**A Project Report**

On

# IMAGE SHAPE MANIPULATION FROM A SINGLE AUGMENTED TRAINING SAMPLE.

***Submitted in partial fulfillment of the***

***requirement for the award of degree of***

Bachelor of Technology in Computer Science and Engineering

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

**Under The Supervision of**
**Mr. V. ARUL**
**Assistant Professor**

**Submitted By:**
VAIBHAV NATH 19SCSE1010199
RUPESH KUMAR DWIVEDI 19SCSE1010010

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING,**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,**

**GALGOTIAS UNIVERSITY, GREATER NOIDA, INDIA**

**December-2021**

## SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
## GALGOTIAS UNIVERSITY, GREATER NOIDA

## CANDIDATE'S DECLARATION

We hereby certify that the work which is being presented in the project, entitled **"IMAGE SHAPE MANIPULATION FROM A SINGLE AUGMENTED TRAINING SAMPLE"** in partial fulfillment of the requirements for the award of the B-Tech CSE submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of July, 2021 to December, 2021, under the supervision of Mr.V.Arul, Assistant Professor, Department of Computer Science and Engineering, of School of Computing Science and Engineering , Galgotias University, Greater Noida, India.

The matter presented in the project has not been submitted by us for the award of any other degree of this or any other places.

Vaibhav Nath: 19SCSE1010199

Rupesh Kumar Dwivedi: 19SCSE1010010

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr. V.Arul
Assistant Professor

**<u>CERTIFICATE</u>**

The Final Project Viva-Voce examination of Vaibhav Nath: 19SCSE1010199 and Rupesh Kumar Dwivedi: 19SCSE1010010 has been held on _____________ and his work is recommended for the award of B.Tech-CSE.

**Signature of Examiner(s)** **Signature of Supervisor(s)**

**Signature of Project Coordinator** **Signature of Dean**

Date: 24 December 2021
Place: Greater Noida

## ACKNOWLEDGEMENT

I would like to thank my guide Mr V.ARUL who gave me this opportunity to work on this project. I got to learn a lot from this project about Image shape manipulation using machine learning algorithms (DeepSIM).

At last, I would like to extend my heartfelt thanks to my guide because without their help this project would not have been successfully completed. Finally, I would like to thank my colleagues who have been with me all the time.

## ABSTRACT

The DeepSIM, a generative model for conditional image manipulation based on a single image. Here, extensive augmentation is key for enabling single image training, and incorporate the use of thin-plate spline (TPS) as an effective augmentation. Our network learns to map between a primitive representations of the image to the image itself.

The choice of a primitive representation has an impact on the ease and expressiveness of the manipulations and can be automatic (e.g. edges), manual (e.g. segmentation) or hybrid such as edges on top of segmentations. At manipulation time, our generator allows for making complex image changes by modifying the primitive input representation and mapping it through the network. Our method is shown to achieve remarkable performance on image manipulation tasks.

# CONTENTS

## List of Figures

## Acronyms

| | |
|---|---|
| DeepSIM | Deep Simulation Model |
| B.Tech | Bachelor Of Technology |
| scipy | Scientific Python |
| numpy | Numerical Python |
| Pytorch | Pytorch is opensource library for computer vision and natural language processing |
| IDNA | Internationalized Domain Names in Applications |

# CHAPTER-1
# INTRODUCTION

## 1.1 Basic Introduction

Deep neural networks have significantly boosted performance on image manipulation tasks for which large training datasets can be obtained, such as, mapping facial landmarks to facial images. In practice, however, there are many settings in which the image to be manipulated is unique, and a training set consisting of many similar input-output samples is unavailable.

Moreover, in some cases using a large dataset might even lead to unwelcome outputs that do not preserve the specific characteristics of the desired image. Training generative models on just a single image, is an exciting recent research direction, which may hold the potential to extend the scope of neural-network-based image manipulation methods to unique images.

## 1.2 DeepSIM

In this project, we introduce - DeepSIM, a simple-to-implement yet highly effective method for training deep conditional generative models from a single image pair.

Our method is capable of solving various image manipulation tasks including:

(i) shape warping (ii) object rearrangement (iii) object removal (iv) object addition (v) creation of painted and photorealistic animated clips.

Given a single target image, first, a primitive representation is created for the training image. This can either be unsupervised (i.e. edge map, unsupervised segmentation), supervised (i.e. segmentation map, sketch, drawing), or a combination of both. We use a standard conditional image mapping network to learn to map between the primitive representation and the image. Once training is complete, a user can explicitly design and choose the changes they want to apply to the target image by manipulating the simple primitive (serving as a simpler manipulation domain). The modified primitive is fed to the network, which transforms it into the real image domain with the desired manipulation. This process is illustrated in Figure.

Several papers have explored the topic of what and how much can be learned from a single image. Two recent seminal works SinGAN and InGAN propose to extend this beyond the scope of texture synthesis. SinGAN tackles the problem of single image manipulation in an unconditional manner allowing unsupervised generation tasks. InGAN, on the other hand, proposes a conditional model for applying various geometric transformations to the image. Our paper extends this body of work by exploring the case of supervised image-to-image translation allowing the modification of specific image details such as the shape or location of image parts. We find that the augmentation strategy is key for making DeepSIM work effectively. Breaking from the standard practice in the image translation community of using a simple crop-

and-flip augmentation, we found that using a thin-plate-spline (TPS) augmentation method is essential for training conditional generative models based on a single image-pair input. The success of TPS is due to its exploration of possible image manipulations, extending the training distribution to include the manipulated input. Our model successfully learns  the internal statistics of the target image, allowing both professional and amateur designers to explore their ideas while preserving the semantic and geometric attributes of the target image and producing high fidelity results.

Our contributions in this project:

• A general purpose approach for training conditional generators supervised by merely a single image-pair.

• Recognizing that image augmentation is key for this task, and the remarkable performance of thin-platespline (TPS) augmentation which was not previously used for single image manipulation.

• Achieving outstanding visual performance on a range of image manipulation applications.

## Related images and animations

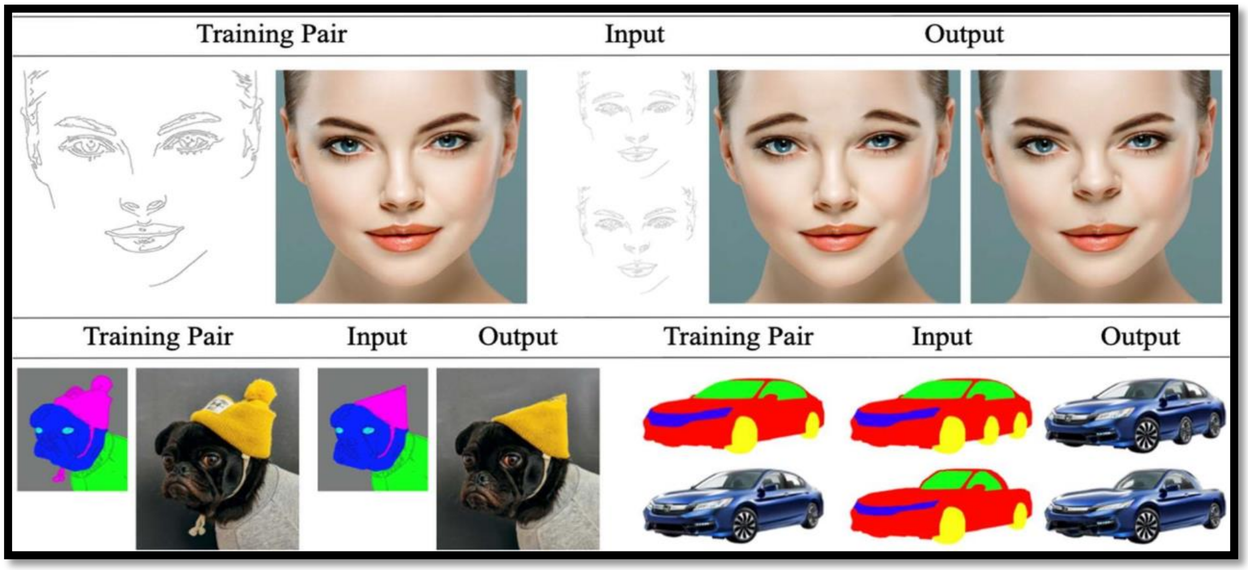

**Figure : 1** Image shape manipulation.

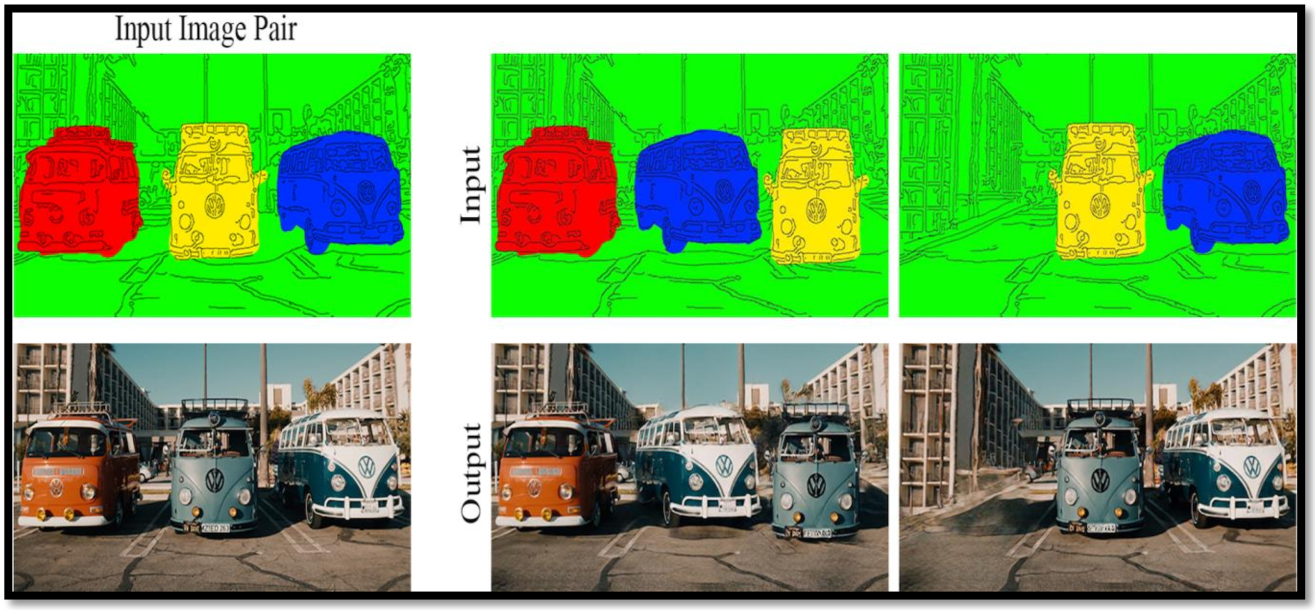

**Figure : 2** Image shape manipulation.

Top row - primitive images. Left - original pair used for training. Center- switching the positions between the two rightmost cars. Right- removing the leftmost car and in painting the background.

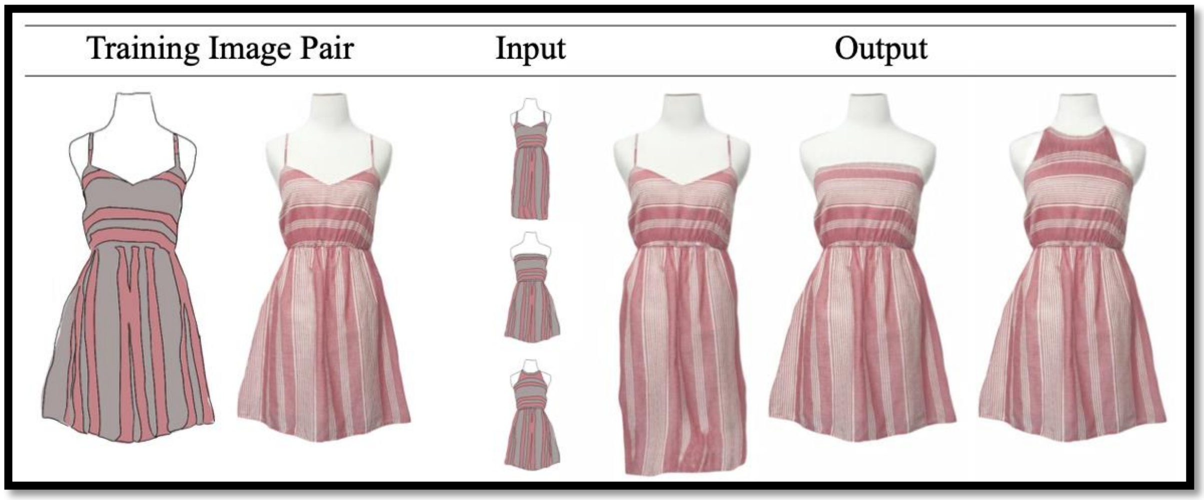

**Figure : 3** Image shape manipulation.

On the left is the training image pair, in the middle are the manipulated primitives and on the right are the manipulated outputs- left to right: dress length, strapless, wrap around the neck.

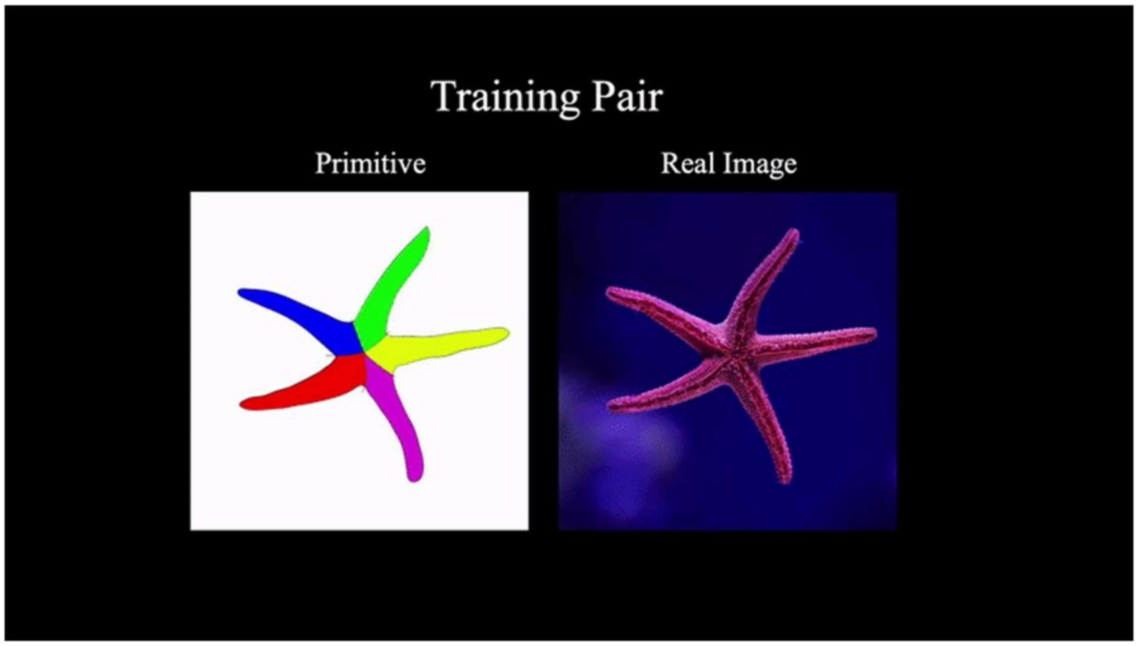

**Figure: 4** Animated video **-**1

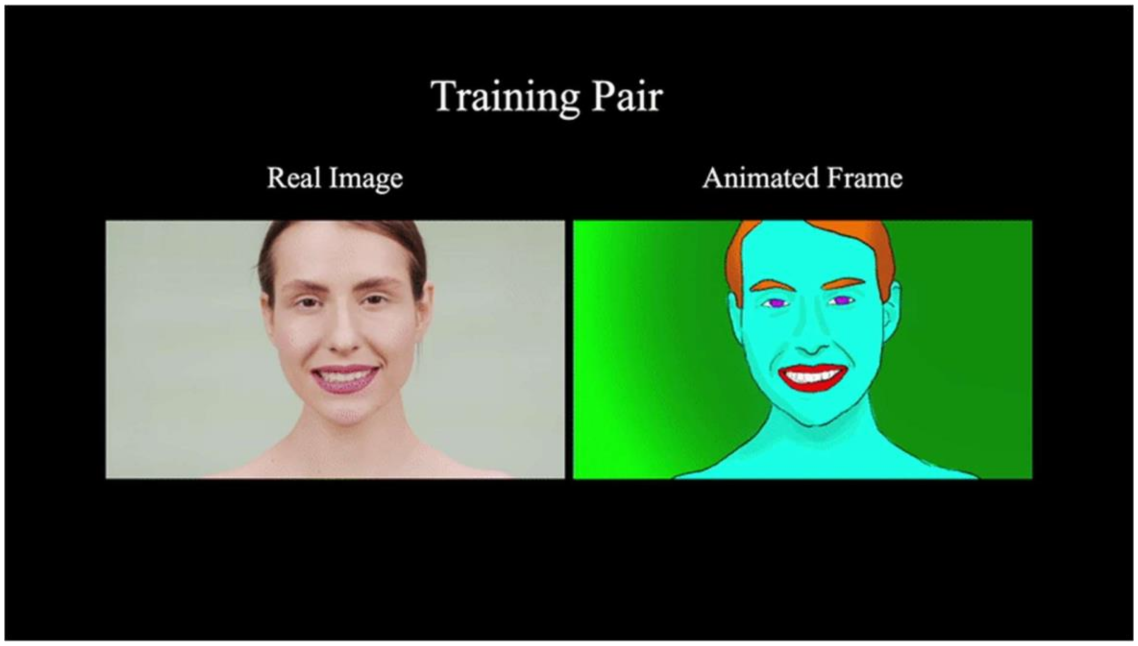

**Figure: 5** Animated video – 2.

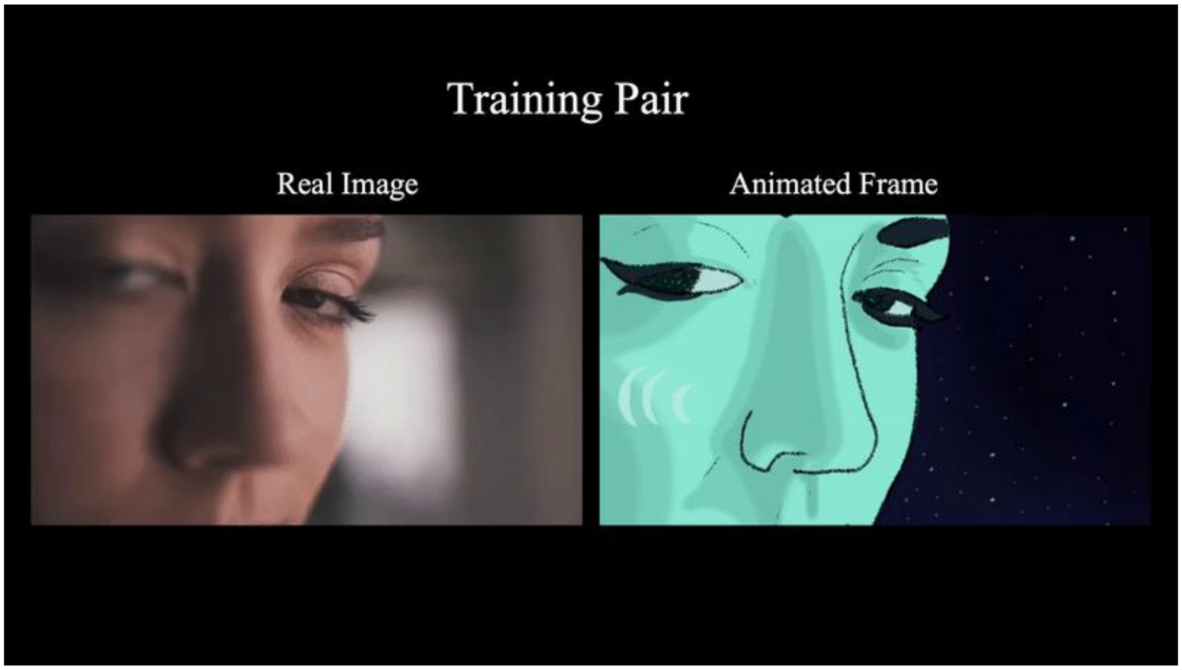

**Figure: 6** Animated video – 3.

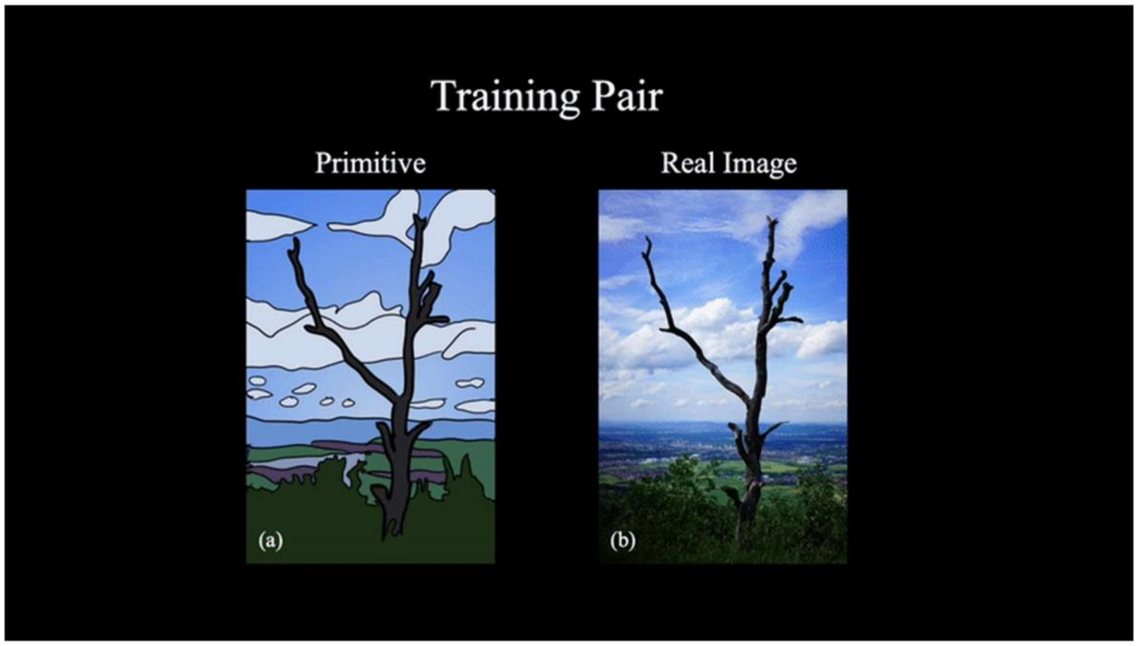

**Figure: 7** Animated video – 4.

# CHAPTER-2
# LITERATURE SURVEY

## 2.1 Semantic Image Manipulation

In 2018, Understanding, reasoning, and manipulating semantic concepts of images have been a fundamental research problem for decades. Previous work mainly focused on direct manipulation on natural image manifold through color strokes, key points, textures, and holes-to-fill. In this work, we present a novel hierarchical framework for semantic image manipulation. Key to our hierarchical framework is that we employ structured semantic layout as our intermediate representation for manipulation. Initialized with coarse-level bounding boxes, our structure generator first creates pixel-wise semantic layout capturing the object shape, object-object interactions, and object-scene relations. Then our image generator fills in the pixel-level textures guided by the semantic layout. Such framework allows a user to manipulate images at object-level by adding, removing, and moving one bounding box at a time. Experimental evaluations demonstrate the advantages of the hierarchical manipulation framework over existing image generation and context hole-filing models, both qualitatively and quantitatively. Benefits of the hierarchical framework are further demonstrated in applications such as semantic object manipulation, interactive image editing, and data-driven image manipulation.

Learning to perceive, reason and manipulate images has been one of the core research problems in computer vision, machine learning and graphics for decades. Recently the problem has been actively studied in interactive image editing using deep neural networks, where the goal is to manipulate an image according to the various types of user-controls, such as color strokes, key-points, textures, and holes-to-fill (in-painting). While these interactive image editing approaches have made good advances in synthesizing high-quality manipulation results, they are limited to direct manipulation on natural image manifold. The main focus of this paper is to achieve semantic-level manipulation of images. Instead of manipulating images on natural image manifold, we consider semantic label map as an interface for manipulation. By editing the label map, users are able to specify the desired images at semantic-level, such as the location, object class, and object shape. Recently, approaches based on image-to image translation have demonstrated promising results on semantic image manipulation. However, the existing works mostly focused on learning a style transformation function from label maps to pixels, while manipulation of structure of the labels remains fully responsible to users. The requirement on direct control over pixel-wise labels makes the manipulation task still challenging since it requires a precise and considerable amount of user inputs to specify the structure of the objects and scene. Although the problem can be partly addressed by template-based manipulation interface (e.g. adding the objects from the pre-defined sets of template masks, blind pasting of the object mask is problematic since the structure of the object should be determined adaptively depending on the surrounding context.

## 2.2 Realistic Image Manipulation

In 2019, Realistic image manipulation is challenging because it requires modifying the image appearance in a user-controlled way, while preserving the realism of the result. Unless the user has considerable artistic skill, it is easy to "fall off" the manifold of natural images while editing. In this paper, we propose to learn the natural image manifold directly from data using a generative adversarial neural network. We then define a class of image editing operations, and constrain their output to lie on that learned manifold at all times. The model automatically adjusts the output keeping all edits as realistic as possible. All our manipulations are expressed in terms of constrained optimization and are applied in near-real time. We evaluate our algorithm on the task of realistic photo manipulation of shape and color. The presented method can further be used for changing one image to look like the other, as well as generating novel imagery from scratch based on user's scribbles.

In 2020, We present an interactive method that manipulates perceived object shape from a single input color image thanks to a warping technique implemented on the GPU. The key idea is to give the illusion of shape sharpening or rounding by exaggerating orientation patterns in the image that are strongly correlated to surface curvature. We build on a growing literature in both human and computer vision showing the importance of orientation patterns in the communication of shape, which we complement with mathematical relationships and a statistical image analysis revealing that structure tensors are indeed strongly correlated to surface shape features. We then rely on these correlations to introduce a flow-guided image warping algorithm, which in effect exaggerates orientation patterns involved in shape perception. We evaluate our technique by 1) comparing it to ground truth shape deformations, and 2) performing two perceptual experiments to assess its effects. Our algorithm produces convincing shape manipulation results on synthetic images and photographs, for various materials and lighting environments.

## 2.3 Video Image Manipulation

Over the past few years, huge steps forward in the field of automatic video editing techniques have been made. In particular, great interest has been shown towards methods for facial manipulation. Just to name an example, it is nowadays possible to easily perform facial reenactment, i.e., transferring the facial expressions from one video to another one . This enables to change the identity of a speaker with very little effort. Systems and tools for facial manipulations are now so advanced that even users without any previous experience in photo retouching and digital arts can use them. Indeed, code and libraries that work in an almost automatic fashion are more and more often made available to the public for free. On one hand, this technological advancement opens the door to new artistic possibilities (e.g., movie making, visual effect, visual arts, etc.). On the other hand, unfortunately, it also eases the generation of video forgeries by malicious users. Fake news spreading and revenge porn are just a few of the possible malicious applications of advanced facial manipulation technology in the wrong hands. As the distribution of these kinds of manipulated videos indubitably leads to serious and dangerous consequences (e.g., diminished trust in media, targeted opinion formation, cyber bullying, etc.), the ability of detecting whether a face has been manipulated in a video sequence is becoming of paramount importance. Detecting whether a video has been modified is not a novel issue per se. Multimedia forensics researchers have been working on this topic since many years, proposing different kinds of solutions to different problems. For instance, in the authors focus on studying the coding history of videos. The authors of focus on localizing copy-move forgeries with block-based or dense techniques. In different methods are proposed to detect frame duplication or deletion. All the above-mentioned methods work according to a common principle: each non-reversible operation leaves a peculiar footprint that can be exposed to detect the specific editing. However, forensics footprints are often very subtle and hard to detect. This is the case of videos undergoing excessive compression, multiple editing operations at once, or strong down sampling. This is also the case of very realistic forgeries operated through methods that are hard to formally model. For this reason, modern facial manipulation techniques are very challenging to detect from the forensic perspective. As a matter of fact, many different face manipulation techniques exist (i.e., there is not a unique model explaining these forgeries). Moreover, they often operate on small video regions only (i.e., the face or part of it, and not the full frame).

# CHAPTER-3
# FUNCTIONALITY & WORKING

## 3.1 Functionality

Our method learns a conditional generative adversarial network (cGAN) using just a single image pair consisting of the main image and its primitive representation. To account for the limited training set, we augment the data by using thin-plate-spline (TPS) warps on the training pair. The proposed approach has several objectives:

i) single image training ii) fidelity - the output should reflect the primitive representation iii) appearance - the output image should appear to come from the same distribution as the training image. We will next describe each component of our method:-

Our model design follows standard practice for cGAN models (particularly Pix2PixHD). Let us denote our training image pair (x, y) where y ∈ Rdx×dy×3 is the input image (dx and dy are the number of rows and columns) and x ∈ Rdx×dy×dp is the corresponding image primitive (dp is the number of channels in the image primitive). We learn a generator network G: Rdx×dy×dp → Rdx×dy×3, which learns to map input image primitive x to the generated image G(x). The fidelity of the result is measured using the VGG perceptual loss ℓperk : (Rdx×dy×3,Rdx×dy×3) → R , which compares the differences between two images using a set of activations extracted from each image using a VGG network pre-trained on the Image Net dataset (we follow the implementation in). We therefore write the reconstruction loss:-

$$\ell_{rec}(x, y; G) = \ell_{perc}(G(x), y) \qquad (1)$$

Conditional GAN loss: Following standard practice, we add an adversarial loss which measures the ability of a discriminator to differentiate between the (primitive, generated image) pair (x, G(x)) and the (primitive, true Figure 6: TPS Visualisation. A random TPS warp of the primitive-image pair. Also see SM. image) pair (x, y). The conditional discriminator D :(Rdx×dy×dp ,Rdx×dy×3) → [0, 1] is implemented using a deep classifier which maps a pair of primitive and corresponding image into the probability of the two being a ground truth primitive-image pair. D is trained adversarial against G. The loss of the discriminator (ℓadv) is:

$$\ell_{adv}(x,y;D,G) = \log(D(x,y)) + \log(1 - D(x,G(x))) \quad (2)$$

The combined loss ℓtotal is the sum of the reconstruction and adversarial losses, weighted by a constant α:

$$\ell_{total}(x,y;D,G) = \ell_{rec}(x,y;G) + \alpha \cdot \ell_{adv}(x,y;D,G) \quad (3)$$

## 3.2 Working of project

- **Collecting Data**

  The first and very important part of this project is to collect data then we have to make it in proper format using some built-in libraries in python.

- **Creating a Dataset of Collected data**

  After making the data in proper format we will create a dataset of collected data using pandas.

- **Training the Data**

  We use DeepSIM to train the dataset for our model.

- **Testing the Data**

  We use our model built in one step above to test the model.

# CHAPTER-4
# ANALYSIS

**Input primitives:-** As segmentations capture high-level aspects of the image while edge maps capture the low-level of the image better, we analyse the primitive that combines both. This choice is uncommon, e.g. Pix2PixHD proposed combining instance and semantic segmentation maps, however, this does not provide low-level details compares the three primitives. The edge representation is unable to capture the eye, presumably as it cannot capture its semantic meaning. The segmentation is unable to capture the details in the new background regions creating a smearing effect. The combined primitive is able to capture the eye as well as the low-level textures of the background region. In Fig. we present more manipulation results using the combined primitive. In the centre column, we switched the positions of rightmost cars. As the objects were not of the same size, some empty image regions were filled using small changes to the edges. A more extreme result can be seen in the rightmost column, the car on the left was removed, creating a large empty image region. By filling in the missing details using edges, our method was able to successfully complete the background (see SM for an ablation).

**Runtime:** Our runtime is a function of the neural architecture and the number of iterations. When running all experiments on the same hardware (NVIDIA RTX-2080 Ti), a 256x256 image e.g. the ”face” image (Fig.) takes SinGAN 72 minutes to train, and 180 minutes for TuiGAN while DeepSIM (ours) takes 49 minutes. As was discussed previously, TuiGAN requires a new training process for each new manipulation whereas our DeepSIM does not.

Is the cGAN loss necessary? We evaluated removing the cGAN loss, keeping just the VGG perceptual loss on the Cars image (see SM). For such high-res images the cGAN was a better perceptual loss. At lower resolutions, the VGG results were reasonable but still blurrier than the cGAN loss.

Can methods trained on large datasets generalize to rare images? We present examples where this is not the case. Fig. showed that BicycleGAN did not generalize as well as Pix2PixHD-MI for new (in-distribution) shoes. We show that in the more extreme case, where the image lies further from the source distribution used for training, current methods fail completely. See SM for further analysis.

Augmentation in deep single image methods: Although we are the first to propose single-image training for manipulation using extensive non-linear augmentations, we see SinGAN as implicitly being an augmentation-based unconditional generation approach. In its first level it learns an unconditional low-res image generator, while latter stages can be seen as an upscaling network. Critically, it relies on a set of “augmented” input low-res images generated by the first stage GAN. Some other methods e.g. Deep Image Prior do not use any form of augmentation.

Failure modes: We highlight three main failure modes of DeepSIM : i) generating unseen objects - when the manipulation requires generating objects unseen in training, the network can do so incorrectly. ii) background duplication - when adding an object onto new background regions, the network can erroneously copy some background regions that originally surrounded the object. iii) interpolation in empty regions - as no guidance is given in empty image regions, the network hallucinates details, sometimes incorrectly.

# CHAPTER-5
# REQUIREMENTS & IMPLEMENTATION

## 5.1 Requirements

Following are the tools we are using in this project to accomplish it:

1. certify
2. charset-normalizer
3. cycler
4. dominate
5. idna
6. imageio
7. jsonpatch
8. jsonpointer
9. kiwisolver
10. matplotlib
11. networkx
12. numpy
13. Pillow
14. pyparsing
15. python-dateutil
16. PyWavelets
17. pyzmq
18. requests
19. scikit-image
20. scipy
21. tifffile
22. torch
23. torchaudio
24. torchvision
25. typing-extensions
26. urllib3
27. imageio-ffmpeg

## 5.2 Implementation

## Source Code for training the model:

```
import time
import os
import numpy as np
import torch
from torch.autograd import Variable
from collections import OrderedDict
from subprocess import call
import fractions


def lcm(a, b): return abs(a * b) / fractions.gcd(a, b) if a and b else 0


from options.train_options import TrainOptions
from data.data_loader import CreateDataLoader
from models.models import create_model
import util.util as util
from util.visualizer import Visualizer
import torchvision.utils as vutils


opt = TrainOptions().parse()
iter_path = os.path.join(opt.checkpoints_dir, opt.name, 'iter.txt')
if opt.continue_train:
    try:
        start_epoch, epoch_iter = np.loadtxt(iter_path, delimiter=',', dtype=int)
    except:
        start_epoch, epoch_iter = 1, 0
    print('Resuming from epoch %d at iteration %d' % (start_epoch, epoch_iter))
else:
    start_epoch, epoch_iter = 1, 0

opt.print_freq = lcm(opt.print_freq, opt.batchSize)
if opt.debug:
    opt.display_freq = 1
    opt.print_freq = 1
    opt.niter = 1
    opt.niter_decay = 0
    opt.max_dataset_size = 10

data_loader = CreateDataLoader(opt)
```

```
dataset = data_loader.load_data()
dataset_size = len(data_loader)
print('#training images = %d' % dataset_size)

model = create_model(opt)
visualizer = Visualizer(opt)
if opt.fp16:
    from apex import amp
    model, [optimizer_G, optimizer_D] = amp.initialize(model, [model.optimizer_G,
model.optimizer_D], opt_level='O1')
    model = torch.nn.DataParallel(model, device_ids=opt.gpu_ids)
else:
    if len(opt.gpu_ids) > 0:
        optimizer_G, optimizer_D = model.module.optimizer_G, model.module.optimizer_D
    else:
        optimizer_G, optimizer_D = model.optimizer_G, model.optimizer_D

total_steps = (start_epoch - 1) * dataset_size + epoch_iter

display_delta = total_steps % opt.display_freq
print_delta = total_steps % opt.print_freq
save_delta = total_steps % opt.save_latest_freq

print("display_delta", display_delta)
print("print_delta", print_delta)
print("save_delta", save_delta)

torch.cuda.empty_cache()
total_time_start = time.time()
for epoch in range(start_epoch, opt.niter + opt.niter_decay + 1):
    epoch_start_time = time.time()
    if epoch != start_epoch:
        epoch_iter = epoch_iter % dataset_size
    for i, data in enumerate(dataset, start=epoch_iter):
        if total_steps % opt.print_freq == print_delta:
            iter_start_time = time.time()
        total_steps += opt.batchSize
        epoch_iter += opt.batchSize

        # whether to collect output images
        save_fake = total_steps % opt.display_freq == display_delta

        a = time.time()
        losses, generated = model(Variable(data['label']), Variable(data['inst']),
                             Variable(data['image']), Variable(data['feat']), infer=save_fake)
        # sum per device losses
        losses = [torch.mean(x).unsqueeze(0) if not isinstance(x, int) else x for x in losses]
        if len(opt.gpu_ids) > 0:
            loss_dict = dict(zip(model.module.loss_names, losses))
        else:
            loss_dict = dict(zip(model.loss_names, losses))

        # calculate final loss scalar
        loss_D = (loss_dict['D_fake'] + loss_dict['D_real']) * 0.5
        loss_G = loss_dict['G_GAN'] + loss_dict.get('G_GAN_Feat', 0) +
loss_dict.get('G_VGG', 0)

        # update generator weights
        optimizer_G.zero_grad()
        if opt.fp16:
            with amp.scale_loss(loss_G, optimizer_G) as scaled_loss:
                scaled_loss.backward()
        else:
            loss_G.backward()
        optimizer_G.step()

        # update discriminator weights
        optimizer_D.zero_grad()
        if opt.fp16:
            with amp.scale_loss(loss_D, optimizer_D) as scaled_loss:
                scaled_loss.backward()
        else:
            loss_D.backward()
        optimizer_D.step()

        ### print out errors
        if total_steps % opt.print_freq == print_delta:
            errors = {k: v.data.item() if not isinstance(v, int) else v for k, v in loss_dict.items()}
            t = (time.time() - iter_start_time) / opt.print_freq
            visualizer.print_current_errors(epoch, epoch_iter, errors, t)
            visualizer.plot_current_errors(errors, total_steps)
            # call(["nvidia-smi", "--format=csv", "--query-gpu=memory.used,memory.free"])

        ### display output images
        if save_fake:
            visuals = OrderedDict([('input_label', util.tensor2label(data['label'][0],
opt.label_nc)),
                                   ('synthesized_image', util.tensor2im(generated.data[0])),
                                   ('real_image', util.tensor2im(data['image'][0]))])
            visualizer.display_current_results(visuals, epoch, total_steps)
        ### save latest model
        if total_steps % opt.save_latest_freq == save_delta:
            print('saving the latest model (epoch %d, total_steps %d)' % (epoch, total_steps))
            if len(opt.gpu_ids) > 0:
                model.module.save('latest')
            else:
                model.save('latest')
            np.savetxt(iter_path, (epoch, epoch_iter), delimiter=',', fmt='%d')
        if epoch_iter >= dataset_size:
            break

    # end of epoch
    iter_end_time = time.time()
    print('End of epoch %d / %d \t Time Taken: %.4f sec\n' %
          (epoch, opt.niter + opt.niter_decay, time.time() - epoch_start_time))

    ### save model for this epoch
    if epoch % opt.save_epoch_freq == 0:
        print('saving the model at the end of epoch %d, iters %d' % (epoch, total_steps))
        if len(opt.gpu_ids) > 0:
            model.module.save('latest')
            model.module.save(epoch)
        else:
            model.save('latest')
            model.save(epoch)
        np.savetxt(iter_path, (epoch + 1, 0), delimiter=',', fmt='%d')

    ### instead of only training the local enhancer, train the entire network after certain
iterations
    if (opt.niter_fix_global != 0) and (epoch == opt.niter_fix_global):
        if len(opt.gpu_ids) > 0:
            model.module.update_fixed_params()
        else:
            model.update_fixed_params()

    ### linearly decay learning rate after certain iterations
    if epoch > opt.niter:
        if len(opt.gpu_ids) > 0:
            model.module.update_learning_rate()
        else:
            model.update_learning_rate()

print('End of training [%s], Time Taken: %.4f sec' % (opt.name, time.time() -
total_time_start))
print('saving the model at the end of epoch %d, iters %d' % (epoch, total_steps))
if len(opt.gpu_ids) > 0:
    model.module.save('latest')
else:
    model.save('latest')
vutils.save_image(data['image'], '%s/%s/%d_image.png' % (opt.checkpoints_dir, opt.name, epoch_iter),normalize=True)
vutils.save_image(generated.data, '%s/%s/%d_fake.png' % (opt.checkpoints_dir, opt.name, epoch_iter),normalize=True)
vutils.save_image(data['label'], '%s/%s/%d_label.png' % (opt.checkpoints_dir, opt.name, epoch_iter),normalize=True)
np.savetxt(iter_path, (epoch + 1, 0), delimiter=',', fmt='%d')
print('End of training [%s], Time Taken: %.4f sec' % (opt.name, time.time() - total_time_start))
```

## Source Code for testing the model:

```
import os
from collections import OrderedDict
from torch.autograd import Variable
from options.test_options import TestOptions
from data.data_loader import CreateDataLoader
from models.models import create_model
import util.util as util
from util.visualizer import Visualizer
from util import html
import torch

opt = TestOptions().parse(save=False)
opt.nThreads = 1   # test code only supports nThreads = 1
opt.batchSize = 1
opt.serial_batches = True  # no shuffle
opt.no_flip = True  # no flip
opt.resize_or_crop = "none"




data_loader = CreateDataLoader(opt)
dataset = data_loader.load_data()
visualizer = Visualizer(opt)
# create website
web_dir = os.path.join(opt.results_dir, opt.name, '%s_%s' % (opt.phase, opt.which_epoch))
webpage = html.HTML(web_dir, 'Experiment = %s, Phase = %s, Epoch = %s' % (opt.name, opt.phase, opt.which_epoch))
```

```
# test
if not opt.engine and not opt.onnx:
    model = create_model(opt)
    if opt.data_type == 16:
        model.half()
    elif opt.data_type == 8:
        model.type(torch.uint8)

    if opt.verbose:
        print(model)
else:
    from run_engine import run_trt_engine, run_onnx

for i, data in enumerate(dataset):
    if i >= opt.how_many:
        break
    if opt.data_type == 16:
        data['label'] = data['label'].half()
        data['inst']  = data['inst'].half()
    elif opt.data_type == 8:
        data['label'] = data['label'].uint8()
        data['inst']  = data['inst'].uint8()
    if opt.export_onnx:
        print ("Exporting to ONNX: ", opt.export_onnx)
        assert opt.export_onnx.endswith("onnx"), "Export model file should end with .onnx"
        torch.onnx.export(model, [data['label'], data['inst']],
                          opt.export_onnx, verbose=True)
        exit(0)
    minibatch = 1
    if opt.engine:
        generated = run_trt_engine(opt.engine, minibatch, [data['label'], data['inst']])
    elif opt.onnx:
        generated = run_onnx(opt.onnx, opt.data_type, minibatch, [data['label'], data['inst']])
    else:
        generated = model.inference(data['label'], data['inst'], data['image'])
    print(data['label'].shape, data['inst'].shape, data['image'].shape, generated.shape)
    visuals = OrderedDict([('input_label', util.tensor2label(data['label'][0], opt.label_nc)),
                           ('synthesized_image', util.tensor2im(generated.data[0]))])
    img_path = data['path']
    print(img_path)
    print('process image... %s' % img_path)
    visualizer.save_images(webpage, visuals, img_path)

webpage.save()
```

```
if opt.vid_mode:
    util.frames_to_vid(webpage)
```

# CHAPTER-6
# CONCLUSION

We proposed a method for training conditional generators from a single training image based on TPS augmentations. Our method is able to perform complex image manipulation at high-resolution. Single image methods have significant potential, they preserve image fine-details to a level not typically achieved by previous methods trained on large datasets. One limitation of single- image methods (including ours) is the requirement for training a separate network for every image. Speeding up training of single-image generators is a promising direction for future work.

## REFERENCES

[1] Nur Arad, Nira Dyn, Daniel Reisfeld, and Yehezkel Yeshurun. Image warping by radial basis functions: Applications to facial expressions. CVGIP: Graph. Models Image Process.,56(2):161–172, Mar. 1994. 3

[2] Yuki M Asano, Christian Rupprecht, and Andrea Vedaldi. A critical analysis of self-supervision, or what we can learn from a single image. arXiv preprint arXiv:1904.13132, 2019. 3

[3] Shai Avidan and Ariel Shamir. Seam carving for contentaware image resizing. In SIGGRAPH, 2007. 2

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. 2

[5] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. SIGGRAPH Comput. Graph., 26(2):35–42, July 1992. 3

[6] Urs Bergmann, Nikolay Jetchev, and Roland Vollgraf. Learning texture manifolds with the periodic spatial GAN. CoRR, abs/1705.06566, 2017. 2

[7] J Canny. A computational approach to edge-detection. Ieee transactions on pattern analysis and machine intelligence, 1986. 17

[8] Wengling Chen and James Hays. Sketchygan: Towards diverse and realistic sketch to image synthesis, 2018. 3

[9] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In CVPR, 2018. 3

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3213–3223, 2016. 17

[11] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. arXiv preprint arXiv:1805.09501, 2018.3

[12] Tali Dekel, Chuang Gan, Dilip Krishnan, Ce Liu, and William T Freeman. Smart, sparse contours to represent and edit images. arXiv preprint arXiv:1712.08232, 2017. 3

[13] Gianluca Donato and Serge Belongie. Approximate thin plate spline mappings. In European conference on computer vision, pages 21–31. Springer, 2002. 2, 5

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, DavidWarde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In NIPS, pages 2672–2680, 2014. 3

[15] Xintong Han, Zuxuan Wu, Zhe Wu, Ruichi Yu, and Larry S Davis. Viton: An image-based virtual try-on network. In CVPR, 2018. 3

[16] Aaron Hertzmann, Charles E. Jacobs, Nuria Oliver, Brian Curless, and David H. Salesin. Image analogies. SIGGRAPH,2001. 2, 18, 25