GENERATIVE MODEL REGULARIZATION FOR FEATURE VISUALIZATION BY OPTIMISATION

Maxwell Clarke – maxeonyx@gmail.com – 300559026

COMP421 Project – Sat 24 Oct 2020 – Victoria University of Wellington

See the code on GitHub for the full experiments

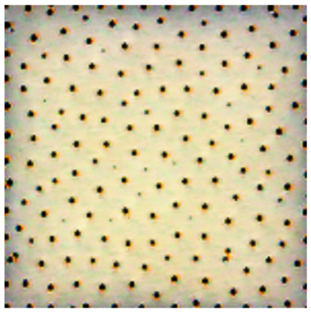

Fig. 1: GoogLeNet Layer 3A

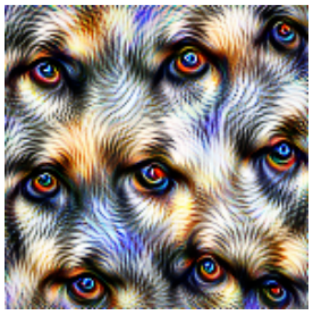

Fig. 2: Layer 4A

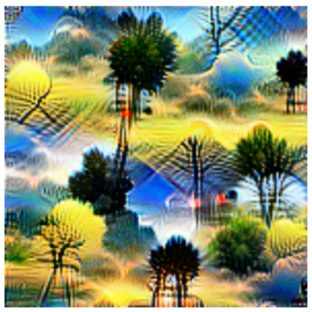

Fig. 3: Layer 4B

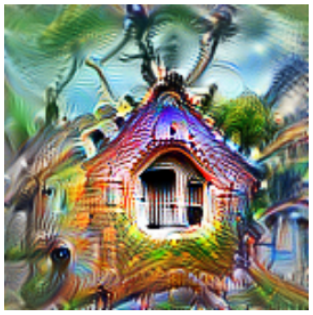

Fig. 4: Layer 4C

ABSTRACT

Feature visualization by optimisation [1] is a method of producing images from a neural network, by optimizing an input to maximize activations of neurons within the network. This can help us to visualize the behaviour of the network, although whether this is useful for finding problems is unclear.

Images produced by this method suffer from high-frequency artifacts and are not very interpretable. To address this, a variety of regularization methods can be used. One such method is constraining the optimization to the latent space of a generative model. This is similar to using dataset examples, but with the benefit that a generative model can produce images not in the dataset.

We implement a classifier on a synthetic dataset, and use feature visualization by optimization to visualize it. We then create a generative model using an auto-encoder, and compare visualizations using naive optimization, transformation robustness regularization and generative model regularization.

Contents

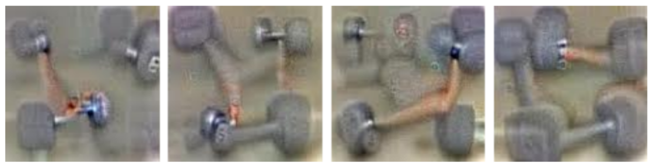

Fig. 5: GoogLeNet undesirably looks for muscular arms on the ImageNet class "dumbbell" [5].

1. INTRODUCTION

1.1. The Paper

In the paper "Feature Visualization" [1], the authors discuss a family of methods for inspecting the behaviour of neural network called *feature visualization by optimization*. With this method, we inspect the behaviour of components of a network such as single neurons, channels, layers and class outputs, using an optimization process to create the "optimal input" for activating some component of a given neural network. Some examples of images produced by this method using a GoogLeNet [2] model trained on the ImageNet [3] dataset are shown in Figures 1 to 4. From this, we can start to see what sort of features in the input distribution activate these network components.

The work is motivated by criticism of deep learning models for having poor interpretability. Neural networks are usually viewed as black boxes, and undesirable behaviour can go unnoticed, such as very poor performance on out-of distribution data, or susceptibility to adversarial examples [4]. The sub-field of neural network interpretability aims to surface these concerns by providing methods for inspecting and visualising the inner behaviour of deep neural networks.

One example of a problem that can be found with these methods is this classic example from [5], shown in Figure 5. These images, produced with feature visualization by optimization on the class output for the class "dumbbell", show that the network believes that no dumbbell is complete without a muscular arm holding it. This is due to having no training examples in the dataset with only a dumbbell. Using these visualizations, we can identify issues like this and correct them.

Feature visualization by optimization existed prior to this paper. For example, it was used to create *adversarial examples* in [4], to create neural network "hallucinations" in DeepDream [6], and to inspect neural networks (as we do here) in [7], [8] and [9]. The contributions of this paper are not in the methods, but rather:

1. Providing an engaging introduction to the ideas.

2. Summarizing prior work.

3. Providing (interactive) comparisons of various regularization and parameterization methods.

Despite the motivation, and existing work in this area, it is still not clear how neural network interpretability is useful in practice. From looking at the produced visualizations, we only gain an informal intuition of what the neural network does. This is helpful for our understanding of a particular model, but not directly helpful when looking for undesirable behaviour, or when making neural network architecture choices. Effectively using neural network visualizations to inform decisions about training and deployment of models would be very useful, but is still a topic of research.

1.2. Feature Visualization by Optimization

How does this method actually work?

Feature visualization by optimization works by performing a similar gradient descent process as training the weights of a neural network, but with respect to the input instead of the parameters. This is also called *activation maximization*.

For clarity, the training step of parameter optimization is as follows, for parameters θ, random input batch x_B, learning rate ϵ, network $f(x, \theta)$ and loss function $L(x, \theta)$.

$$\theta_{k+1} \longleftarrow \theta_k - \epsilon \frac{\delta L(x_B, \theta_k)}{\delta \theta_k}$$

In feature visualization by optimization, we instead optimize an input x against pretrained θ which remain constant. We replace the loss function with O and call it an *objective*.

$$x_{k+1} \longleftarrow x_k + \epsilon \frac{\delta O(x_k, \theta)}{\delta x_k}$$

The objective is the activation value of some component of the neural network. Note that this is then typically gradient ascent, because we are interested in the maximum activation of a component. In principle, the minimum activation could be found as well, however this is not straightforward for ReLU networks, because neurons only output in the positive range.

Let $a_i(x, \theta)$ be a subset of the neural network $f(x, \theta)$ which outputs only the value of neuron i. Because f is a feed-forward network, all other parts of the network can be simply ignored. i is actually a composite index which includes the layer, position and channel. Then, we can formulate some objectives for our optimisation. First, a single neuron objective for neuron i:

$$O(x, \theta) = a_i(x, \theta)$$

A channel objective for a channel c of a convolutional layer l, where W and H are the width and height:

$$O(x, \theta) = \sum_i^W \sum_j^H a_{l,i,j,c}(x, \theta)$$

An objective combining two arbitrary neurons i and j. We could also give them different weights.

$$O(x, \theta) = a_i(x, \theta) + a_j(x, \theta)$$

1.3. Regularization and Parameterizations

Unfortunately, just optimising the objectives is not as simple as it seems. Naively, we end up with images with high-frequency patterns such as those shown in Figures 6 and 7.

These high-frequency patterns *are* revealing something about the network structure, most likely to do with strided convolutions and pooling. However, they are essentially the same no matter the objective we choose, and so are not helping us to understand the network.

In order to fix this, we can introduce regularization. The most straightforward method is *transformation robustness*. This regularization method randomly transforms the image at each optimisation step. As a result, the network can no longer rely on localized high-frequency patterns, because they may be shifted by the transformation. Some examples using our classifier (introduced later) are shown in Figure 7. In particular, the images on the right are created with both random translation and rotation in between optimisation steps.

More generally, we can use any differentiable function f which produces the input x to the classifier.

$$x = f(z)$$
$$O(z, \theta) = a_i(f(z), \theta)$$

We have now re-framed the optimization problem in terms of z instead of x. This is called a parameterization. This is a useful thing to do because it can change the shape of the optimization problem, widening some basins, shrinking others, and constraining or broadening the search space.

The kind of function that can be used is very broad, and includes optimising in the fourier basis before scaling the frequencies, or optimising the weights of a *Compositional Pattern Producing Network*. Even more exotic methods such as differentiable 3D rendering are explored in "Differentiable Image Parameterizations" [10].

We will focus on one particular parameterization, which is the latent space of a generative model.

1.4. Generative Models

We can use a pre-trained generative model to produce images, which in turn can be fed into the classifier. This creates a differentiable mapping between the latent space of the generative model and the classifier neurons which are our objective for feature visualization. We can then optimize this objective by moving within the latent space. This is re-parameterizing the optimization problem based on the generative model.

However, if our goal is to constrain the optimisation to images in the data distribution, it is surely easier to just feed the classifier dataset examples, and sort by the objective value. How is this better than simply searching over images in our dataset?

The benefit of a generative model is that we can in theory sample from any point in the image distribution. A "realistic" image that most strongly activates a particular neuron might in principle exist but not be present in the dataset. A good generative model should be able to create such an image. In addition, it should allow us to achieve complex objectives which might be useful for visualization, such as creating diverse examples (penalising similarity between multiple examples in a batch), or interpolating between objectives.

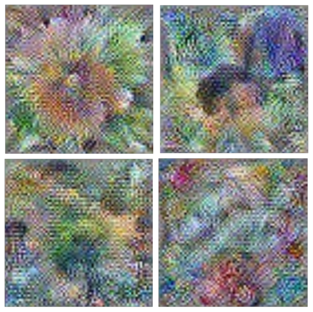
Fig. 6: Channel objectives on GoogLeNet without regularization from [1]

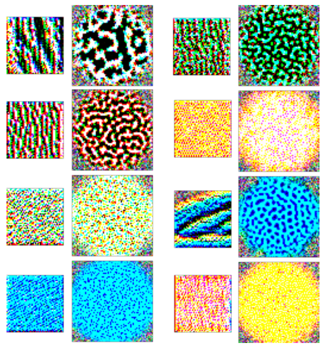
Fig. 7: Channel objectives on our classifier, without (small) and with (big) transformation robustness regularization.

Nguyen et al. implement this method in [11]. They train both a GAN and a classifier on ImageNet, and use the former to produce images based on objectives from the latter, with good results.

Any kind of generative model can in principle be used for this purpose, including GANs, and autoencoder variants such as variational autoencoders or bounded information rate autoencoders [12].

The downside of using a generative model parameterization is the same as using dataset examples. The generative model may have learned biases from the dataset that will cause us to miss issues such as the dumbbell example (Figure 5).

2. OUR CONTRIBUTION

We train a classifier on a synthetic dataset of geometric shapes, and create an implementation of feature visualization by optimization to visualize some components of the classifier.

We then visually compare 4 methods of regularization for visualizing features: no regularization, transformation robustness, dataset examples and generative model parameterization.

For the generative model, we train a basic autoencoder on our dataset. We then attempt to optimize some visualization objectives using the decoder as a differentiable image parameterization.

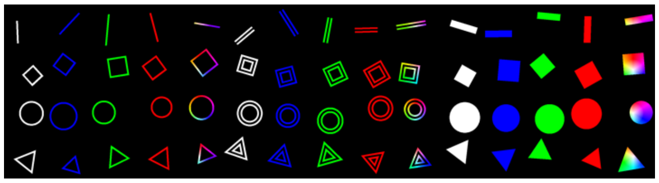

Fig. 8: Synthetic dataset used for the experiments

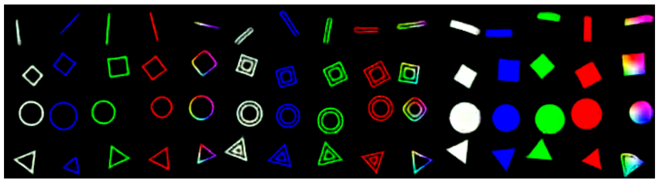

Fig. 9: Auto-encoder reconstructed images

3. EXPERIMENTS

3.1. Synthetic Dataset

We created a synthetic dataset of 24000 48x48 images with 60 classes for the experiments. Example images of all the classes are shown in Figure 8.

The dataset was split into training, testing and validation sets with a $8:1:1$ ratio respectively.

Each image has 5 varying attributes. There is one class for each unique combination of the first three attributes.

- Shape
- Single Line / Double Line / Filled
- Color (including Rainbow)
- Translation
- Rotation

3.2. Classifier

A convolutional network was trained as a classifier on this dataset. It uses a feed-forward architecture with pooling layers. This classifier was then used to create the optimization objectives - it is this model which we visualize when we create the images in the following experiments.

The classifier has 5 convolutional layers followed by 2 fully-connected layers. The final layer has a 60-wide softmax output.

3.3. Feature Visualization

Shown in Figure 7 are 8 pairs of images. These were each produced by maximizing a different *channel objective*. A channel objective is the mean activation over all of the neurons for a particular channel in a convolutional layer. These images visualize each of the 8 channels in the third convolutional layer of our classifier. On the left of each pair, the image was produced with naive optimisation (no regularization). On the right, transformation robustness was used. In particular, before the optimisation step is applied the image is translated (jittered) by 0 to 20 pixels in both x and y (independently), and is then rotated by a random angle from 0 to 2π.

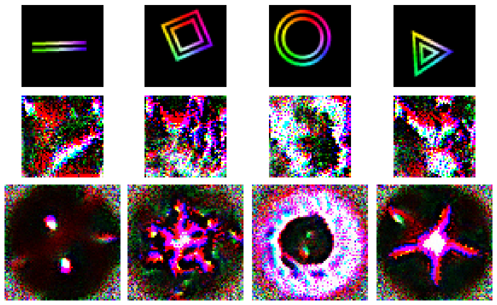

Fig. 10: Class output objectives on our classifier. Top: target class example. Middle: no regularization. Bottom: transformation robustness regularization

We can see that the two images of each pair bear some relation, especially by color, but are clearly different.

Shown in Figure 10 are images produced with a *class output objective*. This is the activation of a single neuron in the final dense layer, before the softmax. Specifically, the objectives are respectively the class outputs 9, 24, 39 and 54, which are “Rainbow Double Line”, “Rainbow Double Square”, “Rainbow Double Circle” and “Rainbow Double Triangle”.

These vizualizations bear little resemblance to shapes, but we can see that transformation robustness regularization helps to make them visually distinct. These images are typical results when optimising for class outputs of the respective shapes.

3.4. Autoencoder

An autoencoder network was trained on this dataset. Like the classifier, it uses a feed-forward architecture with pooling layers, and upsampling layers in the decoder. We use a 6-dimensional latent space, which proves sufficient to get good reconstruction error on this dataset.

Example images from each class after reconstruction by the auto-encoder are shown in Figure 9.

The autoencoder is trained using a mean-squared-error reconstruction loss. No constraints are applied to the latent layer as this is a standard autoencoder. This is not ideal, because the latent space is likely not to be very smooth. Using a variational autoencoder or witness-function-regularized autoencoder would be better for achieving good latent representations. However, those properties are not really neccessary for the proof of concept.

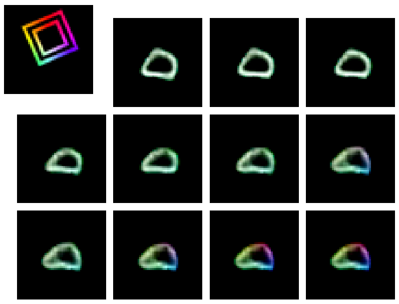

Fig. 11: Optimizing a class output objective in the latent space of the decoder. Due to vanishing gradients, the optimization is slow and very prone to local minima. Top left: Target class example.

3.5. Generative Model Regularization

We then implemented feature visualization using the decoder from the autoencoder to reparameterize the optimization. An example training sequence with a class output objective is shown in Figure 11.

Unfortunately, the results are underwhelming. Even with many training steps, the optimization will never move very far from the random starting position in the latent space. As we see in the figure (which has been selected because it shows more convergence than typical) there is some movement happening, and in what appears to be the correct direction, however the convergence stops at this point.

We believe this behaviour is due to the vanishing gradient problem. After the decoder is prepended to the classifier, there are 13 dense or convolutional layers between the decoder input (latent space) and class output objective. This is likely too many for a reliable gradient. We could fix this with different network architecture choices for both networks, such as using residual layers, or simply reducing the number of layers.

However, instead of making these changes, we simply used random sampling in the latent space to find good latent vectors, instead of gradient descent. 6 dimensions is few enough that we can cover most of the regions of interest with a tolerable number of samples. This allows us to still compare the generative model as a parameterization method despite the vanishing gradient problem.

Shown in Figure 12 are the 8 best images for a class output objective, from 500000 uniform samples in the latent space. Comparing to the rainbow square visualization from Figure 10, we can see that using the generative model is much more helpful for visualizing the class outputs. However, it is terrible for visualizing the lower-level channel objectives that we saw earlier, as we see in Figure 13, comparing to the corresponding objectives in Figure 7.

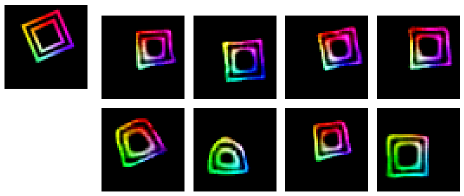

Fig. 12: Generated images corresponding to the best 8 latent vectors for a class output objective. Found by random sampling, not gradient descent. Top left: Target class example.

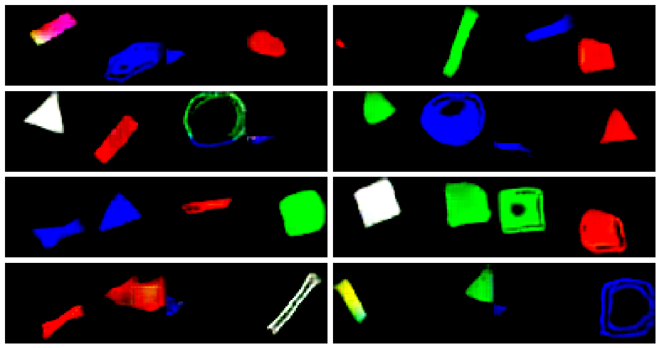

Fig. 13: Generated images corresponding to the best 4 latent vectors for each of 8 different channel objectives. Found by random sampling, not gradient descent.

4. SUMMARY

Feature visualization by optimization is a very useful tool for understanding neural networks. It allows to visualize the low- and high-level features that a neural network responds to, and can help us identify deficiencies in our dataset.

A basic implementation is very simple with modern machine learning frameworks such as Tensorflow, but more sophisticated parameterizations may require care to implement correctly, such as to avoid vanishing gradients.

We are very interested to learn more about extensions to these techniques, such as how they may be usefully applied to non-image domains. We highly reccommend reading "Differentiable Image Parameterizations" (link) for an interesting overview of how these techniques are used for neural art.

5. REFERENCES

[1] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert, "Feature visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization.

[2] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich, "Going deeper with convolutions," 2014.

[3] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li

Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[4] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," 2014.

[5] Alexander Mordvintsev, Christopher Olah, and Mike Tyka, "Inceptionism: Going deeper into neural networks," 2015.

[6] Alexander Mordvintsev, Christopher Olah, and Mike Tyka, "Deepdream-a code example for visualizing neural networks," *Google Research*, vol. 2, no. 5, 2015.

[7] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent, "Visualizing higher-layer features of a deep network," *Technical Report, Univeristé de Montréal*, 01 2009.

[8] Aravindh Mahendran and Andrea Vedaldi, "Understanding deep image representations by inverting them," 2014.

[9] Anh Nguyen, Jason Yosinski, and Jeff Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," 2015.

[10] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah, "Differentiable image parameterizations," *Distill*, 2018, https://distill.pub/2018/differentiable-parameterizations.

[11] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune, "Synthesizing the preferred inputs for neurons in neural networks via deep generator networks," 2016.

[12] D. T. Braithwaite and W. B. Kleijn, "Bounded information rate variational autoencoders," 2018.