MASTER THESIS

An Empirical Study of Deep Learning for Guided Painting Generation and Hypernym Extraction from Text

Jerry Li

Northwestern University

Committee chair: Doug Downey

Committee member: Aravindan Vijayaraghavan

Contact: jerrylijiaming@gmail.com

June 7, 2017

Acknowledgments

I would first like to thank my thesis advisor Prof. Doug Downey for his support of my research in the past three years, for accepting me into his lab when I only had 3 month worth of Computer Science experience, and for raising my interest in machine learning, natural language processing, and in research. My college life and research interests would be very different without his guidance, encouragement, and expertise.

I would also like to thank Prof. Aravindan Vijayaraghavan for guiding me to see the math and the beauty behind machine learning problems and for being a second committee member of my master thesis. Math would be less fun without his patience and his clear explainations to my questions.

I would like to acknowledge my colleagues and supervisors: David Demeter, Nishant Subramani, Sarah D'Angelo, Prof. Darren Gergle for the opportunity to work together and for offering their help when I most need them. I would like to thank Taizan Yonetsuji for inspiring my sketch colorization project and for being friendly and helpful. I would like to thank Rella for providing me the sketches of her artwork and allowing me to use them in this thesis.

I would like to thank my parents for supporting my study in a prestigious college. I would like to thank my friend Ivy Zheng and Yulun Wu for nudging me to study Computer Science. I would like to thank Alan Fu, Michael Wang, and Jimmy Yoon for our endeavor. And I would like to thank all my friends, senpai, and kouhai for their spiritual support.

Contents

1 Introduction to Neural Art

The process of creating something unique is long thought to be the pinnacle of intellect. With the recent advancement in Deep Learning, particularly in fields such as Gaming AI [Silver et al., 2016], object detection [Simonyan and Zisserman, 2014; Szegedy et al., 2015], translation [Johnson et al., 2016; Wu et al., 2016], and classification [Jiang et al., 2010] where efficient deep learning algorithms sometimes outperforms the best human in the field, one may start to wonder what are the limits of the current AI system. As the quote from the famous physics researcher Richard Feynman says "What I cannot create, I do not understand". Thanks to the progress in recent years in deep learning classification models, neural networks are capable of understanding a lot more about the world we live in. As a result, researchers are also putting their eyes on building deep learning systems that are able to create and/or recreate image, text, and voice. The GAN model, first proposed in [Goodfellow et al., 2014], has been especially successful in the area of image generation [Isola et al., 2016; Radford et al., 2015; Reed et al., 2016; Zhu et al., 2016] (An example generated image: 1) and achieved some success in text generation as well [Yu et al., 2016]. Some other non-GAN based image synthesis models include: DRAW [Gregor et al., 2015] and Sketch-rnn [Ha and Eck, 2017] . Recreating voice requires a little bit more ingenuity in terms of the input format and preprocessing, but there are still huge progress in recent years on non-parametric text-to-speech synthesizer. Google's WaveNet [van den Oord et al., 2016] generates raw audio forms frame by frame using a deep generative RNN model. Baidu's Deep Voice [Arik et al., 2017] improves on the WaveNet model and with carefully programmed custom training process tuned specifically for their hardware, Deep Voice can generate voice in real

time.

Figure 1: Faces generated by Face DCGAN in [Radford et al., 2015].

In this paper, we will focus on deep learning algorithms that are capable of utilizing image as well as other non-image data to guide the creation of new images and paintings. Then we will delve into two projects re-implementing and improving on related works in this field: one is neural style and the other one is sketch coloring.

2 Neural Style

2.1 Introduction to Neural Style

Neural Style can be loosely summarized as a technique that uses deep neural networks to take two or more images as inputs, one content image and one or more style images, and outputs a single image which is still largely similar to the content image in terms of its semantics and overall position of objects, but the "style" of which resembles that of the given style images. It is a tricky task to define what exactly does the "style" of an image means. Prior to LA Gatys' work in 2015 [Gatys et al., 2015b], researchers have attempted to interpret style as brush stroke, color choice, and representation and arrangement of semantic objects [Semmo et al., 2015; Shih et al., 2013, 2014] . Each of those previous attempts consisted of carefully hand designed features to capture the "loss" or the difference between styles of two images. In the narrow range of styles for which the algorithms were designed, They achieved a decent amount of success, but the hand crafted features most often failed to apply to the vast amount of artworks created by our human civilization. LA Gatys' paper in 2015 [Gatys et al., 2015b] utilized the power of a pretrained object recognition network to capture the statistical relations needed to describe the style and successfully created visually pleasing results, shown in 2 . Since then, numerous improvements [Chen and Schmidt, 2016; Dumoulin et al., 2017; Dushkoff, 2016; Huang and Belongie, 2017; Li and Wand, 2016; Ulyanov, 2016; Ulyanov et al., 2016a] has been made to make the results looks better and the generation process faster, but almost all of them follows Gatys' work and used pretrained object recognition neural networks to capture the "style" of the input images – thus the field is called Neural Style.

Figure 2: Neural Style transfer outputs. Reprinted from [Gatys et al., 2015b].

We re-implement some of the works in the field, namely back propagation neural style transfer [Gatys et al., 2015b], Markov Random Field(MRF) style transfer [Li and Wand, 2016], feed forward style transfer [Dumoulin et al., 2017; Ulyanov et al., 2016a] , and neural doodle [Ulyanov, 2016]. Further more, we've made some

improvements (as well as failed attempts) to provide more detailed control over the stylization process.

2.2 Neural Style Related Work

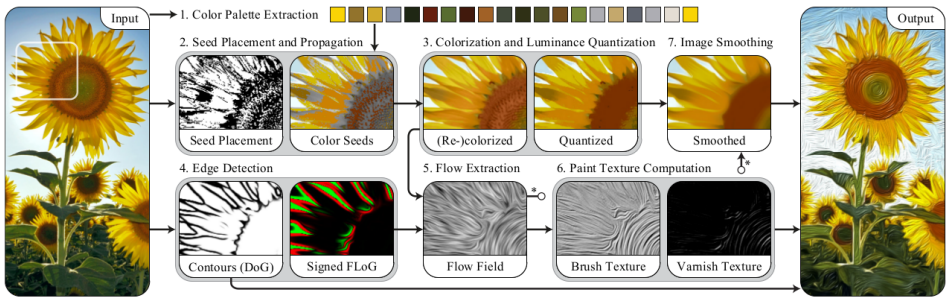


Figure 3: Oil paint stylization pipeline. Reprinted from [Semmo et al., 2015].

Prior to LA Gatys' work[Gatys et al., 2015b], the style transfer task was mostly limited to one specific type of scene, object, or style. Shih et al. [Shih et al., 2013] provided a technique to synthesize photographs at different times of the day by using a locally affine model trained on a database of time-lapse videos. Semmo et al. [Semmo et al., 2015] used re-colorization and non-linear image filtering to stylize photos into oil paints. Shih et al.[Shih et al., 2014] used multi-scale local transfer through computing Laplacian stacks and local energy maps to transfer the style of one head portrait to another. All of them achieved good results by mimicking artistic details such as brush strokes, color usage, textures, etc., but there was an common limitation. It is too costly to manually analyze each style and build a computational model that is not extensible to outside of the original application. (An example of a carefully designed oil paint style transfer model is shown in 3.) As a result, style transfer did not gain public attention, until 2015.

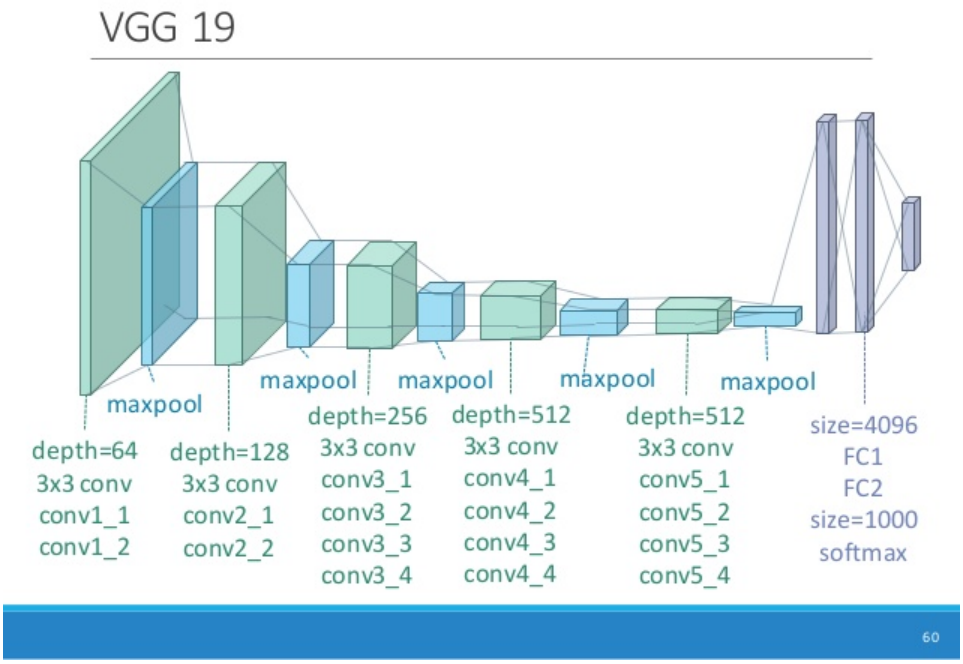


Figure 4: Illustration of the VGG Network. Reprinted from [Chang, 2016].

In 2015, Gatys et al. published his paper "A Neural Algorithm of Artistic Style" [Gatys et al., 2015b] that revolutionized the field of style transfer. The main contribution of this paper is summarized as follows: it was the first to show that pre-trained object detection networks could also be used on style transfer and it was the first to suggest that the Gram matrix - a matrix that computes the statistical correlation of two vectors - can be used as a loss function that directly captures the style of an image but not the content, thus separating the content and the style of an image.

Gatys' work [Gatys et al., 2015b] was rooted in texture transfer. In earlier works of texture synthesis such as Portilla and Simoncelli [Portilla and Simoncelli,

2000], it was proposed that texture can be sucessfully described by the stastistics of patches in "adjacent spatial locations, orientations, and scale" [Portilla and Simoncelli, 2000]. As an extension to that idea, In Gatys previous work [Gatys et al., 2015a], he showed that the VGG19 [Simonyan and Zisserman, 2014], a convolutional neural network (shown in 4) that achieved the state of the art in the 2014 ImageNet Challenge, an image recognition competition, also captured a diverse range of features in each convolutional layer as a by product. By using the Gram matrix, which calculates the inner product of a vectorized feature map(i.e. a flattened convolutional layer in VGG19) with its transpose, it successfully captured enough statistical information about the correlation among local features of the input image. Furthermore, as the convolutional layers gets deeper, the receptive field gets larger and the captured features became more coarse-grained. Therefore, calculating the Gram matrix for different convolution layers in VGG19 enables the model to capture the local features of different sizes, which according to [Portilla and Simoncelli, 2000] describes the overall texture of the input image. A new image that resembles the texture of the input image can be constructed by feeding the texture and a white-noise-initialized image through VGG19 and minimizing the L2 difference between the Gram matrices of their convolutional layers using back propagation on pixel values of the second image. The model is shown in 5.

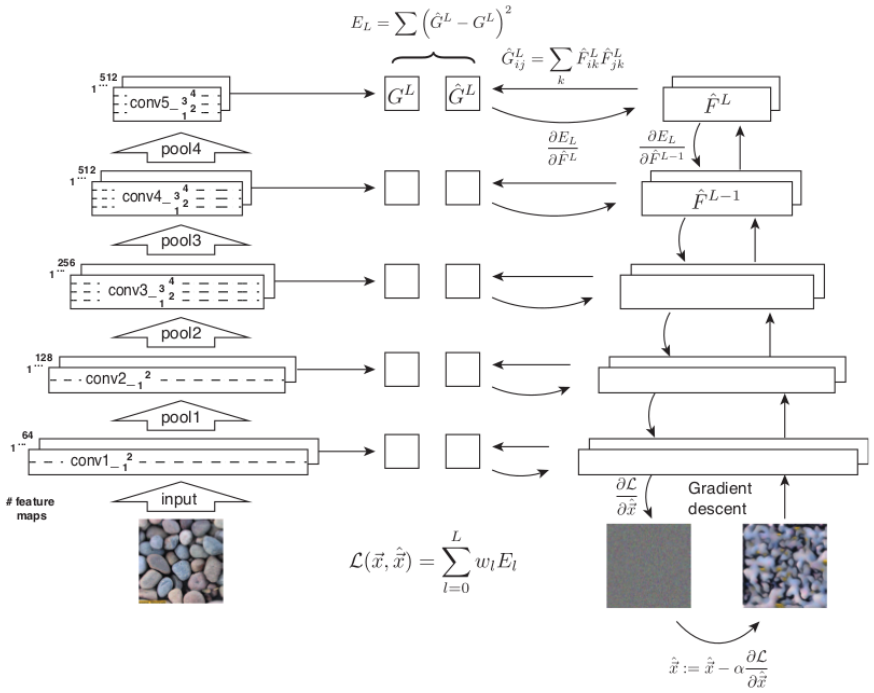


Figure 5: Illustration of the back propagation texture synthesis model. Reprinted from [Gatys et al., 2015a].

In essence, the back propagation neural style algorithm [Gatys et al., 2015a] simply added a content loss on top of the texture transfer techniques described above. The content of an image is represented directly by the feature response of a high level convolution layer in VGG19. One can interpret that as "seeing" the content image through the VGG19 network, which is capable of extracting semantic information but discarding the exact pixel values of the content image. The additional content loss is the L2 difference in the feature responses of the chosen convolution layer of the content image versus that of the output image. As a result of reducing the content and style losses simultaneously, the model matches the texture information of the output with the style images while at the same time keeps the

same semantic meaning and layout of the content image. In summary, the content and style information of the input images are separated using a pre-trained object recognition network and recombined on a new image using an iterative optimization method. A reconstruction of the separated style and content information is shown in 6.

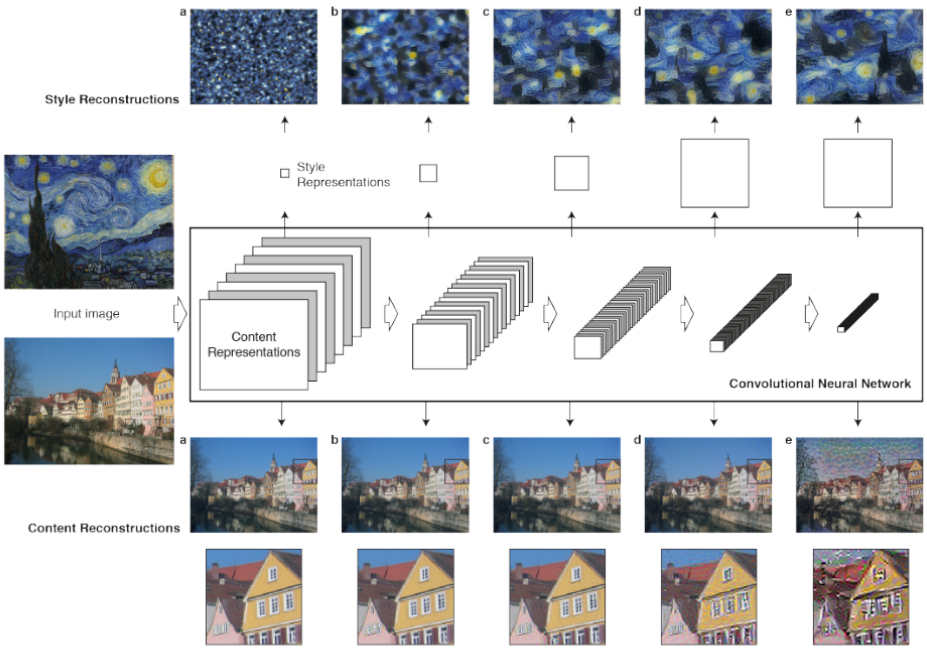


Figure 6: Image synthesis using only style or content loss from one specific layer. Reprinted from[Gatys et al., 2015b].

Following Gatys' work, numerous improvements have been made to improve either the quality or the speed of the output. Li and Wand's paper [Li and Wand, 2016] replaced the Gram matrix with Markov Random Field (MRF) to capture the content and style loss. Similar to using Gram matrix, MRF also makes the

assumption that the style of an image is captured by the local statistical dependencies that can be captured by a pre-trained object detection neural network. It extracts $k \times k$ patches from each convolutional layer and map the target patches to the output patches using nearest neighbor. The L2 loss is then calculated between the mapped nearest neighbor patches. The resulting output often looks more coherent and more similar to the provided style image. A huge drawback of this technique is that computing nearest neighbor for large images is computationally costly and sometimes even infeasible: The computation time for an exact nearest neighbor matching is $O((height * width)^2)$. Another drawback is that it is very dependent on the VGG19's ability to capture the semantics of the style image correctly and matches that with the same object in the content image. Therefore the method often works well when style and content image are semantically similar, but produces less satisfying results when there is a mismatch (as shown in 7).

Figure 7: Style transfer result of [Gatys et al., 2015b] compared with the MRF-based method [Li and Wand, 2016], the latter marked as "Ours". The MRF-based method produces a more coherent looking result, but failed to faithfully reproduce some key features in the content image. For example, the car disappeared since there is no similar object to map to in the style image. One can also notice that the bike in the style image is hallucinated by the MRF-based method. Reprinted from [Li and Wand, 2016].

One intuitive extension of the original neural style includes applying it to videos. [Dushkoff, 2016] used optical flow to guide the stylization process to be temporally coherent throughout a video. The optical flow transfere the style information to its predicted location in the next frame, thus reducing the problem of inconsistentcy between each neighbouring frame and yields interesting results.

Another important improvement on the original algorithm [Gatys et al., 2015b] is the usage of feed-forward CNN to approximate the back-propagation process. Since the backpropagation requires several hundred iterations to slowly transform a white noise image into the desired output, it often takes several minutes on a high-end GPU to complete the style transfer process, depending on the size of the input and output. In contrast, if we can train a feed-forward CNN that learns the style of one style image, then only one single pass through the network is needed, decreasing

the time by over a thousand fold. That was what Ulyanov et al. proposed in his paper [Ulyanov et al., 2016a]. By using a pyramid-shaped generator network conditioned on a white noise image, one can train the generator network to approximate the texture of an image. The texture/style loss is equal to the l2 loss between the style features of the generated image and the target image. The style features are calculated the same way as in [Gatys et al., 2015b]. The generator network is trained using gradient descent. A simple extension from texture generation to style transfer can be achieved by modifying the generator network to be conditioned on the content image and adding the content loss to the total loss. The generator network then learns to do style transfer for one single style image given any content image hundreds of times faster. The results are shown in 8. Later Ulyanov et al. improved the model in his paper [Ulyanov et al., 2016b] by replacing batch normalization [Ioffe and Szegedy, 2015] with instance normalization, where the scale and shift of in the normalization step is no longer approximated by sampling that from each batch, but are directly calculated from each image. Ulyanov et al. explained in his paper that this change makes the network more invariant to contrast change in content image, but it was later proved in [Huang and Belongie, 2017] that the instance normalization does far more than that.

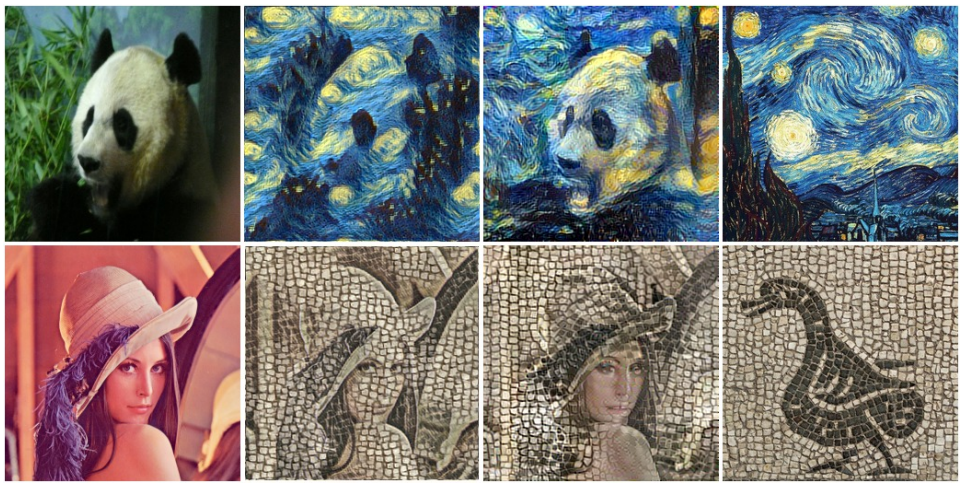


Figure 8: A comparison of the fast forward method with the back propagation method. Reprinted from [Ulyanov et al., 2016a].

Subsequent work soon follows after the so called "fast neural style" is proposed in [Ulyanov et al., 2016a]. One clear drawback of the fast neural style technique is that one must train a network for every single style image, which can take overnight on a high-end GPU (again depending on the input and output size). [Dumoulin et al., 2017] proposed an N-style network that is capable of output multiple styles by additionally conditioning the generator network on an indicator vector where each element in the vector equals to the weight of each style image. The main contribution of that paper is it discovered that the difference between styles can be easily captured by the scale and shift parameter of the instance normalization function. The proposed method replaces instance normalization with conditional instance normalization, which makes the scale and shift vectors style dependent on the weight of each style while keeping the rest of the network the same (as shown in 9). A mixture of two or more styles can be achieved by linearly mixing the scale and shift vectors of various styles. This method greatly reduces the time for training

new networks, because now one can fix the rest of the network and only train on the scale and shift variables for each new style. Examples are shown in 10.

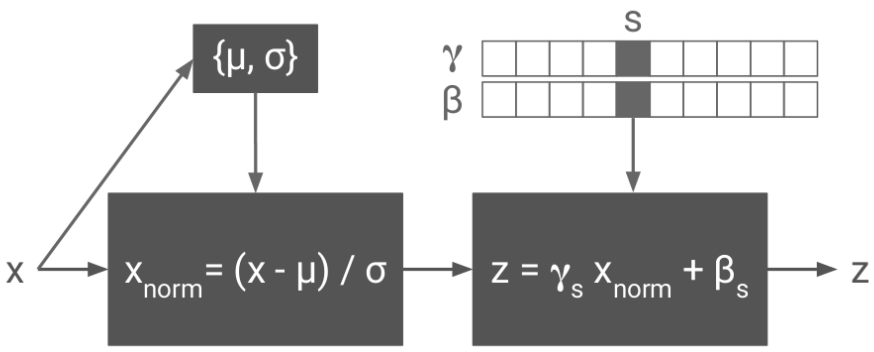


Figure 9: "Conditional instance normalization. The input activation x is normalized across both spatial dimensions and subsequently scaled and shifted using style-dependent parameter vectors γ_s, β_s where s indexes the style label." Reprinted from [Ulyanov et al., 2016a].

Figure 10: Example of the N-style network. Reprinted from [Ulyanov et al., 2016a].

One application of the "fast neural style" is "neural doodle", proposed by Ulyanov first in his blog [Ulyanov, 2016]. Simply put, it is a semantically guided texture generation feed-forward network. By feeding in an additional bitmaps that semantically separates the style image and output image into different regions, one can mimic the texture of each region in the style image and transfer that texture onto the corresponding region in the output. The result is impressive: with only two bitmap image as input (one for the style image and one for the output), one can paint like any grand master, as shown in 11.

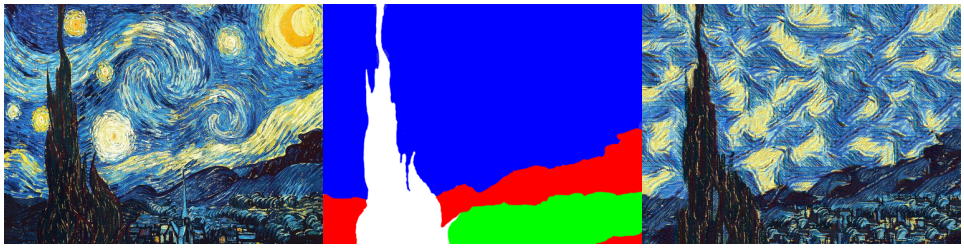


Figure 11: Left: style image. Middle: The semantic mask for both style image and output image. Right: The output image. Reprinted from [Ulyanov, 2016].

The last two recent works we'd like to mention is both on fast style transfer of arbitrary style. To our knowledge, [Chen and Schmidt, 2016] is the first to introduce a fast style transfer method that works for any style. The procedure is referred as "swapping the style". Inspired by the MRF approach [Li and Wand, 2016], the patch based method takes patches generated from the pretrained VGG19 convolutional layers, and for each patch from the content and style image, calculate the normalized cross correlation, and swap each content patch with its closest matching style patch. To find an image that matches the closest to the "style swapped" patches, one can either gradient descent from a white noise image, or to train an inverse generator network that approximates the gradient descent process in one single forward pass. The model achieves a more or less similarly pleasing artistic result with 1/100 of time used by the back propagation on a 300x500 image [Chen and Schmidt, 2016].

Similar to the MRF approach [Li and Wand, 2016], one significant downside of the patch-based method [Chen and Schmidt, 2016] is that calculating the best match between patches takes lots of time and memory. The technique proposed in [Huang and Belongie, 2017] avoided that problem by using a non-patch based method called "Adaptive Instance Normalization" (AdaIN). Similar to the idea of fast multi-style transfer proposed by Dumoulin et al. [Dumoulin et al., 2017], instead of learning the

scale and shift vectors in instance norms for each style, those vectors are computed based on the style input image. An inverse generator network is then trained to project the feature maps back into image space, yielding the final output. The speed increase compared to Chen and Schmidt [Chen and Schmidt, 2016] is 10 to 50 fold depending on the output size, making it a real time algorithm. A comparison of the results from the three fast forward style transfer methods are shown in 12.

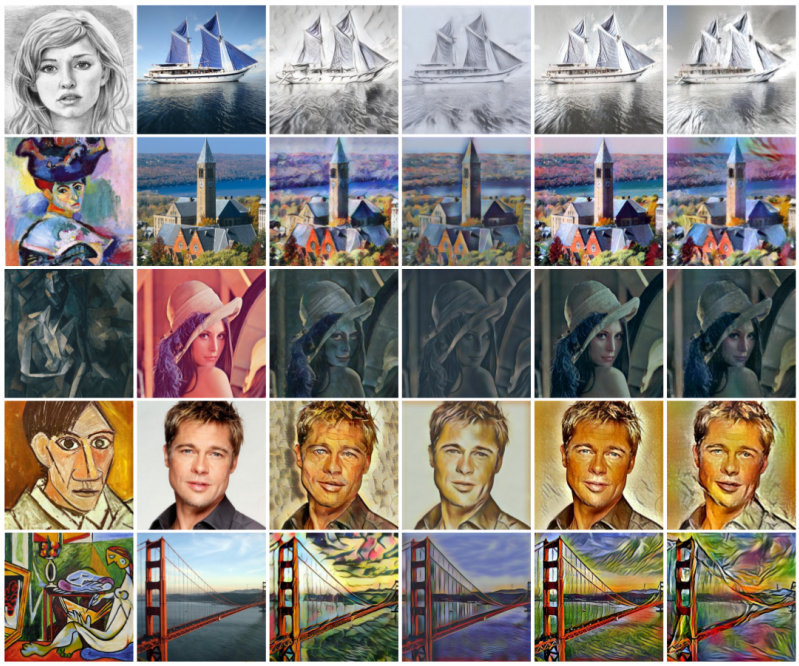


Figure 12: Columns from left to right: Style, Content, [Huang and Belongie, 2017], [Chen and Schmidt, 2016], [Ulyanov et al., 2017], [Gatys et al., 2015b]. Reprinted from [Huang and Belongie, 2017].

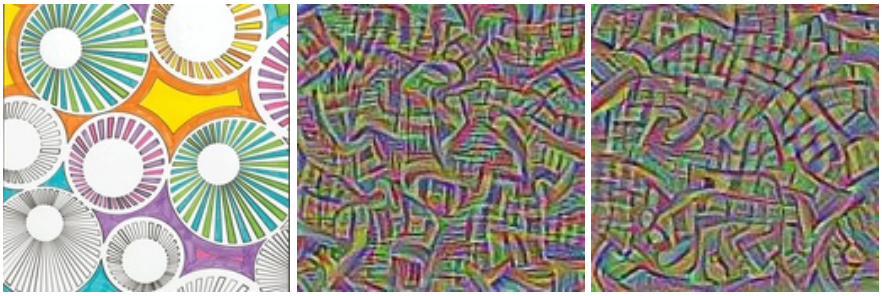
2.3 Controllable stylization

When we re-implemented neural style models [Dumoulin et al., 2017; Gatys et al., 2015b; Li and Wand, 2016; Ulyanov, 2016; Ulyanov et al., 2016a], two questions

continued to puzzle us, both of which involved giving user more control over the stylization process. The first question was: is it possible to control the size of the ”brush” during stylization process. The second one was: is it possible to have different degree of stylization in different areas of the output? In this section, we will delve into these two questions, talk about some solutions that we explored to solve the problem, and the current state-of-the-art approach on them.

2.3.1 Brush size control

Regarding the first problem, in early stages of our research we did not think that there was an easy way to control the brush size. Afterall, the brush size is part of the style and that should be included in the objective function when computing the style loss. (We are using brush size interchangably with style feature size for simplicity, but brush size is only one type of style feature size.) However, when we tried to stylize the same content image twice, the first one being twice as big in width and height than the second one, we found that the size of the brush strokes seem to change with respect to the size of the content image. Smaller content image tend to give a larger brush stroke and larger content image gives smaller brush strokes. Furthermore, we found that the size of the style image matters as well. If we shrink the style image to 50% of its original size, then the brush strokes also seem to shrink approximately 50%. An example is shown in 13



(a) Style image **(b)** Input size: 256×256 **(c)** Input size: 512×512

Figure 13: A comparsion between two different style image input sizes. Notice how the generated texture is more dense when the input size is smaller. Both image is generated using the texture synthesis method in [Gatys et al., 2015a]. The learning rate is set to 10.0 and both run for 10 rounds of back propagation.

Later, we found that the brush size is in fact dictated by the kernel size of the convolutional layer. As a simple example, let us consider a style image with size $m \times n$ and a 1-layer convolutional network. The single convolutional layer has kernel size $3 \times 3 \times 32$, shift equals 1, and stride equals 1. Now let us look at one cell in the convolutional layer. That cell has a receptive field of 3×3. That is, one cell can "see" 3×3 pixels in the style image. Now when the style image is resized to $m/2 \times n/2$, the cell's receptive field does not change, but now the features in that 3×3 field are all resized by a half. When re-creating the style, the one with shrinked style image produces shrinked brush size, because that is the feature being perceived by the convolutional layers.

But how does the size of the content image has to do with the brush size? The brush size is a style feature, which should not be affected by the content. The key lies in the *relative* size between the brush size and the contents. When the content image is shrinked by a half, the ratio between the size of objects and the brush size is decreased by a half because the brush size is unaffected by the content

image. However in human perception, we have an inherit understanding of the size of objects and we make judgement based on the "hidden ruler" in our perception. Therefore to us, the brush size *looks* 2 times larger when content image is shrinked by half, but in fact the absolute brush size stays the same.

Returning to the question of changing brush size, a simple way that follows the discussion above is: to make brush size look x times larger, either increase the size of style image by x or decrease the size of content image (which is also the output size) by x. However, this simple solution is infeasible sometimes due to limits in computational resources. A image 2 times larger will generally take more than twice the amount of time to compute and takes 4 times the memory. Futhermore, enlarging or shrinking the image may sacrifice the resolution of the input images that will in turn affect the output. We have tried to resize the convolutional layers' kernel size, which dictates the receptive field of each layer, but that produces even more artifacts than resizing the inputs.

A solution to this problem was proposed in [Gatys et al., 2016] using a simple yet very effective method, near the end when we were wrapping up our neural style project. Gatys proposed that due to the limit in the receptive field of VGG19, stylization is more successful for images with height and width around 500 pixels. Images larger than that size cannot be effectively captured by the receptive field and style features larger than receptive field cannot be produced due to this limit. In order to produce larger style features on high resolution images, Gatys et al. proposed to first resize the input to around 500 pixels, obtain the stylization result on a low resolution image, then upscale the stylized low resolution image to be the initialization of the high-resolution stylization process. This uses the fact that initialization makes a huge impact on the final stylization output, which was also

observed by [Nikulin and Novak, 2016]. By using the low-resolution as the starting point, the large style features are preserved during the stylization process. The output is a high resolution stylized image that has features with size exceeding that of the receptive field of VGG19.

2.3.2 Spacial control

The second problem we've encountered is that sometimes we would like one region be less stylized than other regions to achieve some artistic effect. For example, if we want to make the main subject of a photograph stand out from the rest, a good way to do so is to stylize everything other than the subject. However in [Gatys et al., 2015b] there is no direct way to achieve such an effect. Inspired by the "neural doodle" [Ulyanov, 2016] we proposed a simple solution to this problem. We use a gray scale guiding mask M with shape same as the content and output image to control the degree of stylization. During the back propagation process of style transfer, for each chosen style layer, we do an element-wise multiplication between the layer and the resized guiding mask M before calculating the gram matrix and the style loss. Thus the graident of the style loss is scaled proportional to the the mask's brightness for each corresponding region. The degree of stylization for each region can be directly controlled by M, which is provided by the user. An example is provided in 14. Similar techniques were also proposed in [Gatys et al., 2016; Selim et al., 2016].

(a) Style image **(b)** Content image **(c)** Mask **(d)** Result

Figure 14: Result of style transfer with spacial control. Regions corresponding to brighter parts of the mask image is stylized more than the darker regions.

2.4 Future directions

Since there are already plenty of implementations of various neural style techniques, this project mainly serves as a learning process of machine perception of image styles. Since [Gatys et al., 2015b], neural style has improved by a lot. The stylization time has decreased by more than 1000 fold using [Chen and Schmidt, 2016; Huang and Belongie, 2017], although at the cost of some decrease in quality. Users now have more control over the stylization process, including style mixing [Dumoulin et al., 2017; Huang and Belongie, 2017], maintaining the color of original content, controlling the brush size (i.e. style feature scale) [Gatys et al., 2016], and the location as well as the degree of stylization [Gatys et al., 2016; Selim et al., 2016]. However, one major drawback of all neural style techniques is their reliance on a pretrained object detection network, such as VGG19, to perform the feature and semantic information extraction. Most networks are trained and tuned for detecting real-world objects. While this works well in a lot of real world applications, such as stylizing photos taken on a smartphone camera using apps, the output decreases in quality when the content image is not real-world photos because the chosen convolutional layer to represent the content cannot semantically understand that

content image. This could improve in the future if we have more labeled data to train object detection networks on non-photo images. Another disadvantage of the style transfer is that it cannot change the object layout of the content image. That might not be a problem if the style used also preserves the scale and positioning of objects depicted, such as van Gogh Starry Night and Wood's American Gothic. However when it comes to artworks such as Munch's The Scream and Dali's The Persistence of Memory, the objects depicted are often geometrically transformed to serve a certain artistic effect. To our knowledge, no current style transfer model is able to achieve such an effect, although in principle they have the ability to do so, given the fact that the pretrained object detection network is able to classify and (depending on the network) even localize semantic objects. Through recreating and transfering artistic styles, we hope that we as Computer Scientists as well as the general public using this technology can gain a new perspective on the meaning of art and style, and hopefully can encourage more future researches in both Computer Science and Art alike.

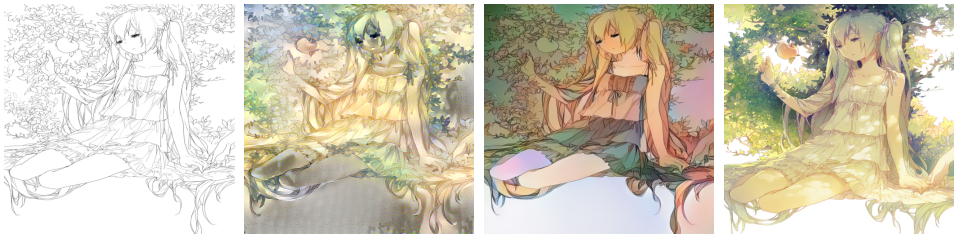
3 Sketch Coloring

3.1 Introduction to Sketch Coloring

The sketch coloring task can be simply described as: given a grayscale sketch, convert that into a plausible-looking colored image of the same scene or object(s). It is common among human painters that before any real painting is done, painters first draw sketches, which contain the layout as well as lighting and shades information. A colored painting can later be painted based on the sketch. We would like to have a program that is given the sketch and automatically completes the coloring step using

the information given in the sketch. Note that this task has two main difficulties. One is that the task requires a semantic understanding of the sketch, without which the coloring would simply be impossible. The other is that the correct output space is inherently ambiguous. A simple example is as follows: if the input is a sketch of an apple, then either a green apple or a red apple, or anything in between is plausible.

The sketch coloring can be seen as a type of neural style transfer in that the grayscale sketch is one style and the colored image is another, but directly applying neural style algorithms to the sketch coloring task will not result in plausible outputs, because the convolutional neural net is pretrained on real world images but not on sketches. It will fail to capture the underlying semantics of the sketch and will therefore fail to output real-looking images. In addition, the assumption that local features captures texture information does not necessarily hold for color information. Some failed examples using style transfer is included in 25. It can also be seen as a harder version of the "coloring grayscale images" task [Iizuka et al., 2016; Larsson et al., 2016; Zhang et al., 2016], since the grayscale image of an object contains more information than the sketch of the same object.



(a) Sketch **(b)** Neural Style [Gatys et al., 2015b] **(c)** PaintsChainer [Yonetsuji, 2016] **(d)** Groundtruth Colored Image

Figure 15: A comparsion between sketch coloring using neural style methods and GAN-based methods. For the neural style method, the sketch was used as content image and ground truth colored image as style image. The neural style transfer program ran for 1500 iterations with learning rate 10.0. The resized colored image and sketch (not available online) are generously provided by Rella.

There are currently two state-of-the-art models for the sketch coloring task, both of which uses CNN to tackle this problem. One of them is the "pix2pix" model [Isola et al., 2016], which is a general supervised method that uses Convolutional Neural Networks to map an image pixel to pixel into another representation of the same image. The other is the "Scribbler" [Sangkloy et al., 2016], which demonstrated the ability to transform sketches of bedrooms, cars, and human portraits into their corresponding colored images. The novelty about this paper compared to pix2pix is that it demonstrated that the user can control the output by overlaying a slight amount of hint for which color to use. The "Scribbler" CNN can learn to fill in colors smartly based on the extra hints without problems of overflowing or making the output look unreal – demonstrating that the network successfully managed to understand the semantic meaning behind the sketches.

Our work is inspired by and closely follows a model called "PaintsChainer" [Yonetsuji, 2016] that applied the techniques in both pix2pix and Scribbler to anime-style sketches. The goal is still to automatically color sketches, but the input is way

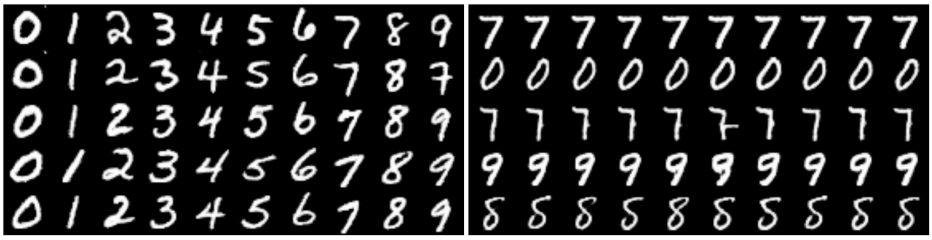
less restricted compared to coloring only one kind of scene or one type of object in pix2pix and Scribbler. It is therefore harder to make realistic looking outputs since the output space is even more uncertain than other more restricted tasks. As an example, 5.1 shows one sketch and 15 different possible outputs created by artists on Pixiv, an artwork submission website. We will discuss the difficulties we've met and the findings we've made when implementing the tensorflow version of the program. We also release a database of around 6000 sketches-colored pairs, both painted by human artists, to encourage research in this area.

3.2 Sketch Coloring Related Work

Previous work such as [Klare and Jain, 2010] mainly focused on image matching between sketches and photos, but not image synthesis. With the huge advances in deep Convolutiona Neural Networks(CNN) and Generative Adversarial Networks(GAN) [Goodfellow et al., 2014] research in recent years, it became possible to turn something as abstract as sketches of shoes, cats, or bags into fully colored real-looking images. Before delving into the details of various sketch coloring models, we would like to first briefly survey the development of GAN in recent years that led to the fast development in generative image synthesis.

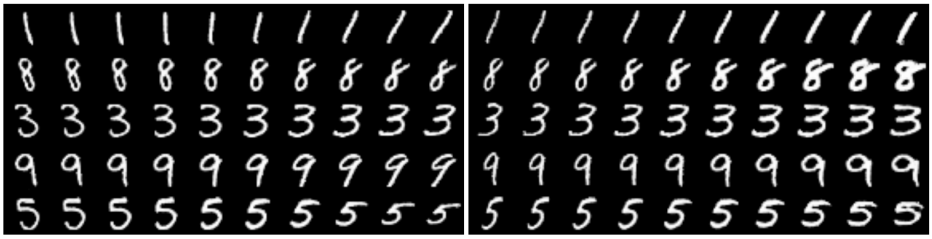
The Generative Adversarial Networks (GAN) was first introduced by Ian Goodfellow in 2014 in his paper [Goodfellow et al., 2014]. The basic structure of GAN is composed of two parts: a generator that is trained to create new instances which closely mimics the training data, and a discriminator that is trained to tell apart the real instances from the ones generated by the generator. Intuitively, one can think of the generator network as trying to fool the discriminator by making the generated instances look more and more like the training data, while the discriminator tries not

to be fooled. Mathematically speaking, the generator works via the principle of maximum likelihood [Goodfellow, 2016]: $model^* = argmax_{model} \prod_{i=1}^{m} p_{model}(x^i)$ where p is the probability that the model assigns to the training instance i. In practice, such a model is approximated by parametrizing it with parameters θ. Thus the objective becomes $\theta^* = argmax_{\theta} \prod_{i=1}^{m} p_{model}(x^i; \theta)$. The paper proved theoretically that the global optimal result is achieved if and only if the probability distribution of the generated instances is exactly the same as that of the real data, and that given enough capacity as well as enough training, the generated probability distribution will converge to the global optimal result.



(a) Varying c_1 on InfoGAN (Digit type)

(b) Varying c_1 on regular GAN (No clear meaning)



(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)

(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

Figure 16: Manipulating latent codes on MNIST. Reprinted from [Chen et al., 2016]

However, the original GAN model suffers from unstable training and was very dependent on choosing the correct hyperparameters and on weight initialization. The actual result differs from the theoratical one because the theoretical result does

not apply directly to parameterized model such as multi layer neural networks or CNN [Goodfellow et al., 2014]. The paper [Salimans et al., 2016] introduced several useful tricks in training GAN networks that can successfully make it converge. The DCGAN network introduced by Radford et al. in [Radford et al., 2015] further improves GAN by combining GAN with CNN and made the training process more stable. Thereafter, all convolutional network for both the generator as well as the discriminator (except the last layer) with batch normalization becomes the most popular structure for CNN based GAN. Some other improvements include InfoGAN [Chen et al., 2016], which added an extra loss term to "maximize the mutual information between a small subset of the latent variables and the observation". This change results in more sensible outputs that can be explicitly controlled through changing the latent variables. For example, one can change the orientation of the generated object by changing only one variable, as shown in 16. Interactive generative adversarial networks b(iGAN) developed by Zhu et al. [Zhu et al., 2016] shows that with GAN one can generate realistic images using just a few brushstrokes. Introspective Adversarial Networks (IAN) by Brock et al. [Brock et al., 2016] generates realistic looking images by taking instructions from users and making slight modifications to a given image. For more information about GAN, we refer the reader to [Goodfellow, 2016].

Figure 17: Example pix2pix model outputs on Google Maps. Reprinted from [Isola et al., 2016]

In [Isola et al., 2016], Isola et al. introduced the pix2pix model that can be applied to a wide range of image-to-image translation tasks. Image-to-image translation task is defined as mapping or "translating" one image into another, such as labels to street scene, aerial photo to map (shown in 17), black-and-white to color, day to night, and edges to photo(shown in 18), etc. The mapping can be one-to-one, or one-to-many. In order to build a general framework for such class of tasks, pix2pix used a generator network structure called U-Net [Ronneberger et al., 2015], which was first used in medical setting to segment biomedical images. What is different between U-Net and the traditional encoder-decoder structure is that U-Net added skip connections between the encoder layers and the corresponding decoder layers, making it easier for the network to use the data available during encoding phase. Traditional decoders use deconvolutional network to enlarge the latent feature matrix provided by the encoder. Therefore it is inherently less accurate

in deciding the precise location of a boundary or a small object. U-Net resolved that problem through the use of skip layers and thus making the precise location information available to the decoder network in case it needs such information. Another innovation made by pix2pix was that it used a special discriminator structure called PatchGAN. The PatchGAN was inspired by neural style transfer: instead of outputting one single number for each image, the discriminator now classifies NxN patches and output a score for each patch. The scores are then averaged to give an overall score for the image. The PatchGAN was added to counter the negative effects produced by a naive l2 or l1 loss function, which tends to produce blurry results and often collapses to the statistically best result but fails to capture the wide range of possible outputs. It was observed that by adding the PatchGAN the outputs became sharper and more colorful[Isola et al., 2016]. Furthermore, the pix2pix network was shown to work relatively well even with small datasets (with sample pairs on the order of 400 or less).

Figure 18: The pix2pix model on edge2color task. Reprinted from [Isola et al., 2016]

The Scribbler model from [Sangkloy et al., 2016] can be seen as a more domain-specific kind of Conditional GAN. The main innovation is its ability to adapt to

user controls. Previous GAN systems gives the user little control over what and how the output is generated. One exception was InfoGAN [Chen et al., 2016] that produced outputs conditioned on semantically meaningful latent variables. The Scribbler can take a sketch of a portrait, a room, or a car, and outputs a colored version of the depicted scene. Different from pix2pix, the Scribbler model used a traditional encoder-decoder structure instead of the U-Net, and the discriminator was not conditioned on the input. During training, random "scribbles" or color strokes were generated by randomly sampling from the target images and are added as part of the input. The result was that the network learned to make use of the provided hints and incorporate the colors into the output. The user was therefore able to control the color of objects simply by putting color strokes on the object. Examples are shown in 19.

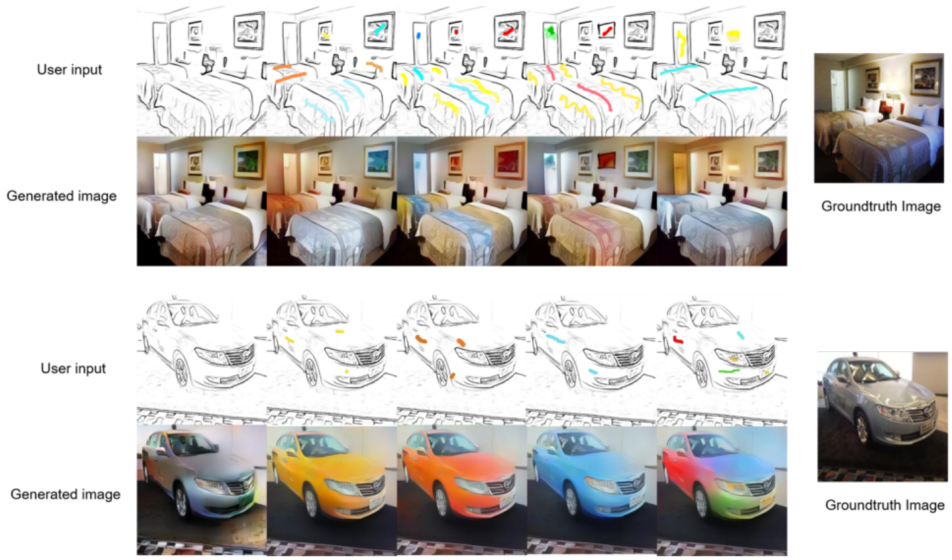


Figure 19: Guided sketch colorization results of the Scribbler model. Without color strokes, Scribber "produces synthesis results that closely follow the input sketch (leftmost column)". With color strokes, Scribber "can adapt the synthesis results to satisfy different color constraints". Taken from [Sangkloy et al., 2016]

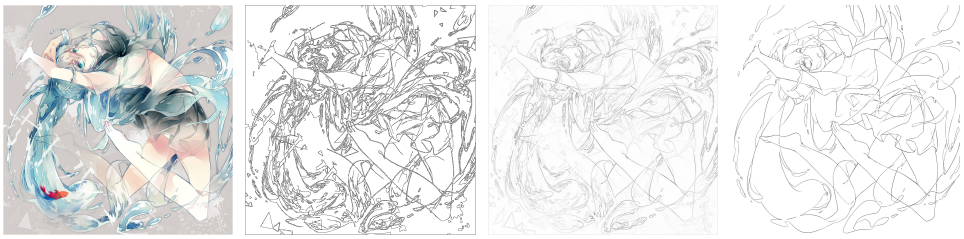
Yonetsuji's work "PaintsChainer", first described in his blog [Yonetsuji, 2016], is a user-controllable sketch coloring program for anime-style sketches. It seems to have taken ideas from both pix2pix and Scribber – it used the U-Net as the generator structure and also supports user control just like the Scribbler. The innovations it made were mainly twofold. One was that it showed the conditional GAN could be applied to a much wider range of objects depicted in anime-style sketches. Anime-style sketches often depict human figures, but also sceneries, animals, machines, and anything else one can imagine. It is not limited to front-facing human portraits or bedrooms. Nor is the target output well defined due to the wide range of possible coloring styles in anime images. The second innovation it made was that it introduced a nice way to automatically generate sketch-like edge

graphs from the original colored images to be used for training. By far we have not talked about the dataset, but dataset is crucial for machine learning tasks. In pix2pix, the dataset is required to be pairwise and it uses existing pairwise datasets for all of its experiments. Notably, the edge-to-shoe and edge-to-bag task dataset taken from [Zhu et al., 2016] using HED edge detector[Xie and Tu, 2015], while the sketch-to-shoe and sketch-to-bag task takes human drawn sketches from [Eitz et al., 2012]. Similarly, the Scribbler model uses a mixture of automatically generated edges, style transfer, and human inputs as its training dataset. We tested the online demo of the pix2pix model [Hesse, 2017] by drawing a random sketch, shown in 20 The result was acceptable, but one can see a lot of artifacts caused by hallucinating furs and even another cat's face appearing in the background.

Figure 20: Sketch to cat model output of the online pix2pix demo [Hesse, 2017].

Despite the artifacts, one can see that although the training data is edges of cat images, it generalizes pretty well to real sketches, both cat and non-cat. In order to generate "sketches" of the colored image, the PaintsChainer program used the difference between the original and the dilated version of the image. It has

advantage over other edge detector models in that the generated sketches looks more like real sketches and have less noise than edge detectors. A comparison of the edge generation techniques are shown in 21. One downside about the sketch generation algorithm is that when two objects have similar color, their boundary is likely to be unclear in the generated sketches. We will talk about how we counter these kind of problems in section 3.3.1.



(a) Colored **(b)** Canny edge detection **(c)** Dilation **(d)** Groundtruth Sketch

Figure 21: A comparsion between different sketch generation techniques. The Canny Edge Detection method uses lower threshold 50 and upper threshold 200. The dilation method uses a 3×3 matrix filled with one as the dilation filter. Colored and ground truth sketch by Reika on Pixiv.

3.3 Dataset

Before we talk about our reimplementation of [Yonetsuji, 2016], we must first obtain a dataset for training. [Yonetsuji, 2016] did not specify the source of the training data but did mention that the overall size of the dataset is around 600,000 images. To get such a large amount of colored anime style images, we scraped Pixiv, a mainly Japanese online artist platform that contain mainly anime style images, using a list of tags called "xxx users入り"[1], which means that xxx number of users has starred

[1]A complete list of those tags can be found on hyperlinks from this pagehttp://dic.pixiv.net/a/users入り

the artwork. The reason we chose such a tag is to collect only the most popular artworks, which tend to be more likely painted by professional or semi-professional artists. We collected all artworks that has been starred by at least 500 users using a Scraper tool called pixiv downloader[2]. The total number of images scraped this way is around 1.5 million. We will make the ids of those image pages available at Pixiv Dataset. The dataset is then cleaned to remove grayscale and monochrome images as well as images with height/width ratio below 0.5 or above 2.0. The remaining images amount to around 1 million. The images are first cropped to a square, then resized to 128×128 and 512×512.

3.3.1 Sketch Generation

The method we first used was proposed in [Yonetsuji, 2016], where sketches are generated by first taking the dilation of the grayscaled input image. Then it calculates the absolute value of the difference between the dilated image and the grayscaled image. Since the pixels where there is no color change in neighboring pixels is left unchanged after dilation, the resulting image has black pixels everywhere except along the boarders, where the pixels are white. Finally, the output is negated to give black sketches on a white background. A step-by-step example is shown in 22.

[2]The pixiv downloader software is developed by Nandaka et al. and is publicly available at `https://github.com/Nandaka/PixivUtil2`

(a) Input **(b)** Grayscale **(c)** Dilation **(d)** L1 **(e)** Output

Figure 22: A step-by-step illustration of the dilation sketch generation method. Original image by Rella.

3.4 Preliminary Experiments

As a preliminary experiment, we tried the PaintsChainer released source code [3] on our dataset, but the result was not very satisfactory. The network worked well on a separate validation dataset with generated sketches as input, but it failed to output images of high quality on real sketches. We contacted the author, and he did mention that the source code released did not entirely reflect the code he used for training the model, since he improved multiple times on the code as the training proceeded. He also mentioned taking a lot of time to fine tune GAN training. Not being able to reproduce his training result, we went on to implement our own version of the sketch coloring task in Tensorflow.

3.5 Method and Network Structure

Given the input s, which is created by concatenating the sketch with random "hints" – color patches randomly sampled from the target image, we would like to learn a generative model G that maps the input s to the target colored image t: $G : s \rightarrow t$. Our objective function is composed of several loss functions.

[3] https://github.com/pfnet/PaintsChainer

The conditional GAN loss function can be written as:

$$L_{cGAN}(G, D) = E_{s,t\sim p_{data}(s,t)}[\log D(s, t)] + E_{s\sim p_{data}(s)}[\log (1 - D(s, G(s)))] \tag{1}$$

$p_{data}(s, t)$ is the real probability distribution of sketch and colored image pairs. However, we often do not have access to the real sketch. Therefore we approximate the probability distribution $s, t \sim p_{data}(s, t)$ by $s', t \sim p_{data}(s', t)$, where s' is the sketches generated using either dilation or a trained neural network. The generative model G is minimizing the loss function while the discriminative model D is maximizing the same function. The optimal generator is $G^* = arg\ min_G max_D L_{cGAN}(G, D)$.

However, GAN training is often unstable and previous work has found that it works better to add either an L1 or L2 loss function: $L_{L1}(G) = E_{s',t\sim p_{data}(s',t)}||t - G(s)||_1$.

In addition, we've found that adding a "sketch loss" will help the model avoid color overflowing problems and make the output look more realistic. The sketch loss, also used by [Yonetsuji, 2016], is essentially a Cycle-consistency function. Cycle-consistency function, also used by [Taigman et al., 2016; Zhu et al., 2017], says that if we have two generator models G and f: G converts the source s into the target t and f does the other way around, then a good loss function is :$L_{cycle} = ||f(t) - f(G(s))||$. In terms of our task, this loss function can be explained as follows: the sketch generated conditioned on the real colored image should be the same as the sketch generated conditioned on the output of G.

Lastly, an anisotropic total variation loss [Mahendran and Vedaldi, 2015; Rudin et al., 1992] of the generated output is added: $L_{TV} = \sum_{i,j}((G(s)_{i,j+1} - G(s)_{i,j})^2 +$

$(G(s)_{i+1,j} - G(s)_{i,j})^2)^{1/2}$ where i, j represent the pixel at coordinate (i, j). The total variation loss is commonly used in image synthesis tasks to encourage less noisy outputs. The overall loss function is: $L = \alpha_{cGAN} L_{cGAN} + \alpha_{L1} L_{L1} + \alpha_{cycle} L_{cycle} + \alpha_{TV} L_{TV}$.'

Following [Yonetsuji, 2016], we used an 18 layer deep U-Net as our generator architecture. More specifically, let ck denotes a Convolution-BatchNorm-LeakyReLU layer with out channel=k, shift=3 and stride=1, and Ck denotes the same structure except with shift=4 and stride=2. Similarly CDk denotes a Deconvolution-BatchNorm-LeakyReLU layer with shift = 4 and stride = 2. The skip connection is between encoder layer Ci and decoder layer CDi (One exception is $CD512$ is connected directly to $c512$, not $C512$). The U-Net architecture is as follows:

encoder: $c32 - C64 - c64 - C128 - c128 - C256 - c256 - C512 - c512$

decoder: $CD512 - c256 - CD256 - c128 - CD128 - c64 - CD64 - c32 - c3$

In order to make the model more robust, we augment our training data by resizing the input images first to $13/12$ of the input height and width, then randomly cropping back to the input size (the same method is used in [Isola et al., 2016; Yonetsuji, 2016]). In addition, a gaussian distributed random number r_s is added to all input channels as well as the target output. Finally, if the input sketch is generated using dilation, the width of the sketch is controlled by the $d \times d$ dilation filter. We vary the d as a random variable between 0 and d_{max} lower bounded by d_{min} (That is $d := max(uniform(0, d_{max}), d_{min})$). With regard to network weight initialization, we use the same method as [Isola et al., 2016], initializing all weights with a 0-mean gaussian distribution with a standard deviation of 0.02. Finally, all target outputs are converted to Lab color space. Conventionally this is the default output color space of numerous image coloring papers, because their

input is a gray-scale image, which they can use directly as the L channel of the Lab color space. We think that because sketch may contain some lighting information, using Lab color space may make the objective function easier to learn.

3.6 Experiments

Using the network structure described above, my program still suffered from the same problem as using the PaintsChainer source code: overfitting the generated sketch and unable to generalize to real sketches. We tried to switch to pix2pix generator model, but that did not solve the problem. This phenomenon was not mentioned in source code or in [Yonetsuji, 2016], therefore we first thought it was due to the dataset we used. After trying several other datasets of various size and reaching the same failure mode, we suspected that the failure could be largely accounted by inaccurate sketch generation method.

3.6.1 Neural Network Based Sketch Generation

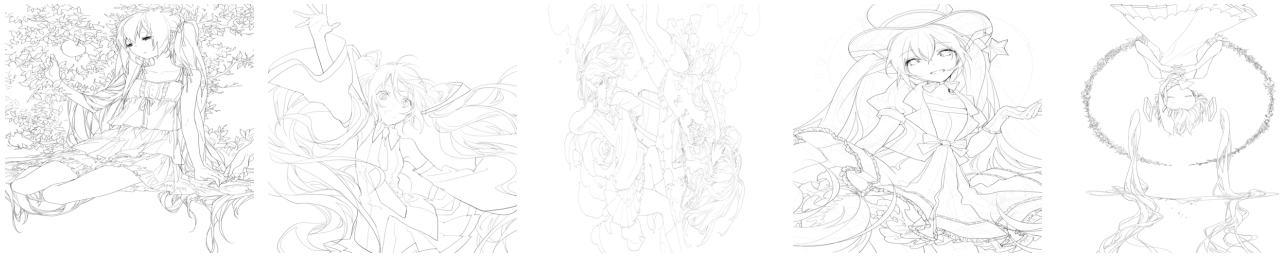
To study the difference between sketches generated through dilation and real sketches, we collected around 6200 sketch-colored image pairs from Pixiv[4]. We used two tags ”塗ってみた”(tried to color) and ”塗らせていただきました”(it was my pleasure to color this) to collect the colored images. Then we scanned the description of the colored paints, hoping to find an url to the original sketch. After getting both the sketch and the colored image, we manually go through each image to make sure that the sketch indeed corresponds to the colored image and there is no misalignment. We found that there is often a huge discrepency between the sketch and the colored image. The sketch usually does not include the background

[4]We decide to again share the ids of the pairs to encourage research in this area: Pixiv Dataset

but the colored image does. We suspect that this is a big reason why generated sketches does not generalize to real sketches. Therefore we proposed a second way to generate sketches: by using a neural network.

Alice Bluestar Takakura Yukimirai 天空魚

(a) Original Image



(b) Ground Truth Sketches

(c) Dilation Generated Sketches

(d) Neural Network Generated Sketches

Figure 23: A comparison of Dilation and Neural Network generated sketches. Art pieces provided by Rella.

It is worth mentioning that in [Sangkloy et al., 2016], it used five different type of sketches for human figures to make sure the network generalizes to real sketches. We proved that edge detection method does not work well on anime style images in 3.3.1. We also tried using style transfer to generate sketches. The result was not so satisfactory 24. The style transfer network uses pretrained object detection network such as VGG19 to extract local features and as previous experiments implied, the VGG19 was not good at extracting semantic features from paintings effectively and thus fails to capture the ”sketches” as a style. The generated images are often distorted, blurry, or biased towards sketches in one certain direction.

(a) Sketch as the Style Image

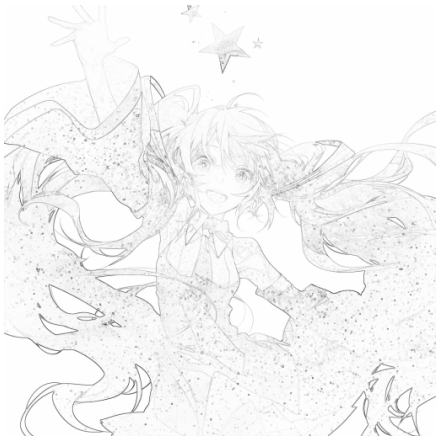
(b) Colored Image as the Content Image

(c) Neural Style [Gatys et al., 2015b] Output

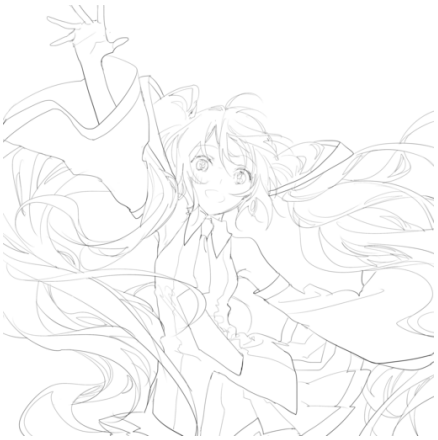
Figure 24: An example of Neural Style generated sketches. Even though the ground truth sketch is already provided as the style image, it looks less like the ground truth sketch than the dilation generated sketch or the neural network generated sketch. It contains too much grayscale and shading information not usually present in human drawn sketches. The problem becomes worse when the style image is not the ground truth sketch.

We retrained the model by using the method in [Sangkloy et al., 2016], which is randomly mixing multiple types of generated sketches to discourage overfitting. The result is shown in 25. Comparing to the previous model trained on only dilation sketches, the output contains a lot less noise, but it is still not comparable to that of the PaintsChainer online demo. Notably, we found that there is still a large amount

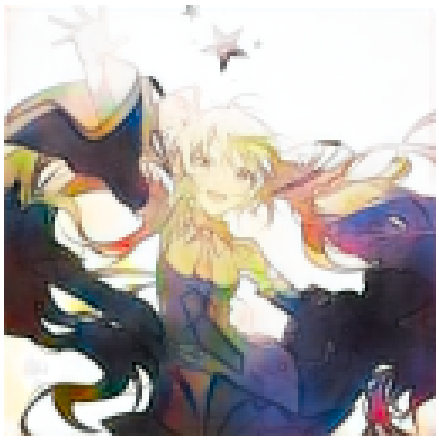
of discrepancy between the output conditioned on generated sketches versus real sketches. The output conditioned on generated sketches, which we used as training input, looked a lot realistic compared to the output conditioned on real sketches.



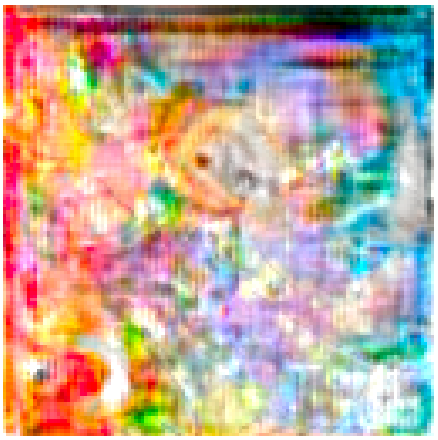
(a) Dilation Sketch



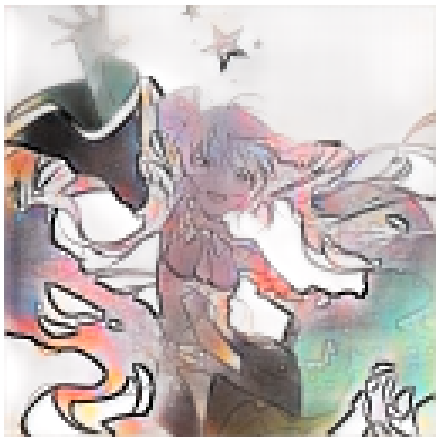
(b) Ground Truth Sketch



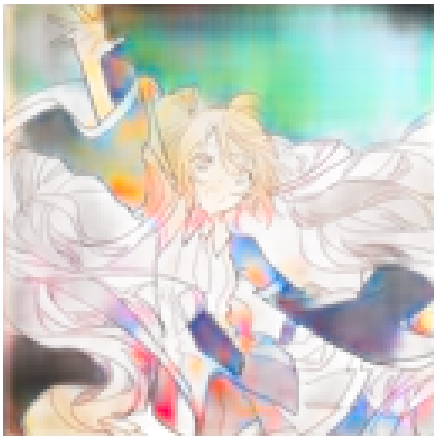
(c) Dilation Sketch Input, model trained using dilation generated sketches



(d) Ground Truth Sketch Input, model trained using dilation generated sketches



(e) Dilation Sketch Input, model trained using Neural Network generated sketches



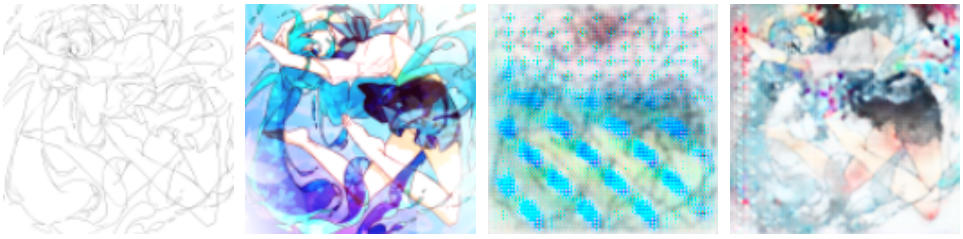
(f) Ground Truth Sketch Input, model trained using Neural Network generated sketches

Figure 25: A comparsion among sketch coloring models trained on dilation generated sketches, neural network generated sketches, and a random mix of the two type of sketches. The models here are trained using WGAN, described in 3.6.2. Colored image and ground truth sketch are provided by Rella.

3.6.2 Colorful images

A major problem in the generated output was the lack of color. The model preferred sepia or grayish images. As described in [Larsson et al., 2016; Zhang et al., 2016], the lack of color is a common problem in coloring tasks and it is often caused by using the l1 or l2 loss, which tends to output the statistically best result. In addition, it is mentioned in [Sangkloy et al., 2016; Yonetsuji, 2016] that the use of GAN greatly increases the colorfulness of the output. In order to solve this problem, we tried two methods: one is replacing the GAN model by the more robust WGAN model [Arjovsky et al., 2017], and the other is to replace the output real valued LAB color space by the quantized LAB color space described in [Zhang et al., 2016]. We show that WGAN is an effective variation of GAN, but color space quantization may be too costly for the potential benefits it brings.

WGAN Proposed in [Arjovsky et al., 2017], the WGAN model replaces the KL divergence used in GAN model by an approximation of the Earth Mover (EM) distance. It is shown that KL divergence, as well as several other common measures of distance between two distributions, suffers from non-continuous and even sometimes undefined loss function, especially when the two distributions does not have enough overlap. By using an approximation to the EM distance, the WGAN model provides a more stable gradient defined everywhere. A second important contribution of the WGAN model is that the approximated EM distance surves as a much more interpretable loss function that resembles the quality of the output. By replacing the GAN with WGAN, we found that it indeed resulted in better performance and more colorful outputs, as shown in 26. Therefore, the WGAN model is adapted into our architecture.



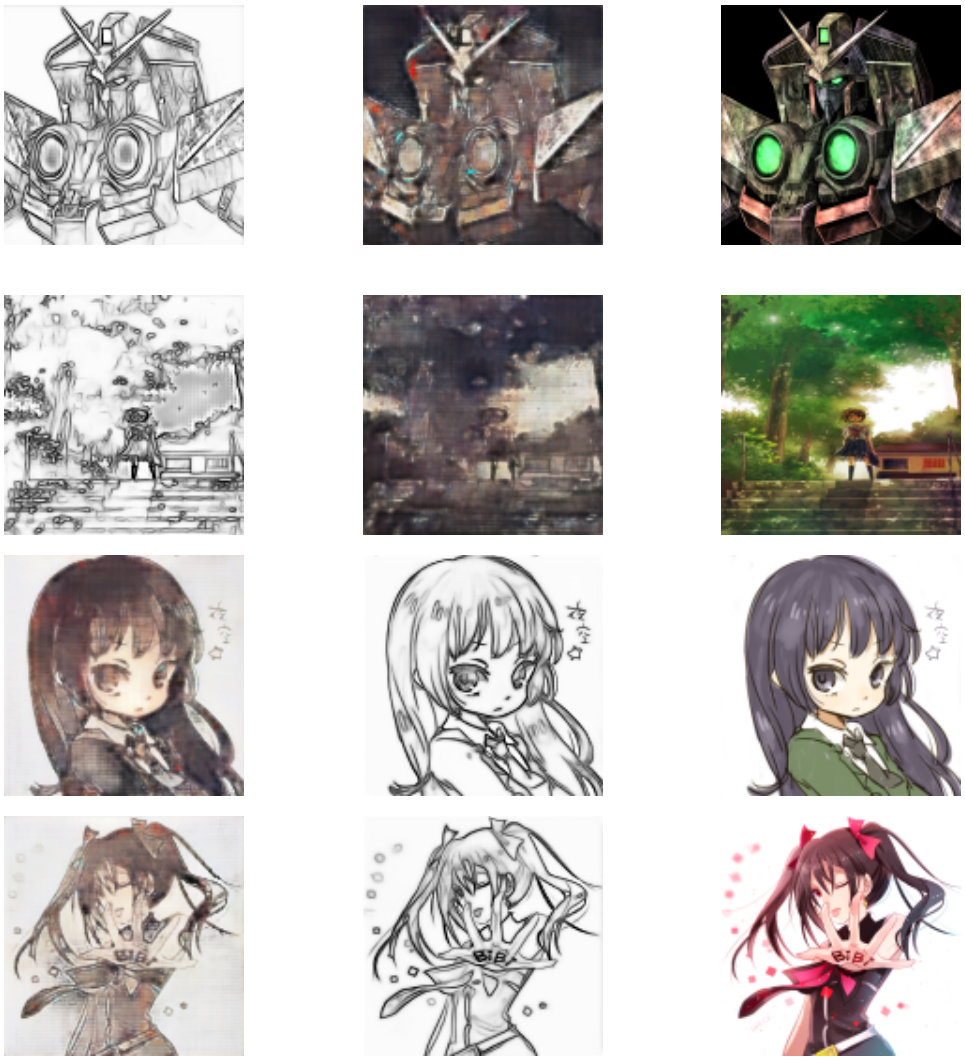
(a) Input **(b)** One possible colorization **(c)** GAN **(d)** WGAN

Figure 26: A comparison between GAN and WGAN adversarial model. Both model is trained on a toy dataset 5.1 for 2000 epochs.

Quantized Color Grid In [Zhang et al., 2016], it is shown that l1 distance, where the goal is to minize euclidean error between the generated image and the ground truth, is not an ideal loss function for colorization tasks. As shown in 5.1, there are simply too many possible colors for objects, especially in our task where imaginary anime characters often have more colorful eyes and hairs so that they can be easily distinguished from one another. As a result, the optimal solution under euclidean distance tend to have less saturated colors. [Zhang et al., 2016] proposed to replace the standard output space $R^{H\times W\times 2}$ with "a distribution over quantized color value outputs" $R^{H\times W\times Q}$, where Q is the number of color bins. Treating the problem as a multinomial classification task, the model assigns each color bin with a probability. The loss is therefore the expected cross entropy between the ground truth and the output. The $R^{H\times W\times Q}$ output is mapped back into $R^{H\times W\times 2}$ using annealed mean: $f_T(z) = \frac{exp(log(z)/T)}{\sum_q exp(log(z_q)/T)}$. z is the probability distribution vector and T is the annealing temperature, which is set to 0.38 in the paper.

We experimented with adding the color space quantization to our model. One difference between our implementation and [Zhang et al., 2016] is that their grayscale

image colorization task's output space is $R^{H \times W \times 2}$ (luminance is provided in the input), but our sketch colorization task has output space $R^{H \times W \times 3}$. As a result, the number of bins Q is increased to $Q^(3/2)$ if we are to keep the range of colors that each bin contains as a constant. Unfortunately, we found that using 216 bins that dividing the LAB color space evenly already slows down training by more than 5 fold. In theory the quantized color space should yield more colorful results, but in practice we abandoned this method after training for 24 hours did not yield much improvement. Some examples are shown in 27.



Quantized Color Grid Model Output | Input | Ground Truth Colorization

Figure 27: Quantized Color Grid model example output. The model was trained for less than half an epoch and more than 24 hours with a GTX 1070 graphics card. No significant improvements over the models trained over the same period of time was observed.[5]

[5]We would like to cite the authors but unfortunately the sources of the images used here were lost. We apologize in advance. Please contact us if you are the author and have any questions or concerns.

3.6.3 Decaying l1 loss

Even though adding WGAN and having two sketch generation methods resulted in improved output, it is still not comparable with the PaintsChainer demo, especially when no hint is present. Out of the numerous experiments we tried , only the decaying l1 loss stood out as really effective in our sketch coloring task. The idea comes from multi-stage training in [Sangkloy et al., 2016; Yonetsuji, 2016], where the training is separated into two stages: one without adversarial loss and one with adversarial loss. The intuition behind the two-stage training process is that training an adversarial network is hard and often takes long before it can generate anything meaningful looking. In conditional GAN, we are able to fast-forward through the first half of training by using a more consistant loss function like l1 loss. This enables the network to quickly learn to produce meaningful yet imperfect results. The second stage then uses the GAN in addition to the l1 loss so that the model can have an output distribution that resembles more closely to the actual output distribution.

Our decaying l1 loss method is similar to the two-stage idea, but provides a more smooth transition in between. During our experiments, we found that the two-stage training process often resulted in sudden decrease in the loss of the discriminator after a period of training. A decrease in WGAN discriminator loss usually indicates that the generative model's output distribution is closer to actual distribution and the discriminator is harder to tell the difference between the two. That was indeed the case, but we found that the sudden collapse of the discriminator loss also resulted in unstable training. The discriminator was no longer able to tell the difference between the real and the generated output distribution, even though

there is a clear difference between the two. As a result, the generated result often contains unrealistic-looking colors. Although this can be prevented by manually terminating the training process earlier on, this kind of rescue is not desired in an end-to-end training process.

We propose a simple fix to this issue: we include the adversarial loss into the training from the beginning, but set the weight of the adversarial loss to be a small value. After each epoch over the training data, we decrease the weight of the l1 loss by α. That is to say, the relative weight between the adversarial loss and the l1 loss increases by α after each epoch, putting more and more emphasis on WGAN as training goes on. In order to prevent l1 loss to be decreased indefinitely, we add a lower bound to l1 loss weight. The result of such a change is shown in 28

Alice Bluestar Takakura Yukimirai 天空魚

(a) Original Image

(b) No l1 loss weight decay, ground truth sketch as input, no hint.

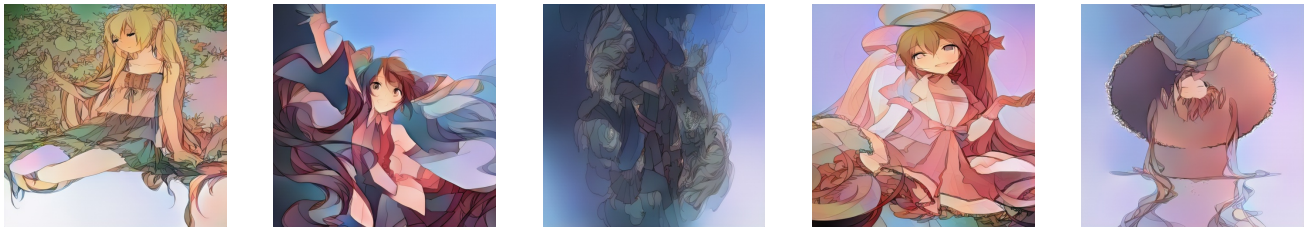
(c) No l1 loss weight decay, ground truth sketch as input, with random hints.

(d) With l1 loss weight decay, ground truth sketch as input, no random hints.

(e) With l1 loss weight decay, ground truth sketch as input, with random hints.

Figure 28: A comparison of model with and without l1 weight decay. For the model trained with l1 weight decay, we set decay rate = 0.9 per epoch and decay lower bound = 50.0. The model with l1 weight decay trained for around 9 epochs on a dataset with around 300k images. The model without l1 weight decay trained for around 7 epochs on the same dataset. Art pieces provided by Rella.

(a) Original Image



(b) PaintsChainer Pretrained Model [Yonetsuji, 2016]

(c) Trained 512×512 model, ground truth sketches as input, no hint.

(d) Trained 512×512 model, ground truth sketches as input, with random hints.

(e) Trained 512×512 model, dilation sketches as input, no hint.

(f) Trained 512×512 model, dilation sketches as input, with random hints.

Figure 29: A comparison of our model with pretrained PaintsChainer model. Art pieces provided by Rella.

3.7 Results

We trained a final version of our model using the settings in 5.2. The sample outputs are compared with the outputs of PaintsChainer online in 29. Notice that the output is a little bit worse than that of the 128×128 model in 23, especially when hint is not present, even though they both trained for 7 epochs (but on a slightly different dataset), implying that even though larger image contains more information to be used, it is harder for the same network to incorporate all the extra information. Also notice that the model does far better on dilation generated sketches, indicating that there are extra information in the dilation sketches not present in human drawn sketches.

Even though we made improvements compared to simply training our dataset using the PaintsChainer source code, our result is still not nearly as natural looking as the pretrained model. The outputs when hints were provided is, in our opinion, comparable to the pretrained model, but the performance of our model without hints is not on-par with the current state-of-the-art. We also noticed that despite all our effort to prevent the model to overfit generated sketches, it still performed far better on those inputs compared to on real human drawn sketches. In fact, when we use the color-sketch image pairs as our validation data, we often see a performance gap of around 0.05 in terms of average l1 loss over all pixels between training data and validation data. This is a sign of the model overfitting the input data. This can be explained by the fact that, no matter how similar the two look to a human, the sketches generated from the colored image contains enough information that is absent in human drawn sketches to cause such a performance gap. A much larger sketch-colored image pair dataset would likely solve the problem, but obtaining

such dataset is expensive. We would like to leave this open question to future researchers.

3.8 Conclusion

In this section, we explored different models for sketch colorization and reimplemented one of them – PaintsChainer. The source code of PaintsChainer did not work out of the box for the dataset we collected. This is likely due to the fact that training and tuning GAN is notoriously hard. In the process of improving the model's performance, we found two things are crucial to natural looking outputs – the use of a neural network generated sketch mixed with dilation generated sketch as input, swapping GAN with WGAN, and the decaying l1 loss. Our model still cannot perform nearly as well as the pretrained PaintsChainer model, especially when no hint is provided as part of the input.

Despite all the downsides of our model, we hope that our research can shine some light onto the training process of a sketch colorization network and the difficulties one may encounter when implementing one. We released our training data as well as the sketch-colored image pairs data and we look forward to more researches on this field.

4 Hypernym Extraction from Text

4.1 Introduction

The task of Hypernym extraction, or more generally relation extraction, is to extract both implicit and explicitly mentioned relations between pair of words or phrases. Given a sentence S composed of tokens $t_1, t_2, t_3, \ldots, t_n$ and relations $R_1, R_2, \ldots, R_n$, find all pairs such that (t_i, R_j, t_k) holds for relations of interest. The possible classes of relation explored in previous works include: part-of, located-at, owns, has instance, hypernym of, etc. Relation extraction is useful for constructing large databases from raw corpora such as web pages, encyclopedias, news, and scientific papers. Arguably one of the successful real world application that may involve relation extraction is the Google Featured Snippet [goo], where a direct answer to a question query along with the webpage from which the answer is cited is returned as the top search result. Although the technical details of the Google Featured Snippet is unknown, relation extraction is one reasonable approach if one wants to build such a system.

Prior to the neural network regaining its popularity, relation extraction was mainly accomplished using carefully chosen features and grammatical analysis of the input sentence, such as in [Agichtein and Gravano, 2000; Banko et al., 2007; Brin, 1998; Etzioni et al., 2005; Kambhatla, 2004; Zelenko et al., 2003; Zhao and Grishman, 2005]. Since [Mikolov et al., 2013a,c; Pennington et al., 2014] , embeddings trained on huge corpora using unsupervised methods such as skip-gram and Continuous Bag of Words (cbow) became the state-of-the-art language models. The word embeddings was shown to contain a rich amount of semantic information useful for relation extraction as well as a range of other tasks, including

but not limited to analogy classification [Mikolov et al., 2013c], machine translation [Mikolov et al., 2013b], named entity recognition[Santos and Guimaraes, 2015], document representation [Kusner et al., 2015], and sentiment analysis [Dos Santos and Gatti, 2014] . It has been shown that the traditionally hand-picked features can be replaced by the word embeddings and still achieve state-of-the-art performance in some tasks [Nguyen and Grishman, 2015]. Numerous methods using LSTM [Huang et al., 2015; Xu et al., 2015] and CNN [Nguyen and Grishman, 2015], was proposed to extract the information in the word embeddings and use that information for relation extraction tasks.

We adapted a CNN-based model for relation extraction focusing on only one single type of relation: hypernym extraction. That is to say, we wish to find all pairs (A, B) such that "A is a B" or "B is a A" holds. The corpus that we mainly focuses on is a collection of Computer Science research papers [6]. We hope through hypernym extraction, we can obtain a highly accurate database of hypernym relations that can be used for semantically indexing research papers and other data-mining tasks. For example, it would be helpful to know that both Convolutional Neural Network and Multi-layer perceptron are both one type of neural network, and neural network is a machine learning model, along with decision trees, naive-bayes model, etc. Without relation extraction, it would take a tremendous amount of human effort to read scientific papers and manually label the hypernym relations in each paper – a task that would require expert knowledge on the domain and is therefore not suitable for crowdsourcing platforms like Mechanical Turk. With the ability to extract hypernym information from paper, the knowledge base could even be updated regularly to add

[6]The corpus was obtained from Semantic Scholar and it has not been not published at the time we write this paper.

results from the newest researches.

Since obtaining labels is costly, our model utilized pre-trained word embeddings as well as active learning and co-training to make learning as efficient as possible on the limited amount of labeled data available. Our main contribution is two-fold. We introduced a pipeline that will make the data collection step of relation extraction more efficient. We also provide a hypernym relation database for Computer Science to encourage future researches.

4.2 Hypernym Extraction from Text Related Work

The first attempts at automatic Relation extraction relied on hand crafted patterns. In [Hearst, 1992], Hearst provided the following example to show how human extract relations: “The bow lute, such as the bambara ndang, is plucked and has an individual curved neck for each string.”. If one is asked what is a Bambara ndang after reading this sentence, the answer will likely be “a bow lute”. Hearst made an observation on human’s reasoning on unfamiliar topics, and concluded that relations can be extracted using patterns like “A such as B” and “A, or other B”. Berland & Charniak in [Berland and Charniak, 1999] used a similar technique to extract meronyms(part-of relation). The upside is its simplicity, but the downside is the numerous special cases that the hand-crafted patterns cannot capture. Another huge downside is that one needs to manually find the good patterns. As a result, the accuracy was low: 66% for [Hearst, 1992] and 55% for [Berland and Charniak, 1999].

A natural extension to the Hand-crafted pattern model is to automate the process of finding the right patterns. The resulting approach is called Bootstrapping [Agichtein and Gravano, 2000; Brin, 1998]. Similar to how human would find the

good patterns, the algorithm starts with a few seed positive examples (p_1, r, p_2), where p_1 and p_2 are related by relation R. Suppose now that unlabeled data is cheap and is a lot larger than the labeled seed data. For each (p_1, R, p_2) in the seed dataset, it goes through the unlabeled data and finds the instances where p_1 and p_2 co-occur near each other. The patterns around the collected instances is then counted and new patterns can be found using the statistics on the candidate patterns. The process can then be repeated by generating new positive examples using the bootstrapped patterns, and the new examples are used to find new patterns. The resulting precision is higher than bootstrapping, but it still suffers from the same problem of using patterns as other pattern matching models – namely the long tail of special cases that are not correctly captured by patterns.

Neither hand-picked patterns nor bootstrapping uses any information about the input words and sentences other than their co-occurrence statistics. To utilize those information, several supervised relation extraction methods were introduced [Agichtein and Gravano, 2000; Banko et al., 2007; Brin, 1998; Etzioni et al., 2005; Kambhatla, 2004; Zelenko et al., 2003; Zhao and Grishman, 2005] . Most of them utilized both internal and external feature generators such as POS tagging, distance to the entity pair to be classified, parsing trees, etc. A classifier then projects those input features to an output representing the probability of (p_1, R, p_2) being a legitimate relation. If the features used includes the parsing tree of the input sentence, then commonly a kernel is used to compute the structural commonness between two trees by computing the percentage of common subparts the two structure shares [Zelenko et al., 2003]. Classifiers such as logistic regression, Support Vector Machine, Maximum Entropy models, and neural networks are employed to classify the instances as positive and negative examples. For example,

Kambhatla [Kambhatla, 2004], the input features are: words, entity type, mention level(Name, nominal, or pronoun), overlap, POS dependency, and Parse tree. A maximum entropy model is then used to incorporate those feature streams. In Sun and Han [Sun and Han, 2014], the parse tree kernel is enhanced with semantic information such as entity type and syntactic information such as lexical patterns. It has been shown that by adding these features, it can achieve a 5.4% increase in F-measure.

Before introducing more recent relation extraction models, it is necessary to first talk about word embeddings, because the adoption of word embeddings as a high quality language model changed the landscape of NLP, including Relation Extraction models. The word embedding model, as well as its predecessor – distributional semantics model(DSM), represents each unit of language, usually one word or one character, as a real-valued vector. An important assumption made to create such language model is that: the meaning of a word can be described by its context. By creating a statistical model of the context of a word, one can capture the semantic and syntactic information about the word. One such statistical model is the co-occurrence matrix M, where each entry M_{ij} is the probability of word i co-occurring with word j. The distributional semantics model(DSM) applies matrix transformation techniques to M to obtain a better vector for each word [Clark, 2015; Erk, 2012; Turney and Pantel, 2010]. The word embedding model builds on DSM and instead of counting the co-occurrence, it directly trains an embedding of a word to maximize the predicted probability of that word being observed with its context [Bengio et al., 2003; Mikolov et al., 2010, 2013a; Pennington et al., 2014]. The supervised model used is often neural networks such as RNN [Mikolov et al., 2010] (or more recently LSTM) and feed-forward neural networks [Bengio et al.,

2003; Mikolov et al., 2013a]. It has been shown in [Baroni et al., 2014] that the word embedding model serves as an excellent language model that consistently out-performs DSM. Numerous applications of the word embeddings model soon follows, including Relation Extraction.

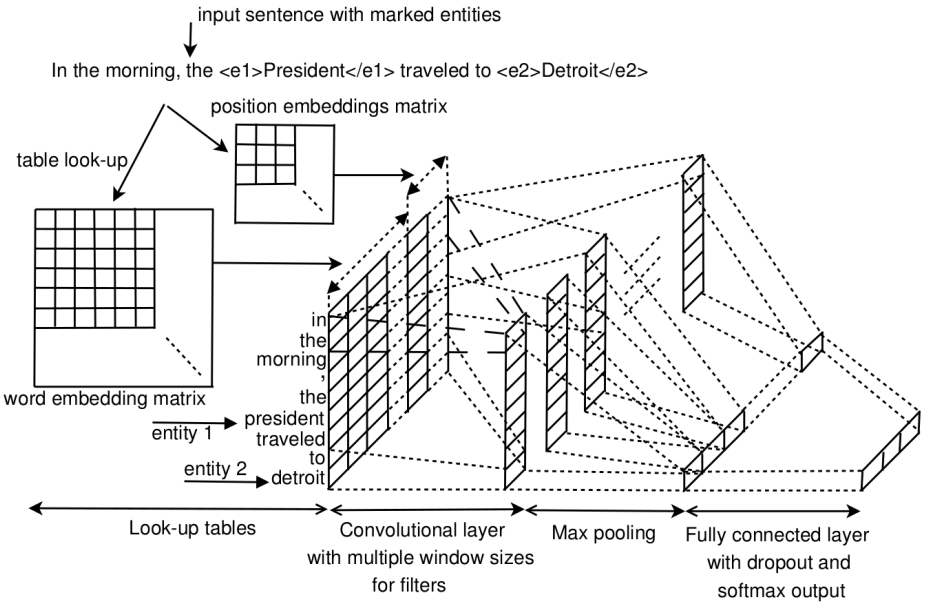


Figure 30: CNN Model for Relation Extraction from [Nguyen and Grishman, 2015].

In [Nguyen and Grishman, 2015], it was shown that by using pretrained word embeddings such as word2vec [Mikolov et al., 2013a], it was possible to discard the heavily hand-designed features used by more traditional approaches and still reach state-of-the-art performance for relation extraction. In order to extract the rich amount of information inside the word embeddings, Nguyen and Grishman took inspiration from the deep learning community and used a Convolutional Neural Network (CNN) with multiple window sizes to capture different levels of informa-

tion in the sentence. The input sentence is first segmented into words, and each word is represented by its word embedding concatenated with two positional embeddings, representing their relative distance to the two target entities to be classified. Max-pooling, fully connected layer with dropouts, and weight regularization are then applied, just like other typical CNN settings. The network structure is shown in 30 The performance of the network on relation classification tasks is on par with previous systems with manually designed features that is not be extendable to languages that currently does not have such feature extractors available.

Our work adapts the CNN model proposed by Nguyen and Grishman and applies it to hypernym extraction task in CS papers. Due to a lack of hand-labeled training data, active learning and co-training techniques are used to enhance the model's performance. Figure 31 gives an overview of the pipeline.

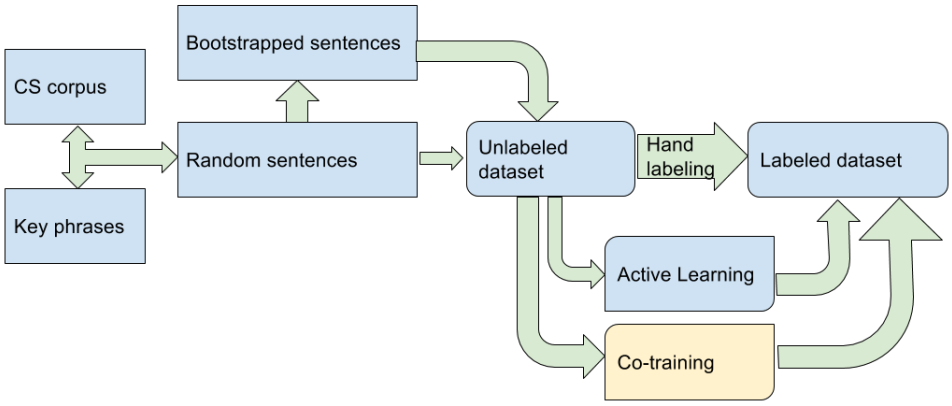


Figure 31: Overall pipeline of our relation extraction model for Computer Science Papers. The Co-training part is colored differently because we could not show it was helpful to our task, which we will discuss in later sections.

4.3 Obtaining and Preprocessing Data

We obtained raw texts of Computer Science papers from Semantic Scholar. We were also provided a list of key phrases. These key phrases will be the main target of this hypernym relation classification task. Keep in mind that the key phrases are by no means perfect. They contain phrases that may not be related to our relation task at all and that will be one of the difficulties we meet later on. First we removed key phrases that occurred in less than 5 papers. We substituted the key phrases that occurred in the corpus with a single token by connecting individual words

with underscores. All words were lower-cased and all non-ascii characters were deleted. Next, we extracted all sentences that contain a pair of key phrases that are within 15 tokens from each other and mark the position of that pair. The sentence along with the marked pair of key phrases was then stored as unlabeled data. We marked all sentence's start and end point using two special tokens, and we padded the sentences to 128 tokens long by adding "⟨PAD⟩" tokens at the end. Finally, we only selected the 40000 words that occured most often as our vocab. Other words were substituted with the "⟨UNK⟩" token. We obtained around 4.5 million unlabled training instances this way.

In order to obtain a high quality set of training data as a starting point, we started with a mixture of random unlabeled instances and a set of bootstrapped instances. The bootstrapped instances is obtained by getting instances with key phrase pairs occuring around the "such as" phrase, so it would look like "A such as B". We then hand-labeled 1300 instances drawn from the mixture.

4.4 Classifier

We adopted the CNN classifier model from [Nguyen and Grishman, 2015], which we will briefly restate here. The model consists of four layers: an embedding lookup layer, a convolution layer, a pooling layer, and a fully connected logistic regression layer. The token embeddings are composed of three parts: the semantic embedding and two position embeddings. For the semantic embedding of dimension m_e, we used a 300-dimensional word2vec embedding pretrained on Google News corpus[7]. Because we need to connect the words in one key phrase into one single token, we represent that combined token by taking the sum of word embeddings

[7] https://code.google.com/archive/p/word2vec/

in each dimension. It has been shown in [Mikolov et al., 2013c] that the sum is a decent representation of the semantic meaning of the phrase. The reason behind the additive compositionality is because a word embedding is related to the logarithm of its probability to appear around a certain context. Thus the sum of two or more words is related to the sum of log of probabilities, which equals the log of their joint probability. Next for each word at position i, we calculate its relative distances to the two key phrases in the sentence. The relative distance is therefore within range $[n-1, n+1]$ where $n = 128$ is the padded sentence length. Each of the $2n-1$ relative distances is encoded using a m_d dimensional random vector, so the total size of the relative distance embedding lookup table is $(2n-1) \times m_d$. Finally the semantic embedding and two distance embedding is concatenated to form a $m_e + 2m_d$ vector for each token in the sentence. The whole sentence is therefore represented as a $(m_e + 2m_d) \times 128$ matrix $\boldsymbol{X}$. Convolutional layers with multiple window size are applied to the sentence matrix $\boldsymbol{X}$. Each convolutional layer f acts as a feature extractor over the tokens in its receptive field. Having multiple window size would allow each convolutional layer focus on the n-grams in its own receptive field so that features at different scale can be extracted. The pooling layer aggregates the extracted features from the convolutional layers using functions such as max and $average$. The pooling layer is followed by a fully connected layer with dropouts. The fully connected layer further summarises the information from the pool layer and outputs one single vector with length c where c is the number of label classes. The dropout serves as a regulation to prevent overfitting. The model is quite shallow compared to other CNN-based models used in computer vision tasks, where the number of convolutional layers ranges from ten to a few hundred. It is nevertheless effective and provides an alternative to the traditional RNN or LSTM

based models.

4.5 Active Learning

Active learning is a useful technique for supervised machine learning, especially when labeling is expensive but obtaining unlabeled instances is cheap. The central idea of active learning is to allow the classifier to pick the instances that it would like to be labeled. The instance picked is called the query, and it is usually assumed to be always answered correctly by an Oracle. The labeled instance is then added into the training data, the algorithm retrains on the new labeled dataset, and a new query is proposed. This loop continues until either we run out of unlabeled instances, or the performance reaches a desired level. Like other supervised setting, the oracle in real applications are usually humans or an emsamble of human judges and therefore can make errors (those are called noisy oracles). The querying process allow an active learning program to get labels that decrease overall uncertainly the most. It has been shown ([Settles et al., 2008; Zhu et al., 2005]) to be able to accelerate training process and is often used when a large amount of unlabeled data is available but labeling is costly. In our task, it is hard to get computer science experts to label potential key phrases that are hypernames of one another and we often get noisy labels even if the person has background knowledge in the field. Therefore using active learning will greatly increase the training speed and reduce the labeling cost.

The querying process can be divided into three main categories, based on how the program receives unlabeled data and how it asks questions. The first cateogry is Membership Query, in which the algorithm generates its own query and ask the oracle for an answer to this query. In order to generate a query, the generator must have an estimate for the distribution of the source. Further restrictions are

imposed on the generator if the query is non-continuous. In addition, answering those generated (and maybe even ill-defined) quries is generally hard for human annotators. Due to those restrictions, designing such a system is usually a little bit more challenging than the other two. The second category is Pool-based Sampling. Unlike Membership Query, in Pool-based Sampling, the candidate queries are not synthesized by a generator but are instead drawn from a large pool of unlabled data. The active learning algorithm then has to decide which of the candidate queries should get labeled, since labeling all of them is too costly. The third category is Stream-based Selective Sampling, which is a slightly restricted version of Pool-based Sampling. Instead of giving the program full access to the pool of unlabeled data, candidate queries are drawn one at a time from the pool and the program has to decide whether to ask an oracle to label that query sequentially (no going back and changing its mind). The upside for Pool-based Sampling and Stream-based Selective Sampling is that the query directly reflects the underlying data distribution. The tradeoff is that the algorithm might not ever get the query that it really want to get labeled.

In our application, we chose the Pool-based Sampling method. Because our task is by nature discrete (words are discrete and there are countable number of frequent words being used), even though each instance is represented by a continuous embedding, it would be hard to map a generated embedding back to a sequence of discrete words and still have it make enough sense to be labeled. On the other hand, the corpus is the unlabeled data extracted from the corpus is available at all times, so it does not make sense to restrict ourselves to processing one potential query at a time using Stream-based Selective Sampling.

Our unlabeled dataset U consists of $|U|$ instances. Since it is prohibitive to

go through all of it for each round of active learning, A batch of N_U instances were drawn from the pool each round from which the queries are picked. We use the largest entropy to choose the query to be labeled. More specifically, for a classifier that outputs probabilities $p_{i1}, p_{i2}, ..., p_{ic}$ for instance i, where c is the number of categories, we rank the instances based on their entropy: $H(p_i) = -\sum_{j=1}^{c} p_{ij} \log p_j$.

Active Learning Result The seed training data was obtained by labeling a dataset which is a random mixture of bootstrapped sentences and randomly drawn ones. The bootstrapping method uses the phrase "A such as B" and outputs all sentences where A and B are key phrases. The randomlyd drawn ones simply outputs all sentences where two key phrases occur within 15 words apart from each other. We labeled 1300 sentences as the seed training data. Next, using the active learning method described above, we labeled an additional 1000 labeled instances. That adds up to around 2300 labeled sentences as our training data. Finally we built a website to collect more data. The website automatically provides the instances with largest entropy to get labeled by labelers. Each hour, the newly labeled instances were collected and were combined into the training data. The classifier was then retrained to provide a fresh set of instances to be labeled. In the case where labelers provided different labels to the same sentence, the label was chosen based on majority vote. When there is a draw in the majority vote, the sentence is labeled as not having hypernym relation. The website provided several positive and negative examples of what counted as a hypernym relation. We provided the website to students of the Machine Learning course at Northwestern University. A total of 24 students participated and labeled 898 sentences.

We compared the performance before and after including the student labeled data on a held out test set. Surprisingly we found that there is a decrease in the Area under Precision Recall Curve. Looking into the additional labels, we found that there are huge variations between the way different students label sentences. When we label our training data, we did our best to use the same criteria for all labels. The criteria used were: the hypernym relation should only hold when both key phrases are noun or noun phrases; the label should be independent of the context of the sentence. The first one is straightforward but the second one might be a little bit confusing. Why would we ignore the sentence context and only judge on the pair of key phrase themselves? To justify our rationale, we provide a simple example below. Given the sentence "`Google`** announced that Facebook, a `social_media_company`**, as its competitor.", one would likely agree that `social_media_company` is not the hypernym for `Google`. Yet if we replace `Google` with `Twitter`, one would say that `social_media_company` is a hypernym for `Twitter`. The two sentence shares the exact same textual context, yet we arrived at a different conclusion because of our background knowledge about Google and Twitter. If we restrict ourself to judging only based on whether the sentence context implies that A and B have hypernym relation, it would be hard to define what exactly constitutes as "imply". Therefore we restricted ourselves to judging based only on whether the pair of key phrases have hypernym relation in general.

Going back to the student labeled data, we found that not all student labelers followed the guidelines we provided. Even when they did, there could still be edge cases where labelers disagree among one another. For example, for the sentence "we return to the question of how they are motivated to make `things`** work in our

discussion of the rules of the game** ", some think this is valid while others think it is too broad. Another example is: given the sentence "hypothetical alignment scores** for pairs of curve** fragments .", some students labeled it as "A is a B" because they think scores can be the result of a curve on an exam. We had to clean up the additional labels and unify the standards before incorporating them into our training data. The final size of labeled dataset is 3229 sentences.

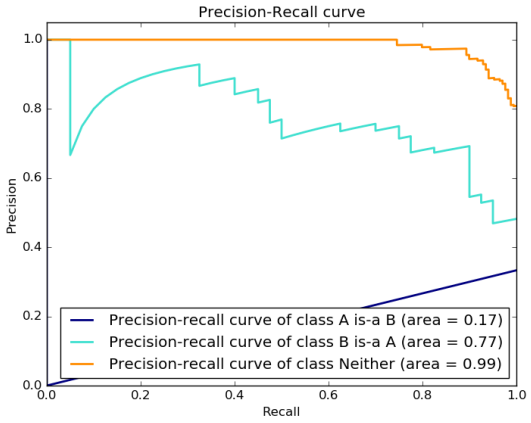
4.6 Embedding

In this section we experimented with three different word embeddings. The first embedding is the pretrained Google News embeddings, which we've used in all other experiments mentioned in previous sections. The second type of word embeddings are directly trained on the Computer Science paper corpus using LSTM with a standard softmax. The weights and embeddings were tied, which is to say there is only a single embeddings space for both context words and target words. The network structure followed [Zaremba et al., 2014]. We suspected that because words may have different semantic meanings when they appear in CS papers and by using an embedding trained directly on the CS corpus, it may be able to incorporate such additional topic-specific semantic information. The third embedding is the previous two embeddings concatenated together. We call that the combined embedding.

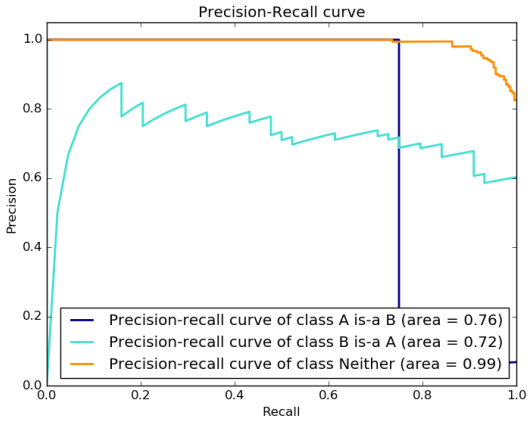
4.6.1 Embedding Experiment Result

The result of running with the Google News pretrained embedding, the LSTM embedding, and the combined embedding is shown in 32. There is no clear winner among the three. The Google News embedding did well on "B is a A" type hypernym relation while the LSTM embedding did better on "A is a B" case. However, note

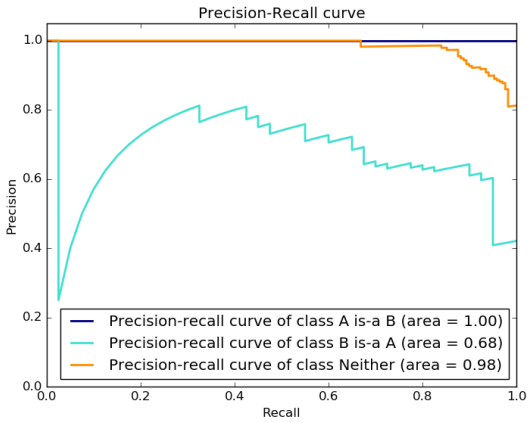
that the labeled data contained lots of “Neither” (no hypernym relation) instances, a few “B is a A” instances, and even fewer “A is a B” instances. Therefore the “A is a B” result is less accurate than the other two categories. The combined embedding did worse on “B is a A” instances but better on “A is a B” instances. However, the combined embedding had doubled the embedding dimension and made training more costly. Since we had more “B is a A” cases in our labeled dataset than “A is a B” instances, we chose the Google News embedding as the default.



(a) Word2vec Google News Embedding



(b) LSTM CS Paper Embedding



(c) Combined Embedding

Figure 32: All three embeddings were trained using the CNN classifier model from [Nguyen and Grishman, 2015]. The labeled data was split into 90% training data and 10% test data. The split was random but was done only once so that all three shared the same inputs. The classifier models were trained for 20 epochs.

4.7 Co-training

Co-training is a technique that uses the unlabeled data to boost performance of a supervised classifier. The concept is proposed in [Blum and Mitchell, 1998] where two conditionally independent view assumption enables the classifier to utilize information from the unlabeled data. For an instance space $X = X_1 \times X_2$, where X_1 and X_2 are two conditionally independent views of the instance X, we can learn two classifier functions f_1 and f_2 such that:

$$f(X) = f_1(X_1) = f_2(X_2) \tag{2}$$

The compatible assumption assumes there exist such two functions f_1 and f_2 satisfying 2. The conditionally independent view assumption can be mathematically expressed as:

$$Pr[X_1 = x_1 | X_2 = x_2, Y = y] = Pr[X_1 = x_1 | Y = y] \tag{3}$$

$$Pr[X_2 = x_2 | X_1 = x_1, Y = y] = Pr[X_2 = x_2 | Y = y] \tag{4}$$

A concrete example is the classification task in [Blum and Mitchell, 1998]: it wishes to classify webpages as either home pages or not. The two views used are the url and the bag of words in the webpage. The two classifiers are trained separately on their corresponding view using a small amount of labeled instances as the seed. At each round of co-training, a bag of instances U' are drawn from the unlabeled instance pool U to be labeled by the two classifiers. Each classifier then picks the most confident p positive instances and n negative instances. Those $2p + 2n$ instances are then added into the training dataset and the U' is replenished

by drawing fresh examples from U. The overall algorithm is shown in 33.

Given:

- a set L of labeled training examples
- a set U of unlabeled examples

Create a pool U' of examples by choosing u examples at random from U
Loop for k iterations:

Use L to train a classifier h_1 that considers only the x_1 portion of x
Use L to train a classifier h_2 that considers only the x_2 portion of x
Allow h_1 to label p positive and n negative examples from U'
Allow h_2 to label p positive and n negative examples from U'
Add these self-labeled examples to L
Randomly choose $2p + 2n$ examples from U to replenish U'

Figure 33: Co-training algorithm from [Blum and Mitchell, 1998].

[Blum and Mitchell, 1998] proved that for the two concept classes C_1 and C_2 corresponding to the two conditionally independent views, "If C_2 is learnable in the PAC model with classification noise, and if the conditional independence assumption is satisfied, then (C_1, C_2) is learnable in the Co-training model from unlabeled data only, given an initial weakly-useful predictor $h(x_1)$." It further proves that even if the compatible assumption is not satisfied, which is likely to be true in practice, it is still possible to boost classifier performance using co-training.

In order to further boost the performance of our classifier, we applied the co-training technique and separated the training data into two views as follows. For each pair of phrases (x_1, x_2), the first view is constructed by finding all sentences that contain both x_1 and x_2 and replace the x_1 and x_2 with placeholders. The second view is the pretrained word embeddings of x_1 and x_2.

Two additional hypothesis must be made in order to satisfy the co-training

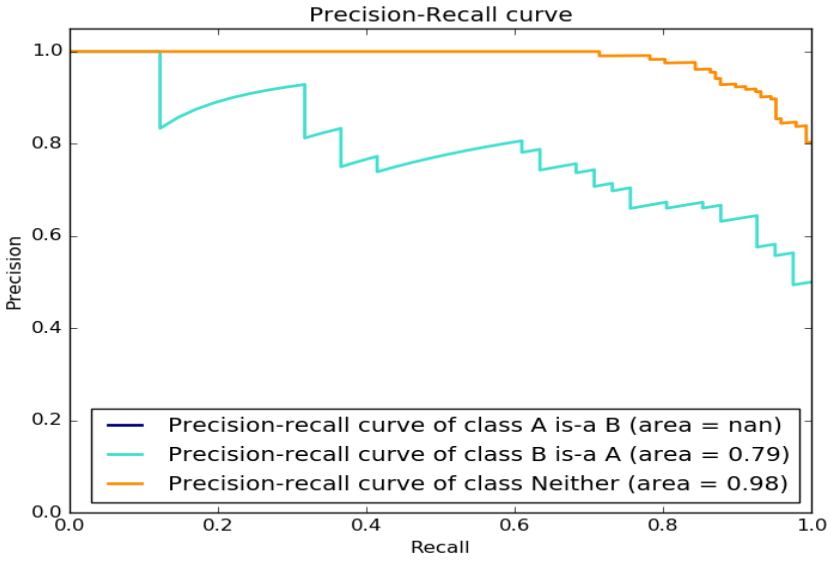
prerequisites. The first is that the pair (x_1, x_2) either has a hypernym relation or not, and that is indepent of the context the pair is in. This may not necessarily true. For example, the word "Google" could mean both the company and the action to search using Google Search Engine. Despite the fact that words have multiple semantic meanings, we made a design choice to only adapt one meaning per word independent of its context. The main reason is that it would otherwise be very hard to obtain labeled data if the label for (x_1, x_2) would depend on the context. Since obtaining data is costly in our task, we choose to prioritize less ambiguity and make the labels independent of the context. It would be interesting if a model conditioned on the context can be developed for our hypernym relation extraction task, but we will leave this to future researchers.

The second assumption that we made was that: if we replace (x_1, x_2) with placeholders in the sentence view classifier, given enough number of sentences containing (x_1, x_2), it will be possible to correctly guess whether (x_1, x_2) has hypernym relation. We were hopeful that even if the two assumptions did not hold all the time, co-training would still increase the performance to some degree.

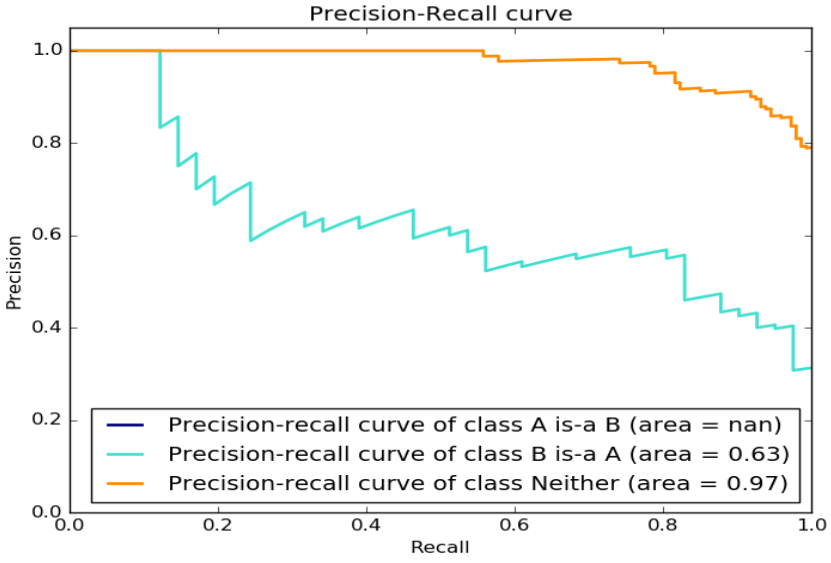
4.7.1 Co-training results

Preliminary experiments In order to test our hypothesis for co-training, we first tried a preliminary experiment where we replaced the word embeddings of the pair of key phrases to be classified by a placeholder embedding. An example would be: "`Northwestern`, a `University` located in Evanston, is a 40 minute drive from downtown Chicago." becomes "`<PlACEHOLDER>`, a `<PlACEHOLDER>` located in Evanston, is a 40 minute drive from downtown Chicago.". We then fed the modified sentence into the CNN classifier and measured the precision recall curve.

The result is shown in 34. The area under precision recall curve got lower as a result, but by closely analyzing the mistakes the classifier made, we noticed that out of the 200 labeled test sentences, 31 out of 32 errors come from the bootstrapped "A such as B" pattern. There were a total of 75 sentences containing bootstrapped pattern, which implies that there was indeed information contained in the context of the sentence that the classifier could use. Notably, the only non-"such as"-related error comes from the sentence "...moved with `velocities**` ranging from 4 to 12 `m/s**`", which we marked as "m/s is a velocity". The best answer should be "m/s is a unit of velocity" but the classifier was only capable of making classifications. This case showed the inherent ambiguity of the problem we are trying to solve.



(a) Original



(b) Replaced pair of key phrases with placeholders

Figure 34: Both were trained for 20 epochs using the CNN classifier model from [Nguyen and Grishman, 2015]. The labeled data was split into 90% training data and 10% test data. It did not include the student labeled data in this experiment.

Given the result of our preliminary experiment, we proceeded with co-training on an unlabeled dataset of around 22000 sentences using 10-fold cross validation on the labeled dataset. That is to say, we picked 90% of the labeled dataset to be the seed training data and 10% as the validation dataset. The two classifier we used were sentence-based CNN classifier with key phrase replaced by placeholders, and key-phrase based two-layer neural network classifier. At each round of co-training, we picked a batch of $|U'| = 1000$ unlabeled instances and each classfier labeled $p = 2$ positives and $n = 30$ negatives. We did in a total of 10 round of co-training, giving us $10 * (2p + 2n) = 640$ additional instances labeled through co-training.

There were two small modification we made to the original algorithm. The first was : instead of replenishing U' by drawing fresh examples from U, we discard all instances in U' if they are not one of the $2p + 2n$ chosen instances and draw a fresh batch of U' from U. The reason behind this change was due to our highly unbalanced dataset. The positive-to-negative ratio in our labeled dataset was around 1 out of 4 because we combined randomly drawn instances with bootstrapped ones. The positive-to-negative ratio of randomly drawn instances was around 5 out of 100. If we did not draw a fresh batch each round, all the positive instances in U' would soon be exhausted without enough positive instances being replenished into U'. Another option would be change our p and n value to match the positive-to-negative ratio, but we did not actually know what the "true" positive-to-negative ratio was in the unlabeled dataset and the inaccurately estimated $p : n$ ratio could still cause U' running out of positive examples to label. The second change was: instead of the two classifiers each picking its most confident positive and negative instances, we calculated their joint predicted probability by taking the product.

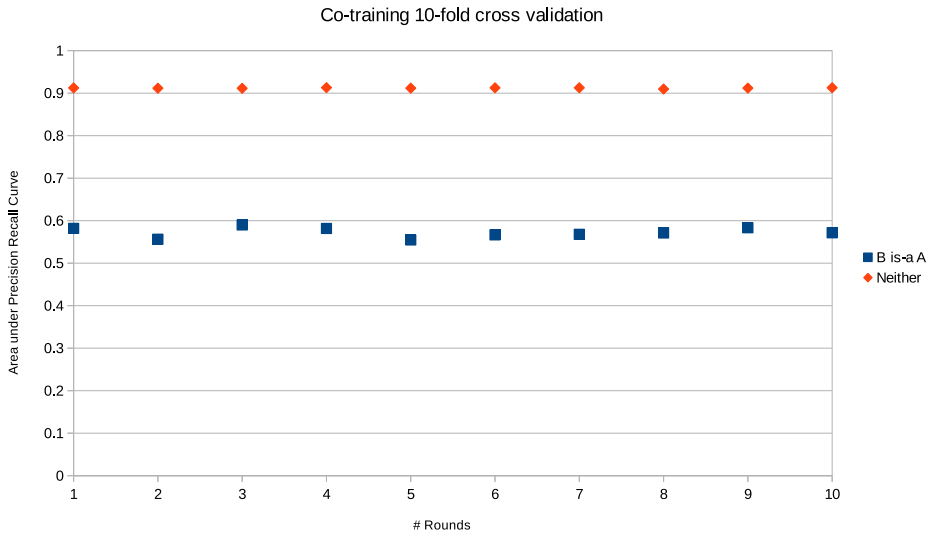


Figure 35: Area under the Precision Recall Curve of co-training averaged over 10-fold cross validation. $|U| = 22000, |U'| = 1000, p = 2, n = 30$.

The result is shown in 35. Even after averaging over 10-fold cross validation, the overall trend still had lots of variance and we did not see a clear increase in area under precision recall curve when tested on the validation set. We suspected that the size of the unlabeled dataset $|U|$ we used was too small. The $10 * (2p + 2n) = 640$ additional labeled instances did not exceed the size of the seed training dataset. Therefore we expanded $|U|$ from 22000 sentences to 1 million sentences. One difficulty that we encountered in this process was that one pair of key phrases could appear in multiple sentences, but due to our context-independent labeling assumption, all pair of key phrases should have the same label independent of the sentences they were in. In order to give a uniform label for all sentences with the same pair of key phrases, we chose to combine the output probability distributions of those sentences to form a single label for one pair of key phrases. We explored

four combination methods which we will discuss below.

Combination methods

- Like method $p_{combined,c} = \prod_i^N (p_{i,c})$
- Noisy or $p_{combined,c} = 1 - \prod_i^N (1 - p_{i,c})$
- Average $p_{combined,c} = \sum_i^N (p_{i,c})$
- Majority Vote $p_{combined,c} = (num\ positive_c + \lambda)/(N + 2\lambda)$. We chose $\lambda = 1$ for all our experiments.

In the methods listed above, all of them were calculated per class. N is the number of sentences with the same pair of key phrases and $p_{i,c}$ is the probability assigned to sentence i for class c. $num\ positive_c$ is simply the number of sentences classified as class c. Finally, $p_{combined,c}$ is not normalized (except the majority vote method) so we do a final normalization to make sure $\sum_c p_{combined,c} = 1$.

As a preliminary experiments, we ran the co-training program several rounds using each of the four combination methods. We found the performance suffered in all four experiments. After taking a closer look, we identified one common problem to all four combination method. For labeling the no-hypernym pairs, the classifier was biasing towards classifying pairs that tends to occur a lot and are easy to classify. For labeling the hypernym pairs, the classifier was biasing towards giving high scores to pairs that occur not so many times. The reason behind such preference was: since our dataset was unbalanced, the classifier was more likely to classify a sentence as negative (no hypernym relation) than as positive (contains one hypernym relation). As a result, if one pair of key phrase appeared in more sentences,

the combined probability would lean more towards the negative label simply by chance. After each round, even though p and n remains the same, the number of added sentences with negative labels is much higher than that with positive labels, often results in an overall positive-to-negative ratio of around 1 to 1000 (When $p = 4, n = 60$), much lower than the 1 to 5 ratio in the seed training dataset. Soon the additional labels added through co-training skews the positive-to-negative ratio too much that the classifier stops classifying anything as positive. An example of this failure case is shown in Table 1.

Table 1: Examples of the failure case where the positive-to-negative ratio balance is broken. The words or key phrases of interest is appended with **.

Label	**# of sentences with the same key phrase pair**	**Example sentence**
Neither	33	(3.10) : if a and** a = / 0 , the claim** clearly holds .
Neither	22	the effectiveness and <UNK>nature of the proposed** pressure observer was** demonstrated using experimental results .
Neither	54	if new data** types** need to be handled , the sparts can be modiied or new sparts can be created to handle them .
B is a A	1	hca** <UNK>global environmental** and occupational health .

We used a simple yet effective method to fix the issue of skewed additional labels. For each unique pair of key phrases, we picked only one random sentence out of all the sentences that the key phrases appeared in. Thus the positive-to-negative ratio of additional labels is kept at $p : n$. Again we did preliminary experiments on each of the combination method and summarized our finding as follows:

- Like method: This method still tends to favor negative pairs that appears a lot of times and positive pairs that appear only once or twice.
- Noisy or: because it is approximately an or-operation, it is exactly the opposite of the Like method: it favors positive pairs that appears a lot of times, thus having higher chance that at least one or more sentence get labeled as positive.
- Average: Because of the unbalanced dataset, larger $|U'|$ works better. The result is shown in 36. Unfortunately, there is still no increase in performance.
- Majority Vote: similar to the Like method.

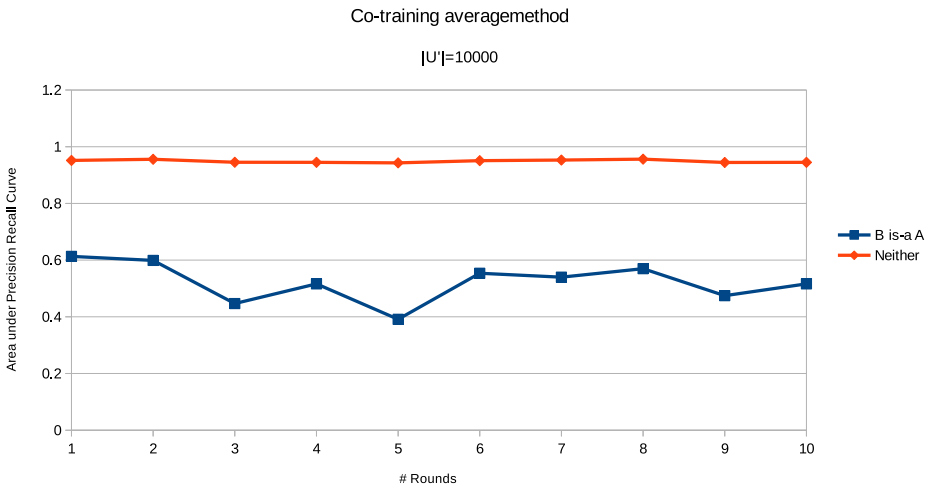


Figure 36: Area under the Precision Recall Curve of 10 rounds of co-training using the Average combination method with $|U'| = 10000$.

Label	Sentence
Neither	lastly , observe that lim_inf** 1 n (g n (x) x) 1 0 lim_sup** 1 n (g n (x) x) 1 .
Neither	if the length of a binary_string** is smaller than the degree of the crc generator** , its crc value is the string itself .
Neither	to make our case , we start** by motivating the need for digital_preservation** storing ¡pad¿ over long periods (2) .
B is a A	ga includes all sub-areas such as metadata , file_data** and block info** .
B is a A	many users** already use social_media such as twitter** for exchanging links to web pages that are of interest to them .
B is a A	each encoded method includes a method id , flags** such as private** or final , and an offset into the code table .

Table 2: Examples of instances labeled by the two classifiers and added to training data during co-training. The words or key phrases of interest is appended with **.

Co-training conclusion The results from 36 indicates that the co-training is unlikely going to work well in our setting. When trained on the seed dataset, there was a big gap between the initial performance of the CNN classifier and the two combined classifiers (sentence based and key phrase pair based), each was assumed to have an independent view of the same classification problem. We think that gap is caused by our wrong co-training assumption. The co-training assumption states that the two concept classes C_1 and C_2 corresponding to the two conditionally independent views are compatible. It further states that even if the two concept classes are not compatible in practice, co-training still **may** improve the performance. However, we think that the conditional independence assumption does not hold for the two concept classes that we used: the sentence-based concept class where we "hide" the key phrases pair from the classfier and the key phrases pair-based concept class where we only show the key phrases pair to the classifier. Not only does it

not hold, in fact the two views are too dependent on each other that we cannot gain much, if any, performance boost from co-training. Remember that the classifiers see not the word themselves, but the word embeddings and the position embeddings representing the words. The word embeddings are trained so that they correspond well to the probability that the word appears in a certain context. That is to say: the word embeddings are trained such that it is possible to predict the "missing word" given its context, and vice versa. Therefore, even though the precondition for co-training can be relaxed a little in practice, in our setting the two views are too highly correlated with one another that ultimately caused co-training to fail.

4.8 Self-training

We also experimente with self-training in the hope to reduce labeling cost and to boost performance. Self-training is the predecessor to co-training, where a classifier is trained on a seed labeled dataset. The trained classifier is used to label more instances in the unlabeled dataset. The instances labeled with the most confidence is chosen and added into the labeled dataset and the classifier is retrained on the new set. The process is repeated until it runs out of unlabeled dataset or until the performance of the model deteriorates (measured on a separate validation dataset).

Due to limit in time, we only ran a preliminary experiment using the CNN hypernym extractor and the same settings as in co-training. The only major difference was that we were no longer using two separate views. The unlabeled dataset pool size was $|U'| = 1000$ instead of 10000. At each round we choose $p + n$ most confident instances from U' and we set $p = 2, n = 30$. The result is shown in 37. We looked at the instances labeled with highest confidence, and saw that their quality was lower than we expected, given other experimental results such as 5.3.

Therefore, even though the overall trend is negative, we suspect that if we have more time, we can fine tune the hyperparameters (Such as $|U'|, p, n$) to achieve better performance.

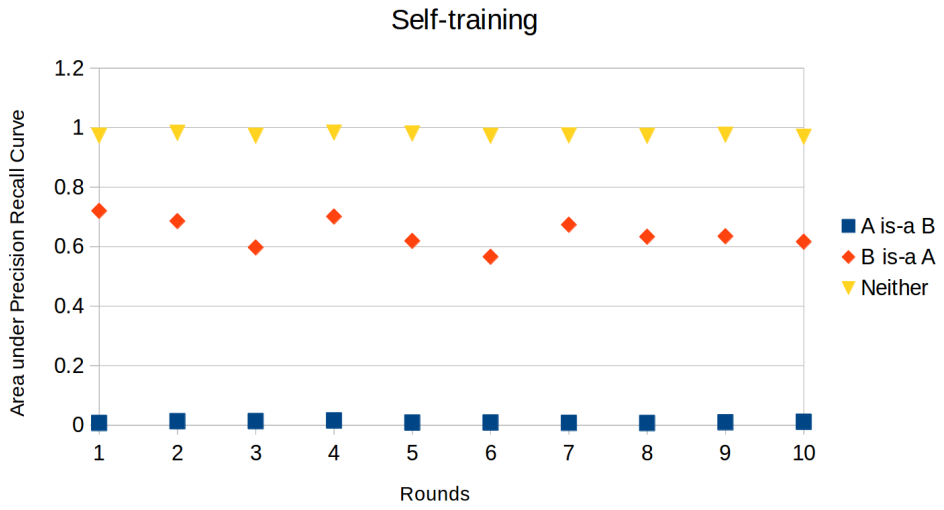


Figure 37: Area under the Precision Recall Curve of the preliminary self-training experiment.

4.9 Results

We provide a list of key phrase pairs that the CNN-based relation extraction model labeled with high confidence. Specifically, taking the labeled data with 3229 sentences as training set(a combination of hand labeling and active labeling), we labeled more than 1 million unlabeled sentence, which was extracted from the first 2% of the CS corpus. We chose the top 500 positively key phrase pairs, because we care more about quality than quantity and the quality of the extracted key phrase pairs quickly deteriorates beyond the first 500. Examples of labeled key phrase pairs along with an example sentence that they appear in is shown in 3. For a list

containing the top 100 instances, please refer to 5.3.

Label	Sentence
B is-a A	enterprise applications : this is the place of high-end business_software** such as erps** or plms .
B is a A	so the demand shocks** such as promotions** should be independent .
B is a A	each encoded method includes a method id , flags** such as private** or final , and an offset into the code table .
B is a A	more recently , [21] study the influence of social links and trust on business transactions** such as auctions** .
B is a A	they were able to exhibit certain physical manifestations** such as materialisation** of spirit form , apports i.e .
B is a A	the dependency is weighted by a correlation_function** such as mutual_information** [14] .
B is a A	in the last_few years several popular msa packages** have been parallelised , for example dialign** [23] or praline [15] .
B is-a A	in particular , it could be_useful in specialty** practices such as ⟨UNK⟩ , ⟨UNK⟩ , and** ⟨UNK⟩ .

Table 3: Examples of unlabeled instances marked by the CNN-based classifier as positive with the highest confidence. Almost all positive labels with high confidence are "B is a A" due to the low occurrence of "A is a B" labels in the labeled dataset. The words or key phrases of interest is appended with **.

4.10 Future Directions

As shown in Table 1, 4, and 3 , a common mistake pattern shared by the CNN classifier, CNN classifier with replaced key phrases, and key phrase classifier was that they often label pairs (A,B) as having hypernym relations when either A or B or sometimes both are not noun or noun phrases. In [Kambhatla, 2004; Sun and Han, 2014], it was shown that adding semantic parsing tree would help relation extraction tasks. Even though [Nguyen and Grishman, 2015] proposed that by using pretrained word embeddings, it was possible to discard such information, we think

that it is still helpful to have additional POS information available. For example, if the word embedding is pretrained using Google News, the semantic meaning of a "tree" may not necessarily correspond to a "tree" in Computer Science. Further more, the word "and" may refer to the conjection word, but it may also refer to the logical operation in our corpus. Having a good POS tagger will likely make such mistake patterns appear less often.

Another interesting question is relation to transfer learning: after the classifier is trained on the CS corpus, can it easily be extended to apply to extracting hypernym relations in other fields of study without too many hand labeled dataset? If the classifier mainly remembers the semantic meaning of words and their relations after being trained, it will not be able to be directly applied to other fields. However, if it relies on sentence contexts but not extensive background knowledge in the area of interest, then it should be applicable to other fields after being trained on the CS corpus. Judging from the preliminary experiment 4.7.1 where we replace the pair of key phrases with placeholders, the classifier is likely using a mixture of background knowledge and context information. We look forward to future researches to prove or disprove our guess.

References

Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the fifth ACM conference on Digital libraries*, pages 85–94. ACM, 2000.

Sercan O Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Jonathan Raiman, Shubho Sengupta, et al. Deep voice: Real-time neural text-to-speech. *arXiv preprint arXiv:1702.07825*, 2017.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Michele Banko, Michael J Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI*, volume 7, pages 2670–2676, 2007.

Marco Baroni, Georgiana Dinu, and Germán Kruszewski. Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *ACL (1)*, pages 238–247, 2014.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb): 1137–1155, 2003.

Matthew Berland and Eugene Charniak. Finding parts in very large corpora. In *Proceedings of the 37th annual meeting of the Association for Computational*

Linguistics on Computational Linguistics, pages 57–64. Association for Computational Linguistics, 1999.

Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, 1998.

Sergey Brin. Extracting patterns and relations from the world wide web. In *International Workshop on The World Wide Web and Databases*, pages 172–183. Springer, 1998.

Andrew Brock, Theodore Lim, JM Ritchie, and Nick Weston. Neural photo editing with introspective adversarial networks. *arXiv preprint arXiv:1609.07093*, 2016.

Mark Chang. Applied deep learning 11/03 convolutional neural networks. `https://www.slideshare.net/ckmarkohchang/applied-deep-learning-1103-convolutional-neural-networks`, October 2016.

Tian Qi Chen and Mark Schmidt. Fast patch-based style transfer of arbitrary style. *arXiv preprint arXiv:1612.04337*, 2016.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

Stephen Clark. Vector space models of lexical meaning. *Handbook of Contemporary Semantic Theory, The*, pages 493–522, 2015.

Cícero Nogueira Dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. 2017.

Michael Dushkoff. A temporally coherent neural algorithm for artistic style transfer. 2016.

Mathias Eitz, James Hays, and Marc Alexa. How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44–1, 2012.

Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653, 2012.

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134, 2005.

Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 262–270, 2015a.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015b.

Leon A Gatys, Alexander S Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. *arXiv preprint arXiv:1611.07865*, 2016.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

Christopher Hesse. Image-to-image demo. `https://affinelayer.com/pixsrv/`, February 2017.

Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *arXiv preprint arXiv:1703.06868*, 2017.

Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Let there be color!: joint end-to-end learning of global and local image priors for automatic image

colorization with simultaneous classification. *ACM Transactions on Graphics (TOG)*, 35(4):110, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

Jianmin Jiang, P Trundle, and Jinchang Ren. Medical image analysis with artificial neural networks. *Computerized Medical Imaging and Graphics*, 34(8):617–631, 2010.

Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Google's multilingual neural machine translation system: Enabling zero-shot translation. *arXiv preprint arXiv:1611.04558*, 2016.

Nanda Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 22. Association for Computational Linguistics, 2004.

Brendan Klare and Anil K Jain. Sketch-to-photo matching: a feature-based approach. In *SPIE Defense, Security, and Sensing*, pages 766702–766702. International Society for Optics and Photonics, 2010.

Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Learning representations for automatic colorization. In *European Conference on Computer Vision*, pages 577–593. Springer, 2016.

Chuan Li and Michael Wand. Combining markov random fields and convolutional neural networks for image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2479–2486, 2016.

Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*, 2013b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013c.

Thien Huu Nguyen and Ralph Grishman. Relation extraction: Perspective from convolutional neural networks. In *Proceedings of NAACL-HLT*, pages 39–48, 2015.

Yaroslav Nikulin and Roman Novak. Exploring the neural algorithm of artistic style. *arXiv preprint arXiv:1602.07188*, 2016.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 3, 2016.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based

noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, 1992.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

Patsorn Sangkloy, Jingwan Lu, Chen Fang, Fisher Yu, and James Hays. Scribbler: Controlling deep image synthesis with sketch and color. *arXiv preprint arXiv:1612.00835*, 2016.

Cicero Nogueira dos Santos and Victor Guimaraes. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*, 2015.

Ahmed Selim, Mohamed Elgharib, and Linda Doyle. Painting style transfer for head portraits using convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 35(4):129, 2016.

Amir Semmo, Daniel Limberger, Jan Eric Kyprianidis, and Jürgen Döllner. Image stylization by oil paint filtering using color palettes. In *Proceedings of the workshop on Computational Aesthetics*, pages 149–158. Eurographics Association, 2015.

Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS workshop on cost-sensitive learning*, pages 1–10, 2008.

Yichang Shih, Sylvain Paris, Frédo Durand, and William T Freeman. Data-driven

hallucination of different times of day from a single outdoor photo. *ACM Transactions on Graphics (TOG)*, 32(6):200, 2013.

YiChang Shih, Sylvain Paris, Connelly Barnes, William T Freeman, and Frédo Durand. Style transfer for headshot portraits. 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Le Sun and Xianpei Han. A feature-enriched tree kernel for relation extraction. 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Yaniv Taigman, Adam Polyak, and Lior Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

Peter D Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37:141–188, 2010.

Dmitry Ulyanov. Feed-forward neural doodle. https://dmitryulyanov.github.io/feed-forward-neural-doodle/, May 2016.

Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *Int. Conf. on Machine Learning (ICML)*, 2016a.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016b.

Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. *arXiv preprint arXiv:1701.02096*, 2017.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1395–1403, 2015.

Yan Xu, Lili Mou, Ge Li, Yunchuan Chen, Hao Peng, and Zhi Jin. Classifying

relations via long short term memory networks along shortest dependency paths. In *EMNLP*, pages 1785–1794, 2015.

Taizan Yonetsuji. 初心者がchainerで線画着色してみた。わりとできた。. `http://qiita.com/taizan/items/cf77fd37ec3a0bef5d9d`. Webpage in Japanese, for English translation see `http://qiita.com/jerryli27/items/f526a7d5b69ae758a3a6`, December 2016.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. *arXiv preprint arXiv:1609.05473*, 2016.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. *Journal of machine learning research*, 3(Feb):1083–1106, 2003.

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

Shubin Zhao and Ralph Grishman. Extracting relations with integrated information using kernel methods. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 419–426. Association for Computational Linguistics, 2005.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Proceedings of European Conference on Computer Vision (ECCV)*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.

Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. *Semi-supervised learning with graphs*. Carnegie Mellon University, language technologies institute, school of computer science, 2005.

5 Appendix

5.1 Example of sketch coloring by human artists

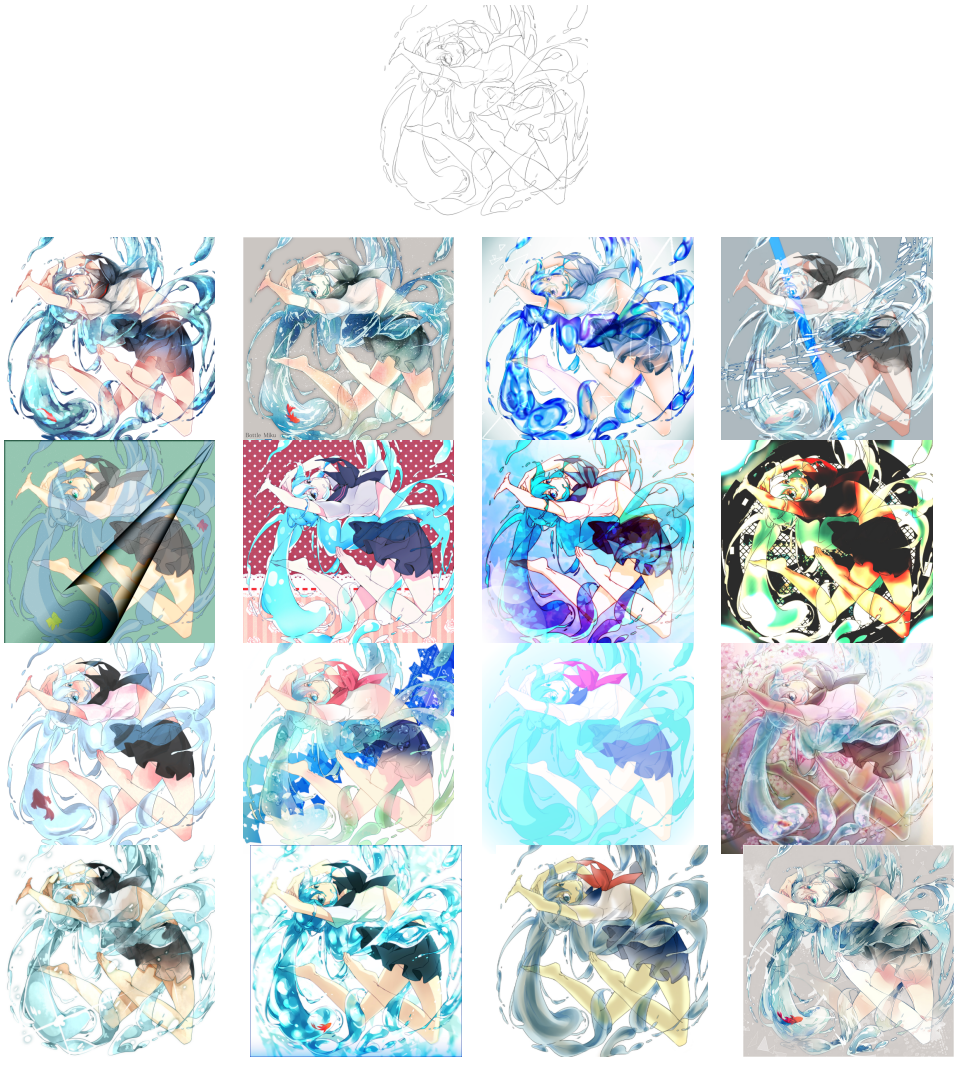


Figure 38: Sketch(top) and different versions of the colored images collected on Pixiv, an artwork submission website, an artist. All images are credited to their original author. From top to bottom and left to right, their corresponding Pixiv image ids are: 33767623, 33933286, 36061742, 38262072, 33805351, 34169540, 36908253, 38312410, 33834204, 34316137, 37599800, 49277980, 33851482, 35384089, 38169369, 49286194, 33743570.

5.2 Sketch Coloring Training Details

For most of our experiments, we settled on one set of hyperparameters. For the weights of each loss function conponent, $\alpha_{cGAN} = 1, \alpha_{L1} = 100, \alpha_{cycle} = 10, \alpha_{TV} = 10$. For the dilation sketch generation, $d_{max} = 6, d_{min} = 3$. For the random noise we add to the input and target output, we use a gaussian random variable with $mean = 0, variance = 0.02$ if the input sketch image is a neural network generated sketch, and $variance = 0.12$ if the input sketch image is dilation generated sketch. All image pixel values are normalized to between -1 and 1. For the models with l1 decay, we set the lower bound of the l1 decay to $alpha_{L1_{min}} = 50$ and decay rate to $\eta = 0.9$ per epoch. So after each epoch, the l1 loss weight is updated as: $alpha_{L1} = max(alpha_{L1_{min}}, alpha_{L1} * \eta$. The learning rate is fixed at 0.0008. In terms of picking random pixels as hints for colorization, we first generate a blank rgba image of the same size as the input sketch. Then we pick 40 random pixels on the target image and replace the corresponding pixel's value on rgba image with that of the target image, with transparency value alpha set to 1. Therefore all other pixels have $alpha = -1$ except for the hint pixels, so alpha also acts as an indicator variable.

During training, usually the model starts to learn skin color first, since that is the most consistent and the most frequently appearing color in anime style paintings. Colors become more and more vibrant as training progresses, usually accompanied by a drop in WGAN discrimination loss.

5.3 Hypernym Extraction - Top 100 Highly Confident Positives

Label	Sentence
B is-a A	enterprise applications : this is the place of high-end business_software** such as erps** or plms .
B is-a A	so the demand shocks** such as promotions** should be independent .
B is-a A	each encoded method includes a method id , flags** such as private** or final , and an offset into the code table .
B is-a A	more recently , [21] study the influence of social links and trust on business transactions** such as auctions** .
B is-a A	they were able to exhibit certain physical manifestations** such as materialisation** of spirit form , apports i.e .
B is-a A	this is especially the case for those technologies that depend in part on iii-v compound semiconductors** such as gaas** .
B is-a A	the dependency is weighted by a correlation_function** such as mutual_information** [14] .
B is-a A	memory can include qualia** such as snapshots** or fragments of the sensory_stream with high_information content .
B is-a A	introduction nlp is gaining prominence due to the importance given to social_media_data** such as twitter** and facebook .
B is-a A	combined host and network idss** such as iss** realsecure [22] can also respond to threats by terminating individual processes .
B is-a A	unlike synthetic models , models in this class are extracted from real world traces** such as handoff** and association in lans .

B is-a A	swollen legs as well as the unlocalized symptoms** such as fever** or cough .
B is-a A	this is the case in games** such as tetris** , pong , and other highly abstract games .
B is-a A	it relates design_properties** such as encapsulation** , cohesion , etc .
B is-a A	the evaluation of each .lter for every ⟨UNK⟩ is a boolean** expression : true** or false .
B is-a A	each provides a set of graphic icons for common peripherals** such as switches** , gauges , and function generators .
B is-a A	in theory this can can be done in a temporal_logic** such as ctl** [7] .
B is-a A	updates need to be applied to all indexes , materialized views , summary synopses** such as histograms** , etc.
B is-a A	4. how well does the 2d-svm-solver work compared to standard software packages** such as libsvm** by chang and lin (2009) ?
B is-a A	, 1994) , and graphical eects** such as highlighting** and blinking (dalal et al.
B is-a A	semantics subobject the semantics subobject is comparable to objects in middleware** such as dcom** and corba .
B is-a A	in addition , xj3d supports various renderers** such as opengl** , java3d etc .
B is-a A	, commercial portals** such as amazon** , yahoo autos , etc .)

B is-a A	consider a web_site** such as amazon** , which sells books .
B is-a A	fourth is knowing how to use a text_editor** such as nano** , gedit , or kate .
B is-a A	if the motion_clip is one of the special motions** such as jumping** , we do not take half of the clip to get node midtimei .
B is-a A	lighting plays a major role in all forms of imagery** such as photography** and video .
B is-a A	the use of intersatellite_links has also been proposed for some nongeosynchronous msss** such as iridium** .
B is-a A	no entropy_coding or other routines** such as dpcm** have been implemented to optimize the bit rates .
B is-a A	as in many other internet-based media** such as forums** and chat rooms , the real identities of bloggers are unknown .
B is-a A	users with severe_motor_disabilities often use assistive_technology** such as switches** and scanning in lieu of a mouse .
B is-a A	all of them used the webpages of the aggregator websites** such as yelp** , google places , foursquare and trip advisor .
B is-a A	such results can be applied to trace_properties** such as authentication** but not to indistinguishability .
B is-a A	large corporations** such as google** have shown its commitment in developing ar technology , e.g .

B is-a A	however , other artefacts** such as xml** documents are increasingly used to develop applications .
B is-a A	therefore , we did not choose slow but accurate trackers** such as particle_filtering** [11] .
B is-a A	this strategy showed limitations when dealing with distortions** such as jpeg_compression** [6] .
B is-a A	emotions** such as anger** or fear can increase blood pressure , increase the respiration_rate and dry the mouth .
B is-a A	simple , intuitive w3 clients** such as mosaic** and netscape have contributed to its tremendous current popularity .
B is-a A	such free services offered by multinationals** such as yahoo** !
B is-a A	until recently , countries** such as india** , malaysia , singapore , and mexico were little known in terms of technology .
B is-a A	others use a variety of curves** such as circles** (morgan et al.
B is-a A	the standard variance shows this kind of ability under most of the audio_processing** such as lpf** , tsm and a/d .
B is-a A	backyard/on-site composting is ideal for households and institutions** such as schools** and hospitals .
B is-a A	[3] there are many other types of identification** such as password** , pin (personal identification number) or token systems .
B is-a A	much of this research has been aimed at data_mining social_media_services** such as twitter** or facebook .

B is-a A	if the seller elects not to transfer water , the alternative_use of the water is application to a crop** such as rice** .
B is-a A	example modules include games** such as mastermind** and space invaders , and utilities such as a calculator and color picker .
B is-a A	lookahead is a traditional_technique for masking communication costs in matrix_factorization** such as lu_decomposition** [2] .
B is-a A	in the south , disasters** such as erosion** , flooding and landslides , and vectorborne diseases are common .
B is-a A	in the place of object , a literal** such as string** or number can be used .
B is-a A	tsujita 's syncdecor system involves the synchronization of pairs of daily appliances** such as lights** , trash boxes , and tvs .
B is-a A	it is the relaxation of a partitioning problem which can in principle be solved by a package** such as cplex** .
B is-a A	experimental results [1] show substantial_performance_advantage of using shorter mac** such as 32-bit* mac .
B is-a A	the interactions provided are rotation_and_translation** such as zooming** (z axis) and panning (x , y axis) .
B is-a A	also , outstanding ratings are valuable for promotion in some agencies** such as army** , air_force , and treasury .
B is-a A	hence , if a scheduling_discipline** such as sced** can schedule a set of connections , so can npedf .

B is-a A	image carries a lot_of_information** such as clarity** , brightness , sharpness , curves etc .
B is-a A	for a change in a crosscutting_concern** such as tracing** or logging , recompilation of the entire system will be necessary .
B is-a A	obesity increases risk for many chronic diseases** such as cardiovascular** disease , diabetes , and numerous cancers .
B is-a A	furthermore , the number of wifi_access points installed inside vehicles** such as buses** , trains or ferries is also raising .
B is-a A	several sat solvers** such as grasp** [marques-silva and sakallah 1999] and chaff [moskewicz et al .
B is-a A	1 but also rams (aside from configuration memories) and unconfigurable logic circuits** such as cpus** .
B is-a A	[2004] use stins to understand the inquiry_learning** forum (ilf**) , a web-based forum for math_and_science teachers .
B is-a A	however , guy fawkes signatures could be implemented as an alternative_option within a cryptocurrency** such as bitcoin** .
B is-a A	in particular , deep depths** such as intersections** can be detected with high_stability by setting up a good threshold .
B is-a A	this opens the door to a number of threats** such as phishing** and impersonation attacks with no easy_way to secure the content .
B is-a A	past_experience of using a dictionary** such as wordnet** shows quite good results [7 ; 17] .

B is-a A	the mobile_radio_channel is characterized by random impairments** such as fading** and shadowing .
B is-a A	x ⟨UNK⟩ the boolean** values are true** and false .
B is-a A	this extends to several_other permissions** such as edit** and download control .
B is-a A	in particular , this applies to popular libraries** such as ant** , antlr , hibernate and lucene .
B is-a A	this can be done using any distance_measure** such as root_mean_square_error** (rmse) .
B is-a A	previous work [5] [6] shows that web_traffic** such as telnet** and ftp followed heavytail distributions .
B is-a A	a particle_filter** such as condensation** can be used for monte-carlo simulation of the underlying distributions .
B is-a A	semantic browsers** such as tabulator** [3] form one type of easily accessible end-user interface to semantic_web data .
B is-a A	professionals** such as press** , ngos , and first responders may own special hardware .
B is-a A	standard benchmarks** such as linpack** are used for ranking the top supercomputers [19] .
B is-a A	this set of threats** includes masquerading , denial_of_service** and unauthorized access .
B is-a A	this type of attack is especially relevant when behavioral traits** such as signature** and voice are used .

B is-a A	, and titles** such as mr.* , ms. , prof. , dr. , rev .
B is-a A	such diverse** groups as marketing** and governmental regulators may also use the requirements .
B is-a A	automatic_interpretation of unusual events_of_interest in wide-area public scenes** such as road** tra.c is highly desirable .
B is-a A	most popular search engines of today** such as google** and yahoo !
B is-a A	-compiler toolkits** such as cosy** [1] , cocktail [18] , ocs [22] , suif [42] , and pim [6 , 17] .
B is-a A	map manipulations** such as panning_and_zooming** are provided by the esri mapobjects library [6] .
B is-a A	retail portals** such as yahoo** !
B is-a A	special topics covered themes** such as emergency_preparedness** and the training of veterinarians about public health .
B is-a A	data and query delivery to the rendezvous_node is implemented by geographical_routing** such as gpsr** [14] .
B is-a A	european_firms pursue sr for concerns of stakeholders** such as government** , regulatory bodies , customers or pressure groups .
B is-a A	in the early 1960s , such a design corresponded to many seasonal sports** such as rowing** , cycling , skating and skiing .
B is-a A	they came to the uk from war-torn countries** such as iraq** , afghanistan and somalia .

B is-a A	line_art analysis line_art analysis operates primarily on curvilinear** type prime image objects , or curve** stroke fragments .
B is-a A	the page displays** different widgets or controls according to the screen** area available for display .
B is-a A	packages** such as gtw** , simkit , warped and cpsim allow the user to debug the parallel_program via runtime messages .
B is-a A	certainly , manufactured items** such as vehicles** have neither dna nor fingerprints .
B is-a A	the sequence uses an object_oriented_language** such as java** .
B is-a A	⟨UNK⟩ analysis ⟨UNK⟩ analysis operates primarily on curvilinear** type prime image objects , or curve** stroke fragments .
B is-a A	by default , wsdl** uses xml_schema** to describe data , but wsdl extensibility allows other data_type systems to be used instead .
B is-a A	by default , wsdl** uses xml_schema** to describe data , but wsdl extensibility allows other data_type systems to be used instead .
B is-a A	those architectures often include non-molecular solids** such as semiconductors** , alloys , and ceramics .
B is-a A	as a boat passing by drops trash** overboard , the gelatinous dwellers discover a treasure** they can not resist .

Table 4: More examples of unlabeled instances marked by the CNN-based classifier as positive with the highest confidence. The words or key phrases of interest is appended with **.