Final Project Report

Keaton Spiller, Huidong Wang, and Sai Kiersarsky

CS 445, Winter 2022, Our implementation

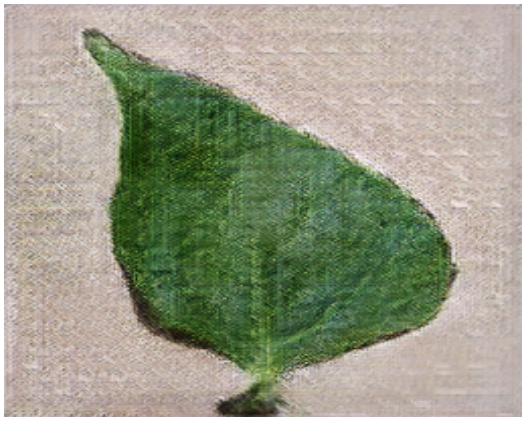
Keaton filtered the database of leaves to use with the GAN network, and helped build and understand the discriminator, generator and GAN. Researched hyper parameter techniques and methods to reduce unwanted characteristics. Huidong helped build the technical parts of the GAN network, in particular generating images from a 64x64 image to 256x256. In addition, troubleshooting the hyper parameters to make a more realistic leaf. Sai created the general overview, structure of the paper, support in coding and research, and possible future directions of General adversarial networks.

For our final project we built a Generative Adversarial Network (GAN) that generates images of plant leaves. The goal of the project is to make as realistic of leaf images as possible after being trained on a dataset of leaf images. This report will consist of the following: First, an overview of the general GAN system, which was mostly new to all our group. Then a description of our particular project including technical details and our thought process. Then finally, some future work that is of interest to us.

GANs are a type of neural network architecture that consist of a generative network and a discriminator network. Both are multi-layer perceptron models which can be composed of both convolutional layers and feed-forward layers. Typically, the discriminator is trained with a

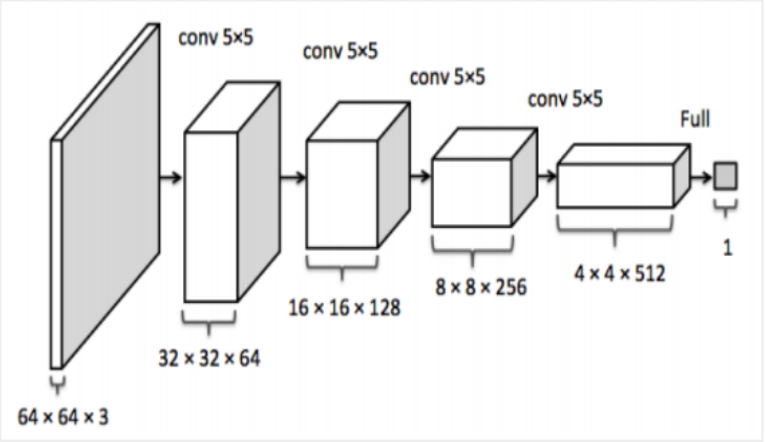
traditional supervised learning paradigm on a dataset consisting of images of whatever class is desired to be generated. Then the generative network is fed pseudo-random noise. The generative network receives as its loss function the degree to which the discriminative network classifies the generated image as belonging to that particular class. The weights of the generative network are updated until it produces an image of satisfactory accuracy to the discriminative network.

Our dataset consisted of various species of plant leaves, ranging from apples to cherries to peaches. There were 15084 total images unevenly distributed among 12 species of plants. We originally decided to use every species so we have maximum training data, yet we ran into issues with images of corn husks taking over and dominating our generated images, possibly because the corn took up the entire frame whereas the other leaf images were centered inside the frame. After reconsolidating the images we had 13,922 images.



This is one of our generated images with preprocessed bach_size=4 (Generated Images)

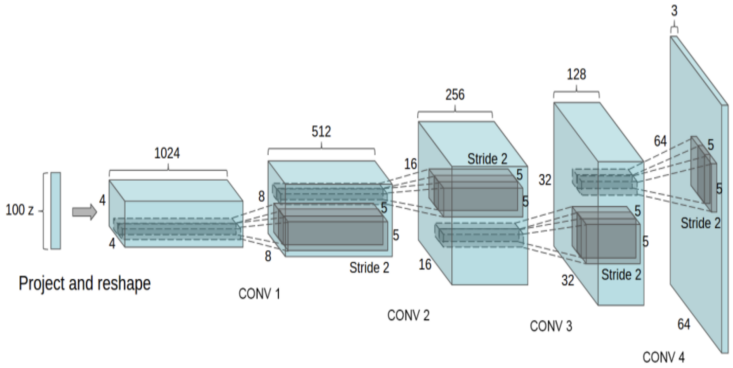
```
Model: "discriminator"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_31 (Conv2D)           (None, 128, 128, 64)      3136

 leaky_re_lu_57 (LeakyReLU)   (None, 128, 128, 64)      0

 conv2d_32 (Conv2D)           (None, 64, 64, 128)       131200

 leaky_re_lu_58 (LeakyReLU)   (None, 64, 64, 128)       0

 conv2d_33 (Conv2D)           (None, 32, 32, 128)       262272

 leaky_re_lu_59 (LeakyReLU)   (None, 32, 32, 128)       0

 flatten_6 (Flatten)          (None, 131072)            0

 dropout_6 (Dropout)          (None, 131072)            0

 dense_19 (Dense)             (None, 1)                 131073

=================================================================
Total params: 527,681
Trainable params: 527,681
Non-trainable params: 0
```



The discriminator convolutional network. This image is from the inpainting paper.

Analogous to the police, the discriminator distinguishes real data from the data generated by estimating the probability that a sample came from the training data. Mean(discriminator_loss), Mean(generator_loss). This uses a sequence of layers of Conv2D, LeakyReLU, and a final "sigmoid" activation. The conv2d uses a convolution kernel that convolves with the layer input to extract the features of the image. LeakyReLU is a Leaky version of a Rectified Linear Unit. When the activation is positive it passes through, otherwise it multiplies alpha by the negative values. This helps reduce neuron death when back propagating. The dropout is in order to prevent overfitting, it sets inputs to 0 when the frequency reaches 0.2, otherwise scales 1/(1-rate). We removed the dropout layer because it removed too much of the data even with a small frequency. This was because of how we combined real and fake images, a separate generator and discriminator for fake and real would be a better solution. The dense part is to pass the activation of the dot product of the input and the kernel along with the bias for an"Activation (dot(input, kernel) + bias)".

```
Model: "generator"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_20 (Dense)             (None, 32768)             4227072

 reshape_13 (Reshape)         (None, 16, 16, 128)       0

 conv2d_transpose_39 (Conv2D  (None, 32, 32, 128)       262272
 Transpose)

 leaky_re_lu_60 (LeakyReLU)   (None, 32, 32, 128)       0

 conv2d_transpose_40 (Conv2D  (None, 64, 64, 256)       524544
 Transpose)

 leaky_re_lu_61 (LeakyReLU)   (None, 64, 64, 256)       0

 conv2d_transpose_41 (Conv2D  (None, 256, 256, 512)     2097664
 Transpose)

 leaky_re_lu_62 (LeakyReLU)   (None, 256, 256, 512)     0

 conv2d_34 (Conv2D)           (None, 256, 256, 3)       38403

=================================================================
Total params: 7,149,955
Trainable params: 7,149,955
Non-trainable params: 0
```



The Generator is a team of counterfeiters, a model that captures data distribution and attempts to fool the discriminator. A hypersphere of random points from a normal distribution in the latent space is passed into sequences of layers of Deconvolution (Conv2DTranspose) and LeakyReLU activations. After dimensions are in the final shape a Conv2D and sigmoid activation convolves the layers of inputs and produces a final image. The Conv2DTranspose is used to go in the opposite direction of convolution and maintain connectivity patterns. This pattern holds the features extracted, and as the GAN is optimized will pick the labels of fake images more frequently.

For each epoch there's a training session between the discriminator and generator using a series of generated images, combined with real images to train the discriminator. The labels of fake and real images are altered using uniform random noise to prevent overfitting the labels. Using a tensorflow GradientTape() we use automatic differentiation with tapes to back propagation through the gradients of each "recorded" computation and optimize towards a minimum. This is done with reverse mode differentiation (tape.gradient), and repeated for

calculating discriminator and generator loss. Once the loss is minimized, we can generate synthetic images recognized to be leaves.

As far as limitations to our work is concerned, one obvious one is that the system is only capable of producing simple leaves. It would be interesting to train the system on a greater variety of images to get more varied results. There are also stylistic components that can be used to modify the output of the GAN, which would be most intriguing. In addition to these factors, the output images of the system look very much like actual leaves, but the final images are rather grainy. There are methods of perfecting the image quality that could be incorporated into the system.

Some hyper parameters we could change are batch_size for pre-processing tensors, noise added to image labels, the activation type of the layers.dense(), or the learning rate used by the Adam optimizer. First we rebuilt the model to use a higher resolution of pixels, 256x256 instead of originally 64x64 images, this gave a much clearer image, and alot more variation. Types of activation we could use are sigmoid, Relu or other types of Tensorsflow specific activations (Link). Trying different versions of batch sizes from the tensor preprocess of images showed us smaller batch sizes gave more realistic leaves. We used 2 to 256 batches, and ended up sticking to 4 batch sizes as the optimal configuration. We attempted different variations of learning rates, noise, and types of activations, but ended up reverting back to the original settings each time. Our main criteria for better parameters were more realistic images.

A future direction of this work is using an algorithm to automatically tune the hyperparameters so that it doesn't have to be done manually. GAN's are notoriously sensitive to hyperparameter configurations, and fully training the networks to assess the results after every

change to a hyperparameter is not efficient if the training takes over an hour to achieve appreciable results (we used Colab's paid Tesla p100 GPUs). To obtain state of the art results, a more sophisticated hyperparameter tuning system must be utilized.

A more long term direction that we would like to take this work in the future is in the form of using GAN-like techniques to probe layers of pre-trained neural networks to examine the role each individual layer plays. Interpretability in machine learning is an area in which the field has much room to expand. A common analogy of neural networks in that of a black box – inscrutable to the outsider. We would like to probe individual layers of neural networks with GAN or GAN-like techniques to probe the 'subconscious' of the network and then subsequently examine correlations between aspects of observable network activity such as degrees of localization/distributivity of information and the aspects of the input being processed at that stage. This may yield insight into the black box of neural networks.

We used a DCGAN, DCGAN is a combination of deep CNN and GAN. The convolutional network is introduced into the generative model for unsupervised training, and the powerful feature extraction ability of the convolutional network is used to improve the learning effect of the generative network. The principle of DCGAN is the same as GAN adversarial generation. It just replaces the G and D of GAN with two convolutional neural networks.

We used a DCGAN instead of LAPGAN which uses multiple GANs to generate levels of detail in a Laplacian pyramid representation of image. The benefit to using a DCGANs instead of LAPGAN is the ability to handle higher resolutions of images. Before the DCGAN, the LAPGAN was unable to generate high resolution images in the same capacity.

References

[1] Github: Code Resource 1 Code Resource 2

[2] Research papers: Ian Goodfellow on GAN networks and

Image Completion with Deep Learning in TensorFlow. Conv2DTranspose