Annotation Invariant Style Transfer

Ben Jourdan

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2020

Abstract

Many cutting edge machine learning models for shape and pose reconstruction [41][19][37] have been trained on synthetic datasets derived from repositories of 3D models such as ShapeNet [6]. These synthetic datasets are usually generated with a simple rendering pipeline which do not produce realistic images. This project defines what I call Annotation Invariant Style Transfer (AIST) which is a process that maps synthetic renders of objects to more realistic images while preserving the underlying shape and pose annotations.

Instead of designing a better model the aim of this project was to create better datasets that improve the performance of existing models on both synthetic and real datasets. AIST is applied to the dataset used by a state of the art shape and pose reconstruction model based on point clouds and differentiable rendering [19]. I show AIST improves median pose error performance by over 30% on synthetic datasets and performs better on a real dataset too[1].

[1]No ground truth is available for real datasets so a rough metric based on the XOR of binary segmentation masks is used.

Acknowledgements

I would like to thank my supervisor Dr. Hakan Bilen for supporting me through this project and introducing me to the fascinating field of shape and pose estimation.

I would also like to thank Professor Amos Storkey for generously providing advisory guidance via video link during the COVID-19 pandemic.

Table of Contents

Chapter 1

Introduction

Humans are capable of performing shape and pose estimation with immense speed and precision. Squash players can predict the position of a ball moving at over 250 mph, drone racing pilots can navigate complex 3D environments at great speed, and my high school maths teacher could juggle 9 bean bags, deforming all the time, at once. A large goal of computer vision research is to develop systems capable of similar feats.

Convolutional neural networks have recently been used to develop models which can perform shape and pose estimation well on synthetic datasets [19][37]. There is however, a common shortfall. Making the jump to real datasets is difficult as supervised methods require shape and pose annotations[1] for training. In many cases it is simply too costly to get hold of these annotations for real objects due to the time and expensive equipment required. Many unsupervised models also need multiple views (images from different directions) of the same object to train. This is fine if digital objects are used but is a very restrictive requirement for real objects.

1.1 Research Question

The motivation for this project is the need to find better ways to deal with this *reality gap* [36]; the discrepancy between what we can simulate digitally and what actually happens in the real world.

In this project my research question was "Is it possible to use existing real and synthetic datasets, image segmentation techniques and an unpaired generative style transfer model to produce realistic images which can then be used to improve shape and pose estimation of existing models on both real and synthetic datasets?". I hypothesized that this is possible.

My hypothesis is based upon the fact that synthetic datasets are often very bland. They lack real world information such as complex material and lighting information which unpaired generative models have been shown able to transfer from one domain to another (horse to zebra, orange to apples etc.)[43]. With suitable modifications to existing unpaired generative models I believe that this real world information can be transferred from a sufficiently preprocessed real dataset to a synthetic one while the shape and pose annotations of the synthetic dataset are preserved. The resulting dataset should let existing pose estimation models perform better on real and synthetic datasets as the training material will be richer.

[1] These annotations vary from one formulation to another. Shape is often represented by 3D voxels, a 3D mesh or a cloud of 3D points sampled from the surface of a mesh. Pose, the orientation of the camera with respect to the target object, is often represented as a quaternion or by elevation and azimuth.

1.2 Project Pipeline

In order to answer my research question I developed the following pipeline.

1. A real dataset of images was processed to extract segmentations of just the target object. This target object is what the final model will be trained to estimate shape and pose of.
2. These segmentations were then normalized and filtered to aid style transfer.
3. A style transfer model was then trained on a synthetic dataset of images of the target object and the normalized, filtered segmentations.
4. The entire synthetic dataset was then stylized with the style transfer model to create a more realistic dataset.
5. Finally a shape and pose model was trained on this more realistic dataset and evaluated to determine the efficacy of AIST by comparing the performance of this shape and pose model to one which had be trained on only the synthetic dataset.

1.2.1 Dataset selection

In this project I used the dataset derived from ShapeNet's cars model repository, from paper [19] as my synthetic dataset and I used the Stanford car dataset [26] as the large dataset of pictures of real objects.

Figure 1.1: A typical render of a car from the ShapeNet database from the dataset used by [19]. Note the lack of detail and low resolution (128 by 128 pixels).

Figure 1.2: A typical segmented image from the Stanford Car dataset. Note the reflections and specular highlights.

Cars were chosen as the target object because real and synthetic images of cars are readily available. Also, shape and pose reconstruction models such as the one I decided to use had

been evaluated on synthetic images of cars already and so could be used as a baseline for models trained on the generated, more realistic dataset. The Stanford Car dataset was chosen largely for its size. I performed extensive preprocessing on this dataset to extract, filter and clean segmentations before using it to train anything so aside from its size, its original characteristics were not so important.

1.2.2 Image segmentation model selection

To perform segmentation on the Stanford car dataset an implementation [3] of Mask R-CNN [15], pretrained on the MSCOCO dataset, was used. It was chosen because the pretrained model had been taught to segment images of cars already as well as being state of the art. I used this implementation as a black box and made no changes to it whatsoever.

1.2.3 Style transfer model starting point

The style transfer model I used is the core of this project. The rest is simply manipulating datasets and using an existing shape and pose model to evaluate performance. I decided upon using an existing implementation of CycleGAN [44][42] as the starting point for my style transfer model. This is because models like CycleGAN have many attractive properties. The most import of which is the fact they are capable of "unpaired image to image style transfer" [43]. This means a style transfer model can be trained even when there is not a 1 to 1 mapping between the source and target domains. Previous works [17][20][23] have used paired datasets for image to image translation tasks. It would have been impossible to use these methods for my task as there is no 1 to 1 relationship between synthetic and real datasets.

1.2.4 Pose estimation model selection

I used the model from the "Differentiable point clouds" paper [19] as the shape and pose model. I refer to it as simply the shape and pose model unless mentioned otherwise from now on. Aside from its strong performance, I chose this model because it could be trained using only multiple views of objects as opposed to other, more supervised, methods that require the actual shape and pose ground truth annotations for training. I use the implementation given in the paper and make no changes whatsoever - again, treating it like a black box.

1.2.5 My contributions

My main contributions are:

- Modification made to the CycleGAN loss function to encourage annotation invariant style transfer.
- Normalization of image segmentations based on the ratio of foreground (car segmentation) to background areas to aid annotation invariant style transfer.

1.3 Report Structure

The chapters following the Background and Related Work section cover the various stages of the project pipeline. The Experiments chapter then covers implementation, optimization details and results.

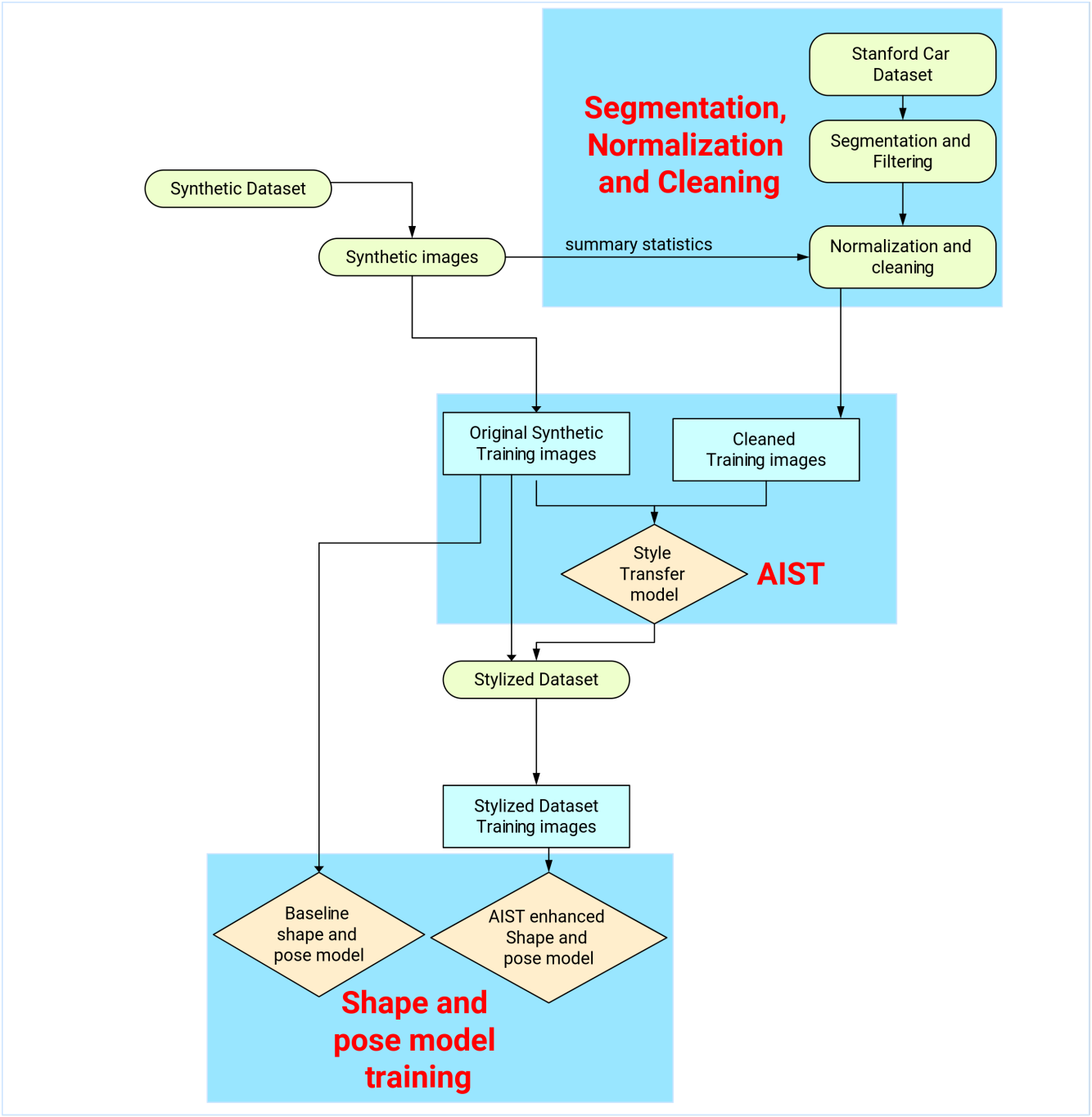


Figure 1.3: A visual description of my project's pipeline. The areas highlighted in blue are covered in the following chapters.

Chapter 2

Background and Related work

My pipeline uses a number of existing models such as MaskRCNN [15] for image segmentation, CycleGAN [43], which I modify (using a ResNet [16] among other things) for style transfer, and an unsupervised shape and pose estimation model [19] to evaluate my pipeline's efficacy. All of these models are built upon Convolutional Neural Networks (CNNs). For those unfamiliar, I have included a whistle-stop tour through the essentials in Appendix B.

As my pipeline uses and builds upon existing models for style transfer, shape and pose estimation, and image segmentation, there are many related works. Below I go over the models I used in this project, some related works as well as a previous approach to tacking the "reality gap".

2.1 Domain Randomization: Addressing the reality gap

Instead of pursuing realism directly to improve performance in the real world, Tobin et al. [36] applied non-realistic random textures to objects in a simulator which were used to train an object detector (a modified version of VGG-16 [35]). The hypothesis was that if the model can cope with sufficient randomness during training then it should be able to cope better in the real world. Indeed when tested in the real world it was found to be accurate to 1.5cm and performed significantly better than when no randomness was applied to the training data. It was

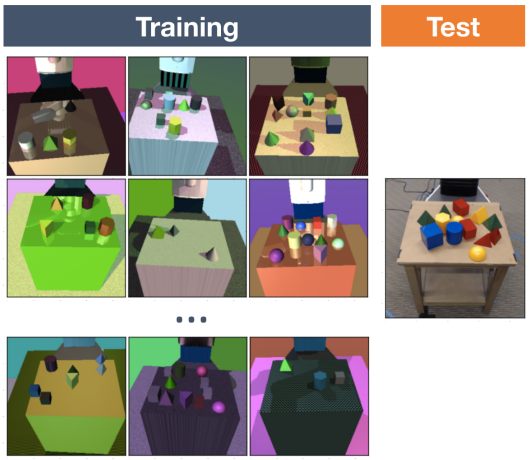


Figure 2.1: Domain Randomization: randomized training and real world testing examples. Taken from [36].

impressive that such a simple technique could improve performance on real images. However I believe that simply applying blind randomization is not optimal. Though the detector may have

become more robust, the goal should be to shift the distribution of the source domain to the target domain to avoid the model learning unnecessary or even unhelpful information.

2.2 Mask R-CNN and Related Work

Mask R-CNN is complicated. It sits at the end of a series of papers that started in 2014 with the paper that created the R-CNN model (Regions with CNN features) [13]. To get a sense of what is going on inside Mask R-CNN it is important to know how R-CNN evolved. At each stage of the evolution the models had two distinct stages. The first stage predicted "regions of interest" that the second stage would go on to further analyse, predicting labels, bounding boxes and in Mask R-CNN's case, binary masks too.

2.2.0.1 R-CNN

Girshick et al. introduced R-CNN [13] to address the question "To what extent do the CNN classification results on ImageNet generalize to object detection?" Unlike Mask R-CNN, the output R-CNN was just class labels and bounding boxes. No masks were predicted.

R-CNN generated region proposals (rough bounding boxes) using an algorithm called selective search [39]. This algorithm is a form of agglomerative clustering, combining smaller regions together by similarity using properties such as texture, size, colour and position. After generating

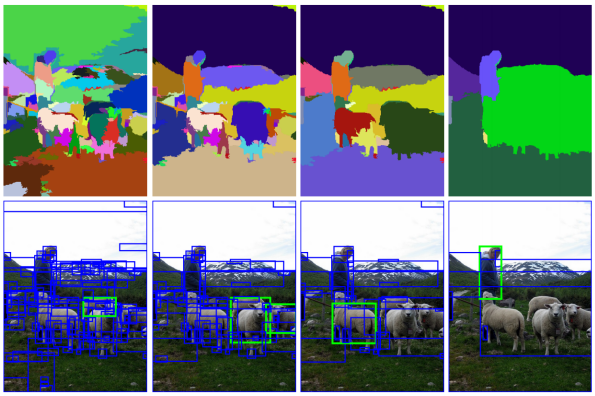


Figure 2.2: Selective search in action. Taken from [39]

region proposals, these rectangular regions were warped to become square (227×227 pixels). Each warped region was then passed through a 5 layer CNN to produce feature maps for each region. An SVM, one for each class, would then score these feature maps. A greedy algorithm called "non-maximum suppression" would then reject lower scoring regions that overlapped higher scoring regions of the same class. After non-max suppression, a regression model predicts the final bounding box for each region selective search proposed, on a per class basis. At the time of release, the RCNN paper improved mean average precision (mAP) by over 30% compared to the previous best result on VOC2012[11], one of the benchmark datasets for image detection at the time.

2.2.0.2 Fast R-CNN

R-CNN was slow. Each proposed region had to be passed through the CNN one at a time. It also had to train a SVM and a bounding box regressor for each class as well as having to train the CNN, SVMs and bounding box regressors separately.

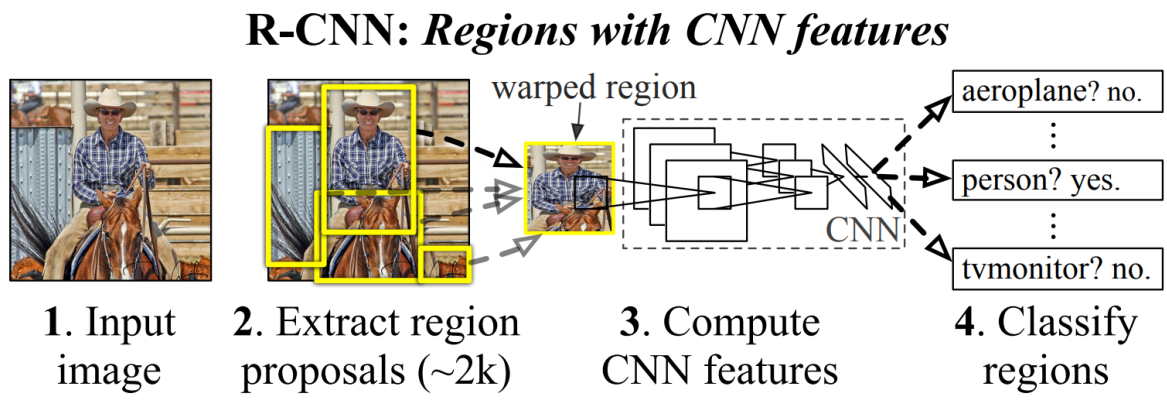


Figure 2.3: RCNN pipeline. Taken from [13]

One of the key insights the Fast R-CNN paper [12] made was that an image could be passed through the CNN all at once and the positions of the region proposals could be used to work out the corresponding regions (called regions of Interest (RoI)) in the CNNs output (called the feature map). These (rectangular) RoI could then be used to perform classification and bounding box regression.

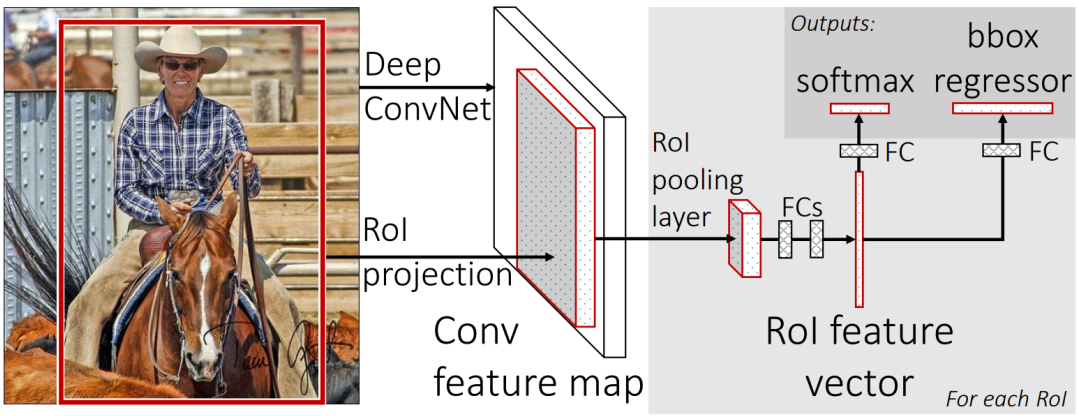


Figure 2.4: Fast R-CNN pipeline taken from [12]. ”FCs” refer to fully connected layers

Region of Interest pooling was then applied to each RoI to produce smaller, square, fixed size, feature maps which were then be passed to classifier and regressor networks. Region of interest pooling works by dividing a rectangular $h \times w$ (h, w variable) RoI into an $H \times W$ (H, W fixed, almost always square) grid of sub-regions. These sub-regions may not be the same size as their boundaries had to line up with the dimensions of the Feature map. Max pooling was then applied, channel independently, to each sub-region, giving a fixed $H \times W$ grid of features. Note the disappearance of the multiple SVMs and bounding box regressors in Figure 2.4 . Fast R-CNN was able to combine the classification and regression tasks as well as the Initial CNN into one network making it easier to train.

Fast R-CNN was a large improvement on R-CNN. It dramatically decreased the time taken to train it as well as the execution time during testing. It managed this while also achieving a better mAP on VOC2012 than R-CNN.

2.2.0.3 Faster R-CNN

There was one remaining flaw with Fast R-CNN and that was region proposal. The solution used by Fast R-CNN and R-CNN, selective search, was painfully slow. After the improvements

made by the Fast R-CNN paper, region proposal became the bottleneck in terms of speed. The Faster R-CNN paper[32] introduced a Region proposal network (RPNs) that dramatically sped up region proposal.

The authors realised that the CNN being used to produce the feature map was extracting similar features to those that were used for region proposal. They therefore devised a method to harness the feature map for region proposal.

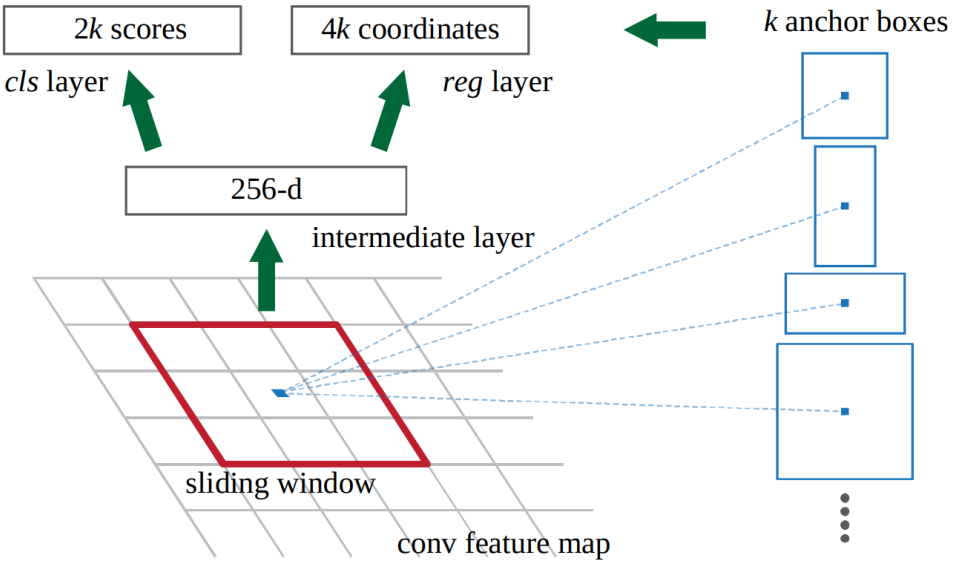


Figure 2.5: An RPN network being slid across a CNN feature map. Note the 3x3 window size. Anchor boxes of various sizes and aspect ratios are visualized on the right. Taken from [32].

They slide a small neural network across the feature map using a 3×3 window. At each position in the feature map the network predicts scores and bounding box positions for k reference boxes of various sizes and aspect ratios, called anchors. This is implemented very efficiently using convolutions. After non-max suppression, the remaining RoIs and their predicted bounding boxes are passed to Fast R-CNN.

Faster R-CNN is capable of running at five frames per second. Considering that the original selective search algorithm alone could take seconds to run this was a phenomenal feat.

2.2.0.4 Mask R-CNN

Mask R-CNN [15] is a simple extension of Faster R-CNN. It adds an extra branch to predict binary segmentation masks along with the existing classification and bounding box prediction branches.

The authors had to modify the RoI pooling stage as they found it did not faithfully preserve pixel level alignments between the RoI and the corresponding regions in the original image. While not as important for class and bounding box prediction this was detrimental to binary mask prediction. They addressed this with RoI alignment, a similar process to RoI pooling that uses bilinear interpolation to avoid the misalignments RoI pooling would otherwise introduce.

Mask-CNN is fast and powerful. It beat all existing single-model entries in every track of the COCO 2016 challenge[27]. Without it I doubt my project would have been successful.

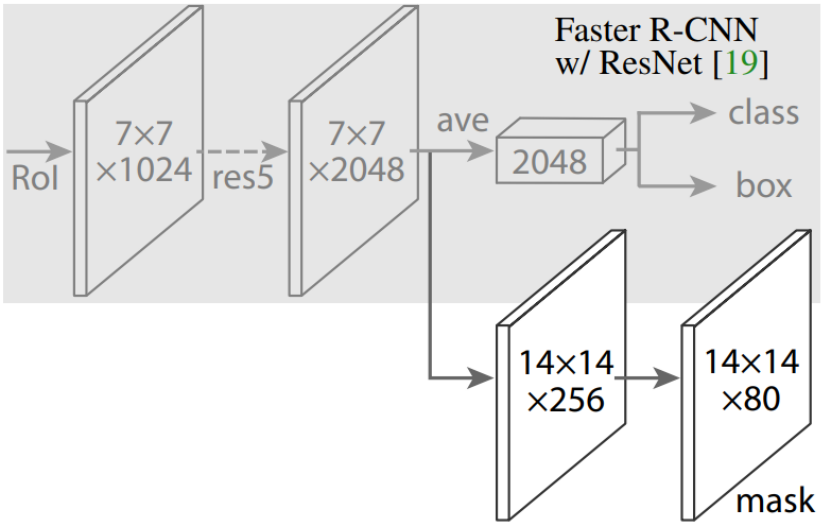


Figure 2.6: showing the extra branch added after RoI pooling. Taken from [15]. Note that the $\times 80$ in the output of the mask branch is due to the fact there were 80 possible classes in the MSCOCO dataset.

2.3 CycleGAN and Related Work

GANs or Generative Adversarial Networks were introduced in 2014 [14]. The paper introduced the idea that two networks could be tasked with competing against each other. GANs addressed the problem of creating fake samples from a target domain, usually images. One network, known as a generator, would try to turn random noise into a convincing fake image and the other, known as a discriminator, would try to tell the real images from the fake ones. A loss function based on the performance of both networks (known as the GAN losses) could then be used to train both networks at the same time. The result would be that the generator becomes good at generating images from a target domain and the discriminator becomes good at detecting fakes. Under particular conditions the loss functions for the discriminator and generator converges during training but in practice this happens infrequently.

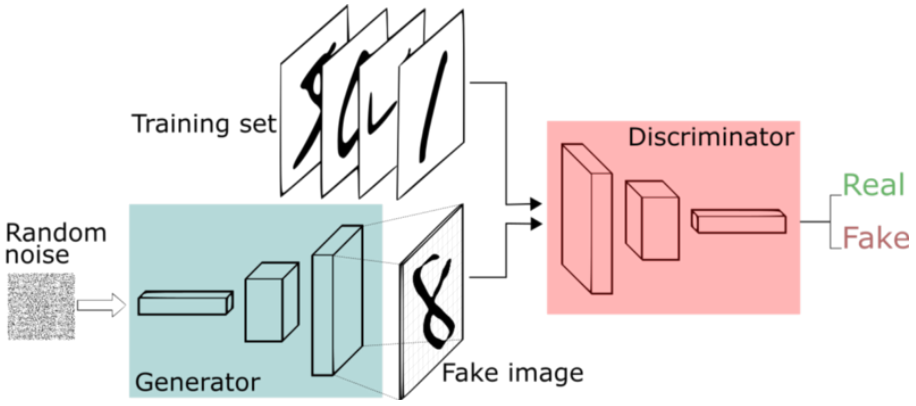


Figure 2.7: Generative Adversarial networks. Taken from [14]

2.3.0.1 CycleGAN

CycleGAN [43] addresses image to image translation in the unpaired setting. This is when the goal is to translate images from one domain to another where there is not a one to one mapping between domains. Instead of generating images from random noise as the original GAN paper

did, CycleGAN generates an image from one domain by starting with an image in the other domain. It does this using "forwards" and "backwards" CNNs as generators.

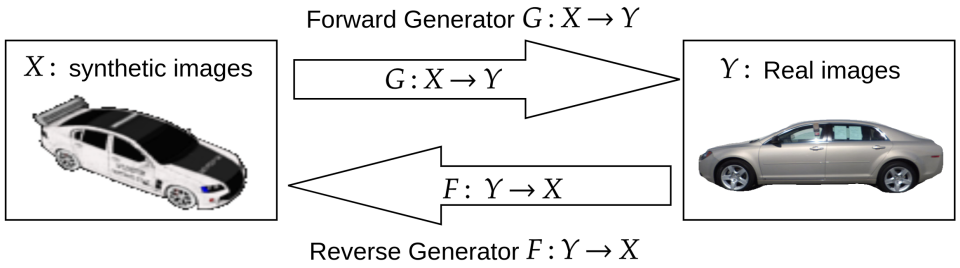


Figure 2.8: The generators of a CycleGAN

In a CycleGAN there are two discriminators, responsible for discriminating between fake and real elements of either domain. The network architecture of either generator is identical, consisting of 3 convolutional layers followed by 9 residual convolutional blocks where each residual block consists of a convolution, a ReLU, and another convolution (the skip connections jump two convolutions). Following the residual layers are two transposed convolutions then a final convolution. This down-scaling followed by up-scaling type of architecture is known as an *encoder decoder network* and is very common. The discriminator networks, again identical, consists of a set of convolutional layers that output a 1x30x30 tensor where each entry corresponds to the classification of a 70x70 area of the original image [21]. The cost functions used by the entire system encourages the generators to produce images which match the target domain well through the GAN loss. It also encourages behaviour such that images generated by one generator can be easily mapped back to the starting domain by the other generator through what is known as the *cycle consistency* loss which gives CycleGAN its name. As I modify the CycleGAN cost functions for this project I leave a more detailed explanation for the chapter on Style Transfer.

CycleGAN achieved impressive results when it was first released. One notable improvement from a more recent paper was the introduction of the Wasserstein cost functions [4]. The cost functions of the original GAN paper sought to minimize the KL divergence between the target domain and the generated images. If the distributions do not overlap at all the KL divergence is constant and so the gradient vanishes whereas the gradients of the Wasserstein cost functions will not vanish. In this project I use a least squares version of the GAN losses [28]. This was shown to perform better than the original at the time and was implemented in the code I modified [42].

There are other papers on unpaired image to image translation that have improved upon the CycleGAN paper. CrossNet [33] introduced "cross-translators". These were used to perform *latent cross-consistency*, a type of regularization on the latent representations of images after being passed through the encoders of each Generator.

I chose to use CycleGAN as my starting point for style transfer above newer, better unpaired image to image translation models because CycleGAN has a very good implementation [42]. I had never written a large model in PyTorch or Tensorflow before so I thought it would be prudent to use an existing implementation instead of spending a massive amount of time writing a slightly better version from scratch. The implementation is also widely used. As of 13th April 2020 it has been forked on GitHub over 3.4 thousand times.

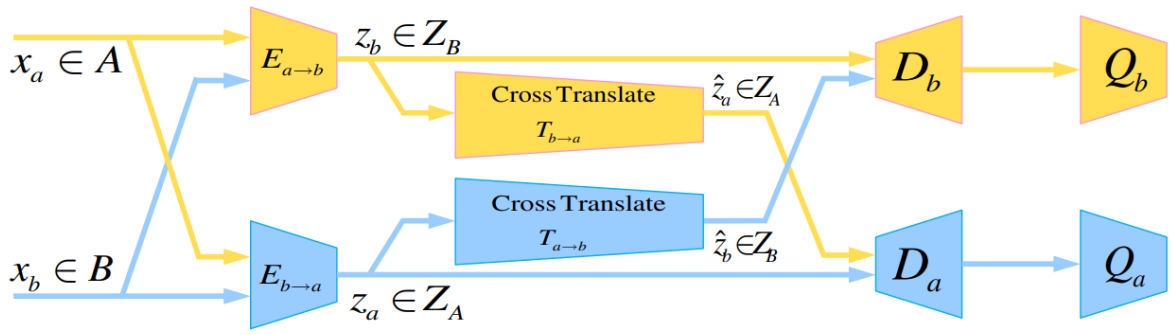


Figure 2.9: CrossNet: The latent spaces of the two encoder-decoder generators are coupled via *cross translators*. The $E_{a\to b}$ and $E_{b\to a}$ networks are the encoder part of the Encoder Decoder networks for the forward and backward Generators respectively. The D networks are the decoder parts and the Q networks are the discriminators. Taken from [33].

2.4 Differentiable Point Clouds and Related Work

2.4.0.1 Differentiable point clouds

Insafutdinov and Dosovitskiy [19] defined an unsupervised model that is capable of predicting the shape and pose of objects from unseen single views. It is unsupervised because it only uses multiple views of different objects to train, not the underlying annotations. It predicts shape by using convolutional layers followed by an MLP to predict a point cloud, represented internally as a vector $3N$ long where N is the number of points. For pose prediction, an ensemble of pose regressors are used. A different MLP branches off from the last layer of the convolutional layers. This further splits into distinct convolution networks. One for each pose regressor. The ensemble of pose regressors is distilled into a single regressor called the student which is trained using a loss based on the angular difference between the student regressor's prediction and the best performing pose regressor in the ensemble.

Given two images of the same object, the point cloud P is predicted using the first image (the rest of the network is ignored). The second image is then used to predict k possible poses, one for each of the pose regressors. The point cloud is then rendered using a differentiable renderer from each of the k possible poses. The loss is then the minimum $L2$ norm between the k renders from each pose and the ground truth of the second image. At test time both a point cloud and pose is predicted.

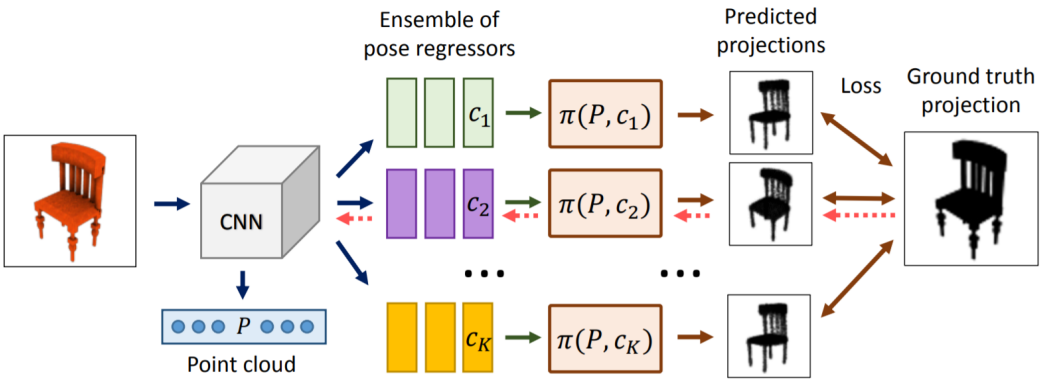


Figure 2.10: Training of the model from [19]. c_i is the ith predicted pose and P is the point cloud predicted using the first image.

2.4.0.2 Differentiable rendering

Insafutdinov and Dosovitskiy [19] use a differentiable renderer π to render point clouds in such a way that gradients can be back propagated through it. These are hard to design as rendering is intuitively a non-differentiable operation due to occlusion. There are many variants; the one used by this paper represents each point in the point cloud as a gaussian density with the same parameters across all points. These points are projected to the image plane with occlusion reasoning handled in a differentiable manner.

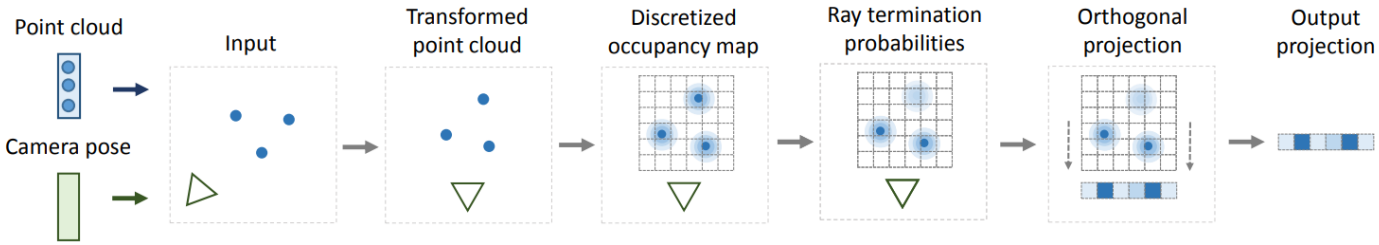


Figure 2.11: Differentiable rendering from 2D to 1D. taken from [19]

Figure 2.11 illustrates the differentiable rendering processes used in the point clouds paper. The occupancy map is created by placing spherical gaussians at each point. Ray termination probabilities are calculated by multiplying the occupancy o_k at position k along a ray by the product $\prod_{u=1}^{k-1}(1-o_u)$. This means ray termination probabilities are high at position k if o_k is high and all previous occupancy's are low. To project the points to the camera plane, the value of each pixel is the sum of the ray termination probabilities times some signal at each point along the ray corresponding to that pixel:

$$\text{pixel value} = \sum_{k\text{th position along corresponding ray}} r_k y_k$$

where r_k is the ray termination probability at position k along the ray and y_k is the value of some signal at position k along the ray. To obtain a depth map $y_k = k/D$ where D is the maximum number of positions along the ray. To obtain a silhouette $y_k = 1 - \delta_{k,D}$.

2.4.0.3 Related work

Along with the differentiable point clouds paper, there are other supervised [18] and unsupervised [38] shape and pose estimation models. Like the differentiable point clouds paper many use reprojection error as the training signal; that is the discrepancy between 2D projections of a predicted 3D object and ground truth projections. Some recent work uses voxels instead of Point clouds to describe shape [38] whereas other have started using meshes [18]. Point cloud and mesh methods have been developed because voxel methods are very inefficient, requiring 3D convolutions and the number of voxels required scales cubically with resolution.

2.5 Datasets

I discuss why I chose these particular datasets in the Introduction.

2.5.1 The synthetic dataset

The dataset of synthetic images and annotations used in this project was taken from the Differentiable point clouds paper [19]. It was generated by rendering 5 views of 7,497 cars from the

ShapeNet repository [6] which had be split into train (5247 cars), test (1500 cars) and validation (750 cars) sets. The accompanying ground truth consisted of a point cloud for each model where points were sampled from the faces of each digital object along with the camera pose for each view of each object. It should be noted that some of these models are not remotely similar to everyday cars, Figure 2.12 for example. A random number of lights with random positions were generated for each render. The camera azimuth and elevation was uniformly sampled from $[0, 360)$ and $[-20, 40]$ (degrees) respectively. This is one of the more significant differences between the synthetic and real datasets; almost no images of real cars are viewed from an elevation of less than zero. This dataset can be downloaded from the point clouds paper's github page[1]. I had hoped to create a similar dataset starting from the original ShapeNet models, sampling camera elevations from a more positive range. Unfortunately neither author has been able to describe how to recreate the dataset exactly from scratch on request. It seems they did not write the rendering code themselves but instead used code from other projects. Updates to the software the rendering code ran on, Blender [8], has changed the behaviour of the rendering scripts. It took many months for the authors to confirm this with me. I assumed they could recreate their own dataset from scratch for most of the first semester. Appendix A details the work I did under this assumption which could not be used.

Figure 2.12: A render (alpha channel coloured black) from the synthetic dataset. The "car" appears to have a machine gun on the roof, snowmobile tracks for wheels and some sort of anti aircraft gun on the back. Not very typical of a real car!

[1] https://github.com/eldar/differentiable-point-clouds

2.5.2 The real dataset

The datasets of real images I chose was the Stanford car dataset [26]. This consists of 16185 images of 196 classes of cars. I resplit the data from a 50/50 test train split to 80/20. I discarded the provided annotations as they only consisted of information such as the car make and bounding box; I needed per pixel segmentation annotations. In its raw form this dataset was not particularly useful. I had to extract the segmentation annotations using MaskRCNN as well as performing a number of preprocessing operations before I could use it successfully for AIST.

Figure 2.13: A typical image from the Stanford Car dataset.

Chapter 3

Segmentation, Normalization and Cleaning

A stock implementation (parameter configuration given in Appendix C) of Mask R-CNN [3] was used for segmentation of the Stanford car dataset. I used pretrained weights obtained by training the model on the MSCOCO dataset [27]. The implementation includes methods to download the pretrained weights easily. The MSCOCO consisted of 80 classes of common objects such as *bicycle, car, airplane, train, truck, person* etc. An extra class was included to detect background. The official training and test sets of the Stanford dataset were merged. A 80/20 test train split was performed after segmentation and Normalization. This was because more than one good segmentation could be recovered from each image and some segmentations were not of high enough quality and so had to be rejected. The approach for filtering segmentations is now described.

Figure 3.1: Original image

Figure 3.2: Extracted segmentations

Figure 3.3: Stock Mask R-CNN segmentation of the Stanford Car dataset

3.1 Filtering the Segmentations

As can be seen from Figure 3.3, Mask R-CNN is able to obtain segmentations for more than one car per image. To reject low quality segmentations and segmentations of the wrong class, segmentations were rejected if any of the following were true:

1. The predicted label was not in $\{\text{car}, \text{truck}\}$.
2. The confidence of the model in the predicted label was less than 0.7.

3. The square root of the area of the bounding box of the segmentation was less than 128.
4. The fraction of the area of segmentation to the area of its *predicted bounding box* was less than 0.5 (not the ground truth bounding boxes provided by the Stanford Car dataset).
5. The image was grayscale.

The value 128 was chosen in the third step because the synthetic images were of dimension 128 by 128; I wanted the segmentations to be at least as large as the synthetic ones. The 0.7 confidence cutoff was arbitrary. Increasing this too high led to far too few segmentations being produced. The 0.5 ratio cutoff was chosen because anything lower would mean accepting segmentations where there is more bounding box than segmentation. I do not believe a good segmentation of boxy objects like cars should have such a ratio lower than 0.5. Grayscale images were dropped as they are not realistic.

Figure 3.4: One of the many successful segmentations

Figure 3.5: The mask wrongly includes some reflections from a puddle

Figure 3.6: The mask is of an object of the wrong class

Figure 3.7: There is nothing wrong with this mask. However the car is cropped severely

Figure 3.8: Bad segmentations showing different failure cases

3.1.1 Initial segmentation

From the original 16,185 images in the Stanford Car dataset, 17,398 segmentations were extracted which passed the aforementioned filters. These were rough filters. For the most part they did a good job but there were exceptions (Figure 3.8). Without doing any normalization or further filtering to the segmentations I found that my style transfer model, described in the AIST chapter, would struggle during training as demonstrated by Figure 3.9.

3.2 Further Filtering

The top right entry of Figure 3.9 is the result of the forward generator from a modified CycleGAN model that I used for style transfer had been trained for over 5 days - longer than any

Figure 3.9: The typical effects of doing no normalization or further filtering on style transfer. Top left: an image from the synthetic dataset. Top right: a stylized (to look real) version. Bottom left: a segmentation that past the filters from the previous chapter. Bottom right: a stylized (to look synthetic) version.

model I actually used. I realised that there were some significant difference between the synthetic images and the segmentations that were affecting the performance of the style transfer model:

1. The dimension of every synthetic image was 128x128 whereas the segmentations came in all shapes and sizes. Most were not even square.
2. The ratio of car to frame in terms of area was far smaller for the synthetic dataset than the segmentations. I believe this was one reason why the forward generator (synthetic to real) seemed to be trying to increase the size of the cars as is evident in the top row of Figure 3.9.
3. Some segmentations consisted of multiple connected components, mainly due to occlusion.
4. Every synthetic image fits nicely in frame whereas there are a large number of segmentations that are of highly cropped cars
5. Some segmentations are of the wrong class.

3.2.1 Frame ratio mismatch

I dealt with the first and second issues above at the same time. The solution I ended up using arose after inspecting the distribution of the ratios of car to frame for both the synthetic and real training sets.

Definition 1. For an RGBA synthetic image or a RGBA segmentation of a real image, I define the ratio of car to frame, or foreground to background, as

$$\frac{\text{number of pixels with alpha channel greater than 0}}{\text{number of pixels with alpha channel equal to 0}}$$

I chose "greater than 0" because the synthetic dataset's images' alpha channels were not binary-some materials were rendered as partially transparent.

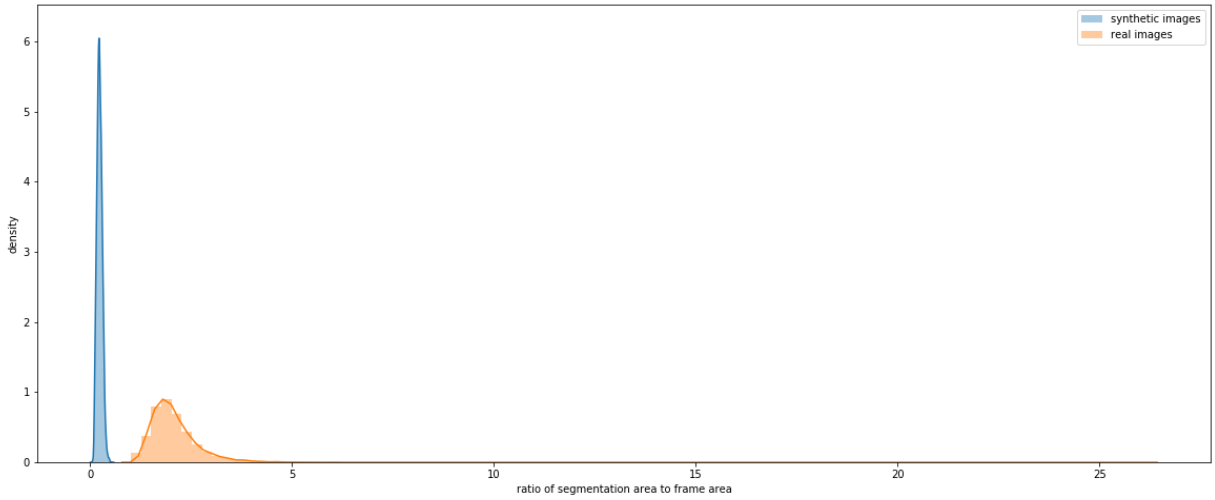


Figure 3.10: Car to Frame ratio distributions for both real segmentations and synthetic images.

The differences between the distributions were stark. Figure 3.10 shows that for synthetic images, as expected, the ratio of car to frame was tightly distributed, centered a little over 0.2. For the real images, the distribution was far more spread out and had a very long tail. The intersection of these distributions was almost zero. This means that this simple statistic could be used to differentiate between alpha layers (masks) of real and synthetic images very well. As I did not want to alter these masks to help preserve the underlying annotations, it was vital I normalized the real images to help the style transfer model. Otherwise, the forward discriminator network in the style transfer model would have no problem telling real and stylized synthetic images apart with this simple test. I believe this is one of the reasons why Figure 3.9 looks so bad.

The solution I settled on was to normalize the real images by first sampling ratios from a normal distribution that had been fitted to the synthetic training set's car to frame ratio distribution. The real images were padded with zeros to become square and then padded again so that their ratios of car to frame agreed with the sampled ratios. I justify using this approach along with the other filters I used with Figure 3.14. It shows that this approach transforms the frame to ratio distributions for the real images so that its distribution is all but indistinguishable from the synthetic one.

To pad images to become square I padded the longer sides of the rectangle by half the difference between the length of the longer side and the length of the smaller side, see Figure 3.12. If the difference was odd then I randomly chose one of the sides to receive the remainder.

Given a ratio r that had been sampled from the fitted Normal distribution, Figure 3.13 describes how to calculate the number of pixels needed to pad each side of the now square segmentation to match the sampled ratio.

3.2.2 Occlusion filtering

To try to filter out segmentations that had been badly occluded I added a filter that would catch any segmentations that consisted of more than one connected component. This dealt with the worst offenders such as Figure 3.11.

Figure 3.11: A typical occluded segmentation

3.2.3 Cropping detection

To check if an image had been cropped, images would be rejected if more than a third of any 2% wide border was covered. The reasoning being if a segmentation consisted of a lot of pixels at the border then the image probably doesn't contain the whole car.

3.3 Post Segmentation and Normalization

From the 17,398 segmentations, 3306 (19%) were rejected by occlusion filtering and cropping detection leaving 14092 segmentations that were made square and normalized. I didn't perform any filtering to try and remove segmentations of the wrong class. The few that made it through haven't seemed to have had a significant impact on the style transfer models. At this point the synthetic and real datasets can now be used to train the style transfer models. I performed an 80/20 train test split to this dataset so I could evaluate the performance of derived shape and pose models on unseen real data.

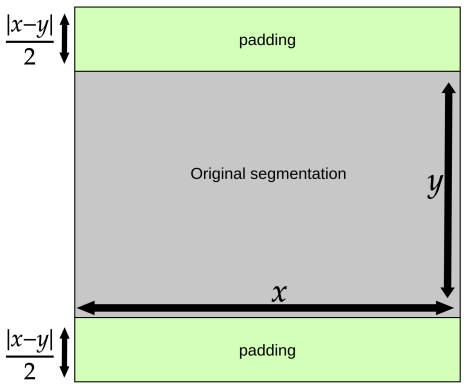
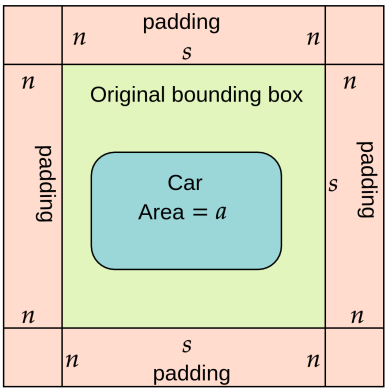


Figure 3.12: Padding rectangular segmentations with zeros to become square



Total Area $A = s^2 + 4n^2 + 4ns$

sampled ratio : $r = \frac{a}{A-a} \iff \frac{a}{r} = A - a$

$\iff$

$0 = 4rn^2 + 4srn + s^2r - a(r+1) \quad (\star)$

with positive solution:

$n = \frac{-sr + \sqrt{r(r+1)a}}{2r}$

chosen padding $= \left\lceil \frac{-sr + \sqrt{r(r+1)a}}{2r} \right\rceil$

Figure 3.13: Padding square segmentations to match a sampled car to frame ratio r. s is the length of each side of the original square segmentation, A is the area of the whole padded image, a is the area of the car and n is the amount of padding required to satisfy $(\star)$ such that the resulting car to frame ratio matches r.

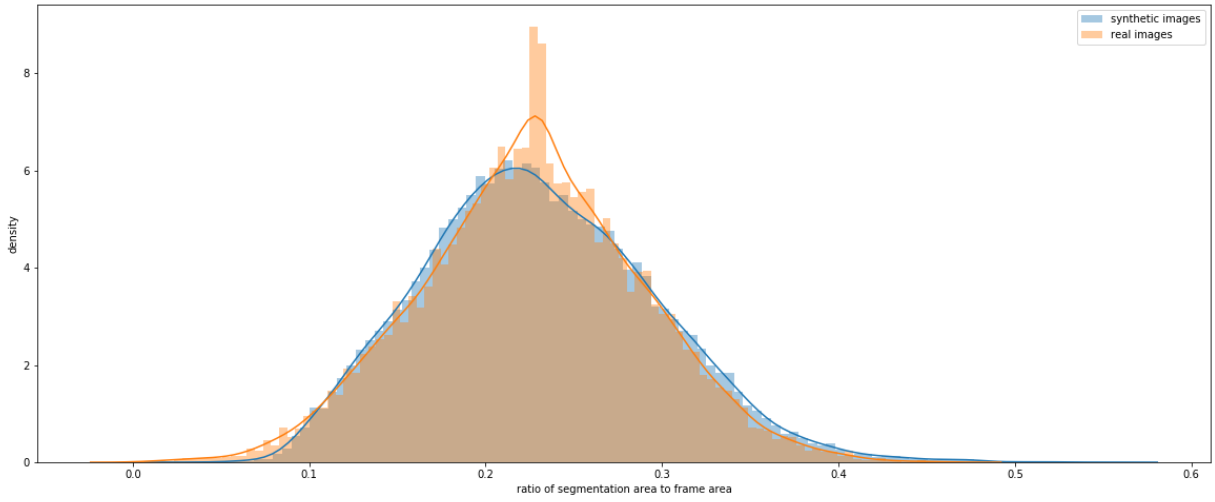


Figure 3.14: Car to frame ratio distributions after occlusion filtering, cropping detection and normalization: car to frame ratios can no longer be used to distinguish between synthetic and real images.

Chapter 4

Annotation Invariant Style Transfer

For style transfer, I started with a stock CycleGAN implementation [42] and modified the loss function of the generators to help keep the annotations, the corresponding shape and pose ground truth associated with every synthetic image, invariant. The stock implementation used the Least Squares GAN objective functions [28] (LSGAN) for the GAN loss instead of the cross entropy objective functions from the original paper[43].

After describing the LSGAN objective functions, Cycle Consistency and identity losses which were used in the stock implementation's loss functions I describe the modifications I made and justify them. I then give the architecture of both the generators and the Discriminators.

Nomenclature:

Let X be the synthetic training images, Y the real training images, $G : X \to Y$ the forward generator, $F : Y \to X$ the backward generator, $D_X : X \to [0,1]$ the synthetic image discriminator and $D_Y : Y \to [0,1]$ the real image discriminator. The discriminators are tasked with mapping fake images to 0 and real(real in the sense that they were not generated) images to 1.

4.0.1 LSGAN objective functions

The LSGAN objective functions for the forward generator G and real discriminator D_Y are defined as:

$$V_{LSGAN_{D_Y}}(G, D_Y, X, Y) = \frac{1}{2} \mathop{\mathbb{E}}_{y \sim Y} \left[(D_Y(y) - 1)^2 \right] + \frac{1}{2} \mathop{\mathbb{E}}_{x \sim X} \left[D_Y(G(x))^2 \right]$$

$$V_{LSGAN_G}(G, D_Y, X, Y) = \mathop{\mathbb{E}}_{x \in X} \left[(1 - D_Y(G(x))^2 \right]$$

During training $V_{LSGAN_{D_Y}}$ is minimized with respect to D_Y so that the discriminator network D_Y can tell real images apart from synthetic ones which have been stylized with G. At the same time V_{LSGAN_G} is minimized with respect to G to try to make D_Y incorrectly label synthetic images as real ones. Due to the inherent conflict between these objectives, the generator and discriminator fight against each other. The hope being that they reach an equilibrium.

The LSGAN objective functions for the backward generator F and synthetic discriminator D_X and they way they are trained is similar:

$$V_{LSGAN_{D_X}}(F,D_X,X,Y) = \frac{1}{2}\mathop{\mathbb{E}}_{x\sim X}\left[(D_X(x))^2\right] + \frac{1}{2}\mathop{\mathbb{E}}_{y\sim Y}\left[(D_X(F(y))-1)^2\right]$$
$$V_{LSGAN_F}(F,D_X,X,Y) = \mathop{\mathbb{E}}_{y\in Y}\left[D_X(F(y))^2\right]$$

4.0.2 Cycle consistency

The cycle consistency loss terms complement the LSGAN objectives. They are used to encourage the generators F and G to behave as inverses of each other. They try to enforce $F(G(x)) \approx x\ \forall x \in X$ as well as $G(F(y)) \approx y\ \forall y \in Y$. An x in X should be mapped to Y and back to itself by applying G then F, likewise for an y in Y.

The cycle consistency loss terms for the forward and backward directions are:

$$V_{CYCLE_{Forward}}(G,F,X,Y) = \lambda_A \mathop{\mathbb{E}}_{x\in X}\left[|x - F(G(x))|_1\right]$$
$$V_{CYCLE_{Backward}}(G,F,X,Y) = \lambda_B \mathop{\mathbb{E}}_{y\in Y}\left[|y - G(F(y))|_1\right]$$

where $|\bullet|_1$ is the L1 norm and λ_A and λ_B are non-negative scalars.

4.0.3 Identity loss

The final loss terms included in the stock implementation were the identity loss terms. The aim of these loss terms is to force the generators to do little to inputs which belong in their mathematical image. This is enforced using the following losses:

$$V_{Idt_G}(G,Y) = \lambda_B\lambda_{idt} \mathop{\mathbb{E}}_{y\in Y}\left[|y - G(y)|_1\right]$$
$$V_{Idt_F}(F,X) = \lambda_A\lambda_{idt} \mathop{\mathbb{E}}_{x\in X}\left[|x - F(x)|_1\right]$$

where λ_{idt} is another non-negative scalar. In words, the goal of the identity losses is such that G will do little to a real image and F will do little to a synthetic image.

4.0.4 Putting it together 1

The combined cost function for the generators is:

$$L_{Generators} = V_{LSGAN_G} + V_{LSGAN_F} + V_{CYCLE_{Forward}} + V_{CYCLE_{Backward}} + V_{Idt_G} + V_{Idt_F}$$

The cost functions for either discriminators are:

$$L_{D_Y} = V_{LSGAN_{D_Y}}$$
$$L_{D_X} = V_{LSGAN_{D_X}}$$

4.1 Modifications to the Generator Loss

I added a few basic sets of terms to the Generator Loss function, $L_{Generators}$, to encourage the generators to preserve shape and pose annotations as best as possible. The first set of terms encouraged the model to preserve the outline of each image by defining a loss on the difference between the alpha layers of the input and output images for each generator. The second set of terms tried to penalize large differences between the input and output images of each generator by defining a loss on the difference between features, extracted by a feature extractor, of the respective input and output images. This is near identical to *Feature Reconstruction loss* defined by Johnson et al. [24].

4.1.1 Alpha loss

The goal of these loss functions was to penalize alterations the generators F and G made to the outline of elements in X and Y respectively. Since the shape and pose model uses reprojection error as its training signal it was vital the outlines of images was as invariant under style transfer as possible.

$$V_{Alpha_G} = \lambda_{Alpha_G} \mathop{\mathbb{E}}_{x \in X} \left[|\text{Alpha}(x) - \text{Alpha}(G(x))|_1^2 \right]$$
$$V_{Alpha_F} = \lambda_{Alpha_F} \mathop{\mathbb{E}}_{y \in Y} \left[|\text{Alpha}(y) - \text{Alpha}(F(y))|_1^2 \right]$$

$\text{Alpha} : \mathbb{R}^{n \times n \times 4} \to \mathbb{R}^{n \times n \times 1}$ takes an RGBA image $x \in \mathbb{R}^{n \times n \times 4}$ and returns the alpha channel - a masking operation. In practice it was a little more complicated. The stock implementation of CycleGAN did not use alpha channels at all so I had to improvise slightly. Rewriting my own version of CycleGAN from scratch would have taken me too long. I reasoned that where the alpha channel of an image is zero there should be no RGB data there either. Hence I used an alternative definition of Alpha, redefining it as $\text{Alpha} : \mathbb{R}^{n \times n \times 3} \to \mathbb{R}^{n \times n \times 1}$ where

$$\text{Alpha}(x)_{ij} = \begin{cases} 1 & \text{Any of the RGB channels of pixel } x_{ij} \text{ are strictly greater than } -1 \\ 0 & \text{otherwise} \end{cases}$$

The -1 was due to the fact that RGB values were normalized by the implementation to lie between -1(nothing) to 1 (saturated). I used the L1 loss squared because I wanted to penalize large errors severely relative to the other loss terms. Similar but not the same as using the L2 loss.

4.1.1.1 Colour correction

This alternative alpha masking function let me add the above alpha loss terms to $L_{Generators}$ without having to rewrite my own CycleGAN implementation. However it also meant I had to apply additional prepossessing to the real and synthetic images. Many synthetic images and the occasional real image (segmentation) had areas with absolute darkness on the car so the alternative alpha masking function would incorrectly assume those pixels were not where the car was. Therefore I prepossessed all the synthetic and real segmentations so that the pixels which had an RGB value of $(0,0,0)$ (before the CycleGAN implementation normalized the images, RGB values were integers in $[0,256]$) and alpha channel greater than zero had their RGB value set to $(1,1,1)$. This was the next least bright possible grey RGB value. In this way

the alternative alpha masking function could distinguish between car and not car as only pixels which were not where a car was could have RGB value $(0,0,0)$.

4.1.2 ResNet feature distance

One problem I encountered was that occasionally the generators would place wheels or head lights on top of windows or car body parts. The alpha terms were not able to discourage this behaviour as the outline of the car would not change. To try to penalize this behaviour I added another set of terms to $L_{Generators}$:

$$V_{\text{ResDist}_G} = \lambda_{\text{ResDist}_G} \mathop{\mathbb{E}}_{x \in X} \left[|\Phi(x) - \Phi(G(x))|_1^2 \right]$$
$$V_{\text{ResDist}_F} = \lambda_{\text{ResDist}_F} \mathop{\mathbb{E}}_{y \in Y} \left[|\Phi(y) - \Phi(F(y))|_1^2 \right]$$

Where Φ is all but the last two layers of pyTorch's built in 18 Layer ResNet [31] that has been pretrained on Image-Net[9] and $\lambda_{\text{ResDist}_G}$ and $\lambda_{\text{ResDist}_F}$ are non-negative scalars. Since I was using all but the last two layers, Φ acted as a feature extractor. Therefore the above terms discouraged large changes in the features extracted by a feature extractor, just like feature reconstruction loss [24]. The intention was these terms would help penalize large changes such as extra wheels being inserted etc. Again I used the L1 loss squared to penalize large errors severely compared to the other loss terms.

4.1.3 Putting it together 2

The modified cost function for the generators was:

$$\begin{aligned} L_{Generators} = \quad & V_{LSGAN_G} + V_{LSGAN_F} + V_{CYCLE_{Forward}} + V_{CYCLE_{Backward}} + V_{Idt_G} + V_{Idt_F} \\ & + V_{Alpha_G} + V_{Alpha_F} + V_{ResNet_G} + V_{ResNet_F} \end{aligned}$$

The cost functions for the discriminators remained the same:

$$L_{D_Y} = V_{LSGAN_{D_Y}}$$
$$L_{D_X} = V_{LSGAN_{D_X}}$$

4.2 Architecture of the Generators

The stock implementation's forward and backward generators had an identical architecture. Input images were first scaled to be 256 by 256 pixels and RGB values were mapped from $[0,255]$ to $[-1,1]$. The generator networks consisted of 9 residual blocks sandwiched between some strided convolutional and transposed-convolutional layers which performed down-scaling and up-scaling respectively. The exact architecture was:

1. A 7×7 convolution of 64 filters, stride 1 and padding 3.
2. A 3×3 convolution of 128 filters, stride 2, padding 1.
3. A 3×3 convolution of 256 filters of stride 2, padding 1.
4. 9 residual blocks. See Figure 4.1.
5. A 3×3 transposed convolution of 128 filters, stride 2, padding 1.
6. A 3×3 transposed convolution of 64 filters, stride 2, padding 1.

7. A 7×7 convolution of 3 filters, stride 1, padding 3.
8. A tanh layer

Each convolution and transposed convolution in the above list is followed by non-affine instance normalization and a leaky ReLU. The leaky ReLU's all used a negative slope of 0.01. The tanh layer ensures the output RBG values fall in $[-1, 1]$. Reflection padding was used - the padded cell next to the left of the top leftmost pixel will have the value of the pixel to the right of the top leftmost pixel etc.

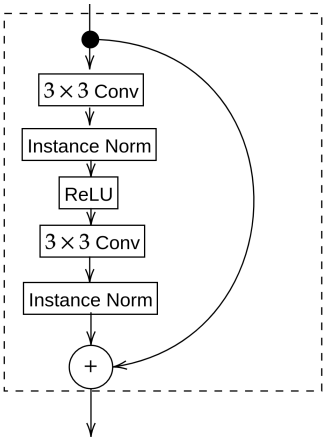


Figure 4.1: A residual block used in the generators. For both 3×3 convolutions there were 256 filters with stride 1 and padding 1. Instance normalization was non-affine and the leaky ReLU's had a negative slope of 0.01.

4.3 Architecture of the Discriminators

The architecture of the discriminators, the same for either one, was that of a PatchGAN [21]:

1. A 4×4 convolution of 64 filters, stride 2, padding 1.
2. A 4×4 convolution of 128 filters, stride 2, padding 1.
3. A 4×4 convolution of 256 filters, stride 2, padding 1.
4. A 4×4 convolution of 512 filters, stride 1, padding 1.
5. A 4×4 convolution of a single filter, stride 1, padding 1.

All but the first convolution are followed by non-affine instance normalization and a ReLU. The first is followed by a ReLU only. The ReLUs all had a negative slope of 0.2 and padding was done with zeros, not reflection.

The discriminator architecture is such that the final convolution produces a $1 \times 30 \times 30$ tensor where each entry corresponds to a classification prediction for 70 by 70 areas of the input image. This affects the structure of the terms involving the discriminator network in the objective functions previously mentioned.

This changes $D_Y : Y \to [0,1]$ to $D_Y : Y \to [0,1]^{30 \times 30}$. For $V_{LSGAN_{D_Y}}$ and V_{LSGAN_G} the MSE between the output of D_Y and the correct label is minimized inside the expectation terms instead of just the square of the difference between the output of the discriminator and the label. As an

example, $V_{LSGAN_{D_Y}}$ and V_{LSGAN_G} become:

$$V_{LSGAN_{D_Y}}(G,D_Y,X,Y) = \frac{1}{2}\mathop{\mathbb{E}}_{y\sim Y}\left[\frac{1}{30^2}\sum_{1\leq i,j\leq 30}(D_Y(y)_{ij}-1)^2\right] + \frac{1}{2}\mathop{\mathbb{E}}_{x\sim X}\left[\frac{1}{30^2}\sum_{1\leq i,j\leq 30}D_Y(G(x))_{ij}^2\right]$$

$$V_{LSGAN_G}(G,D_Y,X,Y) = \mathop{\mathbb{E}}_{x\in X}\left[\frac{1}{30^2}\sum_{1\leq i,j\leq 30}(1-D_Y(G(x)_{ij})^2\right]$$

$V_{LSGAN_{D_X}}$,V_{LSGAN_F} have to be modified similarly.

Chapter 5

Shape and pose estimation using differentiable point clouds

I use the provided implementation from the differentiable point clouds paper [19] to evaluate the performance of AIST in the experiments chapter. I did not modify this implementation in any way.

I use this chapter to describe how the model and associated loss functions works in a bit more depth than in the Background section. I also discuss the metrics used to evaluate performance in this paper. I use the same metrics to understand the effectiveness of AIST.

5.1 Architecture

As mentioned in the Background section this model uses an ensemble of regressors to distil a student regressor for pose prediction.

The model consists of three distinct parts. Four if you also include the differentiable renderer.

The first is an Encoder CNN. This consists of 7 layers:

1. The first layer is a 5×5 convolution of 16 filters with stride 2
2. The other 6 layers are all 3×3 convolutions and come in pairs. The first of each pair has stride 2 and the latter has stride 1. The number of filters increases by a factor of 2 after the first layer of each pair.

The activation function used after each convolution was a leaky ReLU. After the encoder CNN there is a 2 layer MLP with each layer having 1024 units which both use leaky ReLUs. This MLP is shared by both shape prediction and pose prediction.

After the shared MLP two branches split off. The simpler one is the branch for shape prediction. This consists of a single hidden layer with 1024 units using a leaky ReLU which predicts the x,y,z positions of 8 thousand points in the form of a 24 thousand dimensional vector. tanh is then applied to the 8 thousand points to ensure they stay within the output coordinates.

The other branch that splits off is the pose branch. This begins with a single 1024 unit layer using a leaky ReLU activation function. This further splits again. Four branches correspond to each regressor in the ensemble and one corresponds to the student regressor. Both the ensemble and student regressor networks consist of an MLP with 2 layers of 32 units with the first layer using a leaky ReLU as an activation function and the second having no activation function.

These 2 layer MLPs predict pose as a quaternion in the form of a 4 dimensional vector. All leaky ReLUs use a negative slope of 0.2.

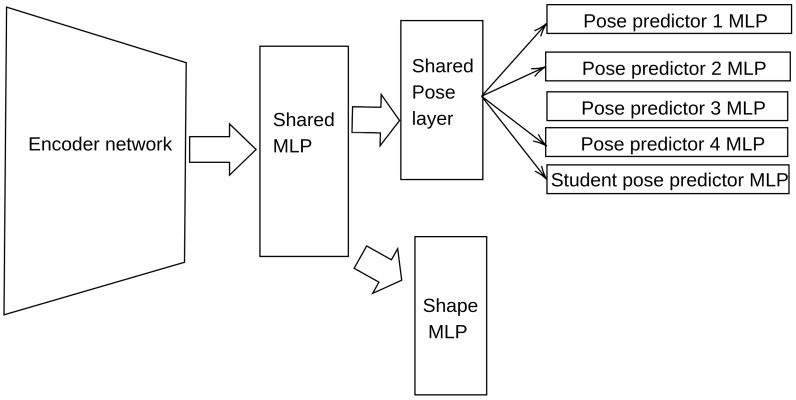


Figure 5.1: Diagram of the differentiable point clouds model architecture

5.2 Loss Functions

During training, batches of 16 images are used: 16 views of 4 different models. For each model, for each possible pair of images (6 total), the first image in each pair is chosen for shape prediction and the later for pose prediction. Explicitly, both images are passed through the encoder and shared MLP. Then the output of the shared MLP corresponding to the first image is passed through the shape branch to predict the shape and the output of the shared MLP corresponding to the second image is passed to the pose branch to predict pose. The contribution to the loss function of such a pair is the minimum L2 norm between the ground truth projection of the second image and the render of the predicted shape (from the first image) from the predicted pose (from the second image), for each regressor in the ensemble of pose predictors. The loss a batch is thus:

$$\mathcal{L}(\theta_P, \theta_C) = \sum_{i=1}^{N} \sum_{1 \le j_1, j_2 \le 4} \min_{c_k, 1 \le k \le 4} ||\widehat{p_{j_1 j_2 c_k}}^{i} - p^i_{j_1}||^2$$

$\mathcal{L}(\theta_P, \theta_C)$ is the loss with the point cloud parameters θ_P and pose prediction parameters θ_C (the weights of either network). N is the number of models in each batch, four in this case. j_1, j_2 sum over the six distinct pairs of views and $p^i_{j_1}$ is the ground truth of the j_1th view of the ith object. $\widehat{p_{j_1 j_2 c_k}}^{i}$ is the projection of the shape predicted with the j_1th view of the ith object from the pose predicted from the j_2th image of the ith object by ensemble regressor c_k.

To train the student regressor the goal was to minimize the angular difference between the pose estimated by the student and the best scoring pose regressor in the ensemble for each pair of images for each model. Using quaternions the paper used the following loss function:

$$\mathcal{L}(\theta_S) = \sum_{i=1}^{N} \sum_{j=1}^{6} 1 - \Re\Big(\frac{q^i_{Sj} {q^i_{Rj}}^{-1}}{||q^i_{Sj} {q^i_j}^{-1}||}\Big)$$

where j runs over each image pair per object, q^i_{Sj} is the pose the student predicts for the jth image pair of the ith object and q^i_j is the pose predicted by the kth regressor that minimized $||\widehat{p_{j_1 j_2 c_k}}^i - p^i_{j_1}||^2$ from the previous expression.

5.3 Optimization

I used the same optimization options that Insafutdinov and Dosovitskiy set in their implementation [19]. The Adam optimizer [25] was used for 600,000 batch iterations where each batch consisted of the 16 images mentioned above with a fixed learning rate of 0.0001. Dropout was applied to the point cloud vector that is predicted by the shape branch because the paper "found it useful to ensure an even distribution of points on the shape". The dropout rate is linearly reduced from 7% to zero over training as far as I can tell from their code. However in the paper it suggests it is reduced from 90% to zero over the course of training instead.

5.4 Evaluation Metrics

The paper makes use of three metrics for measuring the performance of shape and pose estimation models. I will use them to compare models in the chapter 6. The first, concerning shape, is a measure of how well two point clouds match each other:

Definition 2 (Chamfer Distance)**.**

$$d_{Chamf}(P_1, P_2) = \frac{1}{|P_1|} \sum_{x \in P_1} \min_{y \in P_2} ||x - y||_2 + \frac{1}{|P_2|} \sum_{y \in P_2} \min_{x \in P_1} ||y - x||_2$$

Where P_1,P_2 are point clouds and $x, y \in \mathbb{R}^3$ are individual points of either cloud.

If P_1 is ground truth and P_2 is a predicted point cloud then the first sum is the precision of the predicted points while the second sum is the coverage of the ground truth by the predicted cloud.

Concerning pose, two metrics are used. The first is the fraction of predicted poses that were within 30 degrees of the ground truth pose (accuracy) and the second is the median error in degrees between the predicted pose and the ground truth.

Chapter 6

Experiments

I performed three experiments. The first experiment takes the role of the baseline. I simply trained the shape and pose model [19] on the synthetic training set. The optimization details are in the previous chapter. In second experiment I use a dataset derived from the synthetic training set using AIST to train another shape and pose model. I derive this new dataset using my AIST model (the modified cycleGAN), the synthetic training images and the real segmentations that have been filtered and normalized as described in previous chapters. After training the AIST model, the entire synthetic dataset (I included the test set as I wanted to evaluate the baseline on stylized images) was stylized by the model, forming a stylized datasets. A fresh shape and pose model was then trained on the stylized dataset's training set. The third Experiment is identical except different hyperparameters were chosen to train the AIST model. I evaluate the baseline and AIST enhanced models on a test set of synthetic images as well as a test set of real segmentations that were left out during the training of the style transfer models. For all three experiments the shape and pose models are trained in exactly the same way.

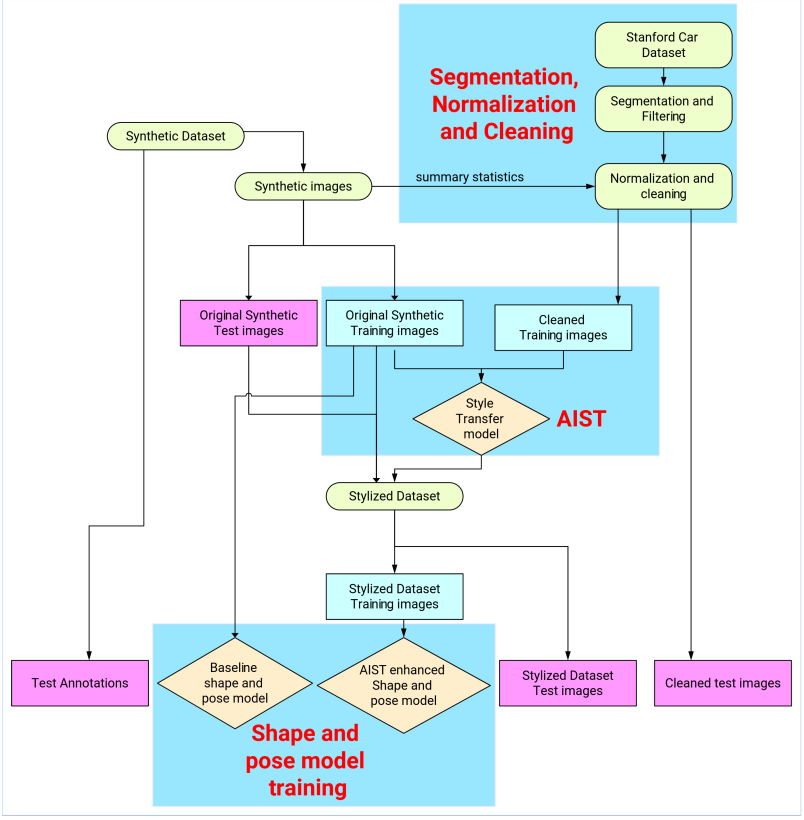


Figure 6.1: A more detailed diagram of my project's pipeline illustrating the flow of data for both the control and AIST enhanced shape and pose models.

6.1 AIST Enhanced Shape and Pose

The second and third experiments are identical aside from some hyper parameter tuning I did to the modified CycleGAN.

Recall the loss function for the generators of my modified cycleGAN:

$$\begin{aligned}
L_{Generators} &= V_{LSGAN_G} + V_{LSGAN_F} + V_{CYCLE_{Forward}} + V_{CYCLE_{Backward}} + V_{Idt_G} + V_{Idt_F} \\
&\quad + V_{Alpha_G} + V_{Alpha_F} + V_{ResNet_G} + V_{ResNet_F} \\
&= \mathop{\mathbb{E}}_{x\in X}\left[(1-D_Y(G(x))^2\right] + \mathop{\mathbb{E}}_{y\in Y}\left[D_X(F(y))^2\right] + \lambda_A \mathop{\mathbb{E}}_{x\in X}\left[|x-F(G(x))|_1\right] + \\
&\lambda_B \mathop{\mathbb{E}}_{y\in Y}\left[|y-G(F(y))|_1\right] + \lambda_B\lambda_{idt} \mathop{\mathbb{E}}_{y\in Y}\left[|y-G(y)|_1\right] + \lambda_A\lambda_{idt} \mathop{\mathbb{E}}_{x\in X}\left[|x-F(x)|_1\right] + \\
&\lambda_{Alpha_G} \mathop{\mathbb{E}}_{x\in X}\left[|\text{Alpha}(x)-\text{Alpha}(G(x))|_1^2\right] + \lambda_{Alpha_F} \mathop{\mathbb{E}}_{y\in Y}\left[|\text{Alpha}(y)-\text{Alpha}(F(y))|_1^2\right] \\
&\quad + \lambda_{\text{ResDist}_G} \mathop{\mathbb{E}}_{x\in X}\left[|\Phi(x)-\Phi(G(x))|_1^2\right] + \lambda_{\text{ResDist}_F} \mathop{\mathbb{E}}_{y\in Y}\left[|\Phi(y)-\Phi(F(y))|_1^2\right]
\end{aligned}$$

Accordingly the hyperparameters I could modify were λ_A, λ_B,λ_{idt},λ_{Alpha_G},λ_{Alpha_F},$\lambda_{\text{ResDist}_G}$ and $\lambda_{\text{ResDist}_F}$. These all control the relative importance of the different loss terms: λ_A and λ_B control the importance of the LSGAN losses, λ_{Idt} (in conjunction with λ_A and λ_B) controls the importance of the identity losses, and the alpha and ResDist terms control the importance of the Alpha and Residual Distance terms respectively.

I was extremely constrained in terms of compute resources. With the single GTX 1080Ti (12GB VRAM) graphics card it took over 4 days to train one of these style transfer models. This was compounded by the fact that each shape and pose model took 3 and a half days to train alone.

Due to these constraints I was only able to train two style transfer models (and their associated shape and pose models) properly. I spent about a week testing the effects of different hyperparameter values on training images during the first few epochs of training before committing to the values I chose. However, my parameter search was very sparse. I just did not have the resources or time to search the hyperparameter space properly. Figure 6.1 enumerates the hyperparameters I chose to train the style transfer model with for the second (constrained) and third (less constrained) experiments. As the names suggest the style transfer model for the second experiment uses slightly more constraining hyperparameters in terms of the losses that I modified CycleGAN with compared to the model for the third experiment.

For both of the style transfer models I did manage to train properly I terminated training after 44 epochs as I saw no noticeable improvements for experiments and did not have the time to let them train longer.

Experiment	λ_A	λ_B	λ_{idt}	λ_{Alpha_G}	λ_{Alpha_F}	$\lambda_{\text{ResDist}_G}$	$\lambda_{\text{ResDist}_F}$
Constrained (experiment 2)	10	10	0.1	2	2	1.75	1.75
Less constrained (experiment 3)	10	10	0.1	1	1	1.25	1.25

Table 6.1: Hyperparameters used to train the style transfer models from the second and third experiments. Aside from λ_A and λ_B whose values I took from the CycleGAN paper [43], the hyperparameters were effectively chosen arbitrarily given the little compute resources I had access to. Significantly better values may exist.

6.1.1 Optimization details

The stock implementation of CycleGAN and the associated paper uses a strategy to reduce "model oscillation" [34] which updates the discriminators using buffer of 50 previous images produced by the generators. The Adam optimizer is used with a batch size of 4 (4 pairs of images from either domain are generated at the same time). The learning rate was kept constant at 0.0002 throughout training and weights were initialized from a normal distribution of zero mean and variance 0.02. The implementation and paper also divides the objective by two when optimizing for the discriminators to slow the rate at which the discriminators learn compared to the generators.

I used the same values of λ_A and λ_B that the cycleGAN paper used. I found that increasing the values of the ResDist terms much higher would cause the network to collapse to the identity mapping. This was to be expected as the minimum loss for these terms is achieved when the mapping does nothing.

6.2 Results on Synthetic Data

After obtaining the shape and pose models for each of the three experiments I evaluated their performance on the 7500 image (of 1500 models) test set from the synthetic dataset and stylized datasets too. In table 6.2 I report the precision and coverage terms of the Chamfer distance along with the Chamfer distance itself (lower is better) multiplied by 100, as Insafutdinov and Dosovitskiy [19] did. I also report the accuracy (higher is better) of pose estimation at the 30 degree threshold along with the median pose error (lower is better) in degrees.

Experiment	Dataset	Precision	Coverage	Chamfer distance	Accuracy	Median Error
Baseline	Original	1.60	**1.35**	**2.95**	0.82	6.95
	Stylized 2	1.61	1.37	2.98	0.83	7.01
	Stylized 3	1.61	1.37	2.98	0.83	7.01
Experiment 2	Original	**1.59**	1.37	2.96	0.83	**4.55**
	Stylized 2	1.57	1.35	2.92	0.84	4.27
	Stylized 3	1.57	1.35	2.93	0.84	4.27
Experiment 3	Original	1.62	1.36	2.97	**0.84**	6.01
	Stylized 2	1.61	1.34	2.95	0.84	5.64
	Stylized 3	1.60	1.34	2.95	0.84	5.65

Table 6.2: Performance of each experiment when tested on the original and stylized datasets' test sets. Stylized 2 and Stylized 3 refer to the tylized datasets created by stylizing the synthetic dataset, the original, with experiment 2 and 3's respective style transfer models. The best results on the original synthetic dataset are highlighted in bold for each metric. Notice the significant reduction in median error that experiment 2 achieves relative to the baseline on the original dataset.

6.2.0.1 Shape

From the results it is is clear that the Chamfer scores are all very close to each other to the point that it seems as though AIST hasn't affected shape prediction at all. I believe this was more of an issue with the model architecture and the relatively low resolution of the synthetic data than any thing else. The baseline struggled to express fine details such as door handles,

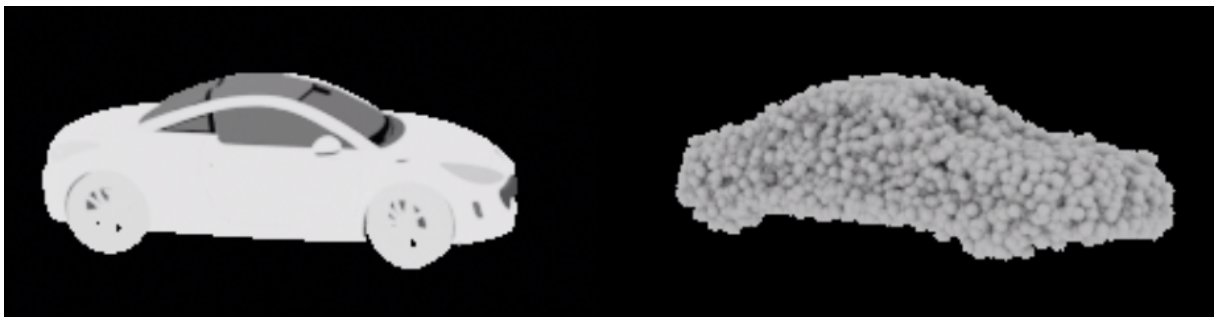


Figure 6.2: A still image from one of the generated animations I created. As discussed in the Background the authors of the point clouds paper [19] could not reproduce their synthetic dataset exactly from the shapeNet repository. I tried to match it as best as possible but there was still some discrepancy. On the left is a synthetic render and on the right is the point cloud the baseline predicted, rendered from the predicted pose using the code for rendering point clouds the same authors provided. Note the apparent error in pose prediction.

wing mirrors and the individual spokes on wheels so I don't think there is much that could have been improved on anyway. Figure 6.2 is a still from one of the animations I generated and demonstrates this lack of expressiveness[1].

6.2.0.2 Pose

On the pose prediction front things look much more promising - the lack of expressive power with regard to fine details did not affect the architectures ability to express pose. A quaternion is just a 4D vector after all. The second and third experiments are both slightly more accurate than the baseline on the original test set with regard to pose accuracy at a threshold of 30 degrees.

They perform significantly better when it comes to median pose error. The less constrained experiment, experiment 3, performs 13.5% better than the baseline on the original test set while the more constrained experiment performs 34.5% better than the baseline. This is remarkable considering the shape and pose model used by experiment 2 never saw any synthetic images during training, only stylized versions. I believe this provides convincing evidence that AIST can be used to improve shape and pose estimation on synthetic datasets.

6.2.0.3 ResNet and Alpha Loss terms on AIST performance

The AIST models of experiment 2 and 3 both failed in interesting ways. For the most part the ResNet losses did their job and stopped wheels etc. being generated in the wrong place. However there were exceptions such as Figure 6.3. The difference between the constrained and less constrained sets of hyper parameters was subtle but non-negligible. Figure 6.4 demonstrated how bright colours were more likely to bleed into the surrounding areas when the less constraining hyperparameters were used. The Alpha loss terms seemed to have worked almost perfectly. I couldn't find any examples where the masks deviated significantly between the synthetic images and their stylized versions.

[1] Find these in the submitted work.

Figure 6.3: A synthetic image on the left. It's stylized version under the AIST model of experiment 2. Notice that a wheel has been rendered in the right place but as if there was no occlusion. This may not have been helped by the fact the view was from a negative elevation which is not something that happens often in real life and in particular in the training set of real segmentations used to train the AIST model.

Figure 6.4: Stylized images of a synthetic car under the AIST model of experiment 2 (left) and 3 (right). Notice that around the right rear light of the right image the red colour bleeds much more into the rest of the car than in the left image

6.3 Results on Real Data

I used the test set of real segmentations which were not used to train the AIST models to test the performance of each experiment on real images. This consisted of 2818 images. Obviously there was no ground truth information I could use to measure performance on the real images directly. What I could measure though was the difference between the masks of real segmentations and the masks of the projections formed by asking each experiment to predict the shape and pose from a real picture and the projecting the predicted point cloud to the predicted pose. The idea being that the better the shape and pose prediction, the smaller the differences between these masks. This was a very rough metric of course as due to pose ambiguity, the difference between masks could be small even if the pose was predicted incorrectly. I used the rendering method provided by the shape and pose model's implementation [19] that uses Blender [8] instead of the differentiable renderer to get sharper projections. An example render is given in the image on the right of Figure 6.2.

Instead of using the Dice coefficient, a popular metric which measures the *overlap* of two segmentations, I used the bitwise XOR to measure the *difference* between segmentation masks.

From the definition of XOR, the XOR of two binary segmentations leaves the pixels on which the segmentations disagreed. After XORing the masks together I would count up the number of pixels left (where the masks disagreed) and normalized this quantity by dividing by the number of pixels in the real image's segmentation. This gave the number of pixel disagreements as a fraction of the number of pixels in the real image's mask, or the normalized number of pixels in disagreement for short.

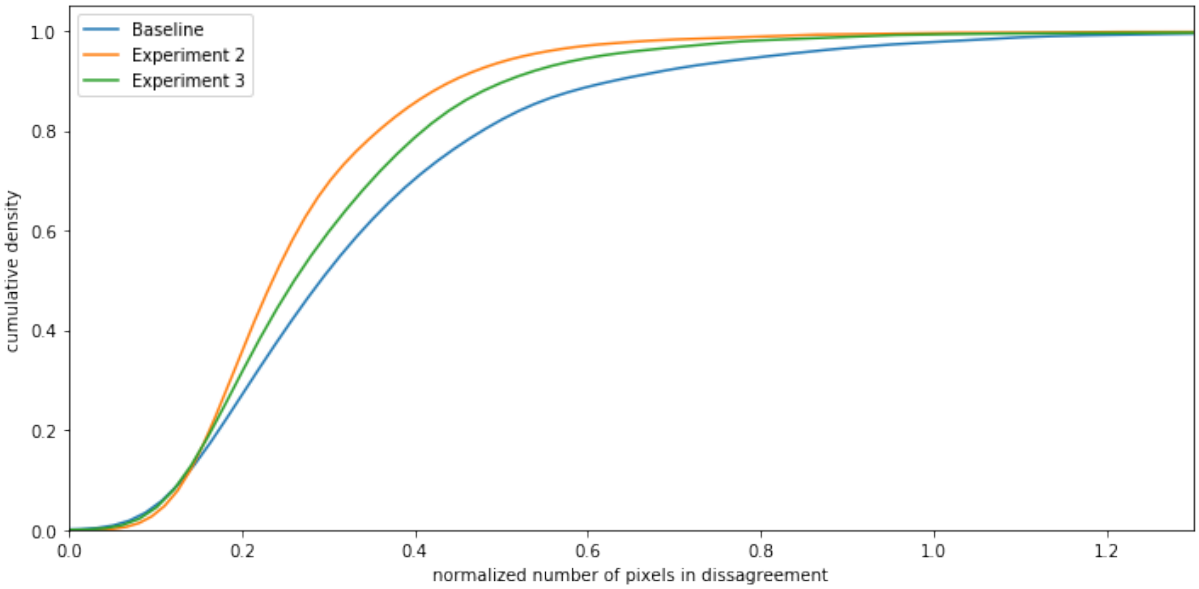


Figure 6.5: Cumulative density plots of three Gaussian kernel density estimates (GKDEs), for smoothness, each fitted to the normalized number of pixels in disagreement for each experiment on the test set of real segmentations.

The clear early rise of experiment 2 and experiment 3 in Figure 6.5 compared to the baseline indicates that experiment 3 and experiment 2 in particular performed better on real segmentations compared to the baseline. As this was a rough measure of performance I can't say much quantitatively due to pose ambiguity. However this is good evidence that using AIST helps existing shape and pose models perform better on real datasets.

Figure 6.6: A number of successfully stylized images generated using the AIST model from experiment 2 and the synthetic dataset.

Figure 6.7: A number of less successfully stylized images generated using the AIST model from experiment 2 and the synthetic dataset.

Chapter 7

Conclusion

I have shown that AIST can be used to improve pose estimation performance with no loss to shape estimation performance on synthetic datasets. I have also shown that AIST improves performance on real datasets, albeit using rough metrics. I have therefore answered my research question.

More importantly, I think the fact AIST improved performance on real datasets may signal the beginning of a new paradigm. In this project I used an unsupervised model to estimate shape and pose which required no explicit annotations to train. It only required images of multiple views of the target object. I believe I could have just as easily used a more supervised model. This could mean that unsupervised models may no longer be the only tools that can be used on real datasets after all.

I pose the following question for future work: "Can the need for shape and pose annotations of real images, *required to train supervised models*, be overcome using AIST?" Based on the increase in performance AIST gave to the unsupervised method I used I believe this may be true.

7.1 Criticism and Future Work

There are many improvements that could be made to the my version of AIST. Figure 6.7 shows some of the many failure cases. The effects of these failures on the corresponding shape and pose model is unknown. For sure they will negatively effect performance but do they also act as some form of regularization for the shape and pose models, similar to Domain Randomization [36]? The fact the AIST enhanced shape and pose models performed better than the baseline on the synthetic dataset indicates that something like this could be the case.

The underlying architecture of AIST, CycleGAN, could be replaced with better architectures such as CrossNet [33]. This may open up the ability to create more effective loss terms that penalize changes to underlying annotations more effectively too. Architectures that implicitly preserve alpha masks may also yield better results compared to the alpha loss terms I shoe-horned into CycleGAN.

I think the hardest question will be how to improve upon what the ResDist loss terms were trying to do. They didn't work particularly well as demonstrated by the example in Figure 6.3 but I can't think of how to improve upon them.

The filtering and segmentation I performed on the Stanford Car dataset was very rough and at times arbitrary. Future work could look into improving these steps in the AIST pipeline. The

reliance on having a Mask R-CNN implementation to segment images should also be investigated. What is possible with only a synthetic dataset and a real dataset where the real dataset does not have the segmentation annotations that are needed to train Mask R-CNN for segmentation? Can segmentations obtained from the synthetic dataset be used in some way? is Mask R-CNN even the best choice?

7.2 Final Words

Figure 7.1 shows there is still a long way to go. It shows performance of the AIST enhanced models is still significantly worse on the real test set than on the synthetic test set. This is not a very fair comparison as there were many nonsense segmentations that passed my filters that were included in the real test set and the metric used is a rough one. Nevertheless, it is clear there is still work to be done to create and train models which perform just as well on real datasets as synthetic ones.

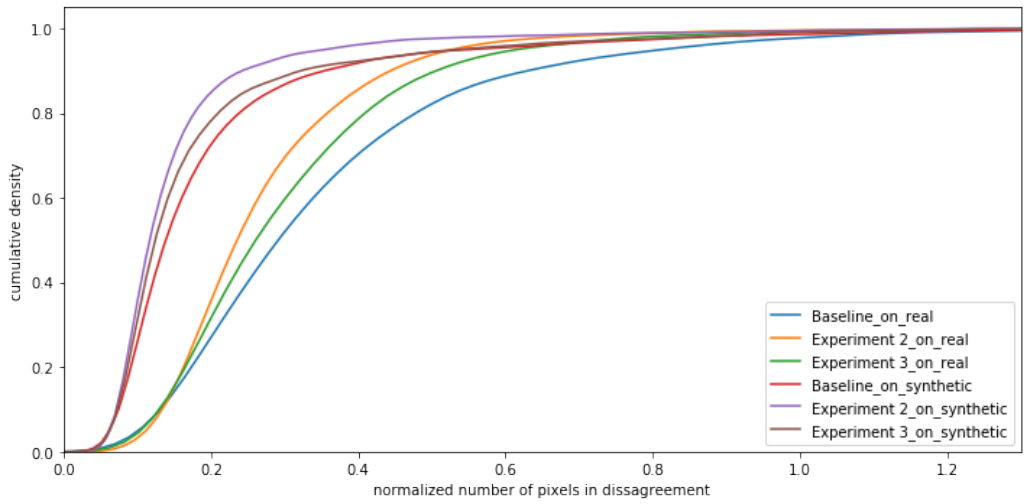


Figure 7.1: The reality gap. This shows Figure 6.5 with the addition of three more GKDEs, each fitted to the normalized number of pixels in disagreement for the different experiments, on 2818 (the real test set had 2818) random images from the test set of synthetic images.

Bibliography

[1] Max pooling. https://deepai.org/machine-learning-glossary-and-terms/max-pooling. Accessed: July 10, 2020.

[2] Redis. https://redis.io/. Accessed: July 10, 2020.

[3] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. https://github.com/matterport/Mask_RCNN, 2017. Accessed: July 10, 2020.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[5] Celery. Celery: Distributed task queue. http://www.celeryproject.org/. Accessed: July 10, 2020.

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[7] Brent Chun and Andrew McNabb. Pssh. https://linux.die.net/man/1/pssh, Jan 2009.

[8] Blender Online Community. Blender - a 3d modelling and rendering package. http://www.blender.org, 2020.

[9] J. Deng, K. Li, M. Do, H. Su, and L. Fei-Fei. Construction and Analysis of a Large Scale Image Ontology. Vision Sciences Society, 2009.

[10] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285.

[11] 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.

[12] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[13] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[15] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[17] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 327–340, New York, NY, USA, 2001. Association for Computing Machinery.

[18] Yoshitaka Ushiku Hiroharu Kato and Tatsuya Harada. Neural 3d mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[19] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[20] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[21] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[22] Jefkine. Backpropagation in convolutional neural networks. `https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/`, Sep 2016. Accessed: July 10, 2020.

[23] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[24] Justin Johnson, Alexandre Alahi, and Fei-Fei Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv e-prints*, page arXiv:1412.6980, December 2014.

[26] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[27] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[28] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, and Zhen Wang. Multi-class generative adversarial networks with the L2 loss function. *CoRR*, abs/1611.04076, 2016.

[29] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[30] Sarah Price. Convolution in two dimensions. `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/low/space/convol.htm`, 1996. Accessed: July 10, 2020.

[31] PyTorch. Pytorch. `https://pytorch.org/hub/pytorch_vision_resnet/`. Accessed: July 10, 2020.

[32] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[33] Omry Sendik, Dani Lischinski, and Daniel Cohen-Or. CrossNet: Latent Cross-Consistency for Unpaired Image Translation. *arXiv e-prints*, page arXiv:1901.04530, January 2019.

[34] Ashish Shrivastava, Tomas Pfister, Oncel Tuzel, Josh Susskind, Wenda Wang, and Russ Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. *arXiv e-prints*, page arXiv:1612.07828, December 2016.

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[36] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

[37] Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Computer Vision and Pattern Regognition (CVPR)*, 2018.

[38] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. *CoRR*, abs/1704.06254, 2017.

[39] Jasper Uijlings, K. Sande, T. Gevers, and Arnold Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.

[40] Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. *CoRR*, abs/1512.06293, 2015.

[41] Jiajun Wu, Yifan Wang, Tianfan Xue, Xingyuan Sun, Bill Freeman, and Josh Tenenbaum. Marrnet: 3d shape reconstruction via 2.5d sketches. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 540–550. Curran Associates, Inc., 2017.

[42] Jun-Yan Zhu. pytorch-cyclegan-and-pix2pix. `https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix`, 2017. Accessed: July 10, 2020.

[43] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[44] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

Appendix A

Appleton Tower Render Farm

I mentioned in the background that the authors of the differentiable point clouds paper [19] whose synthetic dataset I used were not able to recreate their dataset from scratch. Unfortunately this did not become evident for many months after starting the project as it took a long time for them to confirm this. As such I spent a significant amount of time under the assumption that I would be able to make my own dataset from scratch.

I built a distributed render farm using the GPU machines in Appleton Tower. Given synthetic ShapeNet objects, it would generate multiple renders of each object from different views along with a corresponding ground truth point cloud, generated by sampling points uniformly from the surface of each model.

I built this using the terminal program pssh [7] to distribute start up and kill commands to celery [5] workers on DICE GPU machines. These would then communicate with a command program that sent instructions to the workers using the Redis [2] message broker. Images and the corresponding annotations were saved to AFS, the distributed file system that DICE users have access to. The rendering and point cloud generation performed by each worker was done using Blender [8]. During testing it was able to render 101 GB worth of images in 7 hours. This has a range of applications outside of shape and pose estimation which could prove useful to many future Honours project students.

Appendix B

Whistle-stop tour of Convolutional Neural Networks

Convolution networks dramatically reduce the number of parameters that need to be optimized compared to a standard neural network. They do this while still being able to capture many spacial dependencies. Sparse neural networks are vital for vision tasks because the number of parameters a standard neural network would have to learn is very large given the large dimension of images.

In a standard neural network the ith K-dimensional hidden layer L^i is defined as:

$$L_i = f(W^{i,i-1}L^{i-1} + b^i)$$

where b^i is a K-dimensional bias vector and $W^{i,i-1}$ is a $K \times J$ weight matrix where J is the dimension of hidden layer L^{i-1} and f is some nonlinear activation function such as RELUs [29]. The parameters b_i and $W^{i,i-1}$ have to be learned by the network and are dense.

A layer in a convolution network looks rather different. A nonlinear function f is still applied to a linear function of the input/previous layer L^{i-1}. However the structure of $W^{i,i-1}$ and b_i is no longer as simple.

B.0.1 From Continuous to Discrete Convolutions

Here I will go over some of the intuition behind continuous convolutions and why kernels are so crucial.
This goes in to more depth than is needed and so can be skipped. These notes were modified from the year 4 Fourier Analysis course (MATH10051):

Recall the definition of the convolution of two 1-Periodic Riemann Integrable functions $f, g : \mathbb{R} \to \mathcal{C}$:

$$f * g(x) = \int_0^1 f(y)g(x-y)\,dy = \int_0^1 f(x-y)g(y)\,dy = g * f(x)$$

Recall also the definition of the N-th partial sum of the Fourier series of f, $S_N f(x)$:

$$S_N f(x) = \sum_{k=-N}^{N} \hat{f}(k)e^{2\pi ikx}$$

where $\hat{f}(k)$ is the kth fourier coefficient of f:

$$\hat{f}(k) = \int_0^1 f(x) e^{-2\pi i k x}\, dx$$

I will also use the Dirichlet kernel, defined:

$$D_N(y) := \sum_{|k| \leq N} e^{2\pi i k y}$$

Observe:

$$\begin{aligned} S_N f(x) = & \sum_{|k| \leq N} \widehat{f}(k) e^{2\pi i k x} = \sum_{|k| \leq N} \Big(\int_0^1 f(y) e^{-2\pi i k y}\, dy \Big) e^{2\pi i k x} \\ = & \int_0^1 f(y) \Big(\sum_{|k| \leq N} e^{2\pi i k (x-y)} \Big)\, dy = \int_0^1 f(y) D_N(x-y)\, dy \end{aligned}$$

Hence $S_N f(x) = f * D_N(x)$. This is pretty remarkable. It links together Fourier analysis, convolutions and these so called kernel functions. Many results in Fourier Analysis are derived using convolutions involving hand crafted kernel functions. In convolutional networks these kernel functions are learned automatically and instead of helping us with proofs, these kernel functions help us with computer vision tasks.

Another important property from Fourier analysis is that the convolution of any two integrable functions is continuous. This means that the convolution of f and g must in some sense be as smooth or smoother than f and g.

A few papers [40] use Fourier Analysis among other tools to show important properties of convolutional neural networks. For example [40], the deeper the network the more translation-invariant features become.

B.0.2 2D Discrete Convolution

In the continuous case in 1 variable, the convolution $f * g(x)$ can be thought of as flipping the kernel (g) and sliding it over the domain of f and summing the pointwise multiplication of f and the flipped kernel.

The 2D convolution of continuous functions f and k is:

$$f * k(x,y) = \int_{\mathrm{Dom}(f)} \int_{\mathrm{Dom}(k)} f(\tau_u, \tau_v) k(x - \tau_u, y - \tau_v)\, d\tau_u d\tau_v \qquad [30]$$

This is similar but instead of sliding the flipped kernel along one direction we now slide it along two. The discrete convolution in 2D has a similar feel. Given an image I and a kernel $K \in \mathbb{R}^{k_1 \times k_2}$, the convolution is defined as:

$$\begin{aligned} (I * K)_{ij} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m,n) \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m,-n) \qquad [22] \end{aligned}$$

The cross correlation of I and K is defined as:

$$(I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m,n) \qquad [22]$$

Since convolution is the same as cross-correlation bar the flipped kernel, which is learned during training, cross-correlation can be used instead of convolution. This is standard practice. In the context of deep learning, convolution and cross-correlation are synonymous where people normally mean cross-correlation. From now on I refer to cross-correlation as convolution.

B.0.3 Convolutional Layers

Figures B.1 and B.2 describe how an image with a single channel can be convolved with a single kernel. In the case of multiple channels, as is the case with RGB images, independent kernels are stacked on top of each other which are convolved with each channel independently and then summed. A collection of C stacked kernels is called a filter.

The convolution of an Image $I \in \mathbb{R}^{H \times W \times C}$ and a bank of D filters $K \in \mathbb{R}^{k_1 \times k_2 \times C \times D}$ is defined as [22]:

$$(I * K)_{dij} = \sum_{d=0}^{D} \sum_{c=1}^{C} \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K(m,n,c,d) \cdot I(i+m, j+n, c) + b_d$$

where $b \in \mathbb{R}^D$ is a vector of biases. Each filter in K gets its own layer in the output. Note that the same bias is added to each element in the dth layer of the output. Finally the activation function is applied. So the form of the output O of a convolution layer is given by:

$$O_{dij} = f(I * K)_{dij}$$

Figure B.1: Example kernel from [10]

B.0.4 Stride and Padding

Input image I can be padded with p zeros in different ways to modify the dimensions of output O.The kernel can also be slid along in larger jumps, denoted s for stride, than 1. By choosing p and s, it is possible to create a convolution layer that significantly reduces the dimensionality of the input.

let the dimension of I be $i \times i$ and consider a single kernel with dimension $k \times k$. Then the dimension of the output, $I * K$ is $o \times o$ where

$$o = \left\lfloor \frac{i+2p-k}{s} \right\rfloor + 1 \qquad [10]$$

As an example if the input was 256 by 256 pixels and the kernel was 3 by 3, then to down-sample by a factor of two a stride of 2 could be used with 2 pixels of padding. Then the output dimension would be $\left\lfloor \frac{256+4-3}{2} \right\rfloor + 1 = 128$

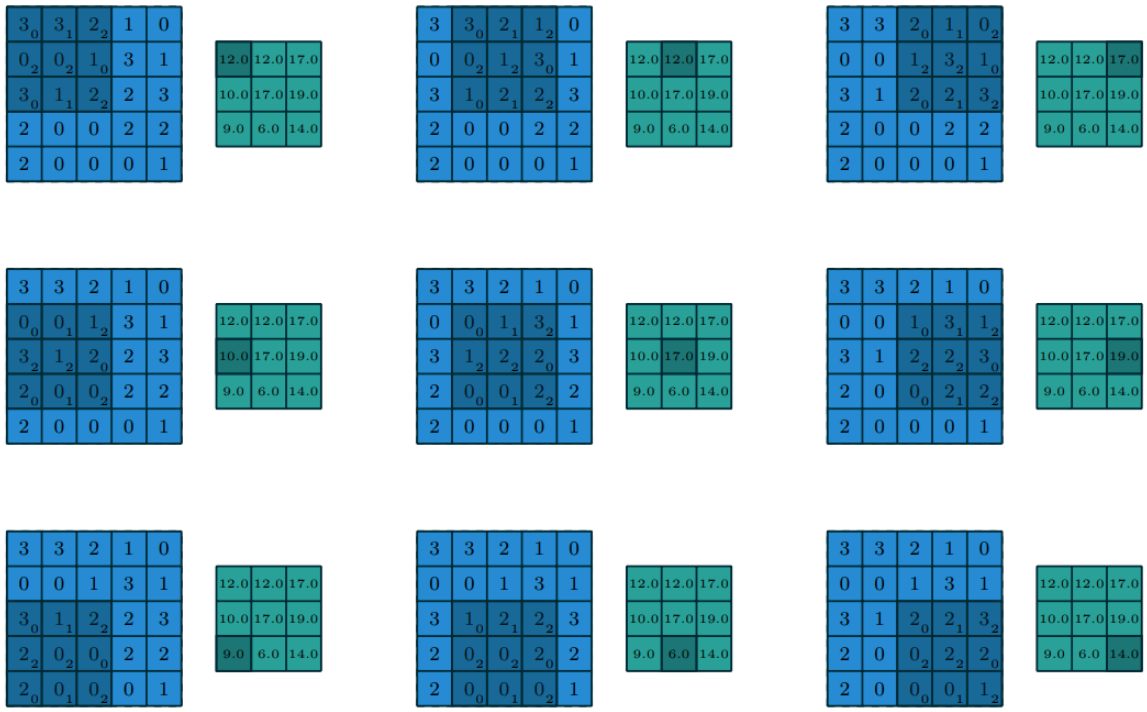


Figure B.2: convolution of a 2d array with the example kernel, taken from [10]. Note how the kernel slides around in two directions.

B.0.5 Max Pooling

A common step performed after a convolutional layer is max pooling. This is involves splitting the input of the previous layer into patches and returning the max of that patch. This helps with overfitting, provides some basic translation invariant properties and eases the computational cost. [1]. A very common type of Max pooling is with a patch size of 2 by 2 and a stride of 2. This will reduce the dimension of input image $I \in \mathbb{R}^{2n \times 2n}$ to output image $O \in \mathbb{R}^{n \times n}$. This operation has no learned parameters.

B.0.6 Transposed Convolutions

As convolutions (without adding the bias term) are linear maps they can be unravelled into matrices. Doing so yields sparse matrices defined by the kernel which the input image is multiplied by to give the convolution [10]. If AB represents this multiplication of the sparse matrix A and B represents the input image then taking the gradient of AB w.r.t. each element of B yields A^T. This is used in the implementation of back-propagation. More interestingly a matrix with the same sparsity structure as A^T, multiplied by an image with the same dimensions as AB yields a matrix of the same dimension as B. This acts as the opposite of what a convolutional layer does in that it can increase the dimensionality of the input image as opposed to reducing it. This is used extensively in networks whose output is another image. The most basic example would be an autoencoder.

B.0.7 ResNets

One significant problem encountered when training Deep Neural Networks (DNNs) is the degradation problem [16].

Intuition says that deeper networks should be able to perform no worse than shallower ones. In-

deed, a deeper network with the same weights as a shallower one followed by identity mappings must perform as well. However it has been shown that performance degrades as the number of layers increases[16] when back propagation is used for training. This led to a natural solution [16] which lets a network easily maintain identity mappings if it needs too:

$$\mathbf{y} = F(\mathbf{x}, \{W_i, b_i\}) + \mathbf{x} \tag{B.1}$$

Where x and y are input and outputs of a layer. $F(\cdot, \{W_i, b_i\})$ represents a number of intermediate layers, normally 2, that contain standard operations such as convolutions and RELUs. In the case where x and y have different dimensions, x is be premultiplied by the appropriate projection matrix.

Equation B.1, known as residual blocks allows gradients to be propagated backwards much more easily. Theses can be stacked together to form Residual Networks (ResNets) that are used widely.

B.1 Omitted Techniques

There are other tools that I use in this project which I haven't covered here such as Leaky ReLUs, instance normalization and dropout. These are all techniques used to aid the training of very deep networks. While important these were not of fundamental significance so I have omitted their details.

Appendix C

Mask R-CNN Configuration

The pretrained model taken from [3] which was used for segmentation has the configuration. This is auto generated by the implementation:

```
Configurations:
BACKBONE                        resnet101
BACKBONE_STRIDES                [4, 8, 16, 32, 64]
BATCH_SIZE                      1
BBOX_STD_DEV                    [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE          None
DETECTION_MAX_INSTANCES         100
DETECTION_MIN_CONFIDENCE        0.7
DETECTION_NMS_THRESHOLD         0.3
FPN_CLASSIF_FC_LAYERS_SIZE      1024
GPU_COUNT                       1
GRADIENT_CLIP_NORM              5.0
IMAGES_PER_GPU                  1
IMAGE_CHANNEL_COUNT             3
IMAGE_MAX_DIM                   1024
IMAGE_META_SIZE                 93
IMAGE_MIN_DIM                   800
IMAGE_MIN_SCALE                 0
IMAGE_RESIZE_MODE               square
IMAGE_SHAPE                     [1024 1024    3]
LEARNING_MOMENTUM               0.9
LEARNING_RATE                   0.001
LOSS_WEIGHTS                    {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0,
'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                  14
MASK_SHAPE                      [28, 28]
MAX_GT_INSTANCES                100
MEAN_PIXEL                      [123.7 116.8 103.9]
MINI_MASK_SHAPE                 (56, 56)
NAME                            coco
NUM_CLASSES                     81
POOL_SIZE                       7
POST_NMS_ROIS_INFERENCE         1000
```

```
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                1000
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```