Project Report: MyriAD: Tailored Dynamic Ads Generation Using Deep Generative Networks

Anupam Samanta
Stony Brook University

***Abstract*—Advertisements have played a major part in any company's success story. A success of great ad lies in the experience of a designer who can lay out the subtle nuances of the design in the ads. Given the recent rise of Artificial neural networks, is it feasible to train a network that can learn such cues based on a dataset of advertisements from a variety of domains, including websites, magazines, posters, etc? Can it be encoded to a new feature space which captures the essentials and aesthetics of a good design? In this project, we try to find this concept by using object detection and autoencoders to allow an artificial system to model these components, and to aid novice and expert users to refine their designs based on the knowledge over tens of thousands of existing advertisements.**

I. Introduction

THIS project aims to caters to an advertisement campaign for businesses on the online platform. Web advertisement has created a huge space for businesses to grow and use their online presence to drive sales and brand awareness. Google Adwords has been a popular ad syndication platform for online campaigns in publisher websites, alongside major social feed syndicators such as Facebook, Twitter, and other social media websites. The reason for web ads popularity is the ease with which a campaign could be set up and the high level of abstraction offered by the syndicator. Many formats and templates make the work simpler for the advertiser, accounting for the multitude of platforms and shapes in which their ad could be displayed, increasing its visibility. This is enough of a motivation for us to model a part of this as a machine-learning problem, specifically targeting ad creation, which is dominated by human ad designers.

In this report, we have captured the various components of a design by using object detection and text detection to extract the bounding boxes of elements of different categories. With this information in our hand, we use these coordinates as a blueprint of the original layout. For each category, we embed the information of the bounding box as one layer for each class. This way all objects of similar classes will have a mask that represents where each object are placed in the final design of the advertisement. This way, the data set can be converted to a series of the mask where each individual component of different categories is separated from the other.

This layer-wise separated image is now fed through an auto-encoder network where the latent dimensions are used as the hidden representation of the design. This representation can be considered as a design embedding of the image and can open up further areas of analysis of design as embeddings, and manipulate them for naively generated design to refine and improve designs.

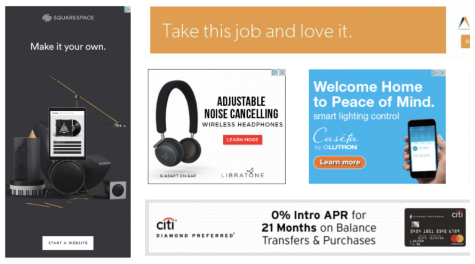


Fig. 1: Sample online ads commonly seen online

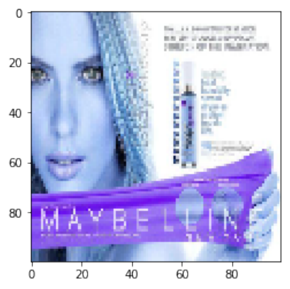
II. Dataset

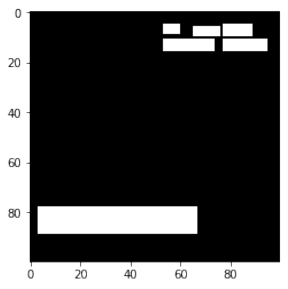
Advertisements have been presented over several mediums over the past few decades. Visual representation of brand logos and text and placement of their product, ranging from food to vehicles are a common way to leave an impression on a large base of customer. Ads can be categorized in generally three categories based on the medium propagated, Images, Videos, and Audios. This project focuses on image ads. Image ads are commonly seen in newspapers, magazines and recently a lot of them have been a popular way to promote a particular brand or cause over the internet. The dataset [1] used here has around 64,000 images from various categories such as an advertisement for electronics products, restaurants, beverages, fashion accessories, etc. Images ranging from high resolution to low resolution and black and white to color were found in the dataset. For uniformity, all images were resized to the scale 100x100 and then to 3 channel images.

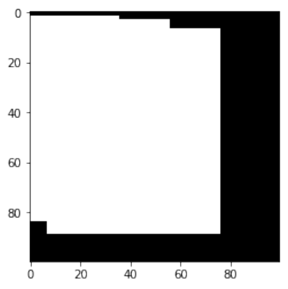
III. Related Work

As there are many components of design in an ad, major ones are text layout, image layout and possible interactive element such as button, etc placement. Similar to the work done by Deka et. al. [3], we start by focusing on only the textual component of the ad.

Choosing textual parts of the image is a challenging part because it has been a major focus of many research work to identify text appearing in wild, especially in multiple orientations. Non-machine learning models such as Stroke width transform are good candidates but are not good enough in our case, where there are many text orientations and possible camouflage with lots of background and foreground images that appear in the design. Turning to modern methods of


(a) Sample dataset image


(b) Mask for text


(c) Mask for detected human

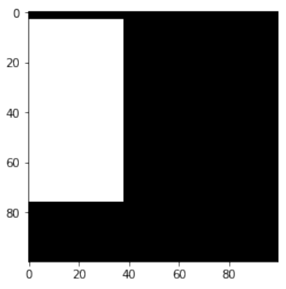

(d) Mask for face

Fig. 2: Dataset image and images obtained after running through the object detection and text detection pipeline

Optical character recognition such as Tesseract-OCR was not viable as they are well suited for text recognition within documents with high contrast between text and the background. As a possible solution, we cropped images for possible areas of text using canny edge detector and contour detection, and over that, applied OCR and SWT. But there were a lot of false positive and false negatives in these methods

Hence, a lot of images in our dataset were not well suited for direct application of OCR methods. So we looked up into more state of the art methods for text detection, especially, Multi-Oriented Multi-Oriented Text Detection with Fully Convolutional Networks (MOPS)[4] and EAST: An Efficient and Accurate Scene Text Detector(EAST)[2]. MOPS is a novel OCR detection approach where they are able to use activation of VGG16 network and deconvolve and sum them to get text boundaries. MOPS has a higher false positive rate, less accuracy and slower in detection. Whereas EAST is fast and accurate text detection system based on PVANET and locality-aware NMS Merging.

For the next part, the remaining visual components of the ads are extracted using state of the art object detection API's. We turned our attention towards the object detection models from the Tensorflow model zoo. The possible options were the single shot detector, Faster RCNN, and Region-Based fully convolutional networks. All of them are well known for their object detection capabilities and availability of trained model along with well documented pipelining methods for inference on our dataset for choosing among these model zoos.

IV. Method

The main idea of the project is rooted to generate latent embeddings of design that can be inferred over the entire dataset of the images but might not be explicitly observed as parameters for the design blueprint. To learn the embeddings, the images are processed through a pipeline, where each image is passed in parallel through two networks, for region detection of text and objects of similar categories. These inferences can be then fed to the encoder-decoder based network. A variational autoencoder can model a distribution from which the dataset is believed to belong to. Our observation is that when a new design is fed forward through this network, there is a variance in the reconstructed design generated. These reconstructions are based on the knowledge of the dataset where it is trained upon.

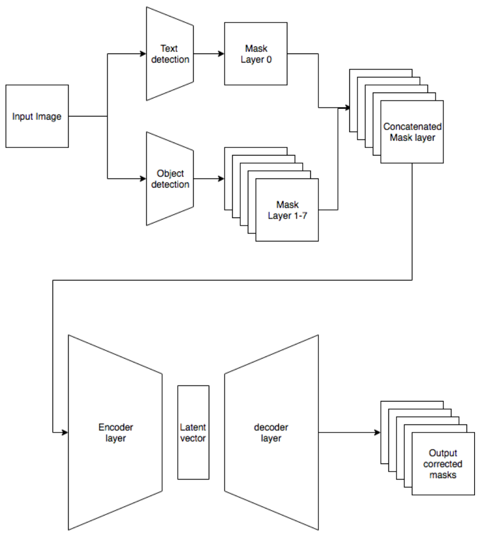


Fig. 3: Network model pipeline

The start of the pipeline is based on two inferences using state of the art inference models for text and object detection. For text detection we are using inference only implementation of the EAST system [2].

Using a Tensorflow implementation of EAST [2], the text in the images of the dataset were detected. The EAST system infers the coordinates of the bounding boxes, returning relative coordinates of the top left and bottom right points. The entire dataset was processed through the text detection system part of the pipeline. For all the images in our dataset, these textbox coordinates were stored separately as a metadata as input for the next stage of the pipeline. We created masks of images where text regions are polygons filled with white whereas rest of the portion of images are left black.

For object detection, we chose from the tensorflow object detection model zoo. Amongst available options of the single shot detector(SSD), Faster RCNN(FRCNN) and Region based fully convolutional networks RFCN, SSD was the fastest whereas FRCNN had the mean average precision score[6]. But both of them were trained on the COCO Dataset[7] and KITTI dataset[8]. The number of classes in both the dataset was around 90. The speedup was not a bottleneck for the pipeline as the object detection and text detection are a one time part of the pipeline. Also, the number of classes for openimage dataset[] was around 650, which was a huge advantage for our dataset as it is a mix of a lot of object classes. There is only 6% difference in mean average precision of RFCN trained on OpenImage Dataset over FRCNN model trained on the dataset. Hence it was more advisable to chose the RFCN over FRCNN even though the latter had a higher accuracy.

Similar to text detection, the output of the inference derived from the object detection API's for each image was stored to be fed later on in the pipeline.

Having obtained these masked images, we start aiming towards a latent representation of the various design elements. The masked images are in a higher dimensional space. We represent this dataset in a smaller lower dimensional space using an auto-encoder. An auto-encoder theoretically is a deep learned compression of the design mask. There are various types of auto-encoders that could have possibly worked here. But we are choosing only two types here, auto-encoder and variational auto-encoder.

For auto-encoders, we used 2 hidden layers of size 2048 and 256 of Relu units. We used a latent vector of size 64 units and trained it for 700 epochs. We resized the mask-images to a uniform size of 100x100 sized image. But the generated images showed a lot of salt and pepper noise using auto-encoders, that resulted in less accuracy between the generated and output image. Auto-encoders were both slow and resulted in more lossy generated images.

Due to this, we switched to Variational auto-encoders that generated images that were visually more coherent than the ones produced by simple auto-encoders. We used a 2 hidden layer auto-encoder with 2048 and 256 sigmoid neuron variational auto-encoder. We ran it for 70 epochs. We ran it on 8-CPU and 4 NVIDIA-P100 GPU's for 28 minutes for each run of training cycle.

An interesting variation to autoencoder is the beta-variational autoencoder[], that introduces another hyperparameter for the network, namely beta. It balances the loss of the latent space loss and the l2 loss incurred from reconstruction. This allows a lesser stress on learning the distribution from the data and also allows more on the reconstruction. Distribution from VAE's is tough to learn as compared to learning from the l2 distance between input and output images, hence increasing L2 loss helps in a faster convergence. Hence we used a beta-VAE with a beta of 0.1.

After training, we are using the encoded input features of the last encoded tensor of size 256 as our feature vector to represent the textual design of the image. We run a similarity search on all images to find similar images across the entire dataset. We show only the top 10 results. These results return the ads that have similar text component placement in the advertisement layout.

V. Results

Running the text detection EAST system took over an hour when ran on 8 NVIDIA Tesla P100 GPU's for over 64832 images. There are on an average 13 text region in each image and a maximum of 892 regions in one image. The distribution follows a Zipfian distribution where most of the images have a smaller text region count revolving around the average.

The object detection API took around 2.1 seconds per image to resolve one image and on an average was able to detect 4 objects per image. The object that was detected the most was 'Poster' given the object detection model consistently recognized half of the images in the dataset correctly as poster's/ads. The next most visible object was the either human's or body parts, signifying the presence of humans in around 1/4th of the ads. There were more than 254 classes detected in all the images among the 654 originally present in the open image dataset on which the open image dataset was trained. Also, the threshold for the confidence for which the object was detected was lowered to 0.3 to allow a number of objects being detected as even in low confidence, a randomly sampled subset of 24 images showed reasonably correct inference on objects detected even with low confidence.

An example of few classes and their counts:

Class Name	count
Poster	28397
Person	20257
Face	17737
Clothing	14437
Woman	11933
Man	8966
Car	6120
Footwear	4378
Bottle	3895
Wheel	3584
Drink	2944
Girl	2744
Mobile phone	1872

TABLE I: Example of classes and their counts

Taking into account the design point of view, we cluster few classes that are similar in a category. This has a 2 fold benefit, first, it reduces the number of layers of the image that is to be fed to the autoencoder as well as increase the number of examples of a particular class. We also filtered classes that appeared in less than 20% of the images in the dataset, to allow better training with less number of classes and more accurate results.

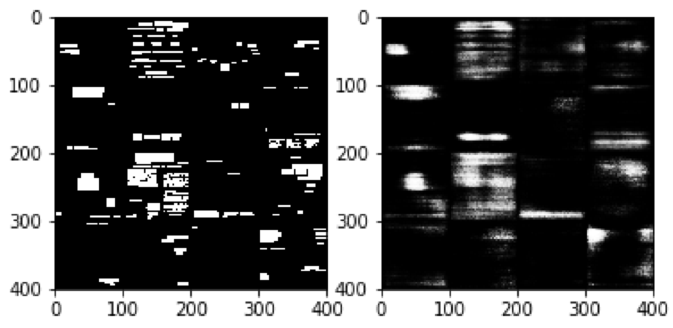
After clustering, we took into account only 4 layers of classes. The categories were: Text regions, Humans, such as man, woman, child, etc; Body parts such as hands, face, legs, etc; and vehicles comprising of cars, trucks, buses, etc.

Class name	classes	count
Human	Girl, Boy, Man, Woman	44429
Body part	Face, Mouth, Head, Eye, etc	24164
Vehicles	Car, Wheel, Land vehicle, Auto part, etc	13831

TABLE II: Top Clustered classes and their counts

For the autoencoder, a 2 layer autoencoder with 256-sized latent vector was observed to be working the best. Using this learned representation, new ad layouts can be generated with the latent vector initialized randomly from a Gaussian distribution. Also passing a new design through the feedforward network creates a new reconstructed image that is an improvement over a naively designed ad layout.

To create an ad layout, we used the output cells of a Jupyter notebook the interactive mode and color-coded different classes. We are able to place components that represent


(a) Text layer

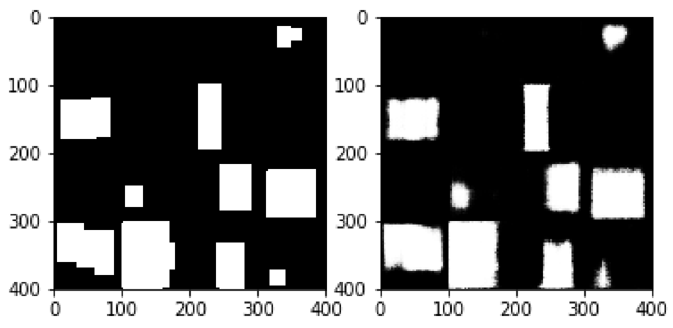

(b) Human mask layer

Fig. 4: Original vs reconstructed mask of randomly selected 4 test images

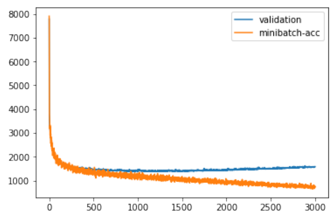


Fig. 5: Validation and train error loss curve over 70 epochs

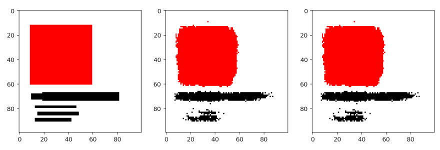


Fig. 6: Reconstruction of a design blueprint. The left image is the original blueprint that was fed to the network, the middle image and the right image are the reconstructed images. The black regions specify text region, whereas the red regions vehicle region. This reconstruction shows centered components of both text inputs regions and vehicle class region.

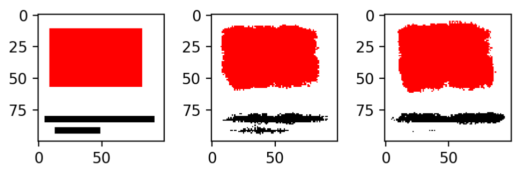


Fig. 7: Reconstruction of a design blueprint. The last image is from tweaking the latent representation, by changing value of the 2nd dimension by a value of -5. In the rightmost image, The network suggests merging of two lines and also moving the lines to the center.

different classes and then passing through the network creates changes in the image and allows for a more refined design layout. Figure [] show that the new designs generated for a naively designed layout. The initial blueprint shows text regions in pink and photos of humans in green regions, as well as black for showing vehicles. We can see that the text are more clustered together and oriented towards the center of the images with more pleasing margins. For the green regions, the network suggests moving the region to a slightly centered position. These subtle movements place the components as learned from the dataset of a similar object Also, using the embedding, playing around with the values of the embedding for the generated layout creates new designs. As we change the values from -10 to 10 for each dimension, each dimension of the embedding shows control for a unique aspect of the design. For example, changing the 2nd dimension of the latent representation moves the components to the center of the image, and the first one moves dilate the components of the image, etc. []

VI. Future Work

As we can see, the latent representation learns design cues from the ad dataset. Modifying the latent representation of any blueprint is leading to different designs being generated. Analyzing the latent representation to infer what are the design cues being learned and correlating it with design ideologies and concepts that are currently in use would be a good place to start for the next step of the project.

Also, autoencoders give masks that are not definitive for inferring boundaries. As the number of the bounding boxes for different classes for different images are variable, there are only two options: creating masks for each class and concatenate them together to create a multi-channel image. That is what we have done in this project. Another approach could be using RNN's that could accomplish input of a variable number of bounding boxes coordinates, and a possible hidden representation of the design of the image using only these coordinate information and the category of the bounding box. This could be similar to what neural translation machines use where they generate embeddings based on the input sentence, but the major difference in our case would be to use the input as the target output. Thus, it could be an RNN based autoencoder, and the output could be the

VII. Conclusion

In this project, we see that it is possible to embed the higher design principles into a 40 dimension latent vector. Training an ad dataset allows the model to learn nuances of the design principles. Currently, this system is still in an early phase but with better object detection API's and better autoencoder representation reconstruction, we can actually generate images that are more cohesive from a designer perspective. Currently, a lot of challenges are there to learn boolean masks for design components and are tough to be learned by variational autoencoders. There might be a better alternative to design embeddings that are rooted from this principal idea: extracting components from the image, learning representation and then using them to aid users in creating designs without a lot of experience in this domain.

References

[1] Automatic Understanding of Image and Video Advertisements. Zaeem Hussain, Mingda Zhang, Xiaozhong Zhang, Keren Ye, Christopher Thomas, Zuha Agha, Nathan Ong, Adriana Kovashka. To appear, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[2] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang *EAST: An Efficient and Accurate Scene Text Detector*.

[3] Biplab Deka, Zifeng Huang, Chad Franzen. Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols and Ranjitha Kumar *Rico: A Mobile App Dataset for Building Data-Driven Design Applications*.

[4] Zaeem Hussain, Mingda Zhang, Xiaozhong Zhang, Keren Ye, Christopher Thomas, Zuha Agha, Nathan Ong, Adriana Kovashka. *Automatic Understanding of Image and Video Advertisements* .

[5] Zheng Zhang, Chengquan Zhang, Wei Shen, Cong Yao, Wenyu Liu and Xiang Bai. *Multi-Oriented Text Detection with Fully Convolutional Networks*.

[6] https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[7] Lin TY. et al. (2014) *Microsoft COCO: Common Objects in Context*. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham

[8] *Vision meets robotics: The KITTI dataset* A Geiger, P Lenz, C Stiller, and R Urtasun The International Journal of Robotics Research Vol 32, Issue 11, pp. 1231 - 1237

[9] Krasin I., Duerig T., Alldrin N., Ferrari V., Abu-El-Haija S., Kuznetsova A., Rom H., Uijlings J., Popov S., Kamali S., Malloci M., Pont-Tuset J., Veit A., Belongie S., Gomes V., Gupta A., Sun C., Chechik G., Cai D., Feng Z., Narayanan D., Murphy K. *OpenImages: A public dataset for large-scale multi-label and multi-class image classification, 2017*. Available from https://storage.googleapis.com/openimages/web/index.html.