

# **Final Project Report: Image2Doodle**

COS 429

December 8, 2020

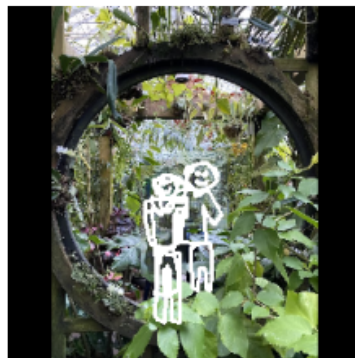
**Koert Chen, Nicholas Sum, and Wendy Ho**

## Introduction

Our project identifies objects from a natural image and replaces these objects with doodles drawn from the Google Quick Draw dataset (*Google Quick Draw Dataset* n.d.). This provides a representation of an image's objects in a simplified form, while still preserving contextual elements and background. This can serve as a method for anonymizing images, as it removes identifying details while maintaining information about image content. It also provides a visually entertaining illustration of several modern computer vision algorithms. Our project maps real objects detected from COCO dataset (Lin et al. 2015) object identification categories to equivalent categories in the Quick Draw dataset (cup to cup, snowboard to skateboard, etc.), but these mappings can be simply and arbitrarily rearranged (person to duck, couch to donut) to create an imaginative reinterpretation of the world that serves as an amusing and educational application of computer vision.



(a) Image



(b) Doodled Image

Figure 1: Image before and after doodling

We decided on a matching based approach over a generative approach, as we felt that it might be more feasible to iterate and improve on a matching based, multi-stage system rather than a deep generative approach. Our project pipeline begins with detecting COCO object categories. Then, we use an inpainting method to replace the bounding boxes of these objects. Finally, we perform edge detection on the regions with detected objects and select a matching doodle, through one of several methods such as nearest-neighbor or local feature matching. Finally, we draw the doodle on the inpainted image. Our project is inspired by the 'Draw This' project by Dan Macnish (<https://danmacnish.com/drawthis/>). This project converted a camera to print doodle representations of images on a white background, like a Polaroid camera. Our project reimplements and then extends this idea with inpainting and a matching system to select the most appropriate doodle.

The intended goals of the project are to create a doodle drawing that preserves the main objects, so that the doodle is still identifiable as the same objects as the original image. In addition, we wish to produce an aesthetically pleasing, or at least amusing, doodle image. Therefore, we evaluate the images using human raters that are asked both to identify objects in the doodled image and rate their aesthetic appreciation of the image. We use well-studied models for object detection and inpainting, so we focus on evaluating the quality of matching doodles.

## Background

### Object Detection

Object detection is the task of identifying objects in images as one of several pre-defined categories and drawing axis-aligned bounding boxes around these objects. To perform object detection, we used CenterNet (Zhou, Wang, and Krähenbühl 2019) through the Tensorflow Object Detection API, pre-trained on the COCO 2017 dataset. CenterNet is a one-stage detector that detects objects as a single center point as a peak on a heatmap, and then other properties of that object are regressed from the image at that center point. In contrast to multi-stage methods that propose potential boxes and predict features for each of those potential boxes, CenterNet provides an optimal speed-accuracy trade-off.

To briefly describe the architecture and design of CenterNet, CenterNet was designed and trained with several backbone architectures, of which we chose the stacked hourglass network (Newell, Yang, and Deng 2016) as it provided the best keypoint estimation performance. This backbone consists of repeated modules of 5-layers of down-convolutional and then up-convolutional layers. On top of the feature layers, keypoint prediction head layers, heatmaps, are trained with a focal loss, a modified version of cross-entropy loss used to mitigate class imbalance in training data. There is one heatmap per class. In training, a ground truth center keypoint is “splatted” across the heatmap, that is, a Gaussian kernel is centered at the ground truth center on the heatmap corresponding to the ground truth class. During inference, the local maximums on the heatmap are the detected keypoint centers. This center keypoint heatmap is the “Center” part of CenterNet, and one center detection corresponds to one object detection.

For other properties of the object, additional head layers are used to predict other properties. First, as these maps are downsampled, there are also an additional two head layers predicting x and y offset to correct this discretization error. These are trained with supervision only at keypoint centers. In addition, to predict bounding box size, there are also two head layers predicting bounding box width and height, trained with L1 loss with supervision only at center points. To detect pose at  $k = 17$  joint locations, CenterNet adds  $k \times 2$  head layers, which represent offsets from the center keypoint to the joint keypoint. These are trained with focal loss analogously to the center keypoints. At inference time, the closest detected joint location is used for each center keypoint.

### Inpainting

When we detect and remove an object from the image, we lose the section of the image where the object was. It would be ideal to have another nonoccluded image of the just the background without the objects, but this is unfeasible. One first step was to just fill in the object as black, but this is not an ideal result. Since the goal is to create an image with the same informational context as the original, the background behind the doodle will be important. Furthermore, doodles are not solid unlike real objects, so we also need to generate background inside the doodle to create a more realistic image.

Image inpainting is the solution to this problem. The problem takes an image and a mask, occludes a part of the image with the mask, and then generates new sections of the image in the mask area. Here, we take the bounding box of the object as the mask. If we use an inpainting model to generate images within this bounding box, we can successfully create a new image with a new background where the object used to be. We note that since we have taken out the object fully with the detection bounding box, we do not have to worry about the object corrupting the background.

In this project, we use the PEN-Net image inpainting model Zeng et al. 2019. This architecture uses a U-Net structure, which encodes pixels into high-level features and decodes it back into an image. This architecture

has three components. First, they use a pyramid-context encoder (PEN) to encode features at all resolutions and levels of detail. Then, they build a multi-scale decoder, which, at each level of the pyramid, uses the information from the deeper level inpainted feature map and the feature map of the level. To do inpainting, they learn affinities between patches inside and outside of the masked region in order to transfer features from the previous level and the level’s feature map. This adds detail at every step while maintaining the semantic plausibility of the inpainted image from the deeper levels. Finally, they use a discriminator to train with an adversarial training loss, as in a usual generative adversarial model. This method allows the inpainted image to maintain high level feature data and large scale semantic detail while also maintaining detail at the shallower levels of the pyramid. Figure 2 shows the architecture. More detail can also be found in the paper Zeng et al. 2019. Code is adapted from paper’s GitHub, linked in README.md in the folder ”inpainting”. Model can be downloaded linked from same readme.

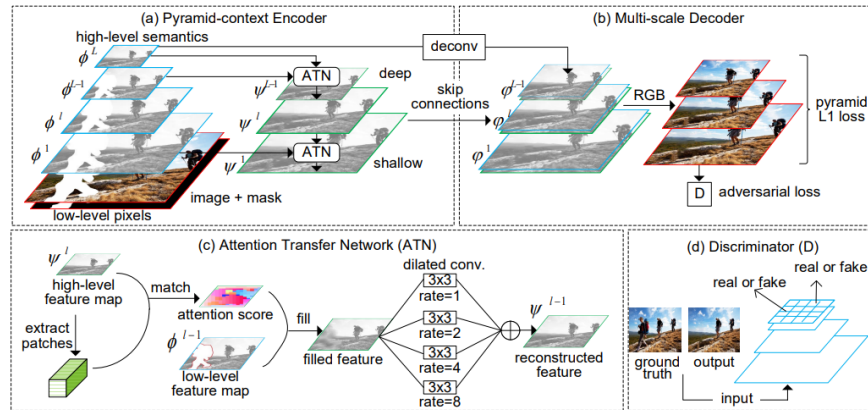


Figure 2: Architecture and training model for PEN-Net

We use this model for a few reasons, which include a fairly well-documented implementation and a pre-trained model. They also present successful results on natural images (trained on the Places2 dataset), which is what we want to fill in for our backgrounds. Finally, the model is also very flexible and can be used with arbitrary mask shapes and sizes. However, this model also has a few problems. First, the implementation only works on low-resolution images (256x256 px). This could be fixed with training another network with larger images and deeper features, but this model is untested in this area. Furthermore, when the object takes a large portion of the image, which happens often for images in the COCO dataset, the model fails to produce meaningful backgrounds. It mostly looks like blurred sections of repeated background in this case. This is also a problem generally with inpainting models in low-context situations. We can mitigate these problems by using a semantic segmentation model in order to make a higher quality mask.

## Doodle Matching and Drawing

After removing the object from the image using image inpainting, we begin the task of filling up the space with a doodle. In this project, the doodles from the Quick, Draw! Google Dataset (*Google Quick Draw Dataset* n.d.) are a collection of over 50 million drawings in 345 categories. The doodles were drawn as part of the Quick, Draw! Game in which participants had 15 seconds to draw a specific prompt like 'car' or 'teddy-bear,' and as such, the doodles are simple, quick representations of a real world object. This poses specific challenges for our project because while a real world car might be photographed from multiple different perspectives, usually the Quick, Draw! car will be a car from the side view and not take into account change in perspectives. On the other hand, because the car is such a simplistic representation, it increases the entertainment factor of our result. We use the default provided subset of 10,000 drawings per category in our implementation.

In our project, we use the OpenCV implementation of the Canny edge detector on the bounding box of the detected object. The Canny edge detector first obtains gradients after Gaussian blurring, then performs non-maximum suppression to thin the resulting edges. After thresholding to distinguish between strong and weak edges, the Canny edge detector applies hysteresis, starting along strong edges and connecting to weak edges, to obtain the best prediction of edges. Using these edges, we match each object to the best doodle that matches it through one of several methods, which are described in more detail our implementation. In general, we tried two main methods, nearest neighbor search between doodles and the image edges, and SIFT descriptors paired with RANSAC. The use of SIFT Descriptors was suggested and explored in **5674030** for the opposite task of sketch-based image retrieval. We hope that for the task of image-based sketch retrieval, it will prove just as adequate.

## Implementation

### Basic Pipeline

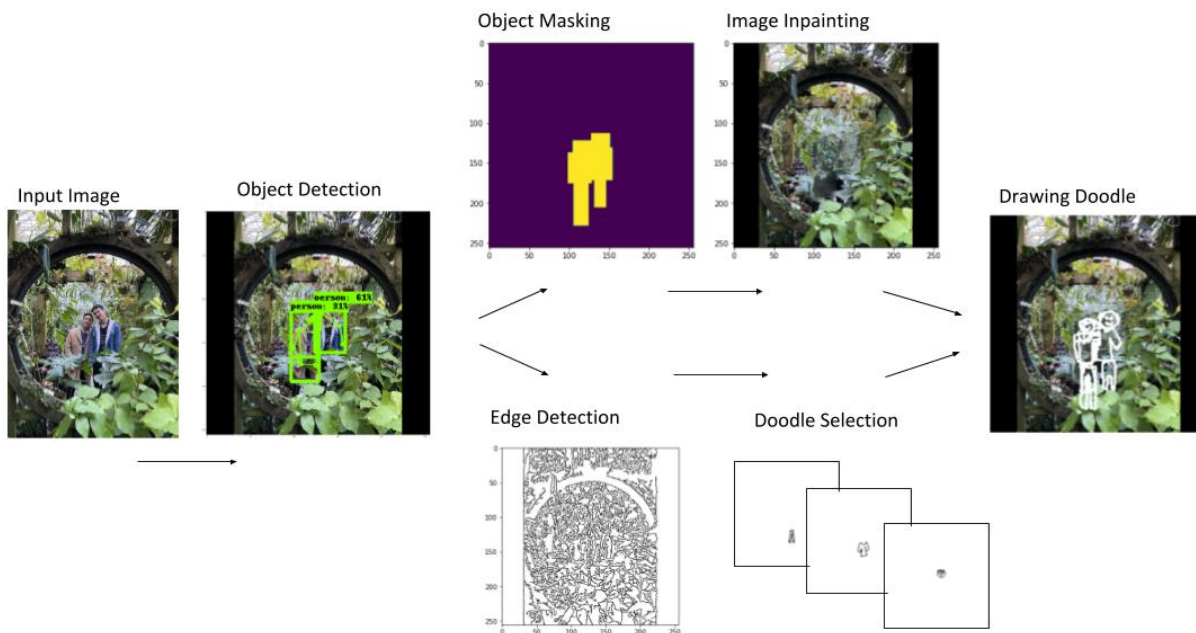


Figure 3: Pipeline Diagram

We begin by taking an input image and downsizing to  $256 \times 256$ , the size required later for inpainting, and then performing object detection with CenterNet. If an object detection passes a confidence threshold of 0.4, we count it as a valid detection. For the surviving detections, we then convert the COCO image category to the equivalent Google Quick Draw category, and prune the detection if no equivalent is configured. Categories were mapped to an equivalent category if they represented exactly the same or visually similar objects, and are mostly similar to the dictionary from the 'Draw This' project (this dictionary is the only part of the code we used from the 'Draw This' project). A partial dictionary is in Figure 4, and the full dictionary is in the code.

COCO Image Category	Quickdraw Doodle Category
pizza	pizza
bus	bus
cat	cat
orange	apple
apple	apple
baseball glove	hand
⋮	⋮

Figure 4: COCO to QuickDraw Mapping

After mapping, we perform doodle matching through one of several methods. First, for each object detection, we perform Canny edge detection on the region defined by the detection bounding box. In our initial working implementation, we then use these edges to perform nearest neighbor search among every doodle in the Quick Draw dataset of the same class as the detection, with the Frobenius norm as the distance metric. The doodles are stretched/downsampled to the same size as the bounding box of the object detection before comparison. For each detection, we save the best, or closest distance, doodle to be drawn on the image. Finally, we take the bounding boxes of the detected objects and mask out the area to perform image inpainting with PEN-Net. On the inpainted image, the best doodles are then drawn in their corresponding bounding boxes. Resulting images from the initial version are compared against the later iterations and improvements in Figure 5.

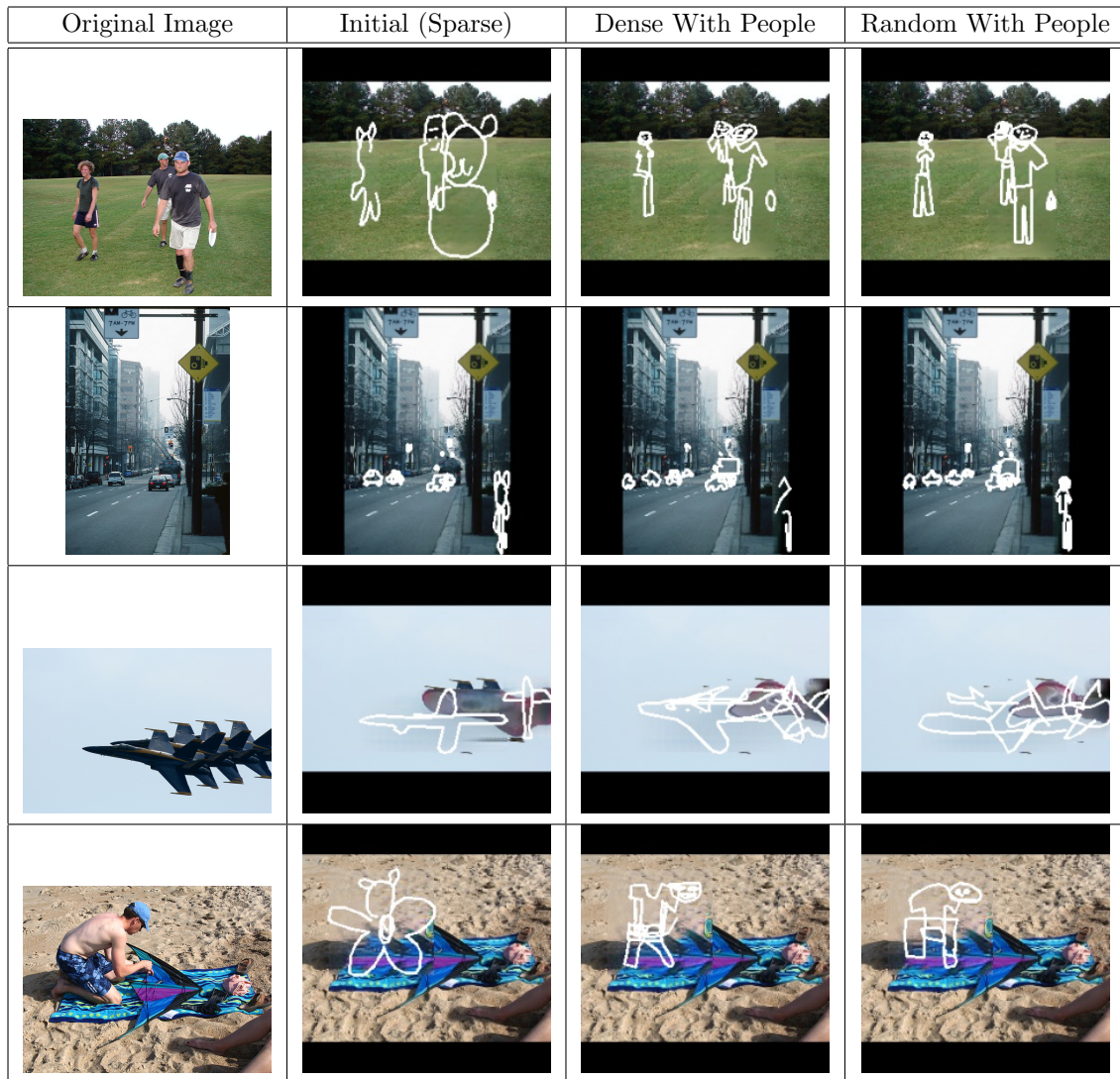


Figure 5: Doodle Matching Method Result Comparison

## Improvements

After creating the initial version, we pursued several avenues of improvement.

**Person matching with pose detection.** The initial version maps persons to teddy bears, as the Quick Draw data set does not include people drawings. This results in amusing but ambiguous results, where a child holding a teddy bear would be replaced with a large teddy bear holding a smaller teddy bear. However, the Quick Draw dataset does include head, t-shirt, and pants drawings. To remove this ambiguity, we used pose detection to manually segment the person bounding box resulting from object detection into several smaller head, torso, and legs bounding boxes. Then, the head boxes were mapped and replaced with head drawings, and, similarly, torsos with t-shirts and legs with pants.

To create the manual body parts bounding boxes, we first switched to using the CenterNet model with pose detection (with joint keypoint prediction heads attached). Then, we found the smallest axis-aligned box that contained the joint keypoints corresponding to this body part, and used this as our new body part bounding

box. For the head, we used the nose, eyes, ears, and shoulders (this box in particular was also extended to to the top of the bounding box height, as the keypoints are all below the top of the head). For the torso, we used the shoulders, and elbows, and hips. For the legs, we used the hips, knees, and ankles. The resulting bounding boxes are rough but sufficient. These 3 synthetic bounding boxes often resulted in decreased performance for inpainting, so we use the original single person bounding box to perform inpainting, but use the synthetic bounding boxes for the matching stages.

Note that the pipeline diagram in Figure 3 actually depicts an instance of this improvement instead of our initial pipeline, though it leaves out the manual segmentation. All trials besides the initial depicted Figure 5 use person matching. The random trial randomly selects a doodle of the desired class, instead of using nearest neighbor matching. This includes the head, t-shirt, and pants classes. The dense line trial uses people matching with nearest neighbor matching, as described below.

**Dense line doodle matching.** In our initial implementation, we downsampled the original doodle to the same size as the bounding box before evaluating distance. In many of the detected objects, the bounding box is small enough to result in thin, dotted, sparse lines, as shown in Figure 6 below.



(a) Original Anvil Doodle



(b) Downsampled Anvil Doodle

Figure 6

These sparse doodles did not necessarily harm the performance of the nearest neighbor matching, as it acted as sort of a weak supervision, selecting for doodles that were mostly empty except matching on points at the detected edges. To attempt to improve this, instead of downsampling images of the doodles, we instead redrew the individual strokes of the doodles at lower sizes, which resulted in more continuous lines, which is why we call this the dense line doodle matching method. We also introduced a Gaussian blur filter, to attempt to offset the greater count of non-white pixels by making the distance metric more forgiving. The resulting images are in the Dense column of Figure 5.



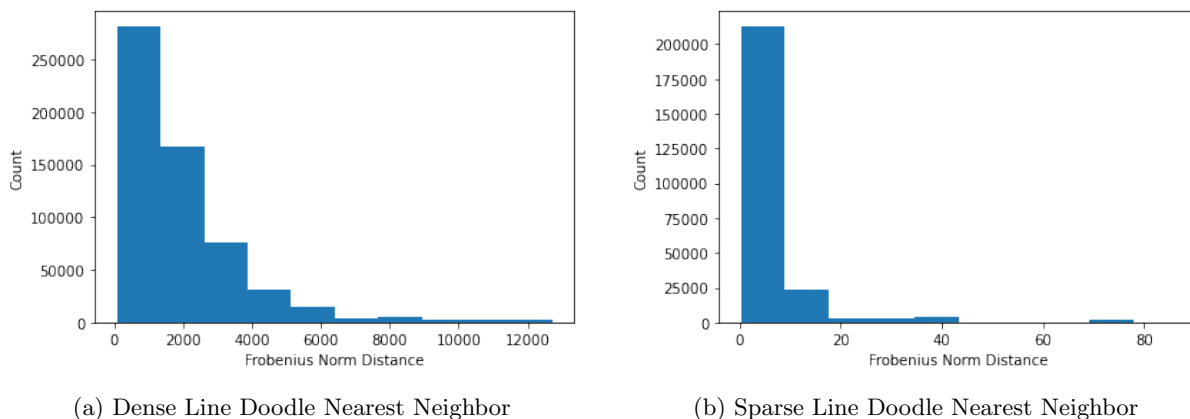


Figure 7: Histograms of Doodle Neighbor Distances

Before we began final evaluations on the actual images, we conducted an initial analysis by recording histograms of distances (norms) of all doodles from their detected object edge image they were compared to in Figure 7. We found that the doodles were mostly clustered at lower distances, which would indicate that nearest neighbor with a Frobenius norm was simply choosing among a large body of similar doodles, and thus not doing a great job of distinguishing between doodles. The graph above does indicate that our improved dense method on the left distinguishes a little more effectively between doodles, as it has a flatter curve with more of the mass distributed away from the origin compared to the initial sparse method on the right. However, altogether, the histograms confirmed our suspicion that nearest neighbor on raw pixels was not likely to be satisfactory, and that we should instead pursue better methods of representing objects and their matching doodles.

**Local feature matching with SIFT and RANSAC.** The next improvement we tried to make was to the matching process between objects and their doodle counterparts. Instead of using a nearest neighbor comparison of pixels themselves, we decided to try manual feature extraction of edge-detected drawings with SIFT and then use RANSAC to do a nearest neighbor matching with each image in the quickdraw dataset. We posited that this method would allow us to make more robust matchings between images as we would be able to use the deeper features of the images.

We created a codebook of SIFT features for every image in the quickdraw dataset that we use as a matched drawing. We extract SIFT features after blurring the line drawing with a Gaussian blur. Before doing the Gaussian blur, SIFT could not find enough features in the image due to the relative sparsity of the drawings, but with the blur, we were able to find enough features to do an effective homography matching. At inference time, we use a Gaussian blurred version of the Canny edge detected object, calculated keypoint matchings with a match finder, and finally found a homography between the two images with RANSAC. Our match is the image with a homography with the most inliers. If we cannot find any image with enough keypoint matchings to create a homography, then we simply abstain and do not draw the drawing on the image. We tried two keypoint matching methods: FLANN with Lowe’s Ratio test as well as a brute-force L2-Norm matcher with cross check, of which, the brute force matcher performed better. Code adapted from our solution from assignment 1 and OpenCV tutorials Mordvintsev and Revision 2013, found in on the GitHub, in the branch “sift”.

Some preliminary results can be found in the following table and figure:

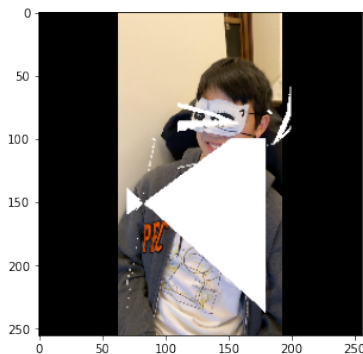


Figure 8: The final matched image

Original Image	Original Keypoints	Matched Drawing	Matched Drawing Keypoints	Transformed

These results are not promising in showing that this would create good image matches or good doodle images. There seems to be a fundamental problem with this approach to image matching for this particular dataset. The doodle drawings are very sparse and SIFT does not find very many keypoints as compared with the Canny edge detected drawing. Many of the detected features in the doodles lie on lines and corners of the drawing, while in the line detected image, they instead lie on the inside structure of the image as well. This makes it hard to do matching in this way on many fronts. First, the features in edge-detection space are mostly different than the types of features in the drawings. Furthermore, the line and corner features that we are able to find correspondences between are hard to differentiate. We cannot find very many matches without weakening the ratio test and other match filtering, making the matching subject to noise and other difficulties. We also cannot be sure that we are making the correct correspondences with the different line features.

This result points to an idea that local features are most likely not the best direction to do line-based matching with doodles. Line features do not make very good features to use for matching. This also points to an idea that drawings are simply fundamentally different from an edge detected image. While drawings mostly focus on the outline and gross details of an object, edge detection finds all different types of edges, including the noise within the object. We could try to use semantic segmentation to find the outline of the object.

This result seems to be at odds with some sketch-based image retrieval results found in the literature, which finds successful image retrieval based on sketches using local features as in Eitz et al. 2011. However, this and other image retrieval results have higher quality datasets in which image retrieval is the goal of the data. They collected data from subjects who were trying to emulate real life objects for an imaginary image retrieval system. In contrast, the QuickDraw dataset, while whimsical and fun, was not meant for this type of matching. Doodles not meant for image retrieval are mostly nonrealistic and symbolic; for example, most busses in the image are 2-D rectangles with four wheels and large square windows. Since these doodles are not realistic, it is hard to use natural features to match with cartoon images' synthetic features.

We do not perform evaluation with this improvement due the general low performance of the preliminary results we did find.

## Evaluation Methods and Results

### Methods

For evaluation, we chose to focus on the doodle matching stage of our pipeline, which is where most of our modifications are focused. We used a well-studied object detection model and chose a cutoff confidence threshold for object detection, 0.4, based off initial exploration, so that we could focus on matching. Similarly, for inpainting, we also used a well-studied model.

To evaluate matching, we considered our various goals. We both wanted to create a recognizable representation as well as an amusing, aesthetically pleasing doodle. Thus, we had human raters evaluate our result images in two ways. For each image, we asked the participant to guess what the original image was supposed to be and to rate the overall quality of the image. Quality responses were on a 1-10 scale, and we intentionally gave broad leeway in our description of quality to the raters: quality in the image is a measure of understandability (can you tell what is happening in the image), amusement (how entertaining is the final result), as well as overall aesthetic (how does this image look as a whole?).

When evaluating the free response answers to 'what was the original image?' we maintained a generous standard in evaluating their interpretation. Each free response was either given a score of 1 (correctly guessing the content of the original image) or 0 (incorrectly guessing the content of the original image). If they left out some objects in the image (such as a bottle in the background), but the overall interpretation was correct, a 1 was given. If they misidentified similar objects such as a cat instead of a dog or a giraffe as 'some sort of animal,' a 1 was given. Only in cases of egregious errors, such as identifying objects with no close visual analogues in the image, were the responses given 0s. This rating system is fairly generous, an intentional choice as it preserves more expressiveness of rating for lower quality doodles. The original doodles are already often difficult to understand, so this rating system is designed to remain useful in face of that. In figure 9, we have included some examples of evaluations and score given.

We performed three trials with three varying methods, the initial (sparse line matching) method, the dense line matching improvement with people matching, and random matching with people matching. The first two used nearest neighbor to select the best doodle, while the final randomly selects a doodle from the class of the object detection. On each trial, we asked 5 human raters to rate 20 images each on both the quality and image recognition tasks, resulting in total for 100 images evaluated for each method. The dataset of images on which we performed evaluation was a random subset of 100 images drawn from the COCO 2017 validation images (Lin et al. 2015).


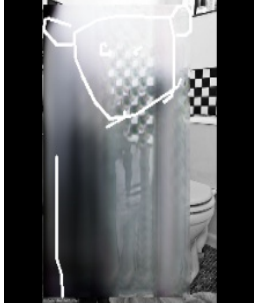




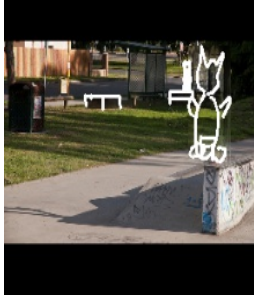
Original Image	Result Image	Correct?	Description	Quality
		0	A shower?	3
		1	Kitchen appliances	10
		1	Bicycle/kids toys	6
		1	person walking in a park	7

Figure 9: Score Examples

## Results

For each trial, we recorded accuracy, the percentage of raters who correctly identified the content of the image according to the criteria above, or that received a 1 in the image recognition question. In addition, we recorded average quality ratings. These two figures for the three trials are summarized in Figure 10 and the quality score frequencies are recorded in Figure 11.

	Initial	Dense With People	Random With People
Accuracy	0.778	0.780	0.844
Quality	5.522	5.733	5.913

Figure 10: Trial Results

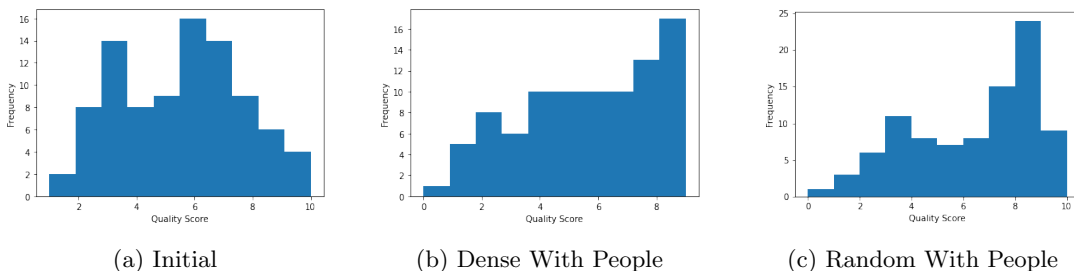


Figure 11: Quality Score Histograms

In Figure 12, we show examples of images from each evaluation with high and low quality scores. Full evaluated image sets are included with the code.


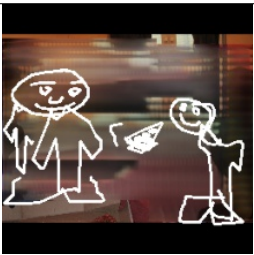

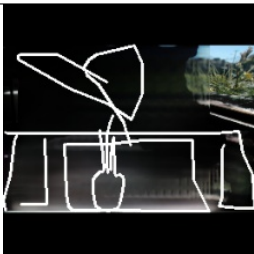
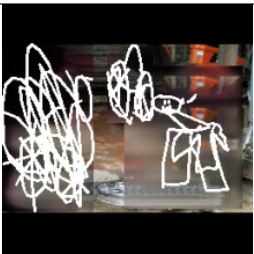

	Initial	Dense With People	Random With People
Good	 Score: 9	 Score: 8	 Score: 9
Bad	 Score: 2	 Score: 2	 Score: 3

Figure 12: Trial Results

## Analysis and Conclusion

The random implementation was rated better in both quality, 5.913, and accuracy, 0.844, than the original implementation and the dense implementation, indicating that our methods to match to a better doodle are actually worse than random matching. One explanation for this is that quick doodle representations of objects often don't look very similar to real world objects. For example, a drawing of a cat might be a simple circle with triangles for ears sticking out of it, but a real world cat might be taken from a side view or a

front view and have a lot more features. By matching the nearest neighbor of the edges of an image, we do not get the 'best' doodle that the average person can identify as an object in that class, but instead a doodle that matches the edges of the real world object, which may turn out to be a terrible doodle in terms of ease of recognition. Furthermore, our nearest neighbor implementation was meant to be the stepping stone to more effective matching and representation techniques, such as SIFT and RANSAC, which would allow us to more effectively identify. As noted in our methods section, however, the feature descriptor method we attempted did not work well enough to even produce simple functionality, likely because our doodles were much more rudimentary than the higher-quality doodles used in similar sketch-based image retrieval systems as in (Mordvintsev and Revision 2013). The nature of the Quick Draw dataset and its collection, a quick web game intended to elicit the simplest recognizable doodle that could be drawn with a mouse, produced doodles with few distinctive features that could be recognized by a hand-crafted descriptor-based retrieval system like SIFT.

Besides summary statistics, we observed empirically from individual quality ratings that the images that performed the best tended to be the ones that have few objects (1-4 objects) and where there was minimal overlap of bounding boxes of the objects. With a higher confidence threshold or perhaps the imposition of a numerical limit to the number of doodles, we could have increased quality scores by preventing overcrowding of doodles. There was a slight improvement in accuracy and quality to drawing humans with 'face,' 'shirt,' and 'pants,' and with implementing a dense line matching rather than a sparse line matching.

As for our two nearest neighbor versions, the initial (sparse line) matching compared with the dense line matching, our improved dense line matching with people matching demonstrates both higher average accuracy and quality, as well as a distribution of quality scores more skewed left. The average quality score increase, from 5.522 to 5.733, is more significant than the accuracy increase, from 0.778 to 0.780, indicating that the techniques helped improve overall aesthetic appreciation more than recognition of the image. Due to time constraints (and the order in which improvements were implemented) we were unable to conduct another trial totally separating the effects of people matching with separate bounding boxes from dense line matching, but these results demonstrate that people matching and dense line matching together were improvements on our original pipeline. This demonstrates that our modifications from our initial model did function as improvements, producing better matchings of doodles to image in both metrics. This verifies the initial exploratory analysis done in Figure 7, where the flatter histogram of distances less clustered at the origin that was associated with dense line matching led us to believe that it would indeed serve as a better matching method.

## Future Work

A promising avenue for future work is improving the nearest-neighbor distance metric. We used a rudimentary metric, the Frobenius norm, on the edges produced by Canny edge detection. This was mentioned in our first lectures as an often ineffective way of comparing raw images, and we intended it mainly as a stepping stone to other image retrieval methods. For example, in this project, we tried to use SIFT local features with RANSAC. We could instead try learning features of the drawings and the line detected images with a neural network. Or, using sketch-based image retrieval methods as a starting point, we could create a model for the reverse: image-based sketch retrieval.

Another approach is through a generative approach. We could use the Quick Draw dataset as training data to generate new doodles based on the test image that we receive, as a type of style-transfer. This would allow us to maintain the structure of the original image and use features of the original image to convert the exact image into a doodle.

One problem we did find with this project was that the doodle data is hard to match with real-life images. We can improve this from both ends. We can get more data, more representative and life-like drawings on the one hand, to make the doodle-space conform more to the real life objects. We can also try to make the real images match better to the doodles through better edge detection methods that match more what someone would produce for a doodle.

## Appendix

All code is available at <https://github.com/wendyho/COS429FinalProject>. (Unmerged) SIFT code on branch sift. Our datasets are drawn from the COCO 2017 dataset, available at <https://cocodataset.org/>, and the Google Quick Draw Dataset, available at <https://quickdraw.withgoogle.com/data>. We used code/libraries from the following sources: Cartoonify project at <https://github.com/danmacnish/cartoonify> (used only for mapping dictionary between categories), Object Detection Tensorflow API at [https://github.com/tensorflow/models/tree/master/research/object\\_detection](https://github.com/tensorflow/models/tree/master/research/object_detection), COCO Dataset API at <https://github.com/cocodataset/cocoapi>, PEN-Net Image Inpainting code and model adapted from <https://github.com/researchmm/PEN-Net-for-Inpainting>, OpenCV APIs at <https://docs.opencv.org/4.5.0/>, SIFT code adapted from [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_feature\\_homography/py\\_feature\\_homography.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html).

## References

- [1] M. Eitz et al. “Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors”. In: *IEEE Transactions on Visualization and Computer Graphics* 17.11 (2011), pp. 1624–1636. DOI: 10.1109/TVCG.2010.266.
- [2] *Google Quick Draw Dataset*. <https://quickdraw.withgoogle.com/data>.
- [3] T.-Y. Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].
- [4] A. Mordvintsev and A. K. Revision. *Feature Matching + Homography to Find Objects*. 2013. URL: [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_feature2d/py\\_feature\\_homography/py\\_feature\\_homography.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html).
- [5] A. Newell, K. Yang, and J. Deng. “Stacked Hourglass Networks for Human Pose Estimation”. In: *CoRR* abs/1603.06937 (2016). arXiv: 1603.06937. URL: <http://arxiv.org/abs/1603.06937>.
- [6] Y. Zeng et al. “Learning pyramid-context encoder network for high-quality image inpainting”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2019, pp. 1486–1494.
- [7] X. Zhou, D. Wang, and P. Krähenbühl. *Objects as Points*. 2019. arXiv: 1904.07850 [cs.CV].