Predicting Memorable Regions Of Images Using Deep Learning and Adversarial Networks

Babar Khan

April 9, 2023

Contents

1 Introduction

1.1 Background

1.1.1 We see a lot of images

We are constantly surrounded by imagery, on the way to work, on the internet, on TV, in stores, etc. Some of them stick out more than others, we see hundreds, if not thousands, of images a day, and yet culturally and individually we all remember the same ones.

1.1.2 Cultural vs Individual Memorability

Due to cultural significance an image can become memorable. The 2015 dress [1], a country's flag, or the 1932 image "lunch atop a skyscraper" [2], come to mind. I would argue that the memorability of these images is tied to the culture surrounding them, not necessarily due to intrinsic properties within. The focus of this research is on the intrinsic properties within an image that make it memorable to an individual, like seeing an advert on a bus and then later recognising the same advert online.

1.1.3 How do we measure the memorability of an image

It has been shown that the memorability of an image is an intrinsic property, independent from the viewer [3, 4, 5, 6]. This was achieved by performing a memorability game where participants are presented with a stream of images, and on an interval shown an image that they had already seen. Most participants were able, or unable, to remember the same images. How often an image was recognised is proportional to its memorability. This is taken further by Akagunduz et al. [7] where, in a similar experiment, they measure which regions of images are memorable.

1.1.4 Why do we want to create a memorable image

1.1.5 How does this research help to create a memorable image

1.2 Motivation

1.3 What are my goals

1.4 Structure

2 Literature Review

2.1 Memorability

In a study by Isola et al., it was found that some images were intrinsically more memorable than others [1]. In another study by Isola et al. [11] they found that correlations between properties such as mean saturation, and the log number of objects, has less impact on the memorability score than object statistics. Categories such as: person, person sitting, and floor, were most helpful image memorability. The categories: ceilings, buildings, and mountains, were least helpful.

In both [1, 11] they tested participants' abilities to remember specific images after repeated exposure. Their goal, similar to that of this paper, was to identify a collection of visual attributes that make images memorable, and to use those to predict the memorability of an image. Their approach was limited by the fact that the object statistics were annotated by hand, this would both make automating the process of determining image memorability impossible, and limit them from finding any abstract properties that helped/hindered memorability.

Recently, the use of deep convolutional networks [12,13] has proven exceptionally successful in image classification and object recognition. It stands to reason that a combination of these systems could perform well at this task. It could recognise which objects are within an image, and to use that to help inform how memorable the image is.

Khosla et al. [4] built on the work in [1, 11] by, instead of determining memorability of an entire image, creating a model that discovers memorability maps of individual images without human annotation. These memorability maps are able to distinguish which areas of an image are remembered, forgotten, or hallucinated. Their approach, similar to Isola et al. in [1, 11], is limited by the arbitrarily picked list of features that define memorability.

2.2 Using Deep Learning to Predict Memorability

Khosla et al. have created a dataset, LaMem [5], containing 60,000 annotated image and memorability pairs. They use deep learning to predict these memorability maps, this is similar to the VISCHEMA plus dataset [7], which I will be working with. In [7] the network output utilises fully connected layers which I believe is unnecessary, and may even hurt performance. More modern computer vision network structures, such as the U-Net [8] or a GAN [9], use fully convolutional networks. These maintain spatial locality which makes them less prone to overfitting, and also allows them to generalise better. Previous work in [1,4,11] relied on human annotated features, using deep learning we should be able to automate this process by learning the features from a sufficiently large dataset such as those presented in [5,7].

I propose that predicting memorability maps is an extension of image segmentation, modern machine learning has found many network architectures that would be worth exploring for this problem. Isola et al. created Pix2Pix [10], a deep learning architecture based on Conditional Adversarial Networks. It has found great success but I think that it is too powerful a network for this task, and may cause overfitting, thus a structure akin to this could prove very effective. The U-Net [8] has also found great success in image segmentation, this system however relies on the output being a classification. Something in between these two could work very well.

3 Methodology

3.1 Requirements Capture

We have a dataset of images that we will aim to accurately to map onto their corresponding labels. Through learning to map these images onto their corresponding VMS labels, we hope that our system will gain a general understanding of the data, such that when another image that matches our distribution is presented, the system can accurately create a VMS label for that. Ideally our system should create accurate mappings, we can test this by using a loss function such as L1 Loss and through qualitative analysis.

3.2 Analysis/Design

I will create a deep learning image to image model that will be trained on the training data portion of the VISCHEMA PLUS dataset, it will be evaluated on the validation data portion of the dataset. The network will be trained to directly output the corresponding VMS map.

This model will be trained as a Generative Adversarial Network, this GAN will contain a generator network and a discriminator network. The Generator will be an image to image network which maps the image onto a VMS map. The Discriminator will take an image and say if it belongs to the distribution of VMS maps or not. My training loop will be composed of an image label pair being taken from the dataset, the image will be passed into a generator, and then a fake label will be produced. The Discriminator will be passed the input image and a corresponding VMS, either produced by the generator or the real one, and through backpropagation its weights will be updated depending on if it answers correctly or not. The generator will have its weights updated through backpropagation depending on if it can fool the discriminator or not.

In the pix2pix network the Discriminator produces 2 dimmensional array of boolean values, rather than a single value, describing whether a region of the image is from the labels of the datset, or if it is generated. I will experiment to see which approach produces better results.

I will also experiment with training the network to output the composition of the image and the label, and then manually subtract the image after. Because the difference between the image and the label is much greater than the difference between the image and the image + label I suspect that this should be easier for the generator to learn the weights for. This

may produce better results and is something I will experiment with, I'm not entirely convinced however because GANs have been shown to struggle when one network is much better than the other, I suspect that the discriminator would have a harder time being able to tell the difference and may make learning impossible.

3.3 Implementation

I will implement this network in Python using the PyTorch machine learning framework. This is because I am familiar with it rather than it having any advantages over other existing frameworks, and the methodology that I describe should produce the same results in any programming language or framework.

3.3.1 Network Architectures

Generator: I will use a U-Net [U-Net paper] as my image to image network. The U-Net makes use of an Encoder block and a Decoder block, there are residual connections between different sections of the blocks that help to preserve image data in the decoding stage. The network has seen great success in the medical field. [U-net success paper].

Discriminator: I will compare two different discriminators. One such that the range is 0—1, either real or fake. An alternative presented in [pix2pix paper] is a PatchGAN, this will produce a 2d array that clsasifies each NxN patch within an image as real or fake.

Training within a GAN is very sensitive to differences in loss between the two networks, if one performs much better than the other it can become impossible for the other to improve, therefore I will experiment with different values of the following variables:

1. The Number of layers in the Generator and Discriminator
2. The number of channels in each layer of the Generator and Discriminator

By tweaking these I should be able to create equally powerful networks.

3.3.2 Loss Functions

The loss for each network is computed differently.

Generator Loss: When training the Generator the loss is the sum of our loss when tricking the discriminator and the pixelwise difference between our

produced label and the real label. Our network will produce a fake label, Lf.

L = MSE(D(Lf, image), 1) + L1(Lf, Lr)

Discriminator Loss: When training the Discriminator the loss is calculated as the mean value of how close our prediction was to the correct answer across a fake and real label.

L = 0.5 * (MSE(D(Lf, image), 0) + MSE(D(Lr, image), 1))

In both networks the loss is computed across an entire batch of images and then the weights are adjusted with backpropagation.

When tracking the progress of the GAN the L1 loss of the produced label vs the real label can be used, as it is independent of the discriminator, and therefore should improve over time. Every epoch I will track the value of this across the validation dataset and use that to inform me about how well our network is training. It should inform me of whether the network is overfitting, underfitting, or training well.

3.3.3 Training Strategy

Typical training (gradient descent steps using the loss of the output vs the label) vs GAN (Training a generator to produce the labels and a discriminator to determine if they are from the dataset or not)

Variables that I will modify: 1. Normalisation method: a. Batch Norm b. Layer Norm c. Instance Norm 4. Dropout 2. Optimisation function: a. Adam b. SGD c. Adagrad d. Adadelta 3. Other Hyperparameters: a. Learning Rate b. Batch Size

Mention the fact that when testing it is not feasible to test every combination of these variables. Instead I will test every variation of normalisation method, pick the best, and then use that when testing for the best optimisation function, then use both of those etc.

3.4 Analysis/Testing

Using all of the different combinations of parameters that I have outlined in my methodology, I will train models to map from the image set to the label set, and then I will evaluate these in multiple different ways. The most important will be the L1 loss across the validation set, this will measure the mean absolute difference between each pixel in the output of the network

and the corresponding label. The smaller this number the closer the network is to achieving a direct mapping from the image set to the label set.

I can also use the L1 loss across the training and validation sets to determine if my model has become overfit to the training data. If the L1 loss has become significantly smaller across the training dataset vs the validation dataset, this will imply that the model has become overfit to the training set, and instead of learning the general pattern of the distribution of images, it is instead memorising the mappings. This will be a useful tool during development, and will allow me to make changes to prevent it. If the L1 loss across my train and validation data is similar then it will imply that my network is able to learn a general mapping of the data. If they both stop improving it may mean that some improvements could be made by making the network more complex (either by adding more layers or adding more channels to my layers).

I will also examine the images qualitatively, do the images look like they are the same, are there any large issues (for example the large blue blobs or yellow areas)

4 Experimental Results

This model was trained on a computer using an RTX 3070 with 8GB of VRAM, testing all of my hyperparameter options took approximately 3 days and I trained my final model over the course of days.

4.1 Best Training Hyperparameters and Model Parameters

I wanted to find which parameter and hyperparameter combinations would perform the best. I decided to automate this process to save time. I wanted to vary the following:

1. The normalisation layer for both the generator and the discriminator.
2. The channel layouts for the generator and the discriminator.
3. The optimiser used for the generator and the discriminator.
4. The learning rates used for each optimiser.
5. If the Adam optimiser was found to be ideal for either the generator or discriminator, then to use different beta options.

I explored the following options:

Normalisation layers:

- Batch Normalisation
- Instance Norm

Generator channel layouts:

- encoder:(3, 64, 128, 256, 512, 1024), decoder:(1024, 512, 256, 128, 64),
- encoder:(3, 32, 64, 128, 256, 512), decoder:(512, 256, 128, 64, 32),
- encoder:(3, 50, 100, 200, 400, 800), decoder:(800, 400, 200, 100, 50),
- encoder:(3, 100, 200, 400, 800, 1600), decoder:(1600, 800, 400, 200, 100)

Discriminator channel layouts:

- (6, 64, 128, 256, 512, 1024),
- (6, 32, 64 , 128, 256, 512),
- (6, 50, 100, 200, 400, 800),

- (6, 100, 200, 400, 800, 1600)

Optimisers:

- SGD, using the following learning rates (0.005, 0.01, 0.02)
- Adam, using the following learning rates (0.0005, 0.001, 0.002), and using the following betas ((0.9, 0.999), (0, 0.999), (0.5, 0.999))
- Adadelta, using the following learning rates (0.5, 1, 2)

Exploring this search space exhaustively is unfortunately not feesible, there are over 5000 different combinations possible, and if I tested each combination once for 100 epochs then it would take approximately 300 days to test. Instead I will have to explore a subset of this search space. I estimated some good default parameter and hyperparameter values, by varying these values I can lower the scope of the search space to approximately 100 combinations, which took $\sim$3 days to test. Unfortunately this did mean that every combination hasn't been tested, however it should give us a good approximation of the best combination.

Note: I didn't need to specify default values for the normalisation layer as these were the first variables I tested.

Default Generator Parameters and Hyperparameters:

- Encoder Channel Layout: (3,64,128,256,512,1024)
- Decoder Channel Layout: (1024,512,256,128,64)
- Optimiser: Adam, betas = (0.9, 0.999)
- Learning Rate: 0.001

Default Discriminator Parameters and Hyperparameters:

- Channel Layout: (6,64,128,256,512,1024)
- Optimiser: Adam, betas = (0.9, 0.999)
- Learning Rate: 0.001

I then iterated over each different parameter and hyperparameter and varied each one sequentially. I tested each combination for 100 epochs and used the L1 loss across the validation dataset as the score, if any parameter options performed better than the default then they would be used going forward. This meant that I only had to iterate $\sim$100 combinations. This took 3 days to test, and at the end I found that the following combination was the best:

Best Generator Parameters and Hyperparameters:

- Normalisation Layer: Batch Normalisation
- Encoder Channel Layout: (3,32,64,128,256,512)
- Decoder Channel Layout: (512, 256, 128, 64,32)
- Optimiser: SGD
- Learning Rate: 0.01

Best Discriminator Parameters and Hyperparameters:

- Normalisation Layer: Batch Normalisation
- Channel Layout: (6,100,200,400,800,1600)
- Optimiser: Adam, betas = (0.9, 0.999)
- Learning Rate: 0.001

With this combination we achieved a loss of $\sim$1.02 across the validation dataset. I found other good results using similar combinations. Using the Adam optimiser for both the generator and discriminator achieved a loss of $\sim$1.04 across the validation dataset. Using the Adatelta optimiser for the generator and the Adam optimiser for the discriminator achieved a loss of $\sim$1.03 across the validation dataset.

Varying Normalisation

Varying Optimiser

Varying Loss Function

PatchGAN vs Boolean Discriminator

Benefits of Residual Connections

4.2 Results

4.3 Evaluation Metrics

4.4 Analysis

5 Conclusion

6 References

[1] BBC. (2015, Feb) Optical illusion: Dress colour debate goes global. [Online]. Available: https://www.bbc.com/news/uk-scotland-highlands-islands-31656935

[2] M. Gambino. (2012, Sep) Lunch atop a skyscraper photograph: The story behind the famous shot. [Online]. Available: https://www.smithsonianmag.com/history/lunch-atop-a-skyscraper-photograph-the-story-behind-the-famous-shot-43931148/

[3] P. Isola, J. Xiao, A. Torralba, and A. Oliva, "What makes an image memorable?" in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 145–152.

[4] P. Isola, D. Parikh, A. Torralba, and A. Oliva, "Understanding the intrinsic memorability of images," in *Advances in Neural Information Processing Systems*, 2011.

[5] A. Khosla, A. S. Raju, A. Torralba, and A. Oliva, "Understanding and predicting image memorability at a large scale," in *International Conference on Computer Vision (ICCV)*, 2015.

[6] P. Isola, J. Xiao, D. Parikh, A. Torralba, and A. Oliva, "What makes a photograph memorable?" *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 7, pp. 1469–1482, 2014.

[7] E. Akagunduz, A. G. Bors, and K. K. Evans, "Defining image memorability using the visual memory schema," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 9, pp. 2165–2178, 2020.