, pert. pose	0.487/0.601/0.407	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, w/o pose	0.623/0.617/0.374	0.702/ N/A /0.000	0.702/ N/A /0.000
Same BG, w/ pose	0.527/0.383/0.956	0.702/ N/A /0.000	0.702/ N/A /0.000
Same BG, pert. pose	0.636/0.445/0.895	0.702/ N/A /0.000	0.702/ N/A /0.000
Same BG, w/o pose	0.461/0.348/0.927	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, w/ pose	0.537/0.385/0.931	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, pert. pose	0.642/0.448/0.868	0.702/ N/A /0.000	0.702/ N/A /0.000
Diff. BG, w/o pose	0.478/0.354/0.911	0.702/ N/A /0.000	0.702/ N/A /0.000

Table 4.2: Accuracy/precision/recall of different detectors on **Task 1** using Algorithm 1 for evaluation and **Algorithm 3** for decision; top: single-view similarity detection, bottom: multi-view similarity detection. Failing models (recall = 0) are highlighted in red.

	MPAAE	Contrastive siamese	Triplet siamese
Same BG, w/ pose	**0.818**/0.695/0.695	0.783/0.635/0.635	0.771/0.616/0.616
Same BG, pert. pose	**0.758**/0.594/0.594	0.753/0.586/0.586	0.746/0.574/0.574
Same BG, w/o pose	**0.714**/0.519/0.519	0.710/0.514/0.514	0.694/0.487/0.487
Diff. BG, w/ pose	**0.816**/0.691/0.691	0.574/0.285/0.285	0.595/0.321/0.321
Diff. BG, pert. pose	**0.754**/0.588/0.588	0.580/0.295/0.295	0.595/0.320/0.320
Diff. BG, w/o pose	**0.703**/0.501/0.501	0.587/0.307/0.307	0.597/0.324/0.324
Same BG, w/ pose	**0.895**/0.823/0.823	0.848/0.744/0.744	0.829/0.713/0.713
Same BG, pert. pose	**0.844**/0.738/0.738	0.815/0.690/0.690	0.816/0.691/0.691
Same BG, w/o pose	**0.812**/0.685/0.685	0.730/0.547/0.547	0.735/0.555/0.555
Diff. BG, w/ pose	**0.883**/0.804/0.804	0.562/0.265/0.265	0.593/0.316/0.316
Diff. BG, pert. pose	**0.838**/0.728/0.728	0.571/0.279/0.279	0.591/0.314/0.314
Diff. BG, w/o pose	**0.801**/0.666/0.666	0.591/0.314/0.314	0.600/0.328/0.328

Table 4.3: Accuracy/precision/recall of different detectors on **Task 1** using Algorithm 1 for evaluation and **Algorithm 4** for decision; top: single-view similarity detection, bottom: multi-view similarity detection; best models highlighted in **bold**.

Chapter 5

Discussion

The results of Sections 4.1.1 and 4.1.2 show that the following modifications lead to an increase in the performance of the similarity detector based on the MPAAE architecture:

- increasing the number of decoders
- decreasing the number of 3D models represented in the training set.

From the perspective of decoders, both actions lead to a decrease in geometric diversity of the data reconstructed by each individual decoder.

For instance, in case of a single decoder for all stages of the same block, the models reconstructed by the decoder mostly share a common general shape and only differ in smaller details. In this case the decoder can quickly learn to reconstruct the common geometric features among the models such as corners and outer edges and must then give more weight to smaller details in order to further improve the reconstruction quality.

Such "attention to detail" of the decoder might propagate back to the encoder and as a result force it to generate more descriptive latent representations which would result in better similarity detection performance also on previously unseen models.

This is illustrated by the reconstructions of unseen objects generated by the single-decoder autoencoder (Figure 5.1a) and one branch of the 4-decoder-autoencoder (Figure 5.1b) previously trained and evaluated in Section 4.1.1.

The single-decoder-autoencoder reconstructions in Figure 5.1a reflect the general shape of the models. The outlines of the objects in the reconstructions

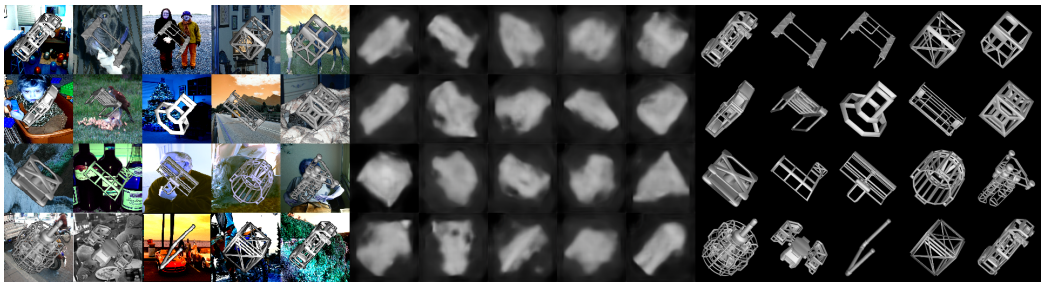
match the outlines in the rendered images with reasonable accuracy, but the inner structure and small details are lost.

In the reconstructions generated by the 4-decoder-autoencoder in Figure 5.1b, on the other hand, the outlines are not preserved and have a stronger resemblance to the outlines of the block `BasicAssembler` which this decoder was trained on. However, the inner structure of each reconstruction bears a certain resemblance to the inner structure of the corresponding rendered objects. It does not accurately reflect the actual shape of the object in most cases but rather it reflects its porosity and the number of smaller appearance features within its boundaries which means that those features are encoded in the latent representations.

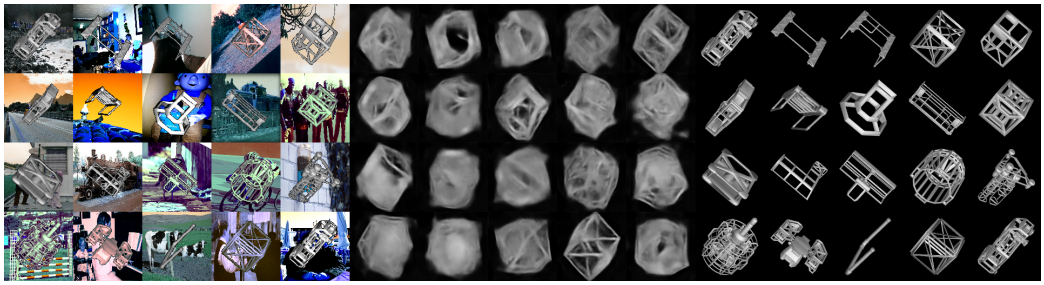
This might explain better performance of the 4-decoder-MPAAE compared to the 1-decoder-MPAAE whose encoder does not capture such features (for instance, compare the reconstructions of the two rightmost objects in the top row by each of the autoencoders). When trained with multiple decoders, the encoder receives information about the relevant features from multiple sources with each source being better at extracting and providing this information due to increased geometric similarity of the reconstructed data.

The results of Section 4.1.1 suggest that this effect can be further amplified by increasing the number of decoders while simultaneously decreasing the number of 3D models represented in the input data of each decoder.

This indicates that multi-path autoencoder architectures might not only be useful for extracting pose features shared between different objects, as Sundermeyer et al. show in their original paper [12], but also for extracting shared appearance features in order to generate more descriptive latent representations. This suggests that the usage of multi-path autoencoders in image classification, object tracking and re-identification and other computer vision tasks which rely on descriptive latent representations is a promising direction of future research.



(a) Reconstructions by the single-decoder MPAAE



(b) Reconstructions by one of the branches of the 4-decoder MPAAE

Figure 5.1: Reconstructions of unseen objects generated by different MPAAE models evaluated in Section 4.1.1 (left part: input images, middle part: reconstructions, right part: ground-truth rendered images).

Chapter 6

Conclusion

In this thesis we have implemented and evaluated multiple solutions to the block similarity detection task posed by GoodAI and achieved the top accuracy of 89.5% on the task of distinguishing between different completeness stages of the same block and 98.9% on the task of distinguishing between different blocks.

We have demonstrated the applicability of the novel Multi-Path Augmented Autoencoder architecture [12] to a new task of image similarity detection and described the benefits of the multi-path architecture for the descriptiveness of latent representations with respect to the appearance features of the objects in the images.

We have demonstrated that the augmented auto-encoder architecture can be trained to bridge the gap between training and application domains using data augmentation techniques, while Siamese architectures benefit more from training purely on images from the synthetic data domain.

We have demonstrated the key role of data augmentation techniques, in particular brightness/contrast variations and partial texturing, in the generation of descriptive domain-invariant latent representations using the Multi-Path Augmented AutoEncoder architecture and the negative influence of excessive training dataset diversity on the representation quality.

Bibliography

[1] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. In Gang Hua and Hervé Jégou, editors, *Computer Vision – ECCV 2016 Workshops*, pages 850–865, Cham, 2016. Springer International Publishing.

[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, 2019.

[3] Dahjung Chung, Khalid Tahboub, and Edward J. Delp. A two stream siamese convolutional neural network for person re-identification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[4] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[6] Mohsen Heidari and Kazim Fouladi-Ghaleh. Using siamese networks with transfer learning for face recognition on small-samples datasets. In

2020 International Conference on Machine Vision and Image Processing (MVIP), pages 1–4, 2020.

[7] Philip T. Jackson, Amir Atapour-Abarghouei, Stephen Bonner, Toby Breckon, and Boguslaw Obara. Style augmentation: Data augmentation via style randomization, 2019.

[8] Yan Ma, Kang Liu, Zhibin Guan, Xinkai Xu, Xu Qian, and Hong Bao. Background augmentation generative adversarial networks (bagans): Effective data generation based on gan-augmented 3d synthesizing. *Symmetry*, 10(12), 2018.

[9] Henrique Morimitsu. Multiple context features in siamese networks for visual object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

[10] Kate Saenko, Brian Kulis, Mario Fritz, and Trevor Darrell. Adapting visual category models to new domains. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 213–226, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[11] Lingxue Song, Dihong Gong, Zhifeng Li, Changsong Liu, and Wei Liu. Occlusion robust face recognition based on mask learning with pairwise differential siamese network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[12] Martin Sundermeyer, Maximilian Durner, En Yen Puang, Zoltan-Csaba Marton, Narunas Vaskevicius, Kai O. Arras, and Rudolph Triebel. Multi-path learning for object pose estimation across domains, 2020.

[13] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3d orientation learning for 6d object detection from rgb images, 2019.

[14] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns, 2017.

[15] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[16] Meng Zheng, Srikrishna Karanam, Ziyan Wu, and Richard J. Radke. Re-identification with consistent attentive siamese networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.