CS230

Your F*ai*vorite Face

Simon Kim
Department of Computer Science
Stanford University
spkim@stanford.edu

JeongWoo Ha
Department of Computer Science
Stanford University
jwha@stanford.edu

Kye Gahyun Kim
Department of Computer Science
Stanford University
ghkim@stanford.edu

Abstract

Our implemented model extracts characteristic facial features from given images and create a new face from them. It incorporates a deep convolutional neural network classifier to first understand the images, the results of which is fed into a generator model to create an image of a new human face. The generator has been trained via a Generative Adversarial Network, which also has a discriminator to distinguish images created by a generator. In the process of training, we experimented with different ideas, such as, but not limited to, overlaying the output of the CNN classifier to the noise vector to be fed into the generator as input, batch normalization, and one-sided label smoothing.

1 Introduction

This project started with the idea that, given several photos of human faces, we could build a model that understands their key facial features and use the knowledge to build a new face that also shares those features. To accomplish the task we needed to build a complicated model containing two models: a deep convolutional neural network (CNN) model that would extract facial features from given pictures, and a generator model trained via a Generative Adversarial Network (GAN) model that would take in the output layer of the CNN model to recreate a human photo of dimensions (218,178,3). In the process we experimented with batch normalization and one-sided label smoothing. The results of this research showed that applying bias on the noise input vector of a GAN model allows us to influence the output image to have the facial characteristics we expected it to have.

2 Related work

Discriminative neural network models have had a huge impact in image detection. However, previous works on the generator model for human images have showed that the progress in learning generative models capable of producing novel examples of natural data has not quite kept the par with the discriminative modeling. For instance, samples produced by the generative models often lacked semantic information such as consistency in gender, or lacked high frequency information, missing key details or often appearing blurry. Thus, to expand on previous works we tried to train a discriminative classifier that can learn representations of the image that are salient to humans, which we can use this classifier to augment the objective function corresponding to a generative model.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

We have used Large-scale CelebFaces (CelebA) dataset, which provided 202,599 images of celebrities with 40 attributes defined (style of hair, eyebrows, glasses, etc.). The raw image was in shape of (218,178,3), which we fed into the CNN classifier to train the model to extract relevant features. Since CelebA dataset provided preprocessed features for each of the facial image, our main focus for the first part was to tune the hyperparameters for the CNN classifier to achieve the highest accuracy. We then used the output to train the GAN model to generate novel images. For testing purposes, considering the large dataset we had, we divided the dataset into 98% training set and 2% dev set.

4 Methods

We used two different models to generate novel facial images reflecting the users' preferences. First, we trained a CNN classifier that would extract 40 facial features given the input image of size (218,178,3). We used this output layer to create the noise input vector for the GAN model, which simultaneously trained the generator and the discriminator. After training we used the generator to create a new face that would match the user's preferences.

4.1 Deep Convolutional Neural Network

The CNN model has 5 layers, with the output layer representing 40 facial attributes. Considering the high dimension of the original image, we used three convolutional layers to process the image, and then flattened it into a dense layer with dimensions 1024. We used rectified linear unit activation functions for each layer, and sigmoid function to generate the output. Because this is a multi-class multi-layer task, we used binary cross entropy as our loss function. We used Adam optimizer with learning rate 0.001.

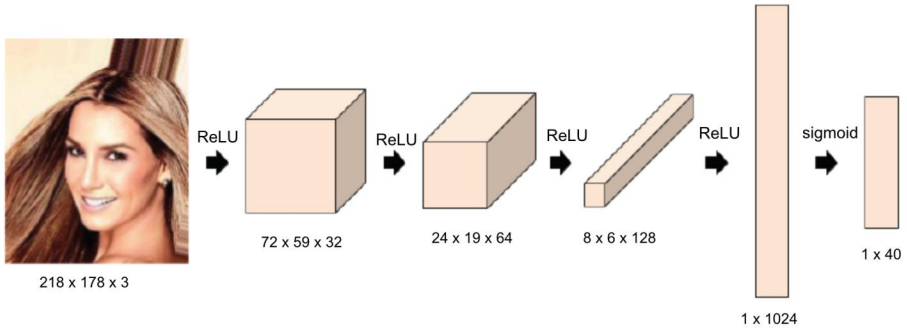

Figure 1: CNN Classifier

The main challenge was to tune the hyperparameters for the convolutional network to achieve the highest accuracy. We isolated each important hyperparameters, namely the filter size, kernel size, pool size, strides, and dropout values to see which value performs the best.

4.2 Generative Adversarial Networks

The output layer of the convolutional neural network has 40 units. As an input to the generative model we've generated a normal distribution of -1 to 1 of dimension 160, and overlaid it with the average of CNN outputs from multiple images. We've included two hyperparameters to create the noise input: α and β. We acquired the noise input as follows:

$$noise\_input[i] = \begin{cases} \alpha \cdot random[i] + (1-\alpha) \cdot output[i\%40] \\ \alpha \cdot random[i] - (1-\alpha) \cdot output[i\%40] \\ \beta \cdot random[i] + (1-\beta) \cdot output[i\%40] \\ \beta \cdot random[i] - (1-\beta) \cdot output[i\%40] \end{cases} \quad (1)$$

where random vector is a 160-dimensional vector of normal distribution, and the output vector is a 40-dimensional output vector of the images from CNN. By adjusting the hyperparameters α and β we could adjust how much we want the generator model to be influenced by the input image. Through empirical study we found that separating parameters α and β granted us more control over how much influence we want the input image to have on the output of the generator model.

We would train the GAN model in two steps: first, we would train the discriminator model to understand the difference between real and fake images. As inputs we fed in the actual images and the generated images from the generator model created from the noise vector described above. For the loss we used binary cross entropy.

Next we trained the generator model by training the GAN model overall. The GAN model was a concatenation of the generator and the discriminator, and had the noise vector as input and the discriminator's validity as the output. It was trained so that the generator would output an image that the discriminator would perceive as an actual image. We were careful not to train the generator at this step, as it could lead to the discriminator being trained twice as fast as the generator and thus the latter would overfit to the discriminator and output the same image regardless of its input noise vector. After training, we used the generator to output an image.

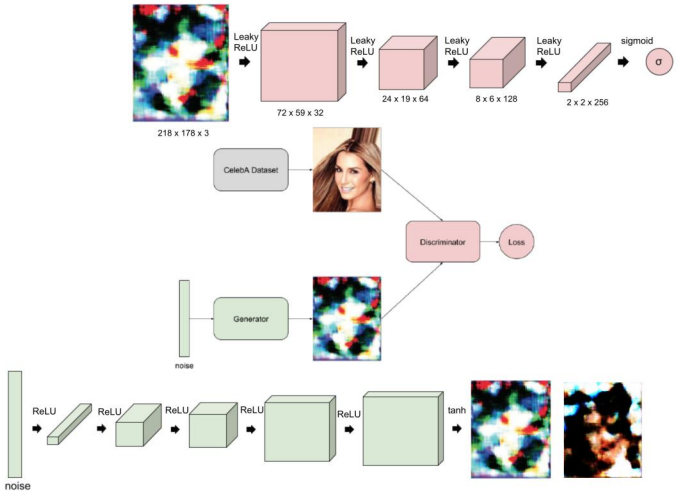

Figure 2: Generative Adversarial Network model

5 Experiments and Discussion

5.1 Deep Convolutional Neural Network

We tried various hyperparameters to get the best accuracies for both the training and dev sets. As you can see from the chart below, we consistently saw the dev score to be higher than train in two-layer ConvNets, a clear sign of under-fitting. Thus We incorporated additional layer to make it more complex. We continued to see signs of under-fitting in three layers, so we increased the filter sizes to (32, 64, 128), and adjusted the kernel, pool sizes, strides, dropouts, and learning rates accordingly.

For our final hyperparameters, we had filter sizes of (32,64,128), and kernel, pool size, and strides of 3. We set the dropout rate low to be 0.3, and used ReLU activation function for the hidden layers. After convolution layers we fed it in to a dense layer of 1024 units, which would be connected to a output layer of 40 units. In our attempt to resolve under-fitting issues, we were curious whether the last hidden layer should be a dense layer or another convolutional layer. After research we found out that having a fully connected dense layer showed higher train and dev scores than a convolutional layer, and as a result decided to incorporate a dense layer connected to the output layer.

We also experimented with two loss functions: sigmoid cross entropy, and binary cross entropy. Although good for multi-class classification tasks, we found the sigmoid cross entropy to be unsuitable for multi-class multi-label classification tasks. As a result we've used the binary cross entropy to calculate the loss. You can see the results in the chart below.

(a) Accuracy distribution

(b) Training Accuracy by Epoch

Figure 3: Hyperparameter Tuning Accuracies

5.2 Generative Adversarial Networks

The GAN model was tricky to train, as the network is complex and is composed of a number of hyperparameters to train. Also, because the GAN model took the output layer of CNN as input, it was required that we first build a well-performing CNN model.

Initially we trained the GAN model with 2,000 real images per epoch, which meant in each epoch discriminator would learn from 2,000 real and fake images each, and the generator would generate 4,000 images. The loss functions for both models were binary cross entropy. After 1,000 steps we found that this led to a discrepancy of learning between two models and caused discriminator to consistently achieve high accuracy and generator to get high loss. We determined this divergence was a result of the huge time gap in training between discriminator and generator, causing discriminator to always outperform the generator. This made the generator converge to a local maximum and always output the same image. As a result we decided to reduce the batch size per each epoch to 32.

Setting the learning rates also required careful adjustments. We initially set them to be equivalent for both models, at 1e-3. However, we determined that the generator was not learning as quickly as the discriminator, as the generator loss failed to go down after sufficient number of steps and the discriminator continuously earned high accuracy. Classification tasks for neural networks are much simpler in dimensionality than generating new data from small vector. Specifically, the discriminator was only a binary classification given an input image of (218,178,3), but the generator had to expand a (160,) vector into that of dimension (116412,). Considering the discrepancy of the difficulty of each task, we've set the learning rate of the generator as 4e-4 and the discriminator as 2e-4.

In addition, we incorporated an idea called one-sided label smoothing. In each epoch we would feed into the generator 16 real and 16 generated images with labels 1 and 0 correspondingly. With a low probability $\gamma = 0.05$, however, we flipped the labels so that real images would be falsely labeled as fake. The simple intuition was that this would make the discriminator be more difficult to be trained, thus preventing the generator from converging and being stuck in a local maximum. *(Salimans et. al. 2016)* This would prevent the discriminator from creating a clear divide between two labels, giving generator more room to generate different images to trick the discriminator. After empirical research we found this to allow generator to better create different images given different input noises. To add additional generality to the model, we've added dropout rates of 0.3.

We added batch normalization after each layer in the generator and the discriminator because, as this is a deep convolutional generative adversarial network, it would be easy for the model to fall into exploding / diminishing gradient issue without normalization and lead to diverging losses for previous GAN models.

For generating the noise input vector for generator, we first tried simply concatenating the 40-dimension output layer of CNN directly after a 100-dimension noise vector, but after 2,000 epochs of training we found that the generated image did not look natural. Previous researches showed empirical studies implying the noise input vector should be a normal distribution with a mean of 0, and thus we were inspired to instead overlay the output layer directly on top of a random normal distribution vector to guarantee a distribution close to normal. After hyperparameter tuning we determined $\alpha = 0.6$ and $\beta = 0.8$ yielded the best results.

6 Results

6.1 Deep Convolutional Neural Network

After training, we have acquired a final train accuracy of 89.43% and loss 0.241, and dev accuracy of 89.40%. You can see that after careful hyperparameter tuning, dev follows traits similar to train dataset, and there are no over- nor under-fitting issues.

(a) Accuracy (b) Binary cross entropy loss

Figure 4: Accuracy and loss across epochs

6.2 Generative Adversarial Networks

In the initial hyperparameter tuning process, the output images contained much noise, as seen in figure 5(a). As epochs progressed, the images started to have distinguishable human features.

(a) 100 epochs (b) 500 (c) 1000

Figure 5: Output images for each epoch trained

Here are the images generated after training 3,500 epochs. Each pair of image was generated from combining same set of images, thus share similar facial features such as gender, hair color, etc. You can see the generator has also learned to disregard the background and focus on facial features. Note that this is only at 3,500 epochs and, considering its high dimensionality, there are still room for improvements in resolution as the number of trained epochs grow.

Figure 6: Output images for each epoch trained

7 Conclusion

In this project, we successfully extracted key facial features from given images using a deep convolutional neural network, and merged the output layers of different images together to generate a new face of similar facial features using a generative adversarial network. Our methods of generating the noise vector for generator proved to be successful in biasing the output image, and we believe our network framework will provide a stepping stone for future works related to generating faces specific to user interests.

References

[1] Karras, T., Timo, A., Laine, S., Lehtinen, J. (2017) Progressive Growing of GANs for Improved Quality, Stability, and Variation.

[2] Berthelot, D., Schumm, T., Metz, L. (2017) BEGAN: Boundary Equilibrium Generative Adversarial Networks

[3] Radford, A., Metz, L., Chintala, S. (2016) Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks

Github Link

https://github.com/spkimda/cs230.git