IS470 Guided Research in Computing
Identity Obfuscation in Person Images

Agenda...

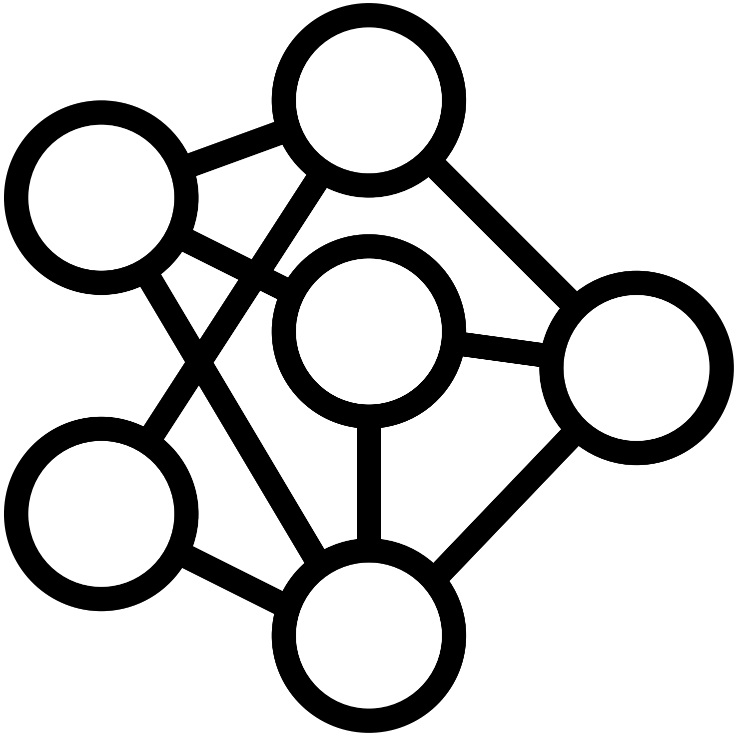
01 Introduction

02 Literature Review

03 Methodology

04 Experiments

05 Conclusion



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:

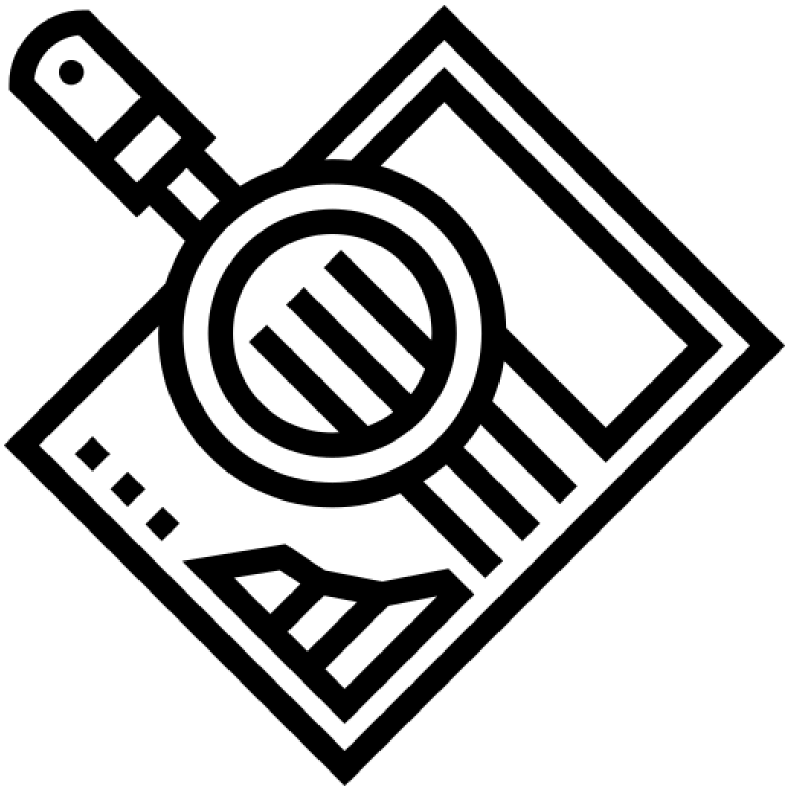
1. Face landmarks
2. Face attributes
3. Image context

Stage 2

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

Stage 3

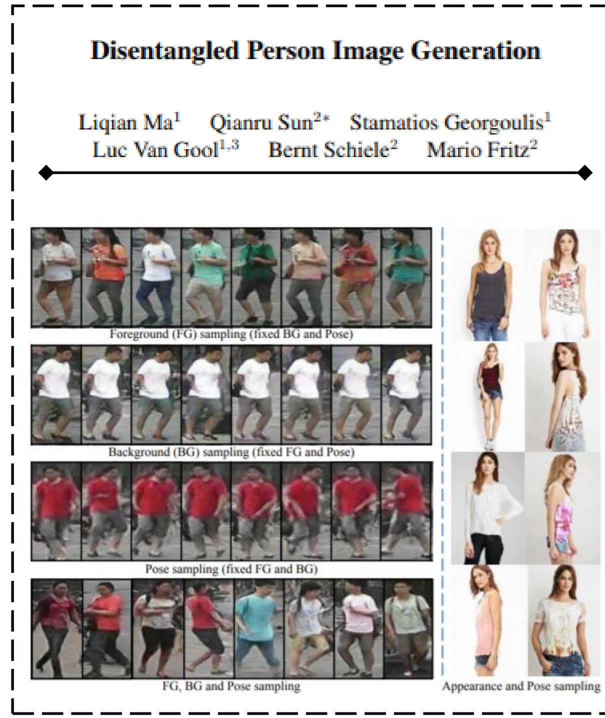
Generate new face images by face attribute manipulation for purpose of identity obfuscation



Literature Review

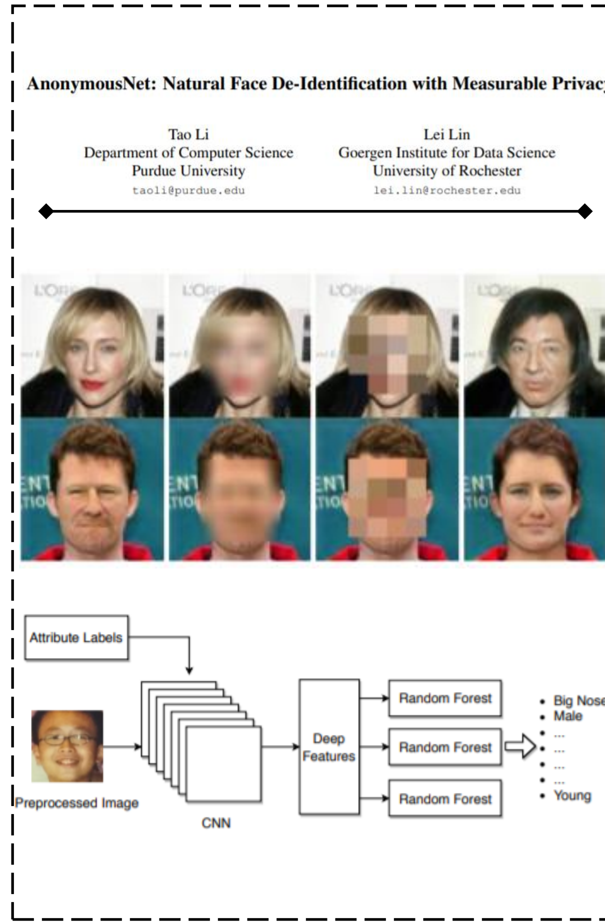
Semi-Supervised Image Reconstruction

- Novel technique of person image generation via sampling disentangled factors of a person image
- Entirely self-supervised pipeline
- Manipulate all three factors for purpose of person re-identification task



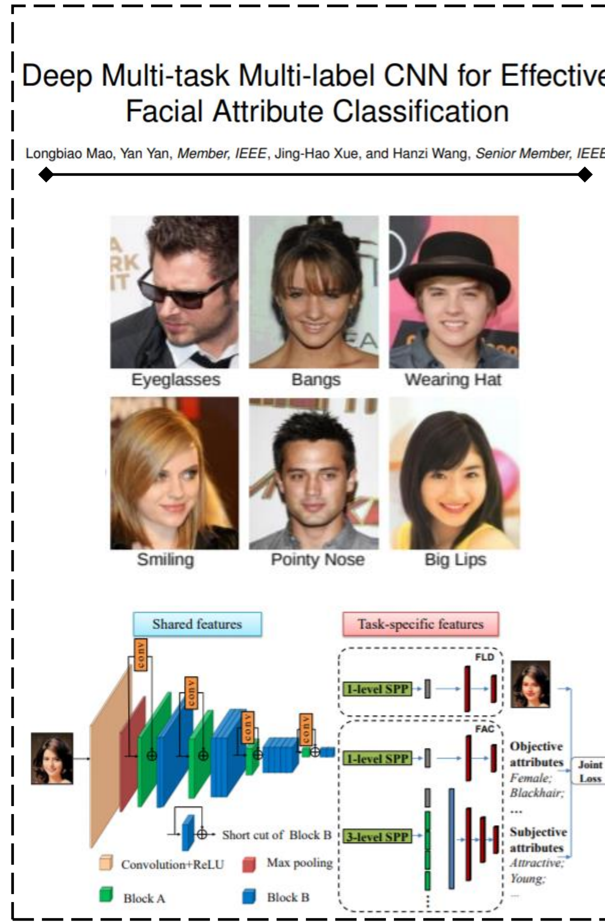
Conditional Image Generation on Face Attributes

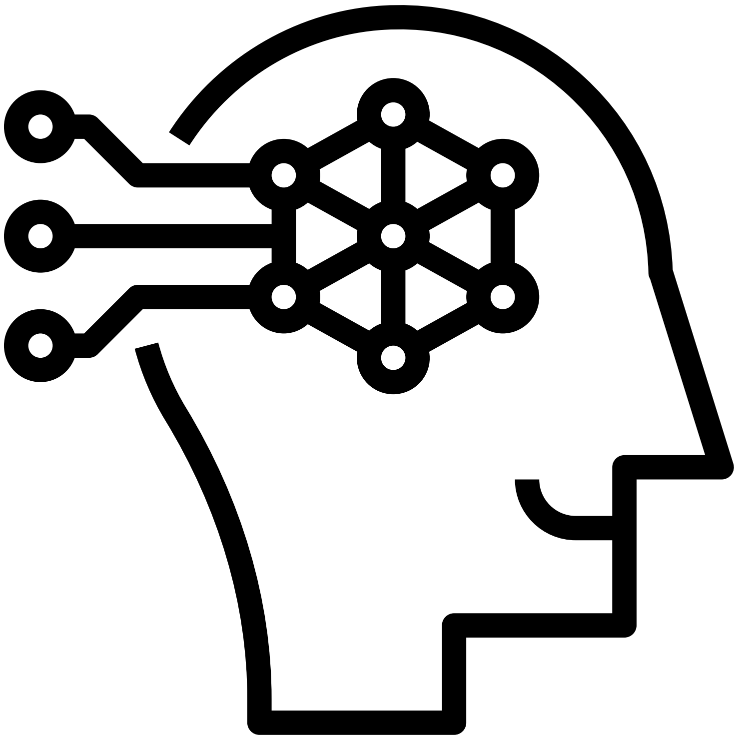
- Higher level understanding of face image obfuscation
- Privacy-preserving attribute selection (PPAS) algorithm
- Do not consider relationships between attributes
- Each attribute is classified with a single random forest classifier



Multi-label Face Attribute Classification

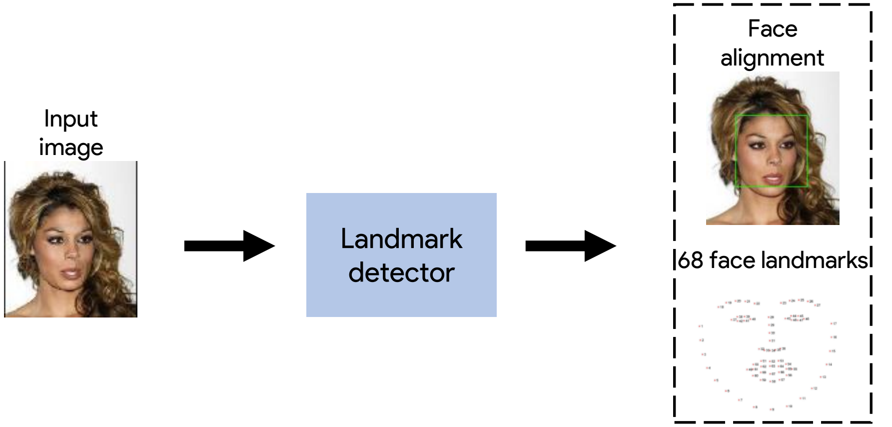
- Novel method for face attribute classification task
- Split learning into two stages, shared and task specific
- Grouped attributes into 2 classes





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

<u>Landmark Detection</u>

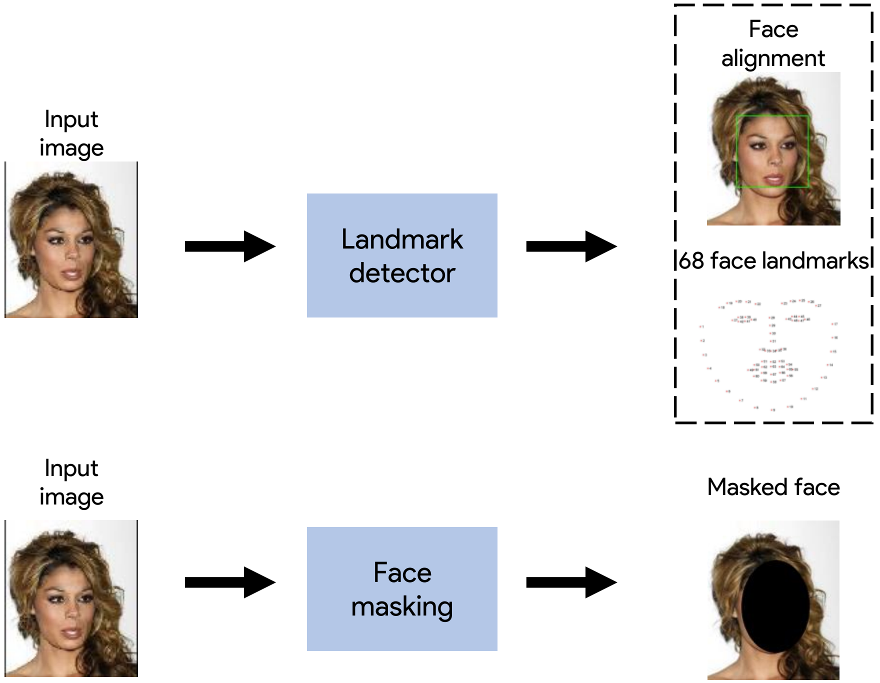
- Pre-trained detector provided by dlib
- Detect face bounding box
- Detect 68 key face landmarks

<u>Face Masking</u>

- *ellipse()* method
- Specific landmarks

<u>Face Attribute Classification</u>

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

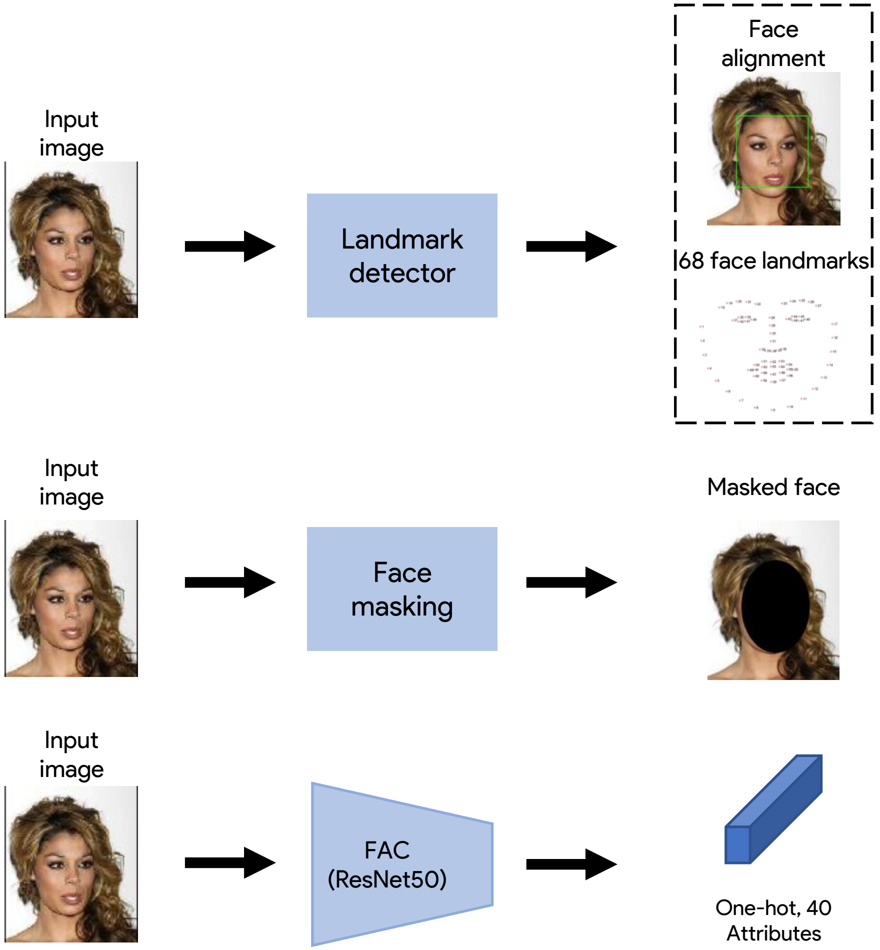


Stage 1
Image disentanglement

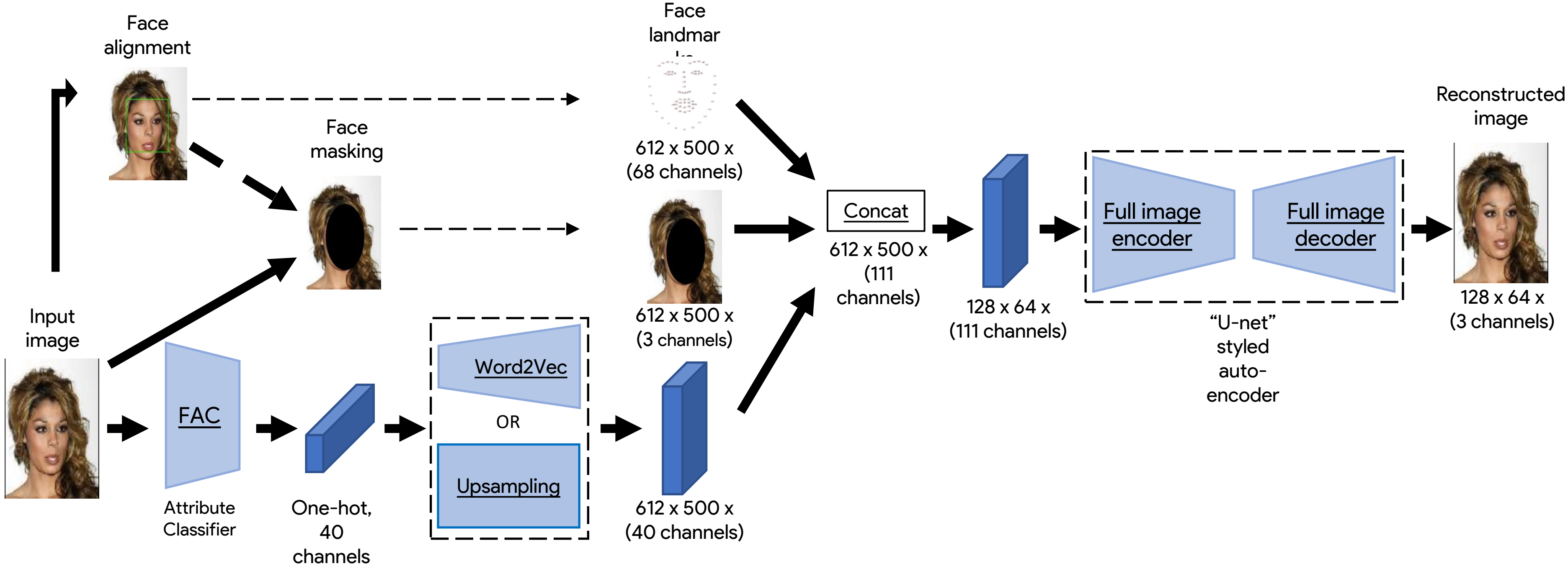
<u>Landmark Detection</u>
- Pre-trained detector provided by dlib
- Detect face bounding box
- Detect 68 key face landmarks

<u>Face Masking</u>
- *ellipse()* method
- Specific landmarks

<u>Face Attribute Classification</u>
- ResNet50 architecture
- Remove final layer, add flatten, add FC dense
- Sigmoid activation
- Adam Solver
- Binary-cross entropy loss

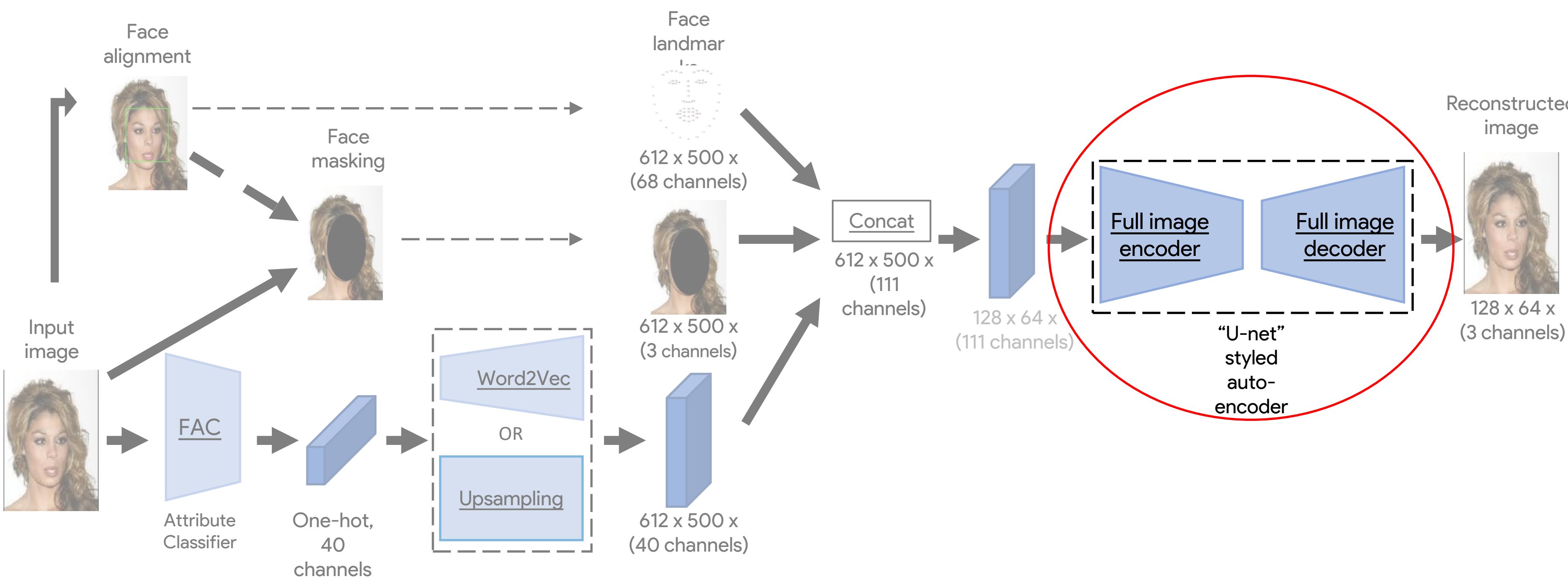


Stage 2
Reconstruction



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

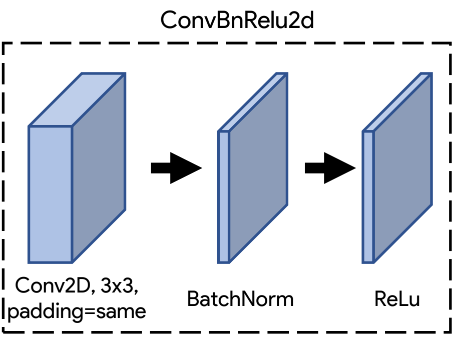
Stage 2
Reconstruction

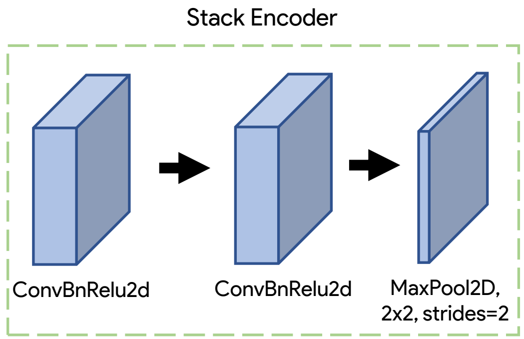


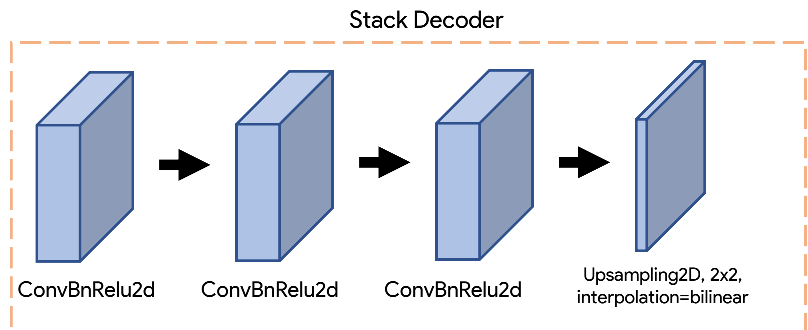
- Self-supervised auto-encoder 'U-Net' based architecture
- Concatenate all outputs from Stage 1
- Word embeddings to increase dimensionality
- SSIM loss

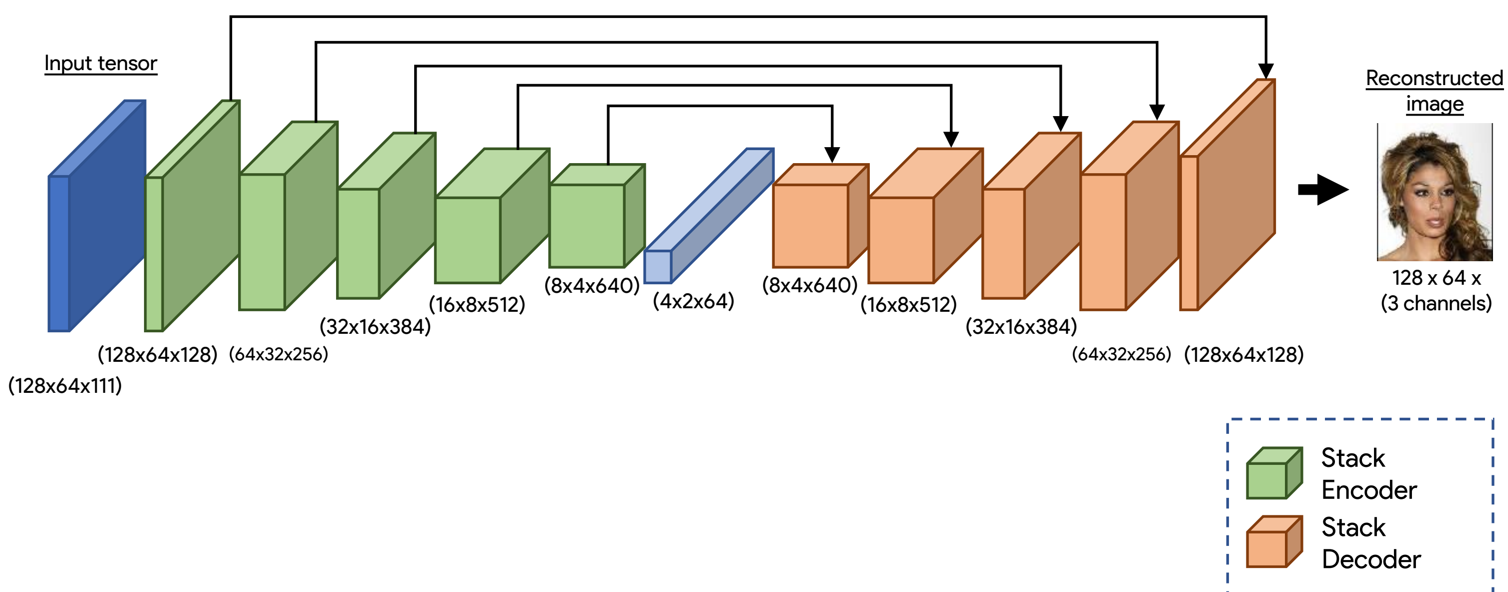
Stage 2

Recon-Net

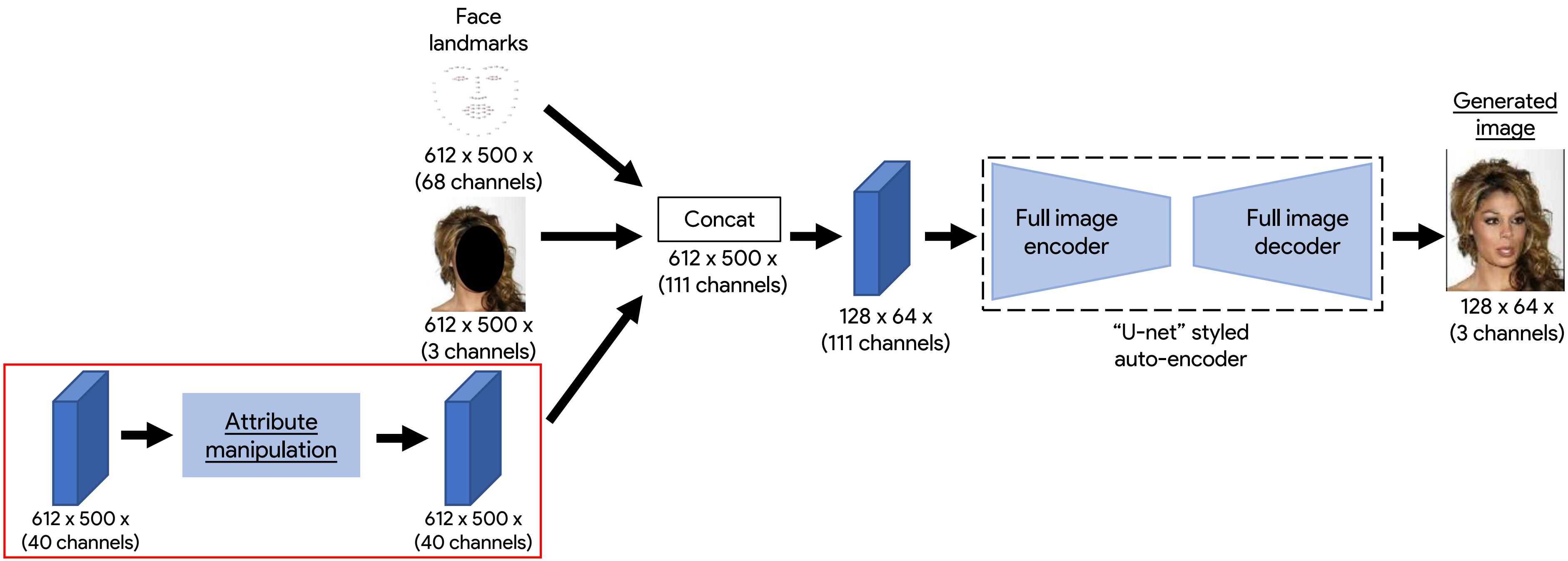








Stage 3
Generation

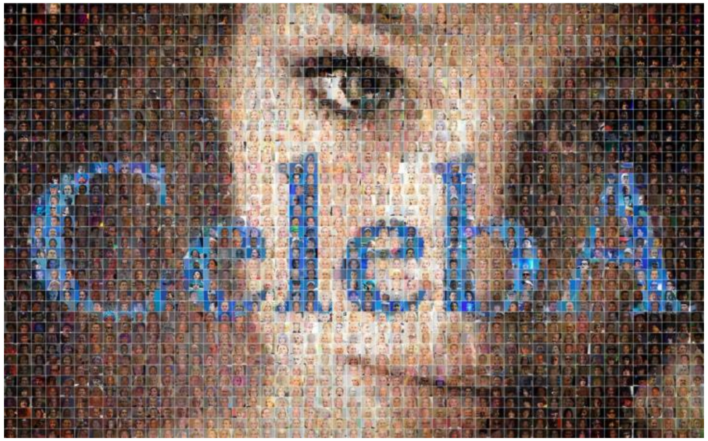


Experiments

Experiment Setup

<u>CelebA Dataset</u>

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



<u>Tools</u>

- 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

<u>Face masking</u>

- 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)

<u>Pre-processing steps</u>

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

<u>FAC</u>

- 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)

$$L^j_{FAC} = -\sum_{i=1}^{2} t_i log(s_i) = -t_1 log(s_1) - (1-t_1) log(1-s_1)$$

$$L_{FAC} = \sum_{j=1}^{40} L^j_{FAC}$$

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)

```python
class DataGenerator(tf.keras.utils.Sequence):
    'Generates data for Keras'
    def __init__(self, dataset, image_name_ls, face_attr, landmark_dict, \
                 batch_size=32,
                  dim=(32,32,32), n_channels=1, shuffle=True):

        'Initialization'
        self.dim = dim
        self.batch_size = batch_size
        self.image_name_ls = image_name_ls
        self.dataset = dataset
        self.n_channels = n_channels
        self.shuffle = shuffle
        self.face_attr = face_attr
        self.landmark_dict = landmark_dict
        self.on_epoch_end()

    def __len__(self):
        'Denotes the number of batches per epoch'
        return int(np.floor(len(self.image_name_ls) / self.batch_size))

    def __getitem__(self, index):
        'Generate one batch of data'
        # Generate indexes of the batch
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        # Find list of IDs
        list_IDs_temp = [self.image_name_ls[k] for k in indexes]
        # Generate data
        X, y = self.__data_generation(list_IDs_temp, indexes)

        return X, y

    def on_epoch_end(self):
        'Updates indexes after each epoch'
        self.indexes = np.arange(len(self.image_name_ls))
        # create array of indexes
        if self.shuffle == True:
            np.random.shuffle(self.indexes) # shuffle in place

    def __data_generation(self, list_IDs_temp, indexes):
        """
        private method

        list_IDs_temp contains name of the images in this batch

        will resize input images to (128x64)
        """
        'Generates data containing batch_size samples'
        # X : (n_samples, *dim, n_channels)
        # Initialization
        X = np.empty((self.batch_size, *self.dim, self.n_channels))
        y = np.empty((self.batch_size, *self.dim, 3), dtype='float64')
        # rgb channels
        new_dim = (64,128)
        # Generate data
        for i, ID in enumerate(list_IDs_temp):
            # Store sample
            train_image_mask_arr = np.asarray(Image.open(os.path.join(
                                   '{}_images_masked'.format(self.dataset),ID)))
            X[i,] = cv2.resize(self.__concat_all_channels(self.face_attr, \
                                          self.landmark_dict,
                                          train_image_mask_arr, ID), new_dim)

            # Store orig image
            train_images_label = np.asarray(Image.open(os.path.join(
                                   '{}_images_resize'.format(\
                                       self.dataset),ID)))/255.

            y[i] = cv2.resize(train_images_label, new_dim)
            # will return the original images
        return X, y

    def __concat_all_channels(self,face_attr, landmarks, mask, image_id):

        """
        for the case where face attr is one hot encoded only (baseline model)
        returns:
        final_arr: array of shape (612, 500, 111)
        """
        face_attr_temp = face_attr[face_attr['image_id']==image_id].\
            drop(columns='image_id').to_numpy().reshape((40,))
        face_attr_new = np.zeros((612,500,40))
        face_attr_new[:,:,face_attr_temp>0] = 1
        # if attribute is present, change to np.ones

        new_array = np.concatenate((mask/255., landmarks[image_id],\
                                    face_attr_new), axis=2)
        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

```python
# Batchnorm epsilon
BN_EPS = 1e-4

tf.keras.backend.set_floatx('float32')

class ConvBnRelu2d(layers.Layer):
    # Convolutional -> Batch norm -> ReLU

    def __init__(self, out_channels, kernel_size=(3, 3), padding='same'):
        super(ConvBnRelu2d, self).__init__()
        self.conv = layers.Conv2D(filters=out_channels, \
            kernel_size=kernel_size, padding='same',
                                  use_bias=False)
        self.bn = layers.BatchNormalization(epsilon=BN_EPS)
        self.relu = layers.ReLU()

    def call(self, x):
        x = self.conv(x)
        x = self.bn(x)
        x = self.relu(x)
        return x

class StackEncoder(layers.Layer):
    # ConvBnRelu -> ConvBnRelu -> * -> max pool (2x2)
    #      * store tensor for concatenation with expansive path

    def __init__(self, y_channels, kernel_size=3, dilation_rate=1):
        super(StackEncoder, self).__init__()
        self.encode = keras.Sequential([
            ConvBnRelu2d(y_channels, kernel_size=kernel_size, padding='same'),
            ConvBnRelu2d(y_channels, kernel_size=kernel_size, padding='same')
        ])

        self.max_pool = layers.MaxPool2D(pool_size=2, strides=2)

    def call(self, x):
        x = self.encode(x)
        x_small = self.max_pool(x)
        return x, x_small


class StackDecoder(layers.Layer):
    # Upsample (2x) -> concatenation -> ConvBnRelu -> ConvBnRelu -> ConvBnRelu

    def __init__(self, y_channels, kernel_size=3):
        super(StackDecoder, self).__init__()
        self.decode = keras.Sequential([
            ConvBnRelu2d(y_channels, kernel_size=kernel_size, padding='same'),
            ConvBnRelu2d(y_channels, kernel_size=kernel_size, padding='same'),
            ConvBnRelu2d(y_channels, kernel_size=kernel_size, padding='same'),
            ]
        )


    def call(self, x, down_tensor):
        x = layers.UpSampling2D(size=(2, 2), interpolation='bilinear')(x)
        x = layers.concatenate([x, down_tensor], axis=3)
        x = self.decode(x)
        return x

class UNet_2d(keras.Model):
    def __init__(self):
        super(UNet_2d, self).__init__()

        self.down1 = StackEncoder(128, kernel_size=3)
        self.down2 = StackEncoder(256, kernel_size=3)
        self.down3 = StackEncoder(384, kernel_size=3)
        self.down4 = StackEncoder(512, kernel_size=3)
        self.down5 = StackEncoder(640, kernel_size=3)

        self.center = ConvBnRelu2d(64, kernel_size=3, padding='same')

        self.up5 = StackDecoder(640, kernel_size=3)
        self.up4 = StackDecoder(512, kernel_size=3)
        self.up3 = StackDecoder(384, kernel_size=3)
        self.up2 = StackDecoder(256, kernel_size=3)
        self.up1 = StackDecoder(128, kernel_size=3)

        # Final prediction uses three channels
        self.classify = layers.Conv2D(filters=3, kernel_size=1, use_bias=True)

            # if filter more than one will raise error at tf.squeeze



    def call(self, x):

        out = x
        down1_tensor, out = self.down1(out)
        down2_tensor, out = self.down2(out)

        down3_tensor, out = self.down3(out)
        down4_tensor, out = self.down4(out)
        down5_tensor, out = self.down5(out)

        out = self.center(out)

        out = self.up5(out, down5_tensor)
        out = self.up4(out, down4_tensor)
        out = self.up3(out, down3_tensor)
        out = self.up2(out, down2_tensor)
        out = self.up1(out, down1_tensor)

        out = self.classify(out)
        out = keras.activations.sigmoid(out)

        return out
```

Stage 2
Recon Net

Epoch 3

Epoch 10

Epoch 15

Stage 2
Recon Net

Epoch 3

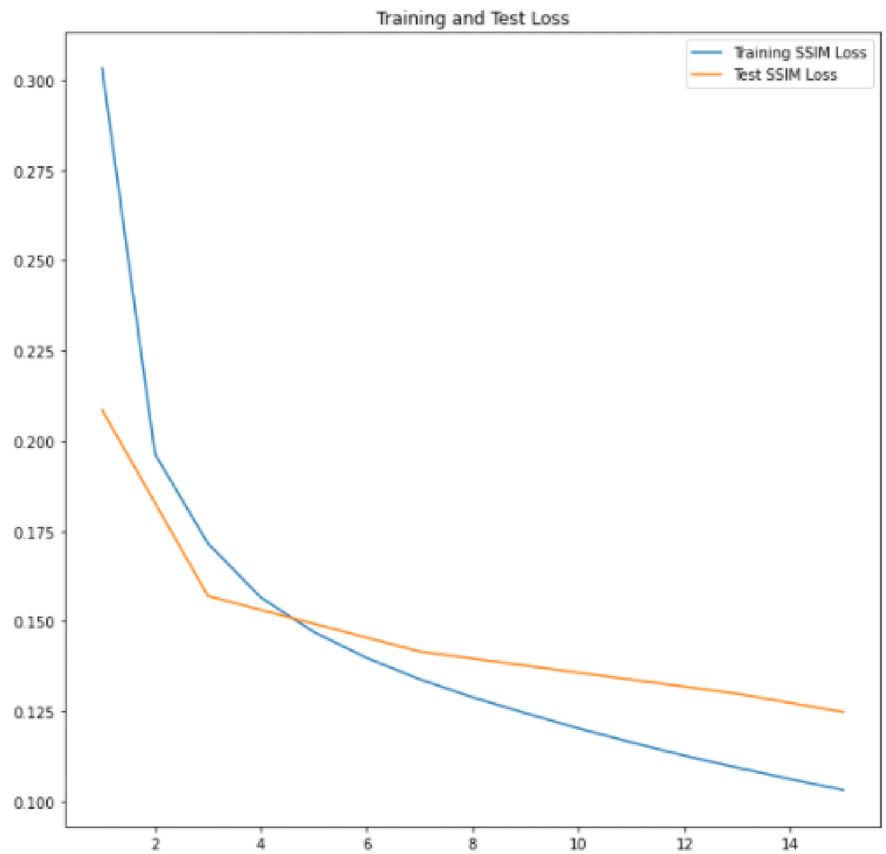
Epoch 10

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



Stage 3
Generation

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

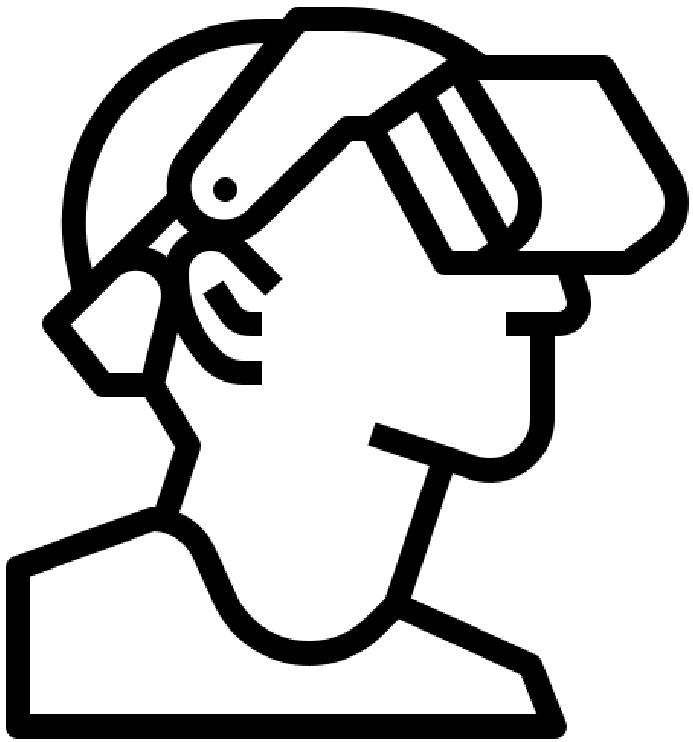
Stage 3
Image Triplets

Stage 3

Image Triplets

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																					



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


DATA
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 !