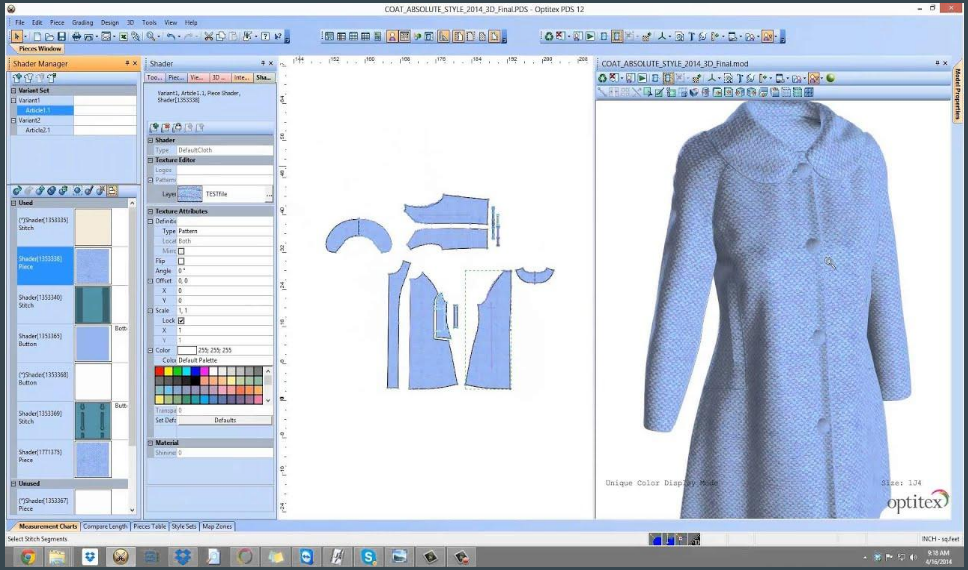
LANGUAGE DRIVEN OUTFIT STYLE ENHANCEMENT

Team 09
Shrey Shrivastava(17CS30034)
Snehal Reddy Koukuntla(17CS30020)
Deepank Agrawal(17CS30011)
Kanishk Singh(17CS30018)

Motivation

- A big challenge in the fashion and graphics industry is for any person to stylise their self-designed clothes.

- A natural way for a layman to describe his idea is through words and hence the best way to take input is natural language in form of textual descriptions.



Formal Statement and Plan

- Let's formally present the problem statement we seek to solve:
    - Given an input image of a fashion item ***I***, and a textual description consisting of a set of key-words, we aim to generate transformed image ***T***, with the style corresponding to the textual description imposed on top of the input image ***I***.

- We divide the problem statement into **three** major parts to ease task formulation:
    - Generate the texture from the textual description input from the user.
    - Segmentation of targeted cloth item from input image.
    - Generated texture is superimposed on the segmented region.

Step 1: Texture generation from input textual description

- We employ StackGAN for the task of texture generation.
- GANs have performed effectively in generating images from textual descriptions, however the synthesized images lack details and vivid object parts.
- To tackle the challenges , the proposed approach decomposes the problem into two more manageable sub-problems.
  - A low resolution image is generated using STAGE-I GAN conditioned on text description.
  - STAGE-II GAN generates realistic high resolution images conditioned on the low resolution image and the corresponding text description.
- Since we only need a patch for our original task , we only deployed the STAGE-I GAN in our algorithm.
- The image output from the first stage is a low resolution image consisting of rough shape and basic colours conditioned on textual description.

Stage-I Generative Adversarial Networks (GANs)

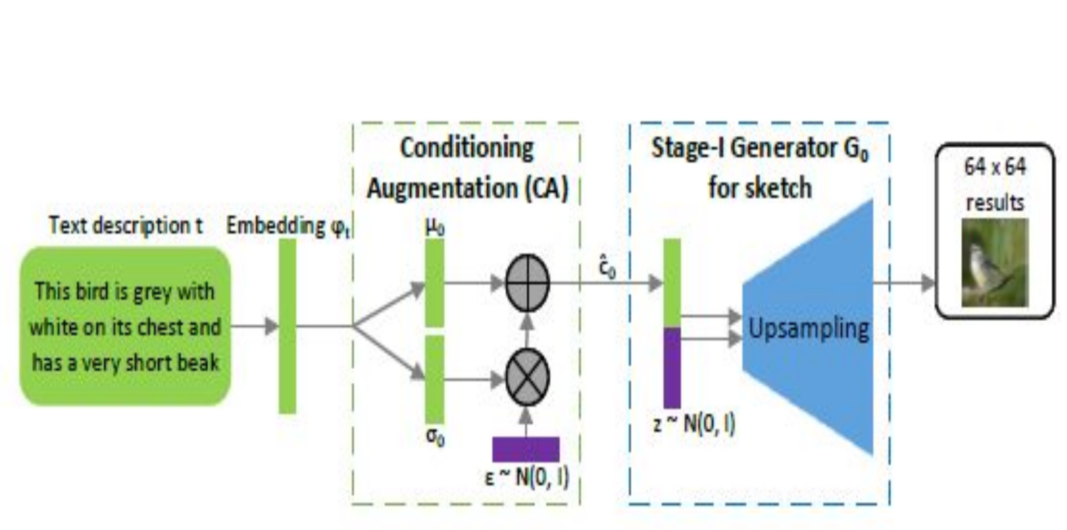
- Composed of two models that are alternatively trained to compete with each other

  - The Generator G
    - Optimized to generate images that are difficult for the discriminator D to differentiate from real images.

  - The Discriminator D
    - Optimized to distinguish real images from the synthetic images generated by G.

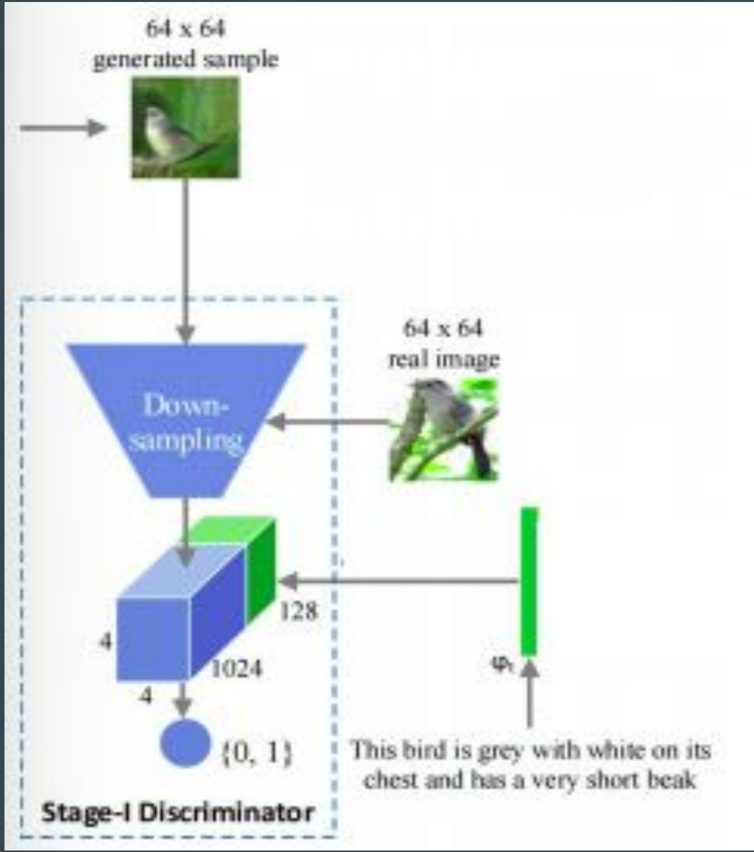
Stage-I Generator

- Text description **t** is encoded to form text embedding.
- Conditional Augmentation is added to improve the diversity of generated images.
- To avoid over-fitting, **KL divergence** between standard and conditioning Gaussian distribution is added during the generator training.



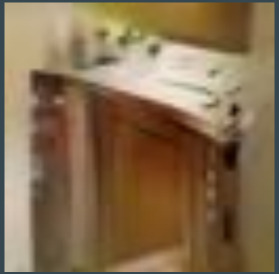
Stage-I Discriminator

- Text embedding is initially compressed and spatially replicated to form tensor of shape [4,4,128].
- The image is also down sampled from [64, 64, 3] to [4, 4 ,1024].
- The resulting concatenated tensor is later fed to [1,1] convolution layer followed by a [4,4] convolution layer.
- The decision of this discriminator is done by a fully connected layer with one node.



Results

- Sample images generated from STACKGAN



"Wooden Door"

"Furniture"

"Cheese Pizza"

"Christmas Tree"

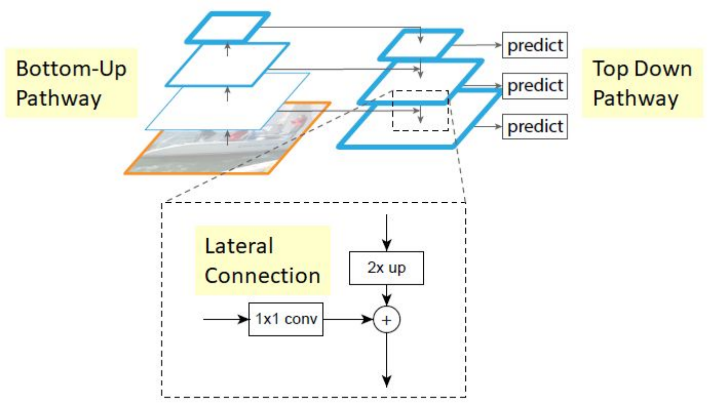
"Beach"

Step 2: Segmentation of cloth piece

- We employ Mask R-CNN for the task of image segmentation.
- Mask R-CNN was initialised with ResNet-50 backbone pre-trained model for segmentation on MS COCO dataset.
- DeepFashion 2 is a comprehensive fashion dataset. It contains 491K diverse images of 13 popular clothing categories from both commercial shopping stores and consumers.
- We used this dataset to further fine-tune the Mask RCNN. This can segment different classes of items separately on the same image like the top, bag, boots, pants etc.
- The resultant segmentation is then passed on to the next part of the pipeline.

Architecture

- Mask RCNN has three major differences:
    - Use of FPN’s
    - Replacement of ROI-Pool with ROI-Align
    - Introduction of additional branch to generate masks.
- The extra mask head allows us to pixel wise segment each object and also extract each object separately without any background
- The Mask RCNN architecture we chose had a Resnet 50 backbone with Feature Pyramid Network.

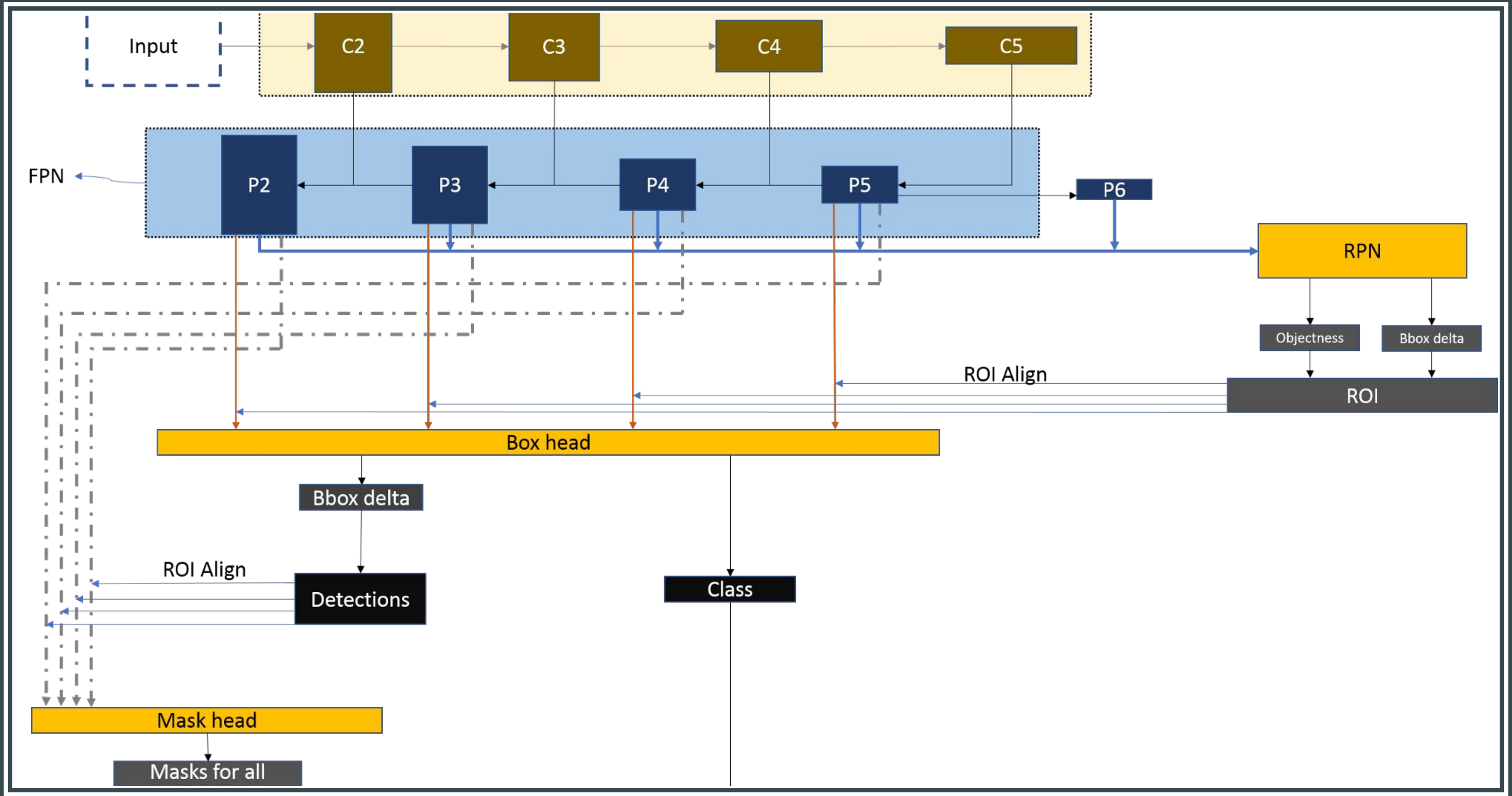



```python
def get_model_instance_segmentation(num_classes):
    # load an instance segmentation model pre-trained pre-trained on COCO
    model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)

    # get number of input features for the classifier
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # replace the pre-trained head with a new one
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # now get the number of input features for the mask classifier
    in_features_mask = model.roi_heads.mask_predictor.conv5_mask.in_channels
    hidden_layer = 256
    # and replace the mask predictor with a new one
    model.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask,
                                                       hidden_layer,
                                                       num_classes)


    return model
```



Input
C2
C3
C4
C5
FPN
P2
P3
P4
P5
P6
RPN
Objectness
Bbox delta
ROI Align
ROI
Box head
Bbox delta
ROI Align
Detections
Class
Mask head
Masks for all

Example: Segmentation

Step 3: Texture application over the segmented image

- After the segmentation of the target cloth piece, we superimpose the texture over the segmented region.

- The size of resulting image is same as the input image.

- The resulting image is passed into a pre-trained TextureGAN model.

Step 3: Continued...

- Since a generalised pre-trained model was available we decided not to train from scratch. We used ‘clothe’ pre-trained model of TextureGAN.
- The pretrained model is trained on DeepFashion Dataset.
- The generated image is combined with the input image to generate final output.

TextureGAN

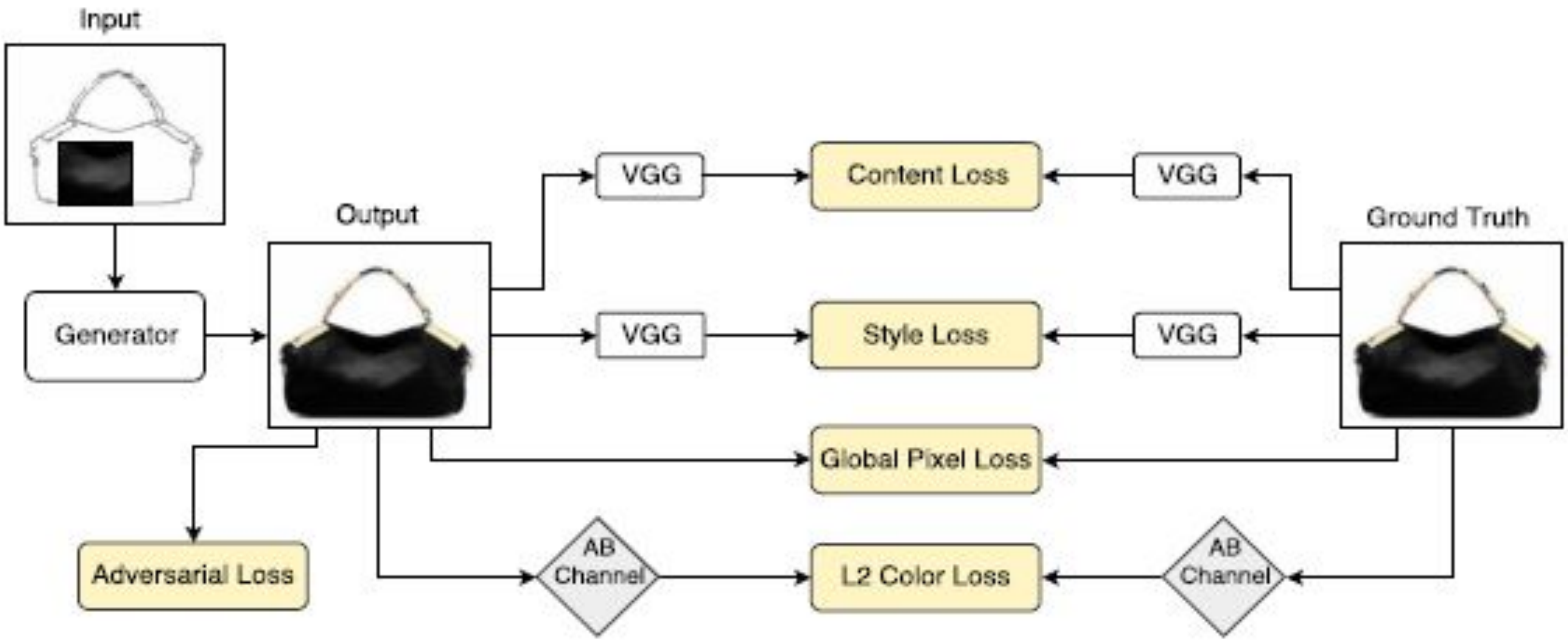
- First deep image synthesis method which allows user to implicitly control object texture.
- The network realistically applies these textures to the target object.
- The model is trained in two stages:
  - Ground-truth Pre-training.
  - External texture Fine-tuning.

Published results from the paper

Ground-truth Pre-training

- The aim of this stage is propagate the texture information in the entire object.
- The input image is converted into LAB color space.
- Various losses are introduced to ensure controlled texture generation:
  - Losses imposed on L channel:
    - Feature loss: To control “leaking” of texture from boundaries.
    - Adversarial loss: Standard GANs loss.
    - Texture loss: To control the variation in generated textures.
      - Style loss: To propagate texture details as per the input texture patch.
      - Pixel loss: Claimed to stabilise the training and generation of faithful textures.
  - Loss imposed on AB channels:
    - Color loss: To enforce user’s color constraints.

$$\mathcal{L} = \mathcal{L}_F + w_{ADV}\mathcal{L}_{ADV} + w_S\mathcal{L}_S + w_P\mathcal{L}_P + w_C\mathcal{L}_C$$



TextureGAN pipeline for the ground-truth pre-training

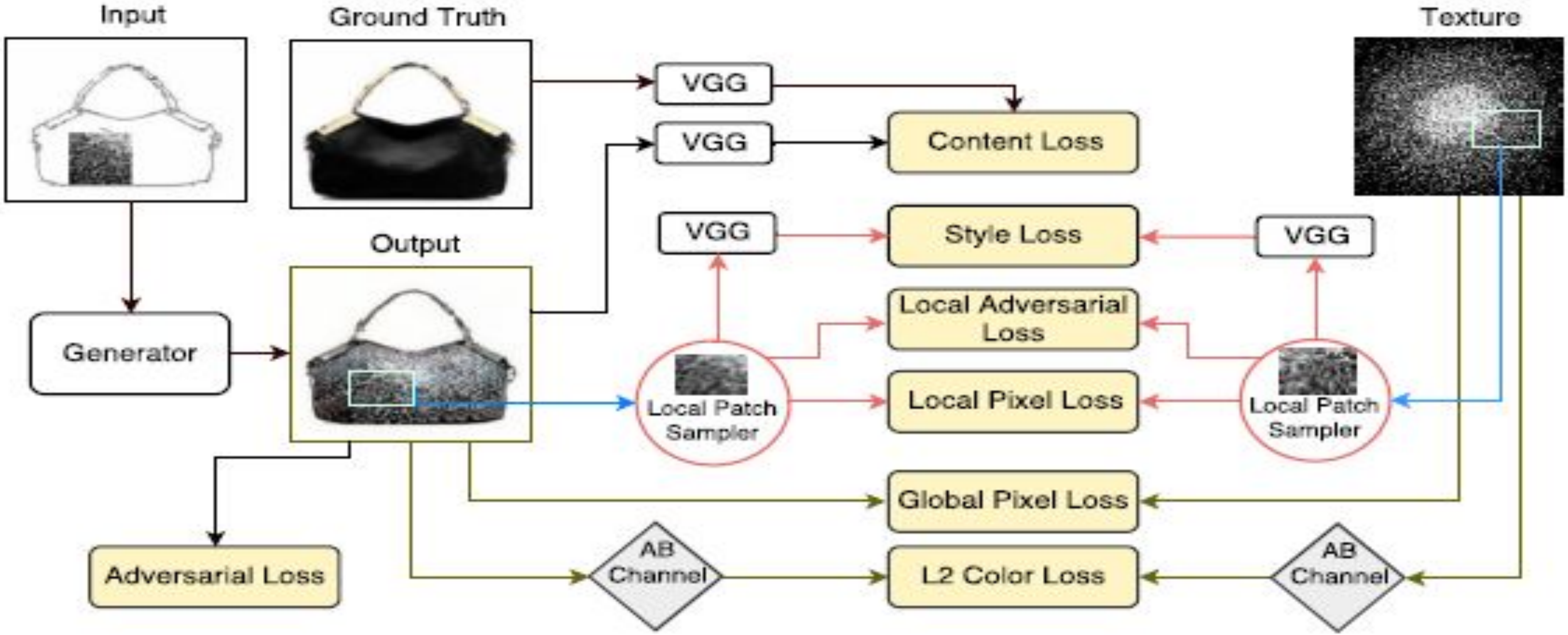
External Texture Fine-tuning

- The network is trained on limited textures in the 1st step. And it has been observed that network performs poorly in propagating rare high level texture details.
- In this step, TextureGAN is generalised to support a broader range of textures and to propagate unseen textures better by fine-tuning the network with a separate texture-only database.
- The main difference is that, in the 1st step, model was “faithful” to the output image while in this step the model is “faithful” to the input textures.

Re-definition of Loss functions

- To ensure “faithfulness” towards the input texture, some loss functions has to be re-defined.
- Feature loss and Adversarial loss remains unchanged.
- Pixel loss and Color loss are changed to compute loss between the texture and generated image.
- For better propagation of texture, Local texture loss is introduced:
  - Local Adversarial loss.
  - Local Style loss.
  - Local Pixel loss.

$$\mathcal{L}_t = \mathcal{L}_s + w_p \mathcal{L}_p + w_{adv} \mathcal{L}_{adv}$$
$$\mathcal{L} = \mathcal{L}_F + w_{ADV} \mathcal{L}_{ADV} + w_P \mathcal{L}'_P + w_C \mathcal{L}'_C + \mathcal{L}_t$$



TextureGAN pipeline for the External Texture Fine-Tuning

Results

Image 1

Input image

The StackGAN output with input “Purple Flower”.

The final output with input “Purple Flower”

Conclusion

- With the presented system, we have successfully combined a wide range of recent research developments to produce a tool where a user can textually describe the details of generated stylised clothing.
- Our results show that this pipeline can handle a wide variety of textual inputs and generate texture compositions that follow the sketched contours satisfactorily.
- Still the proposed methodology is not end-to-end trainable. We have used pre-existing models and cascaded them to generate the output images.
- For future research towards an end-to-end model, our analysis of related research work suggests using extra losses and degenerator modelling in vanilla C-GAN would be necessary for achieving the desired results.

Thank You