

How to Interpolate Between Images: StyleGAN and CLIP Image Manipulation

Jason Uwaeze NetID: ju6

December 2, 2023

Link to Google Colab Notebook: [Notebook Link](#) & [Notebook2 Link](#)

Abstract

Introduction to Computer Vision with Dr. Guha Balakrishnan

1 StyleGAN

The following are the images for Problem 1 section a.

1.1 a. Interpolating Between Images

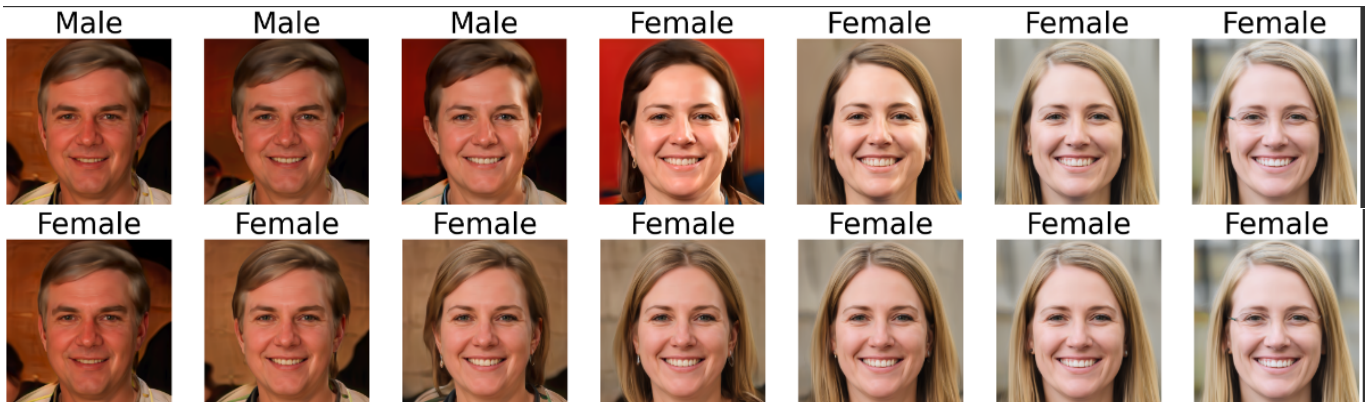


Figure 1: Interpolation between z_0 and z_1 (Top), and interpolation between w_0 and w_1 (Bottom)

iii. Question: What differences do you notice when interpolating in latent space versus style space? Do the intermediate faces look realistic?

Answer: A difference I noticed interpolating in the latent space versus the style space is that in the style space the image becomes "female" a lot sooner than when interpolating in the latent space. This might be because of the points that were selected between z_0 , and z_1 versus the points selected between w_0 and w_1 . Another thing that I noticed in the set of images, is that in the latent space, the male features such as the hair, were held on to a lot longer than images in the style space. For example the background, hair color, and even skin color of the images from the latent space became darker or more masculine before becoming lighter and more like the female image. The images in the style space started looking like the final image w_1 a lot sooner than images in the latent space.

Answer: The intermediate faces created by interpolating points from w_0 to w_1 in the style space look more realistic than intermediate faces created by interpolating points from z_0 to z_1 in the latent space. The images associated with the latent space had images which looked like a combination or average of both male and female images(i.e. z_0 and z_1). The images associate with the style space

looked more like a binary classification, male or female. Odd enough, points from the style space (i.e. between w_0 and w_1) were mostly if not all female faces, or most of the images looked more like w_1 .

1.2 b. Image manipulation with latent space traversals

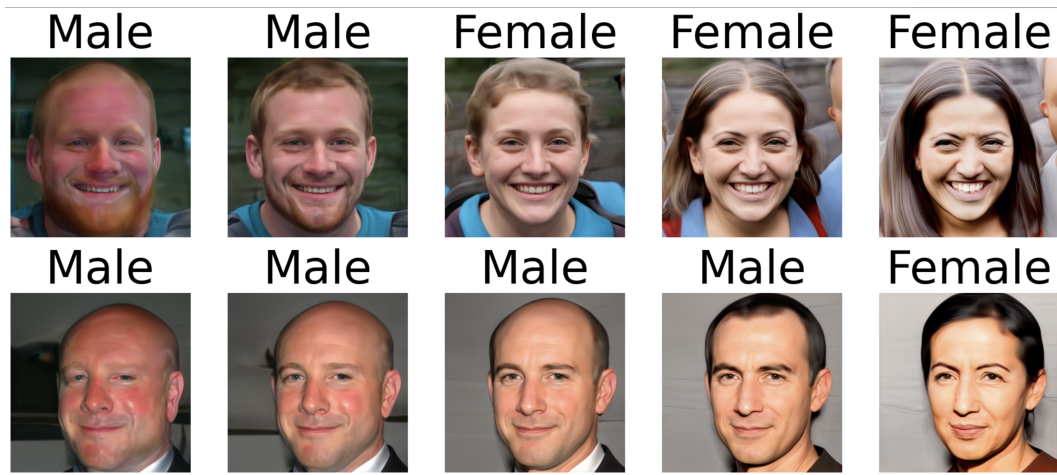


Figure 2: The center image will be the image generated by 'w'. The two images to the left will correspond to moving toward the “more male” direction, and the two to the right will correspond to “more female”. To generate the latter 4 images, move along the SVM hyperplane’s normal vector in both directions using some appropriate step size.

For this solution I used the $new_w = old_w + \alpha + v$ as my equation for manipulating the gender of the image, where α was my variable, and old_w and v were my constants. My step size was 15 such images seen above were associated to α values ranging from $[-30, 30]$ and the center images had a α value of zero, indicating that those images were create with an original w value or $new_w = old_w$.

Question: Do you notice any facial attributes that seem to commonly change when moving between males and females? Why do you think that occurs?

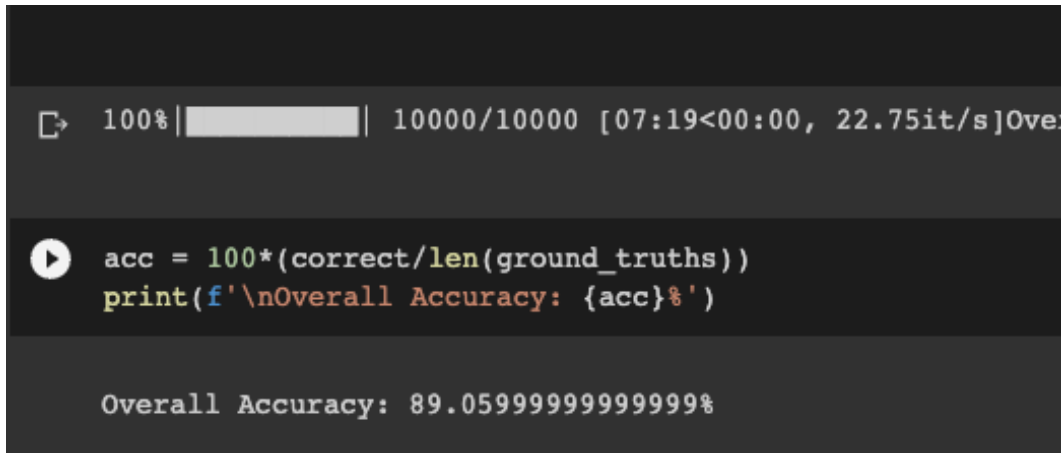
Answer: facial attributes that commonly change when moving between males an females include face hair, such that moving to more male image, facial hair increase. When moving to more female images, facial hair decreases. The smile of the person in the image seemed to stay constant, such that if there were no teeth shown in the original image, there would be no teeth shown in the manipulated images. I think these changes and no changes in facial attributes occur because of the sample dataset. The model had picked up patterns in the facial hair for example. They dataset may have included a bias, such that females tend to have more hair than males, and males tend to have more facial hair than females. Maybe even males tend to smile less than the females in the dataset.

2 Using CLIP for Zero-Shot Classification

2.1 a. Perform classification of each test image

To do so, create 10 different captions (e.g., “An image of a [class]”) corresponding to each of the 10 object classes. Then, for each image, store the label that provides the highest probability score. Report overall accuracy.

Answer: CLIP model achieved a Zero-Shot Classification accuracy score of 89%

A terminal window showing a progress bar at 100% and a code execution block. The code calculates the overall accuracy by dividing the number of correct classifications by the total number of ground truths and multiplying by 100. The output shows an overall accuracy of 89.05999999999999%.

```
100% | ██████████ | 10000/10000 [07:19<00:00, 22.75it/s]Over
▶ acc = 100*(correct/len(ground_truths))
  print(f'\nOverall Accuracy: {acc}%')

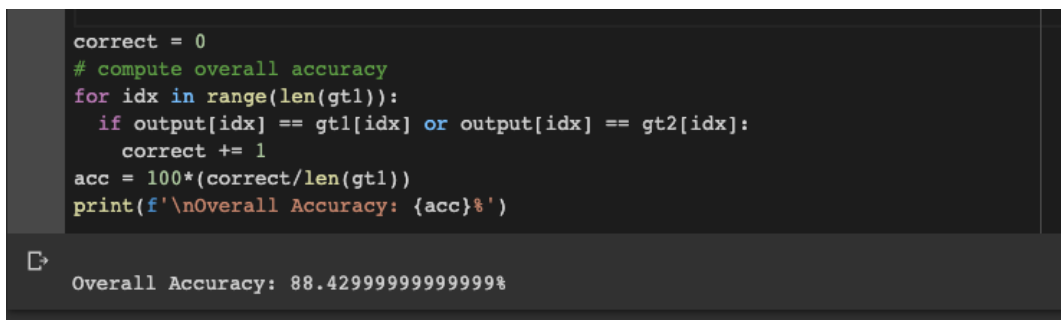
Overall Accuracy: 89.05999999999999%
```

Figure 3: Overall Accuracy Calculation

2.2 b. Performance Evaluation

Engineer the caption prompts to try to obtain better accuracy. To do so, give a set of possible captions per class instead of just one. For example, “A bad photo of a [class]” or “A drawing of a [class]”. Report your accuracy.

Answer: CLIP model achieved a Zero-Shot Classification accuracy score of 88%. The possible captions included, “A large [class]”, “A small [class]”. Adding these options did not improve the model accuracy.

A terminal window showing Python code that iterates through ground truths and compares them with model outputs. The code counts correct classifications and calculates the overall accuracy. The output shows an overall accuracy of 88.42999999999999%.

```
correct = 0
# compute overall accuracy
for idx in range(len(gt1)):
    if output[idx] == gt1[idx] or output[idx] == gt2[idx]:
        correct += 1
acc = 100*(correct/len(gt1))
print(f'\nOverall Accuracy: {acc}%')

Overall Accuracy: 88.42999999999999%
```

Figure 4: Overall Accuracy Calculation

Answer: CLIP model achieved a Zero-Shot Classification accuracy score of 90%. The possible captions included, "A photo of a [class]", "An imagery of "[class]". Adding these options improved the model accuracy from 88% and 89% accuracy to 89.59% accuracy.

```
correct = 0
# compute overall accuracy
for idx in range(len(gt1)):
    if output[idx] == gt1[idx] or output[idx] == gt2[idx]:
        correct += 1
acc = 100*(correct/len(gt1))
print(f'\nOverall Accuracy: {acc}%')
```

100% | ██████████ | 10000/10000 [13:54<00:00, 11.98it/s]
Overall Accuracy: 89.56%

Figure 5: Overall Accuracy Calculation