



IS470 Guided Research in Computing Identity Obfuscation in Person Images

Agenda...



01 Introduction



04 Experiments



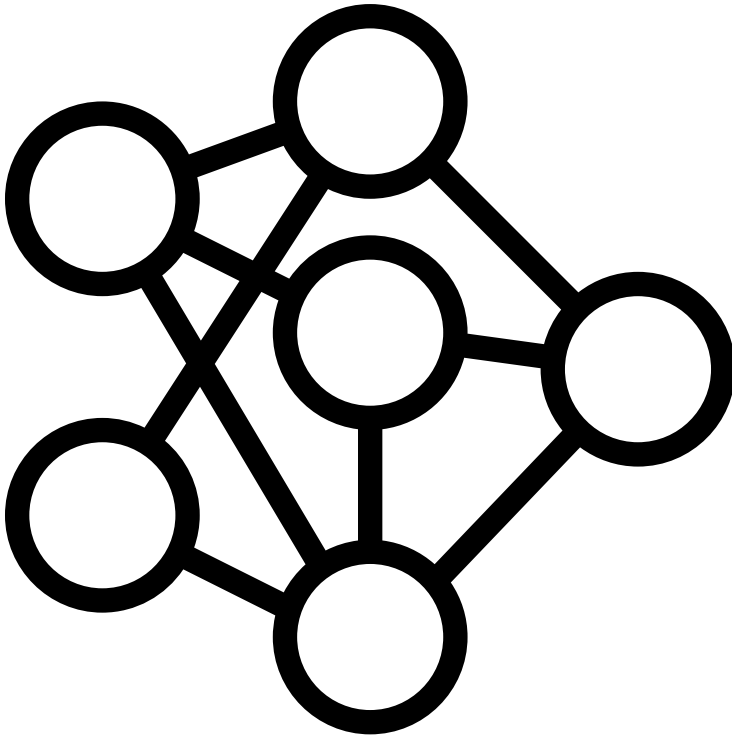
02 Literature Review



05 Conclusion



03 Methodology



Introduction



Problem Statement

- Social media usage on upwards trend
- Privacy prone to being compromised
- Ethical norms are challenged
- Face images contain private information
- Current identity obfuscation methods are insufficient
- Automated/machine face image obfuscation



The Secretive Company That Might End Privacy as We Know It

A little-known start-up helps law enforcement match photos of unknown people to their online images — and “might lead to a dystopian future or something,” a backer says.



Objectives & Goals

01

Explore

New methods of
obfuscation in person
images

02

Improve

Quality of face image
generation

03

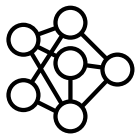
Word
embeddings

The impact of word
embeddings on conditional
image generation

04

Conditional
Generation

The impact of face attribute
manipulation on face image
generation



Overview

3 stage image generation framework

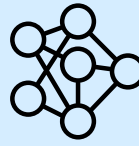
Stage 1



Disentangling person images into three factors:

1. Face landmarks
2. Face attributes
3. Image context

Stage 2

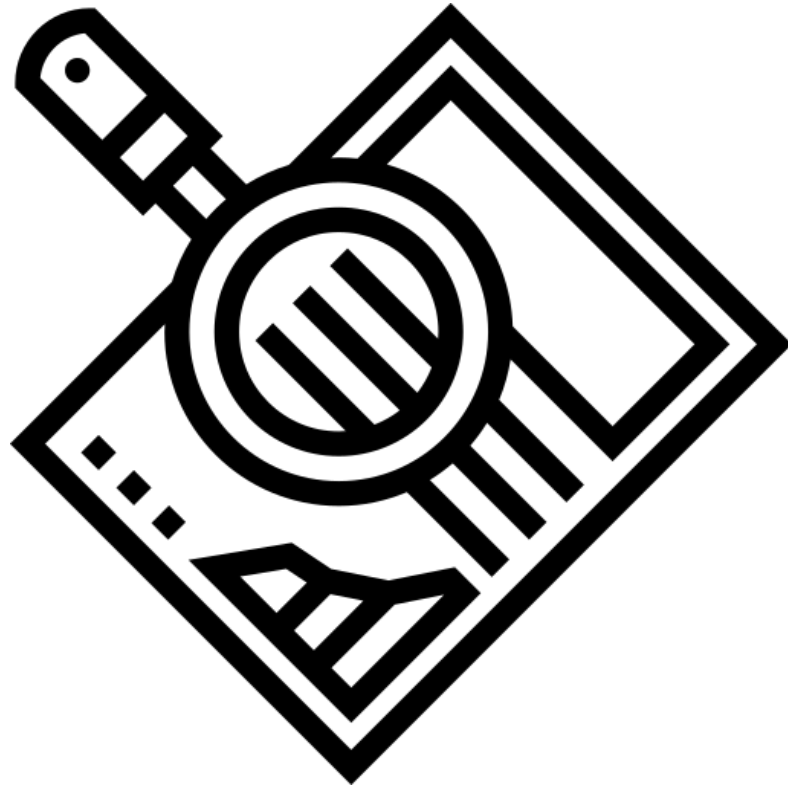


Train auto-encoder to reconstruct original face image from the three disentangled factors

Stage 3




Generate new face images by face attribute manipulation for purpose of identity obfuscation



Literature Review

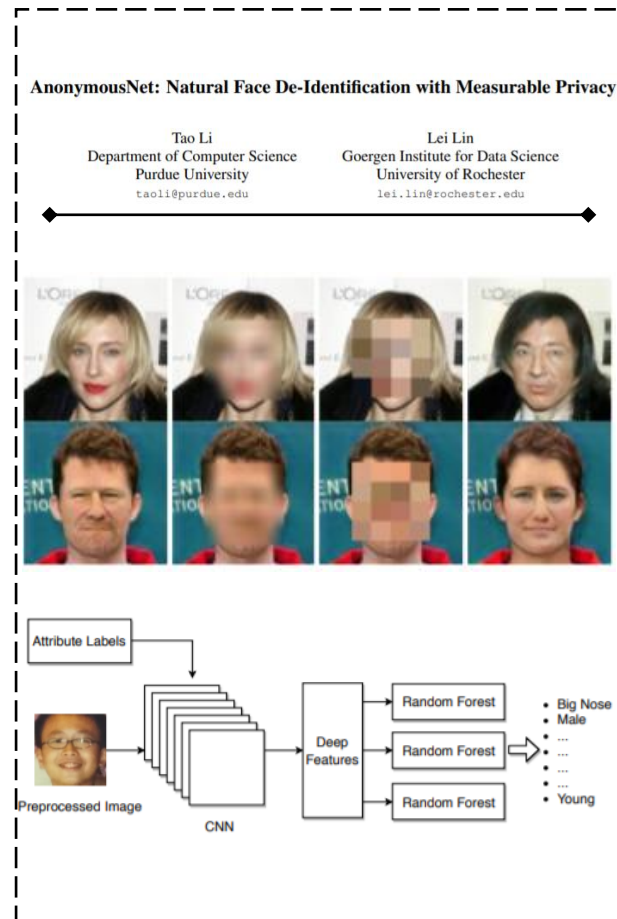


- # Disentangled Person Image Generation
- Liqian Ma¹ Qianru Sun^{2*} Stamatis Georgoulis¹
Luc Van Gool^{1,3} Bernt Schiele² Mario Fritz²
-
- 
- Foreground (FG) sampling (fixed BG and Pose)
- Background (BG) sampling (fixed FG and Pose)
- Pose sampling (fixed FG and BG)
- FG, BG and Pose sampling
- Appearance and Pose sampling



Conditional Image Generation on Face Attributes

- Higher level understanding of face image obfuscation
- Privacy-preserving attribute selection (PPAS) algorithm
- Do not consider relationships between attributes
- Each attribute is classified with a single random forest classifier



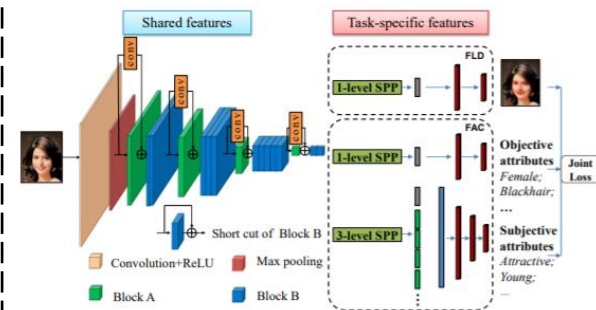


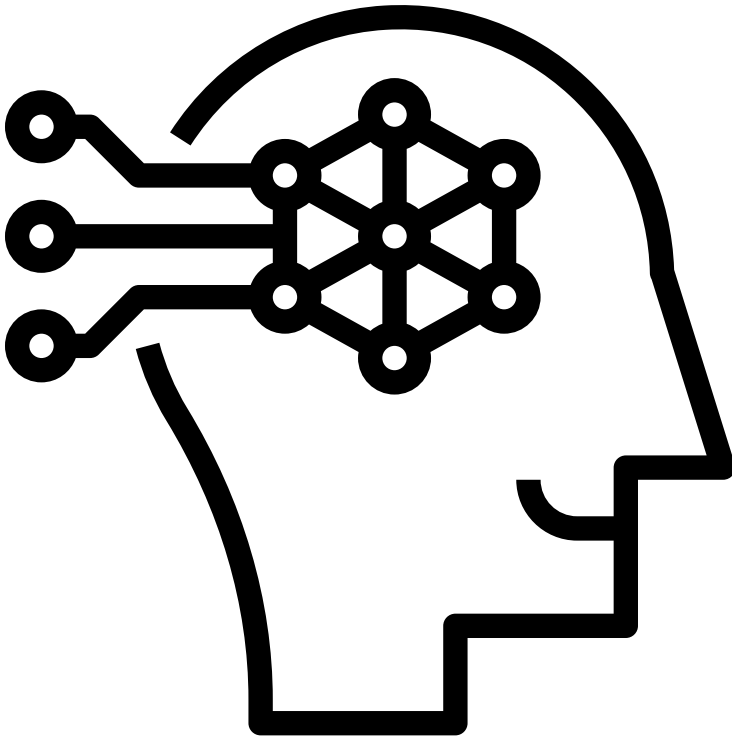
Multi-label Face Attribute Classification

- Novel method for face attribute classification task
- Split learning into two stages, shared and task specific
- Grouped attributes into 2 classes

Deep Multi-task Multi-label CNN for Effective Facial Attribute Classification

Longbiao Mao, Yan Yan, *Member, IEEE*, Jing-Hao Xue, and Hanzi Wang, *Senior Member, IEEE*





Methodology



Stage 1

Image disentanglement

Landmark Detection

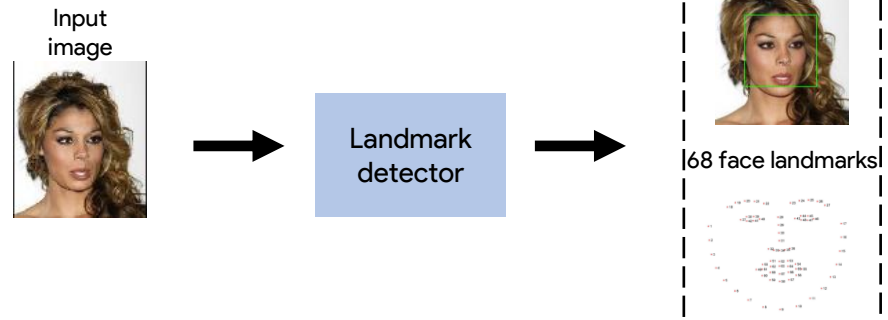
- Pre-trained detector provided by dlib
- Detect face bounding box
- Detect 68 key face landmarks

Face Masking

- *ellipse()* method
- Specific landmarks

Face Attribute Classification

- ResNet50 architecture
- Remove final layer, add flatten, add FC dense
- Sigmoid activation
- Adam Solver
- Binary-cross entropy loss





Stage 1

Image disentanglement

Landmark Detection

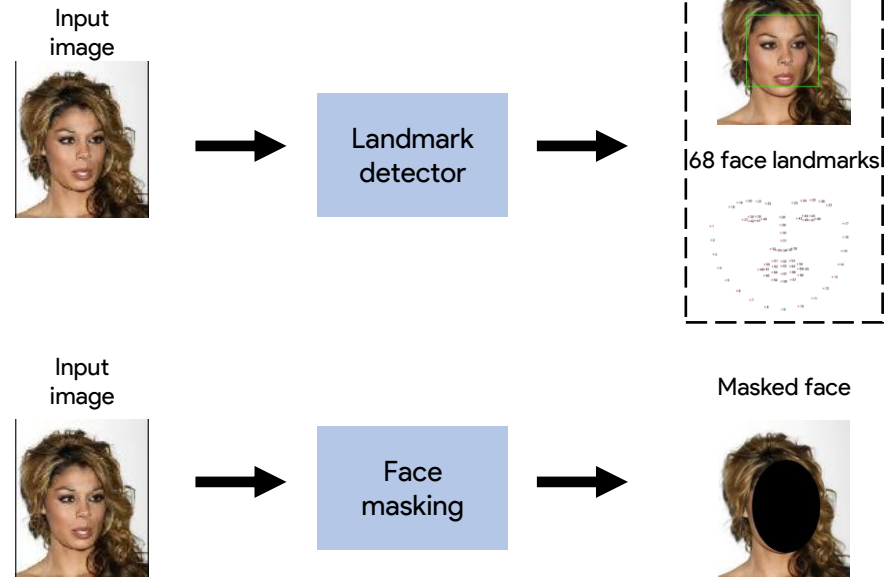
- Pre-trained detector provided by dlib
- Detect face bounding box
- Detect 68 key face landmarks

Face Masking

- *ellipse()* method
- Specific landmarks

Face Attribute Classification

- ResNet50 architecture
- Remove final layer, add flatten, add FC dense
- Sigmoid activation
- Adam Solver
- Binary-cross entropy loss





Stage 1

Image disentanglement

Landmark Detection

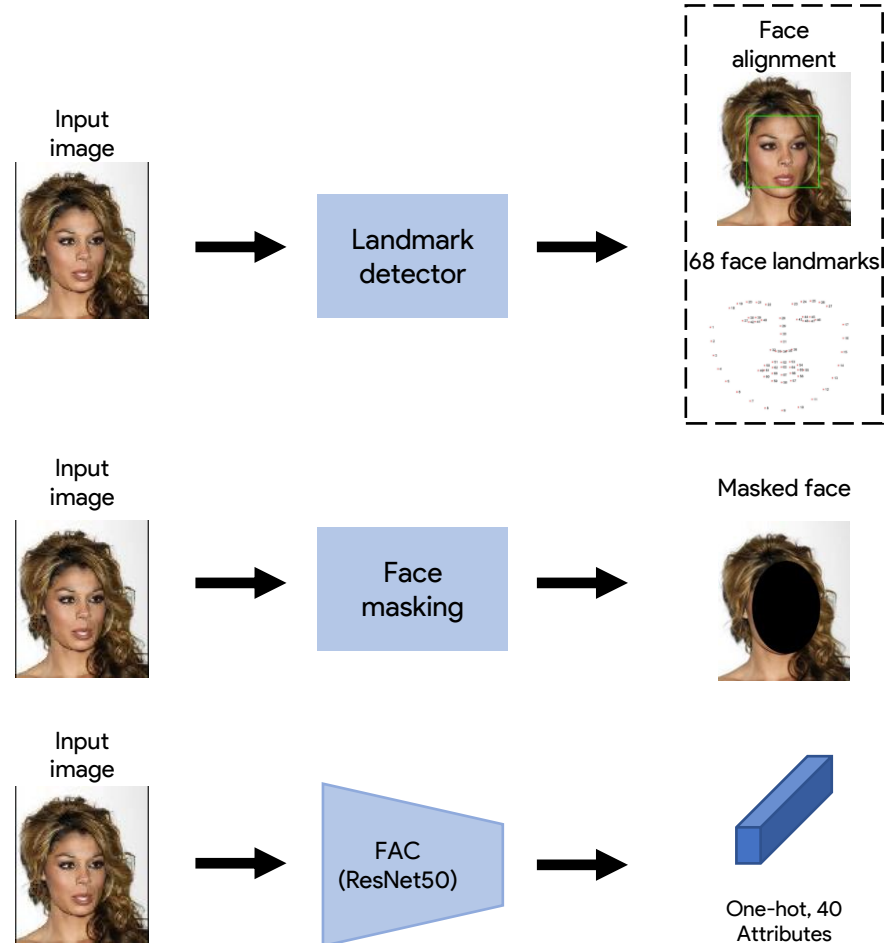
- Pre-trained detector provided by dlib
- Detect face bounding box
- Detect 68 key face landmarks

Face Masking

- *ellipse()* method
- Specific landmarks

Face Attribute Classification

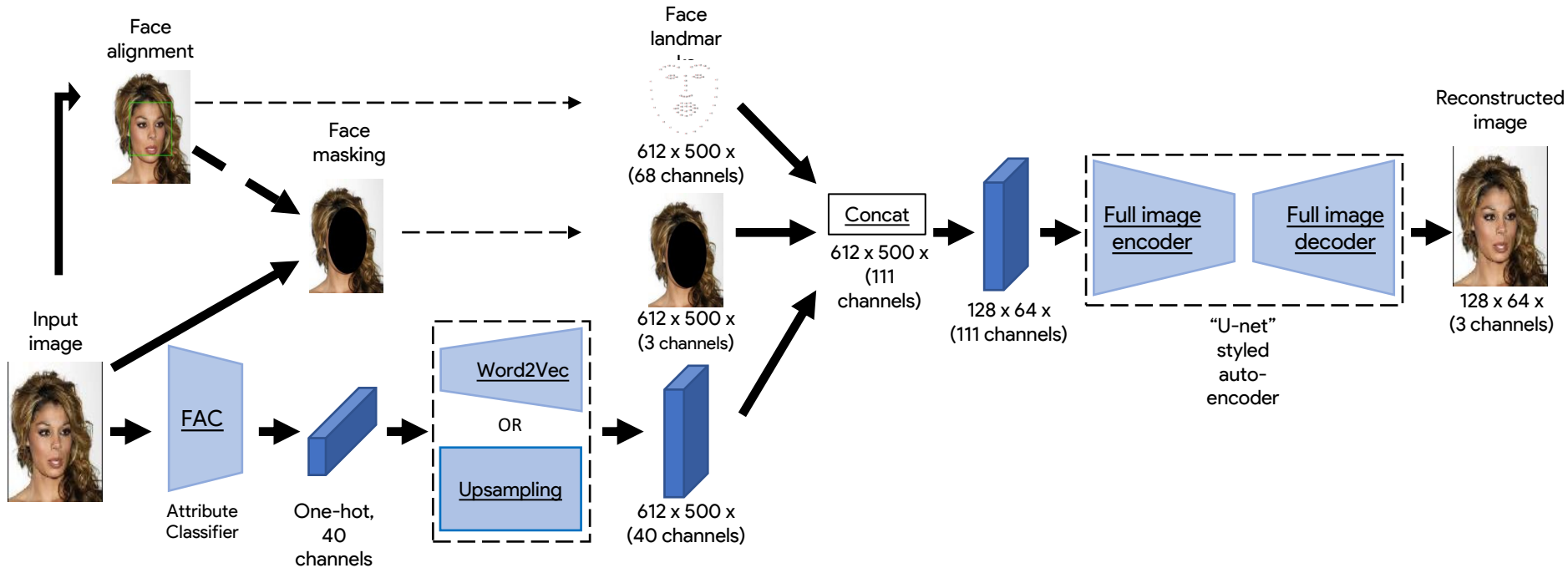
- ResNet50 architecture
- Remove final layer, add flatten, add FC dense
- Sigmoid activation
- Adam Solver
- Binary-cross entropy loss





Stage 2

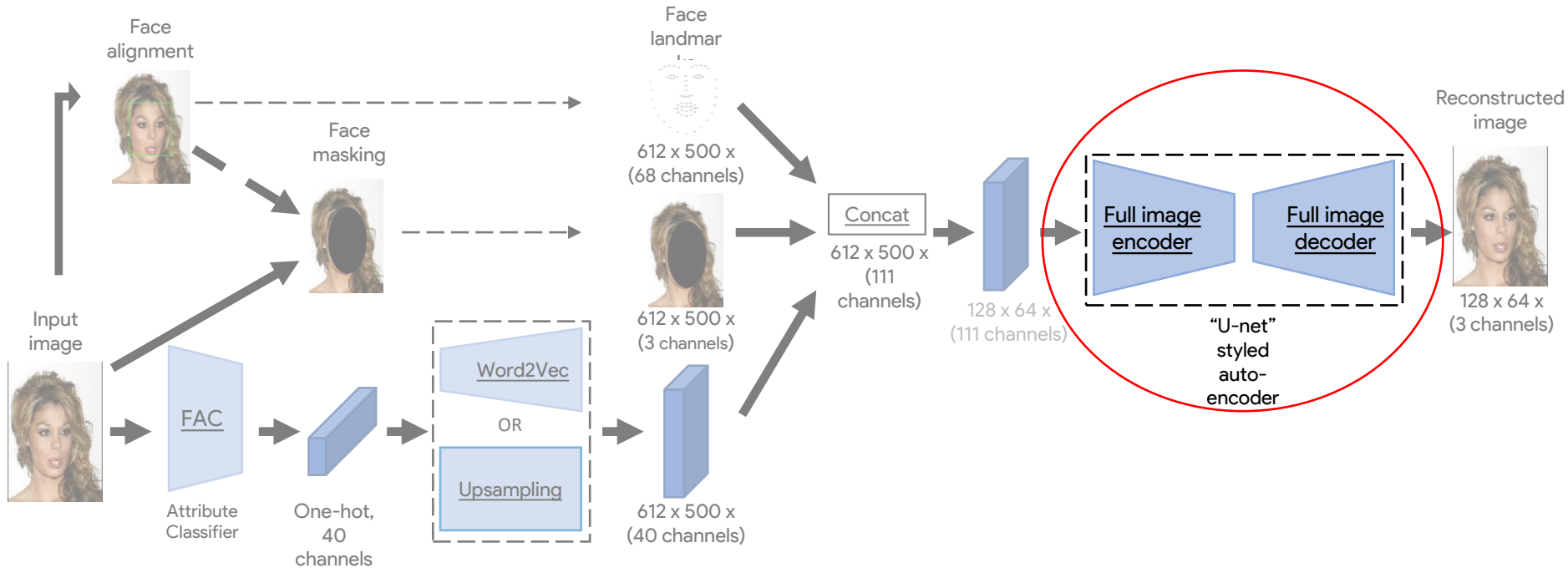
Reconstruction



- Self-supervised auto-encoder 'U-Net' based architecture
- Concatenate all outputs from Stage 1
- Word embeddings to increase dimensionality
- SSIM loss



Stage 2 Reconstruction

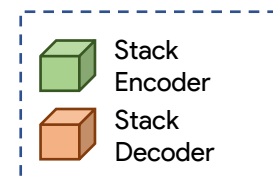
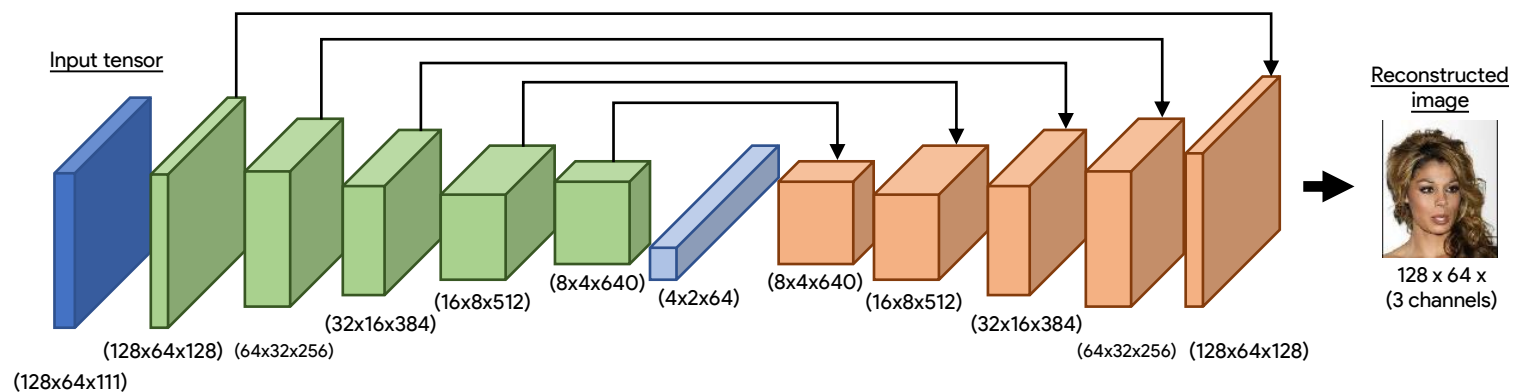
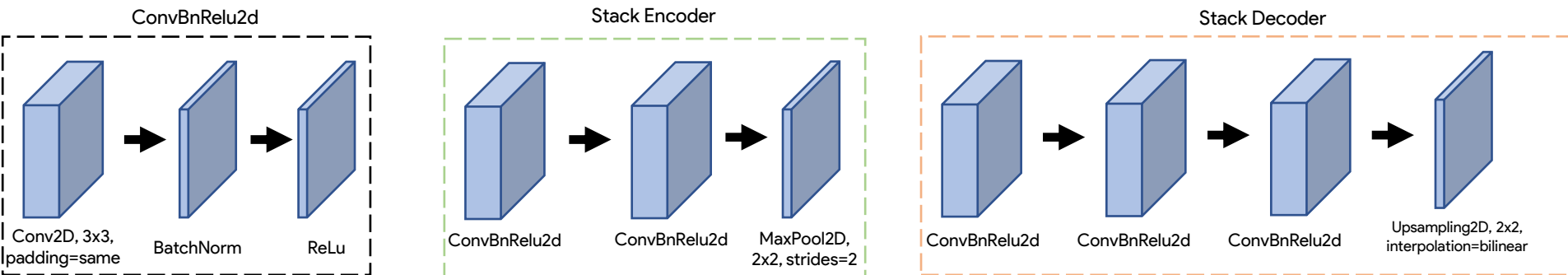


- Self-supervised auto-encoder ‘U-Net’ based architecture
- Concatenate all outputs from Stage 1
- Word embeddings to increase dimensionality
- SSIM loss



Stage 2

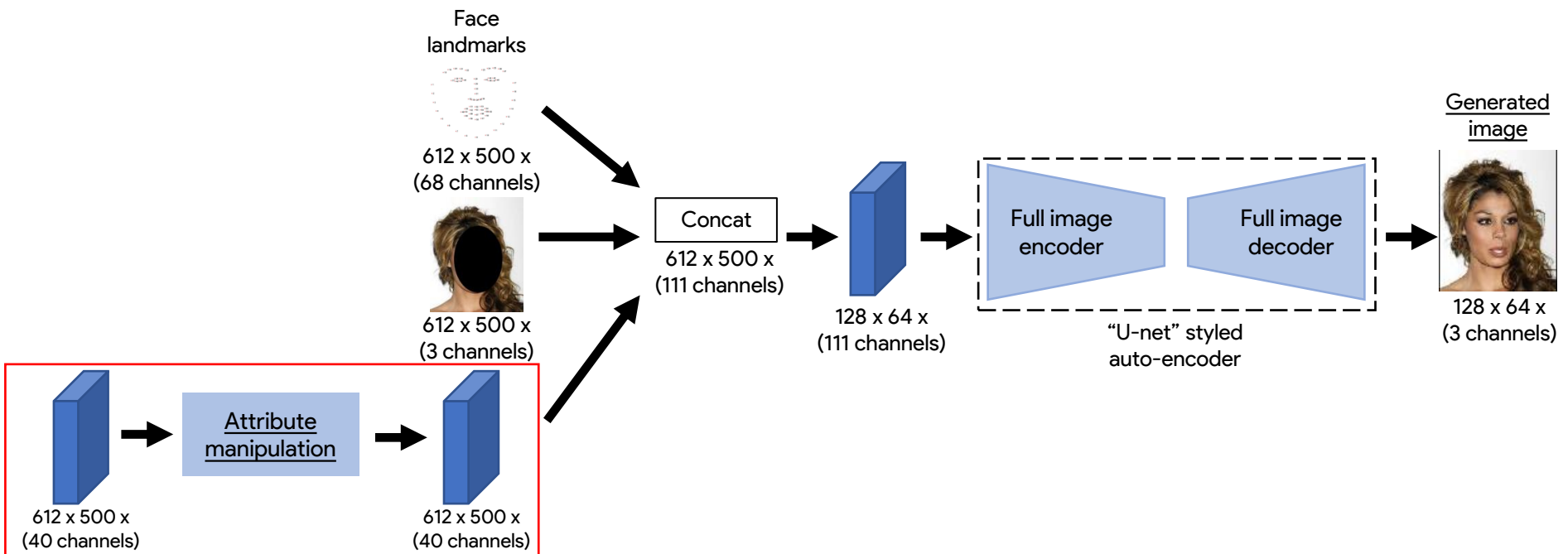
Recon-Net

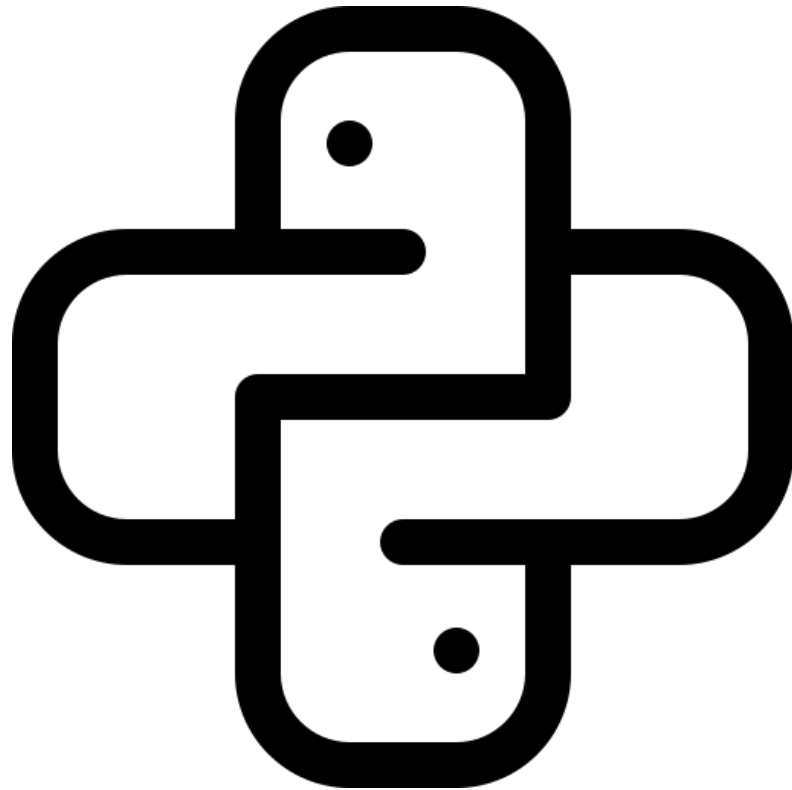




Stage 3

Generation





Experiments

Experiment Setup

CelebA Dataset

- 202,599 face images (218,178,3)
- 10,177 identities
- 40 binary attributes
- Train, val, test partitions
- Subset=0.3

Tools

- Python
- cv2, keras, tensorflow, dlib
- Jupyter notebook
- GeForce RTX 2080ti



Stage 1

Landmark Detection

Pre-processing steps

- Take subset of original data splits
- Train = 48831, Val = 5960, Test = 5988
- Resize image to (612,500)

Face alignment & Landmark detection

- Filter undetected faces in images
- Train = 47345, Val = 5787, Test = 5840
- Outputs (68,(x,y))

Algorithm 1: Face & landmark detection

```
Result: face landmark dictionary: imageID:(68,x,y)
initialization set dictionary;
for set in image sets do
    for image in set do
        resize image to (612,500);
        detect face in image;
        if face detected then
            predict 68 face landmarks;
            update landmark dictionary
        else
            continue to next image
        end
    end
end
```

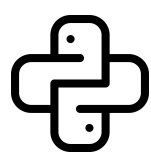
Stage 1

Face Masking

Face masking

- Face masking determined via key points from 68 landmarks
- `cv2.ellipse()` method used to create customized face mask
- Required arguments, {
center, axesLengthwidth,
axesLengthheight,
axesLength
}
- Output is array of shape (612,500,3)





Stage 1

Face Attribute Classification (FAC)

Pre-processing steps

- Normalize all data images such that pixel intensity is between 0,1
- One hot encoding for 40 face attributes

FAC

- Supervised learning, model architecture is raw resnet50
- Activation for FC is sigmoid
- Adam solver
- Binary cross entropy loss
- Custom metric: Average accuracy
- Batch size = 64
- Epochs = 15
- Output is one hot encoded array (1,40)

$$L_{FAC}^j = -\sum_{i=1}^2 t_i \log(s_i) = -t_1 \log(s_1) - (1 - t_1) \log(1 - s_1)$$
$$L_{FAC} = \sum_{j=1}^{40} L_{FAC}^j$$

Table 1: Face attribute classification results

Setting	Average Test Accuracy	Runtime
adam	89.07%	53 mins
adam with lr=0.0001	87.90%	50 mins

Stage 2 Concatenation

Pre-processing steps

- Convert initial landmark shape of (68,2) to (612,500,68)
- Upsample face attributes from (1,40) to (612,500,40)
- Resize concat array to (128, 64, 111)

DataGenerator

- Concat array train batch of size (47345, 128, 64,111) will cause out of memory(OOM) issue when loading data to disk
- Instance DataGenerator class to yield training data in batches of 16 (pre-processing is done on the fly)

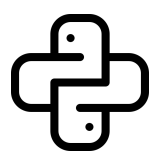
```
1 class DataGenerator(tf.keras.utils.Sequence):
2     'Generates data for keras'
3     def __init__(self, dataset, image_name_ls, face_attr, landmark_dict, \
4         batch_size=32,
5         dim=(32,32,32), n_channels=1, shuffle=True):
6
7         'Initialization'
8         self.dim = dim
9         self.batch_size = batch_size
10        self.image_name_ls = image_name_ls
11        self.dataset = dataset
12        self.n_channels = n_channels
13        self.shuffle = shuffle
14        self.face_attr = face_attr
15        self.landmark_dict = landmark_dict
16        self.on_epoch_end()
17
18    def __len__(self):
19        'Denotes the number of batches per epoch'
20        return int(np.floor(len(self.image_name_ls) / self.batch_size))
21
22    def __getitem__(self, index):
23        'Generate one batch of data'
24        # Generate indexes of the batch
25        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
26
27        # Find list of IDs
28        list_IDs_temp = [self.image_name_ls[k] for k in indexes]
29        # Generate data
30        X, y = self.__data_generation(list_IDs_temp, indexes)
31
32        return X, y
33
34    def on_epoch_end(self):
35        'Updates indexes after each epoch'
36        self.indexes = np.arange(len(self.image_name_ls))
37        # create array of indexes
38        if self.shuffle == True:
39            np.random.shuffle(self.indexes)
40
41    def __data_generation(self, list_IDs_temp, indexes):
42        '''
43        private method
44
45        list_IDs_temp contains name of the images in this batch
46
47        will resize input images to (128x64)
48        '''
49        'Generates data containing batch_size samples'
50        # X : (n_samples, 'dim', n_channels)
51        # Initialization
52        X = np.empty((self.batch_size, *self.dim, self.n_channels))
53        y = np.empty((self.batch_size, *self.dim, 1), dtype='float64')
54        # n_channels
55        new_dim = (64,128)
56        # Generate data
57        for i, ID in enumerate(list_IDs_temp):
58            # Store sample
59            train_image_mask_arr = np.asarray(Image.open(os.path.join(
60                self.dataset, ID)))
61            X[i, :] = cv2.resize(self.__concat_all_channels(self.face_attr, \
62                self.landmark_dict,
63                train_image_mask_arr, ID), new_dim)
64
65            # Store orig image
66            train_images_label = np.asarray(Image.open(os.path.join(
67                self.dataset, ID)))
68            y[i] = cv2.resize(train_images_label, new_dim)
69            # will return the original images
70        return X, y
71
72    def __concat_all_channels(self, face_attr, landmarks, mask, image_id):
73        '''
74        for the case where face attr is one hot encoded only (baseline model)
75        returns:
76        final_arr: array of shape (612, 500, 111)
77        '''
78        face_attr_temp = face_attr[face_attr['image_id'] == image_id].\
79            drop(columns='image_id').to_numpy().reshape((40,))
80        face_attr_new = np.zeros((612,500,40))
81        face_attr_new[:, :, face_attr_temp] = 1
82        # if attribute is present, change to np.ones
83        new_array = np.concatenate((mask/255., landmarks[image_id].\
84            face_attr_new), axis=2)
85        return new_array
```

Stage 2

Recon Net

- Model architecture is adapted from (Ma, 2018) and from (Olaf, 2015)
- Consist of conv blocks and skip connections
- No cropping operation needed as padding = “same” for all layers
- Total of 45,121,667 params
- Adam solver
- SSIM loss
- 15 epochs

```
1 # Batchnorm epsilon
2 BN_EPS = 1e-4
3
4 tf.keras.backend.set_floatx('float32')
5
6 class ConvBlock(layers.Layer):
7     # Convolutional -> Batch norm -> ReLU
8
9     def __init__(self, out_channels, kernel_size=(3, 3), padding='same'):
10         super(ConvBlock, self).__init__()
11         self.conv = layers.Conv2D(filters=out_channels,
12                                   kernel_size=kernel_size, padding='same',
13                                   use_bias=False)
14         self.bn = layers.BatchNormalization(epsilon=BN_EPS)
15         self.relu = layers.ReLU()
16
17     def call(self, x):
18         x = self.conv(x)
19         x = self.bn(x)
20         x = self.relu(x)
21         return x
22
23 class StackDecoder(layers.Layer):
24     # ConvBlock -> ConvBlock -> * -> max pool (x2)
25     # * store tensor for concatenation with expansive path
26
27     def __init__(self, y_channels, kernel_size, dilation_rate=1):
28         super(StackDecoder, self).__init__()
29         self.encode = keras.Sequential([
30             ConvBlock(y_channels, kernel_size=kernel_size, padding='same'),
31             ConvBlock(y_channels, kernel_size=kernel_size, padding='same')
32         ])
33         self.max_pool = layers.MaxPool2D(pool_size=2, strides=2)
34
35     def call(self, x):
36         x = self.encode(x)
37         x_small = self.max_pool(x)
38         return x, x_small
39
40 class StackEncoder(layers.Layer):
41     # Upsample (x) -> concatenation -> ConvBlock -> ConvBlock -> ConvBlock
42
43     def __init__(self, y_channels, kernel_size):
44         super(StackEncoder, self).__init__()
45         self.decode = keras.Sequential([
46             ConvBlock(y_channels, kernel_size=kernel_size, padding='same'),
47             ConvBlock(y_channels, kernel_size=kernel_size, padding='same'),
48             ConvBlock(y_channels, kernel_size=kernel_size, padding='same')
49         ])
50
51     def call(self, x, down_tensor):
52         x = layers.UpSampling2D(size=(2, 2), interpolation='bilinear')(x)
53         x = layers.concatenate([x, down_tensor], axis=3)
54         x = self.decode(x)
55         return x
56
57 class Net_2d(keras.Model):
58     def __init__(self):
59         super(Net_2d, self).__init__()
60
61         self.down1 = StackEncoder(128, kernel_size=3)
62         self.down2 = StackEncoder(256, kernel_size=3)
63         self.down3 = StackEncoder(512, kernel_size=3)
64         self.down4 = StackEncoder(1024, kernel_size=3)
65         self.down5 = StackEncoder(2048, kernel_size=3)
66
67         self.center = ConvBlock(64, kernel_size=3, padding='same')
68
69         self.up1 = StackDecoder(640, kernel_size=3)
70         self.up2 = StackDecoder(320, kernel_size=3)
71         self.up3 = StackDecoder(160, kernel_size=3)
72         self.up4 = StackDecoder(80, kernel_size=3)
73
74         # Final prediction uses three channels
75         self.classify = layers.Conv2D(filters=1, kernel_size=1, use_bias=True)
76
77         # If filter more than one will raise error at tf.squeeze
78
79     def call(self, x):
80         out = x
81         down1_tensor, out = self.down1(out)
82         down2_tensor, out = self.down2(out)
83         down3_tensor, out = self.down3(out)
84         down4_tensor, out = self.down4(out)
85         down5_tensor, out = self.down5(out)
86
87         out = self.center(out)
88
89         out = self.up1(out, down5_tensor)
90         out = self.up2(out, down4_tensor)
91         out = self.up3(out, down3_tensor)
92         out = self.up4(out, down2_tensor)
93         out = self.classify(out)
94         out = keras.activations.sigmoid(out)
95
96         return out
```



Stage 2

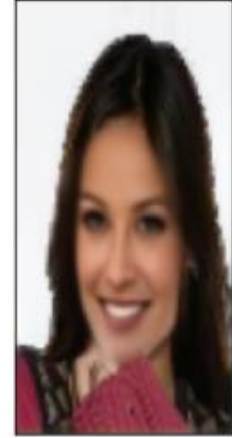
Recon Net



Epoch 3

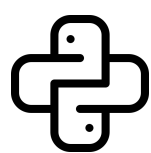


Epoch 10



Epoch 15





Stage 2

Recon Net



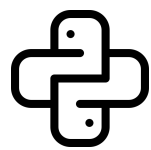
Epoch 3



Epoch 10



Epoch 15

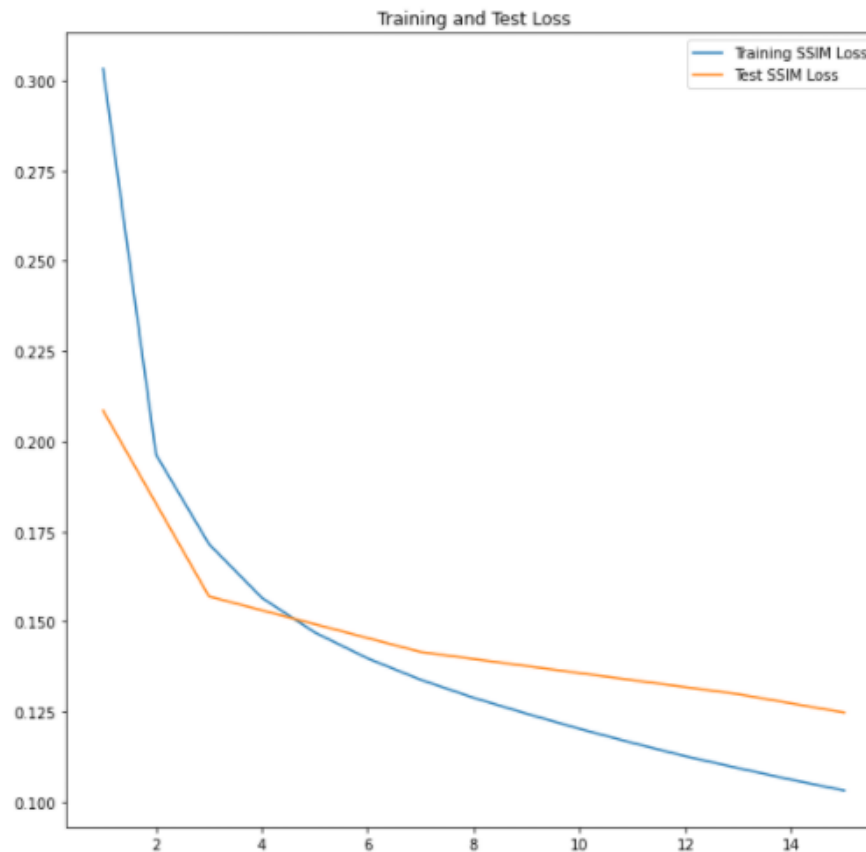


Stage 2

Recon Net Test Results

Table 2: Face reconstruction results

Setting	Test SSIM Score	Runtime
adam ($lr=2e-5$, $\beta_1 = 0.5$, $\beta_2 = 0.999$)	0.8752	30 hrs

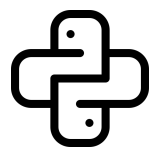


Face attribute manipulation

- Attribute manipulation must be reasonable
- Manipulated attributes hypothesized to have strong obfuscation properties
- Manipulated
'pointy_nose',
'smiling', 'young',
'big_nose'

Algorithm 2: Face Attribute Manipulation

Result: face attributes: `face_attributes_array`: (612,500,40)
initialization set `array_mask` # index which we want to manipulate;
for `face_attributes_array` in batch **do**
 for `i` in `array_mask` **do**
 if `face_attributes_array[:,i] == 1` **then**
 `face_attributes_array[:,i] = 0`; # invert
 else
 `face_attributes_array[:,i] = 1`; # invert
 end
 end
end



Stage 3

Image Triplets



ground_truth



reconstructed



obfuscated

Stage 3

Image Triplets



ground_truth



reconstructed



obfuscated

Stage 3

Image Triplets



ground_truth



reconstructed



obfuscated

Evaluation

- Train a person-recognizer using a subset of 20 identities
- Alexnet architecture extended with 6 dense layers
- Adam optimizer
- Categorical cross-entropy loss
- Test against original images and the obfuscated images
- Trade-off between obfuscation and human perception of images

Table 3: Evaluation of obfuscation

Image settings	Test Accuracy
original	0.6000
obfuscated(all)	0.3000
obfuscated('pointy_nose', 'smiling', 'young' and 'big_nose')	0.3667



Original



Obfuscated



Obfuscated
all attributes

Timeline

TASK	START	END	17-Aug-20	22-Aug-20	27-Aug-20	1-Sep-20	6-Sep-20	12-Sep-20	16-Sep-20	21-Sep-20	26-Sep-20	1-Oct-20	6-Oct-20	11-Oct-20	16-Oct-20	21-Oct-20	26-Oct-20	31-Oct-20	5-Nov-20	10-Nov-20	15-Nov-20	20-Nov-20	27-Nov-20
Proposal Phase	17-Aug-20	12-Sep-20																					
Literature Review	17-Aug-20	12-Sep-20																					
First Draft	21-Aug-20	28-Aug-20																					
Second Draft	28-Aug-20	12-Sep-20																					
Proposal Submission	12-Sep-20	12-Sep-20																					
Mid Term Phase	12-Sep-20	11-Oct-20																					
Extended Literature Review	12-Sep-20	25-Sep-20																					
Architecture of first stage	12-Sep-20	25-Sep-20																					
Training of first stage	25-Sep-20	11-Oct-20																					
Architecture of second stage model	25-Sep-20	11-Oct-20																					
Presentation Phase	11-Oct-20	20-Nov-20																					
Training of second stage model	11-Oct-20	18-Nov-20																					
Testing of second stage model	11-Oct-20	18-Nov-20																					
Testing third stage model	18-Nov-20	20-Nov-20																					
Evaluation	18-Nov-20	27-Nov-20																					
Slide Deck	18-Nov-20	20-Nov-20																					
Term Paper Phase	11-Oct-20	27-Nov-20																					
Refine term paper	11-Oct-20	27-Nov-20																					

Introduction

Literature Review

Methodology

Experiments

Conclusion

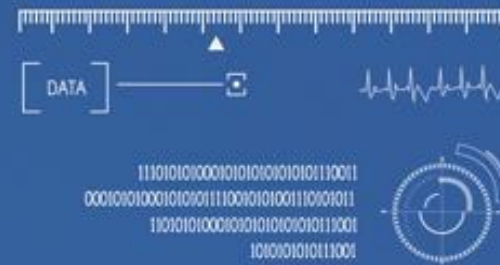


Conclusion



Future Work

1. Try using another loss function for Recon-Net to compare with the baseline model
2. Image augmentation when training in stage 1 and stage 2
3. Adversarial training instead of using auto-encoders
4. Transfer learning on a different dataset, such as labelled faces in the wild (LFW)
5. Pruning network in stage 2 to speed up training time



NO: ONE PERSON
GENDER: MAN
AGE GROUP: YOUNG MAN
ETHNICITY: CAUCASIAN
HUMAN BODY PART: HUMAN FACE
TIME: 167 S
DETECTION: 63621 POINTS
POS (X/Y/Z): 1322 / 856 / 21

thank you !

