# QUERYING DEEP NEURAL NETWORKS WITH GRADIENTS

A Thesis Presented

by

Henry Z. Lo

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

Master of Science

June 2016

Computer Science Program

QUERYING DEEP NEURAL NETWORKS WITH GRADIENTS

A Thesis Presented

by

Henry Z. Lo

Approved as to style and content by:

Wei Ding, Associate Professor
Chairperson of Committee

Dan Simovici, Professor
Member

Nurit Haspel, Assistant Professor
Member

Kourosh Zarringhalam, Assistant Professor
Member

Dan Simovici, Program Director
Computer Science Program

Peter Fejer, Chairperson
Computer Science Department

ABSTRACT

QUERYING DEEP NEURAL NETWORKS WITH GRADIENTS

June 2016

Henry Z. Lo,
B.S., University of Massachusetts Boston
M.S., University of Massachusetts Boston

Directed by Associate Professor Wei Ding

Despite their impact on computer vision and face recognition, the inner workings of convolutional neural networks (convnets) are considered uninterpretable. To counter this, we propose prediction gradients to understand how convnets encode concepts. Existing efforts to understand convnets focus on visualizing units and classes in pixel space. In contrast, prediction gradients measure how individual units in convnets contribute to prediction, and is thus more general than pixelization methods. Calculation is also extremely efficient. Experiments verify the ability of the prediction gradients to measure performance impact of units in a convnet. Using a standard face recognition data set, we also show concepts are distributed in convnets. Finally, we show how the prediction gradient can find the most distinguishing features of faces, what part of one image looks like a target class, and how to use that information to fool classifiers. These use cases demonstrate that prediction gradients can provide unique insight into how neural networks operate.

# Table of Contents

# List of Tables

# List of Figures

# CHAPTER 1

# INTRODUCTION

The high-profile successes of neural networks in competitions [KSH12] and real data [LRM12] have helped spark a "deep learning" movement, which has reintroduced older neural net models, [LBB01], spurred interest in the popular media, and have led companies like Google and Facebook to hire deep learning researchers and use their models in products [SLJ15].

However, unlike decision trees or linear models, neural networks are widely regarded as uninterpretable. Given a model which predicts cancer, which of a patient's genetic mutations is most associated with that prediction? What is the most distinguishing feature of a person's face? A neural network which excels at classification clearly contains this information, but how to extract it is not obvious.

In this thesis, we introduce the gradient of a neural net's predictions to solve this problem. This prediction gradient allows us to find which features, regions, and latent variables contribute most to a neural net predicting a label.

Prediction gradients have many nice properties over existing methods of probing into neural nets. These gradients are calculated in the course of backpropagation, and hence require no further computation than training. Mathematically, the gradient is a linear approximation of the prediction surface in neural space, and hence gives information about where the current input lies in the space learned by the neural net.

In this thesis, we focus on convolutional networks, which are primarily used in computer vision. The next chapter introduces this model. Chapter 3 introduces the prediction gradient and its rationale, comparing it against existing work. The final chapter shows some examples of how a prediction gradient can be used to probe how a convnet learns to recognize faces.

# CHAPTER 2

# NEURAL NETWORKS

In this chapter, we introduce the standard multilayer perceptron (MLP) neural network, and then the convolutional neural network (convnet), which we use in later experiments. For the following sections, our notation is the same as the excellent introduction given in [Bis95].

## 2.1 Multilayer Perceptrons

### 2.1.1 Structure

The simplest and most widely used neural network is the multilayer perceptron (MLP). The structure of an MLP (along with notation) is shown in Figure 2.1.

The MLP consists of multiple layers, each of which has many units. The input of a unit $i$ in layer $\ell$ is $z_i^\ell$. This net input is a linear combination of the outputs (activations) of units in the previous layer:

$$z_i^\ell = \sum_j w_{i,j}^\ell a_j^{\ell-1} + b_i^\ell$$

The $w^\ell$ coefficients are *weights*, which along with the constant bias $b_i^\ell$, are the parameters which are learned by the MLP. The unit's output is:

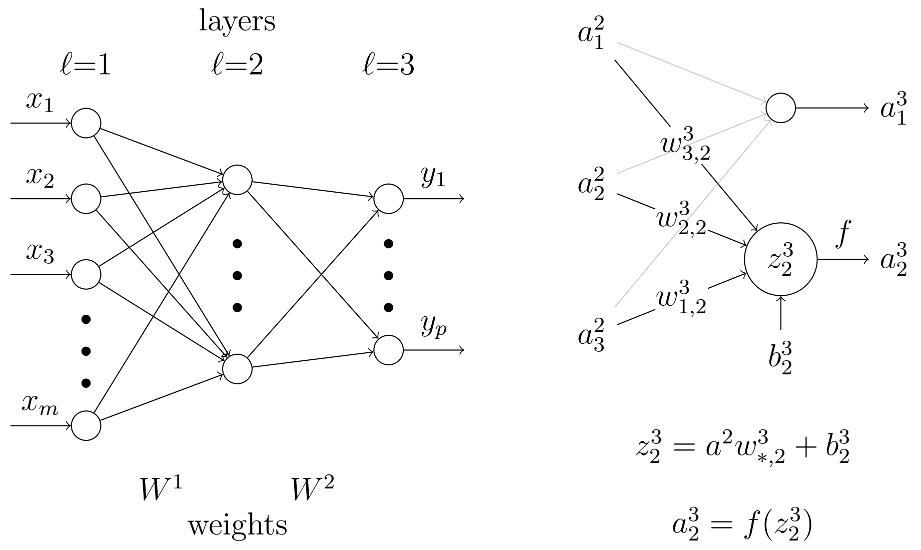


Figure 2.1: Diagram and notation for neural network. Left figure: $x$ denotes inputs to the neural net; $y$, the prediction. Each layer is indexed by $\ell$, and each $W^\ell$ specifies the weight matrix between layer $\ell - 1$ to layer $\ell$. Right figure: each unit's activation is denoted $a_i^\ell$, where $\ell$ is the layer of the unit and $i$ is its index within layer $\ell$. Weights $w_{i,j}^\ell$ connect unit $j$ of layer $\ell - 1$ to unit $i$ in layer $\ell$. $b_i^\ell$ indicates bias of unit $i$ in layer $\ell$. $z_i^\ell$ denotes the net input to node $i$; when $f$ is applied, this results in $a_i^\ell$.

$$a_i^\ell = f(z_i^\ell)$$

In the above equations, $f$ is a vectorized nonlinear function, meaning it is applied elementwise to each coordinate of $W_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell$.

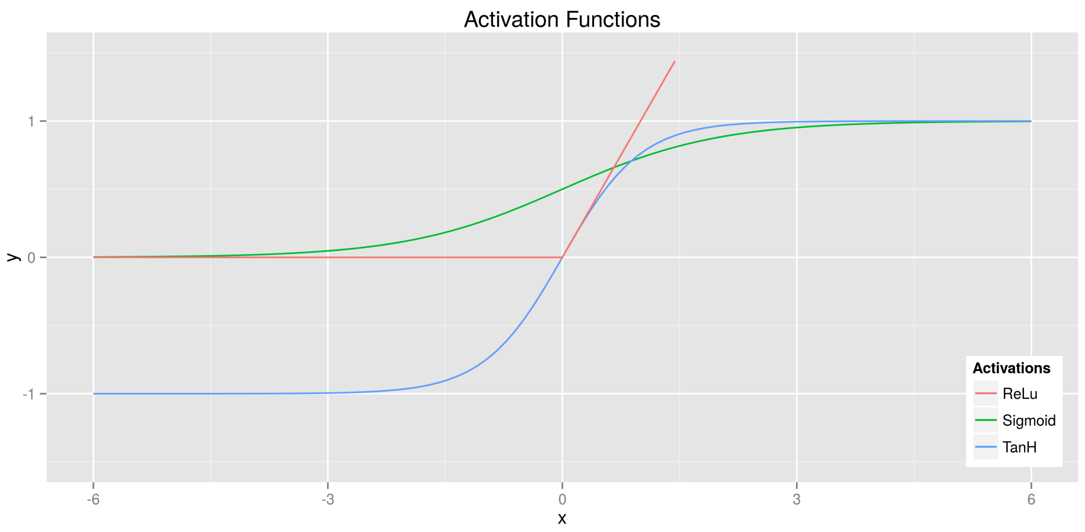


Figure 2.2: Three common activation functions $f$ in neural nets: the sigmoid, tanh, and relu. $y = f(x)$.

### 2.1.2 Activation

$f$ is called the activation function, or nonlinearity. $f(z)$ is usually monotonically increasing in $z$[1], but need not be. The most commonly used activation functions are:

- Perceptron: 0 if $z \leq 0$, 1 otherwise
- Logistic (sigmoid)
- Hyperbolic tangent (tanh)
- Rectified linear unit (relu): $\max(0, z)$

Figure 2.2 shows these activations.

[1]A proof of the universal approximator theorem rests on this assumption. [Cyb89]

### 2.1.3 Output Layer

The last layer varies depending on the task of the neural network.

For supervised learning, the values of the final layer are either taken as regression targets or used to obtain class probabilities $p(y|x)$. The normalization required is done using the softmax function.

In the autoencoder variant of neural nets, the output layer attempts to approximate (and is the same dimensions as) the original input $x$ [BK88].

### 2.1.4 Example

The MLP is a nonlinear classifier, meaning it is able to solve problems such as XOR, which is not linearly separable. However, the parameters of the neural net ($W^\ell$ and $\mathbf{b}^\ell$ for each layer $\ell$) encode a set of affine transformations. In addition, the classifier at the last layer gives a linear decision boundary in the output space of the preceding layer.

To see how this works, consider the XOR problem shown in Figure 2.3. The original points are first linearly transformed, then run through the perceptron nonlinearity. This yields a space which is linearly separable.

The effect of monotonic nonlinearities is a "squashing" action. Hence, each layer of a neural net can be thought of as a rotation / projection with scaling, followed by a space compression (usually $|f(x)| \leq |x|$). Sequential application of this procedure transforms the input space to the target space.

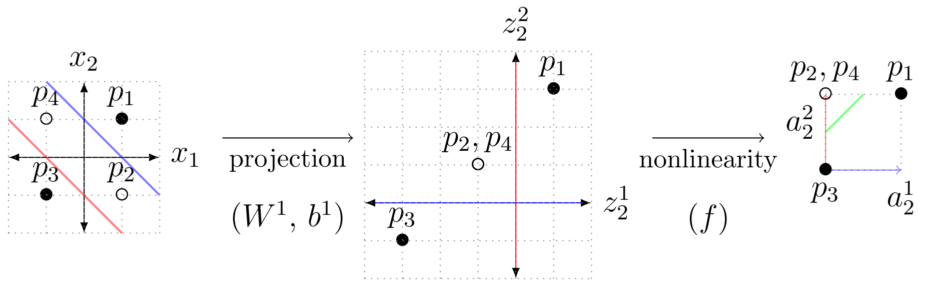


Figure 2.3: The XOR problem and the first layer of an MLP that solves it. The goal is to separate the clear points $(p_2, p_4)$ from the filled points $(p_1, p_3)$; note that this problem is not linearly separable. First, $W^1$ and $b^1$ perform an affine transformation on the input space. Then $f$ "squashes" the original points into a linearly separable space. The final layer can then separate these points linearly to classify perfectly (not shown).

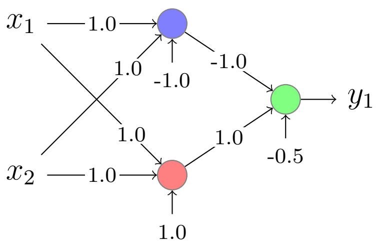


Figure 2.4: An MLP (with perceptron nonlinearities) which solves the XOR problem in figure 2.3. Numbers show weights / biases. Vertical arrows show biases.

### 2.1.5 Learning via Backpropagation

In order to learn the parameters $W$ and $b$, the neural network takes in a set of data points and corresponding labels (as with most supervised learning methods).

Let $x$ be a data point with label $t$. Then the objective function for a three layer MLP is as follows:

$$\begin{gathered}\text{argmin}_{W^1,W^2,b^1,b^2} L(y,t) \\ y = f(W^2 f(W^1 x + b^1) + b^2) \end{gathered} \tag{2.1}$$

Here $L$ is a loss or cost function. For regression, squared error is often used, and for classification, cross entropy is appropriate:

$$L(y,t) = t \log y + (1-t)\log(1-y) \tag{2.2}$$

Learning is done using gradient descent. The update for each weight $w$ (a cell in some $W^\ell$) is as follows:

$$w \leftarrow w + \lambda \frac{\partial L(y,t)}{\partial w} \tag{2.3}$$

where $\lambda$ is the learning rate or step size[2].

The partial derivative $\frac{\partial L(y,t)}{\partial w}$ is straightforward to calculate for weights for the final (third) layer:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial f}\frac{\partial f}{\partial w} \tag{2.4}$$

For weights in all other layers (in this example, the first layer), the backpropagation algorithm uses the chain rule to obtain the partial derivatives:

$$\frac{\partial L}{\partial w^1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial f}\frac{\partial f}{\partial z^3}\frac{\partial z^3}{\partial a^2}\frac{\partial a^2}{\partial f}\frac{\partial f}{\partial w^1} \tag{2.5}$$

[2]The step size may change throughout the course of learning. It is considered best practice to use different step sizes for each weight, and there are many techniques for doing so.

In practice, neural networks are often constructed using automatic differentiation systems, which calculate these partial derivatives automatically. The notion of layers in neural networks corresponds nicely to the concept of computation nodes in AD systems such as Theano. This greatly simplifies the design of neural networks, as long as all components of the network are differentiable.

## 2.2 Convolutional Neural Networks

### 2.2.1 Convolutional Filters

Convolutional neural nets are specially designed for image recognition.

In computer vision, most methods use the pre-processing step of *edge detection*. This simplifies what needs to be learned by the model, and there is strong empirical evidence of the primal role of edges in both human and computer vision.

Edge detectors are often encoded using hand-crafted *convolutional filters*. See figure 2.5 for some examples.

Convolutional filters are parameterized by a square matrix. Consider a $3 \times 3$ filter. The output of the filter for a $3 \times 3$ region of image would be the sum of each pixel in this region multiplied by the corresponding filter weight (see Figure 2.5). For the whole image $X$ (which we can flatten to $x$), the filter would output a new image $X'$ with two pixels less in height and width.

If $X$ is the image as a matrix, and $X_{i',j'}$ is the pixel intensity at coordinates $i'$, $j'$, then the output of a filter $C$ at $X'_{i,j}$ is:

$$X'_{i,j} = \sum_{k=0}^{w} \sum_{\ell=0}^{w} C_{k,\ell} X_{i-w/2+k,j-w/2+\ell} \tag{2.6}$$

where $w$ is the width of the square filter.

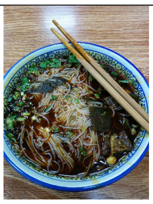
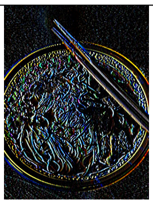
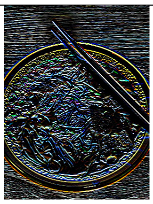
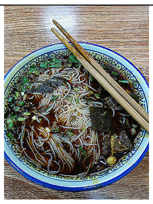
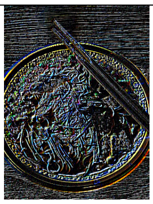
| Original | Sobel X | Sobel Y |
|---|---|---|
|  $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  $\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 1 \end{bmatrix}$ |  $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ |
| **Blur** | **Sharpen** | **Random** |
| $\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$ |  $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  $\begin{bmatrix} -4 & 1 & -1 \\ 2 & -3 & 4 \\ 0 & -2 & 3 \end{bmatrix}$ |

Figure 2.5: Several examples of $3 \times 3$ convolution filters applied to an image. Filters include blurs, but are usually edge detectors. Random filters usually detect edges.

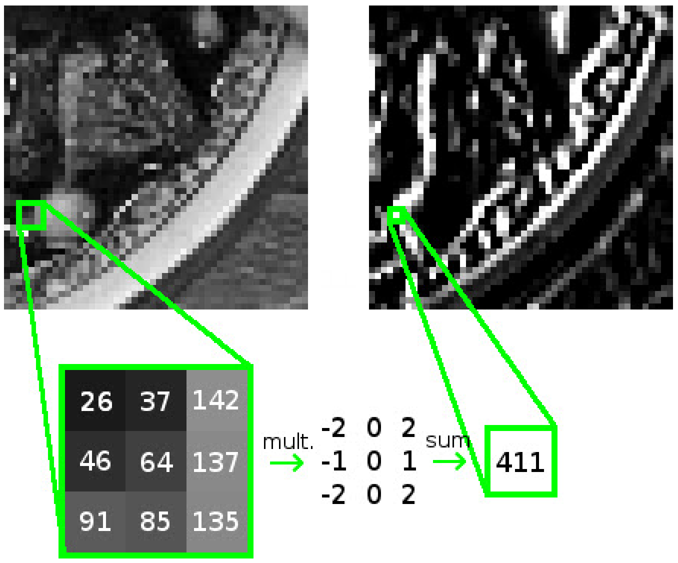


Figure 2.6: Example of convolution. Green region on left is the region of interest. Each pixel intensity in this region is multiplied by the corresponding cell in the filter (bottom middle), then added together, resulting in the pixel value on the right. Filter used is Sobel X, a horizontal edge detector.

### 2.2.2 Convolutional Layers

The key insight of convolutional neural nets (convnets) is that filter parameters can be learned rather than set manually. The convolution is a linear transformation, meaning that its output can be expressed as $Wx$ for some matrix $W$. Hence, the convolution to be specified as a neural network layer.

Consider the action of a single convolutional filter over an entire image, as shown in figure 2.6. The $3 \times 3$ window on the left corresponds to 9 nodes in the input layer, and the right value corresponds to one node in the second layer. There are 9 edges connecting the nodes of these layers. There are as many nodes in the second layer as pixels in the output image. As with the MLP layers, the convolutional output is put through a nonlinearity.

Every set of 9 edges should have the same weights in order to replicate the action of a convolution. This is done by using *tied weights*, which just sets the weights of all these edges to be the same.

Often, multiple filters are learned per layer. These filters share the same input nodes, but have different output nodes and different weights.

This construction specifies the convolutional layer. Each convolutional layer can be thought of as outputting a set of images, each one like the right side of figure 2.6.

### 2.2.3 Architecture

The standard convolutional network first uses convolutional layers, then uses pooling layers. Pooling layers further reduce the size of each image outputted by the convolutional layer, by applying a pooling function over a window of values, and returning one. Each window of values contains a gap, and thus the image is downsampled roughly proportionally to the size of this *stride*. Common pooling functions include max, average, etc. Pooling is not usually parameterized.

The structure of a typical convolutional neural net then usually follows a pattern like:

$$\text{conv} \to \text{pooling} \to \text{conv} \to \text{pooling} \to \text{FC} \to \text{softmax} \tag{2.7}$$

Here, FC specifies the fully connected layers as in the MLP. The synapse between the final pooling layer to the FC layer feeds treats each outputted pixel intensity as a node, as in the naive approach.

As convolutional layers are essentially a special case of FC layers, and as pooling is differentiable, convnets can be learned using backpropagation / gradient descent.

# CHAPTER 3

# PREDICTION GRADIENT

In this chapter, we introduce and justify the prediction gradient, and compare it to existing work.

The prediction gradient was first introduced in our paper [LCD15].

## 3.1 Motivation

Suppose you trained a convnet to accurately identify the presence of tumors in an image. If this model performs very well, it would be interesting to know which of the learned features indicate the presence of the tumor, according the convnet.

One straightforward approach to visualize the output of each filter at each layer. After an image is passed through the convnet, the unit activations for each filter can be separated to visualize the internal representation. This approach is shown in Figure 3.1.

This visualization shows the internal representation for how the convnet processes an image. For example, the type of edge detectors learned by the filters is apparent. However, there is no clear representation of how these activations are related to prediction. For example, one filter may have very high activations, but its output may be flipped, zeroed out, or scaled down in later layers. Thus, high activation does not equal high predictive importance.

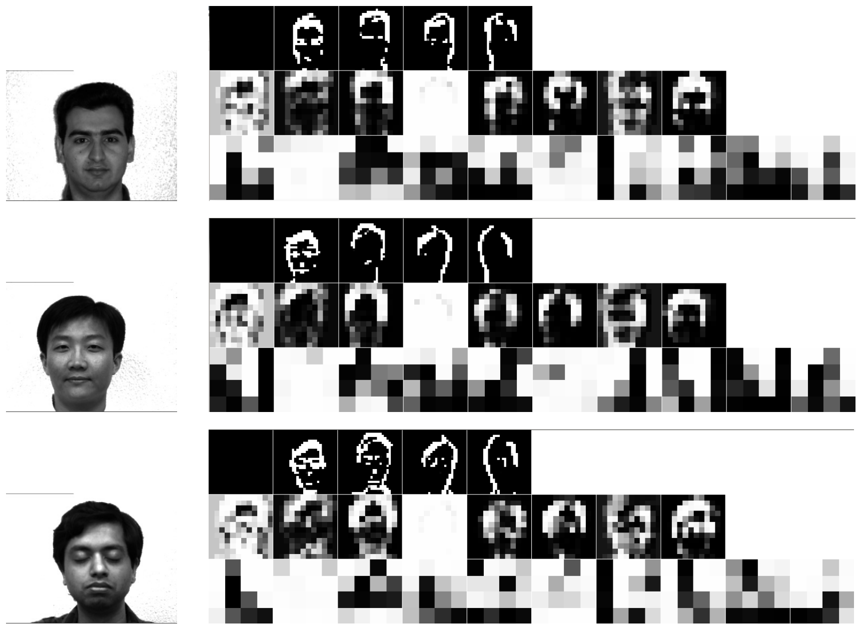


Figure 3.1: Convolutional outputs for three images. For each face in the left column, the images in the right show the output of each convolutional filter. Each row of smaller images corresponds to a layer.

In order to quantify how much a unit contributes to prediction, we introduce the prediction gradient.

## 3.2 Prediction Gradient

The prediction gradient quantifies the predictive importance of a unit $z$ for a class $y$ [LCD15]:

$$PG(a) = \nabla_a y \tag{3.1}$$

$y$ is a nonlinear function which depends on $a$, and $a$ depends on the input image $x$. $PG(a)$ is the linear approximation of how $a$ changes $y$ at the current input $x$.

The PG is already calculated during the course of training in neural nets. In a 3-layer MLP, the PGs for units in layer 2 are obtained during calculation of the weight derivatives $w$ in layer 1:

$$\frac{\partial L}{\partial w^1} = \frac{\partial L}{\partial y} \underbrace{\frac{\partial y}{\partial f} \frac{\partial f}{\partial z^3} \frac{\partial z^3}{\partial a^2}}_{PG(a^2)} \frac{\partial a^2}{\partial f} \frac{\partial f}{\partial w^1} \tag{3.2}$$

In general, the computation of a PG at layer $\ell$ is a part of the computation of the weight gradients leading to layer $\ell$.

### 3.2.1 Why not Loss Gradient?

$L$ is the "error" of the network. Thus $\nabla_a L$ measures unit importance to the error. In addition, it is even easier to compute than the prediction gradient. So why not consider this amount?

Because $L$ is a function of all class predictions $y$, $\nabla_a L$ would simultaneously quantify two things:

- How much $a$ contributes to predicting the correct class (similar to PG).
- How much $a$ *contributes against* predicting the wrong class.

The first quantity is similar to PG in that it quantifies unit importance wrt the true class $t$. However, the prediction gradient is more general, allowing for querying importance wrt other classes (what facial feature of mine makes me look like you?). In order to use the loss gradient to determine a unit's importance to other predictions, one would have to feed in a fake class label in place of $t$.

Classifiers learn to identify both features which contribute to correct classification, and features which contribute against misclassification. The latter set of features are numerous and negative, whereas the former are smaller in number and more "distinctive". Since we want to know what attributes distinguish the class, rather than what attributes count against it, we former focus on just the prediction $y$, rather than the error $L$.

### 3.2.2 Variants

$\nabla_a y$, when applied to units in a fully-connected layer, indicate how much that unit $a$ contributes to prediction $y$.

In convolutional layers, rather than having a single vector of units, we have conceptually a mode 3 tensor of units; the first mode for each image generated by a filter, and modes two and three corresponding to the coordinates in this "image". Each "pixel" in an output image is a unit output.

The contribution of each "pixel" by itself may not be interesting as the contribution of each region, or each filter. This yields two ways of combining convolutional filter PGs.

- By collapsing across the first mode, we can find the aggregate PG for each "pixel" in the filter output. This scores each pixel according to some aggregate of PG. We call this saliency.
- By collapsing across the second and third modes, we can score each filter according to some aggregate of PG. We call this the convolutional gradient.

Note that the saliency here is with respect to the "image" precedent, rather than the true image input. It is a generalization of the saliency proposed in [SVZ14].

Methods for aggregating PG include using the Frobenius norm, max, and L1 norm.

## 3.3 Related Work: Visualization Methods

Since convnets are specifically for image processing, concepts learned by units in convnets can be visualized in pixel space. Existing work has focused on visualizing units such as filters and higher-level activations, using some variant of $\nabla_x a$.

Simonyan et al. used the gradient of a predicted label $y$ wrt pixels $x$ ($\nabla_x y$) to visualize the "spatial support" of a class label for a given image [SVZ14]. This measured, in the space of the given image, which pixels contributed and detracted from a prediction of a given class.

Zeiler and Fergus used deconvnet layers [ZKT10] to visualize the image "concepts" learned at each layer [ZF13]. Simonyan showed that this is conceptually related to visualizing $\nabla_x a$, where $a$ is a unit activation [SVZ14]. Figure 3.4 shows this approach applied to the second-layer units of the Alexnet model for some given textures.

Erhan et al propose an objective of finding the image which maximizes a given

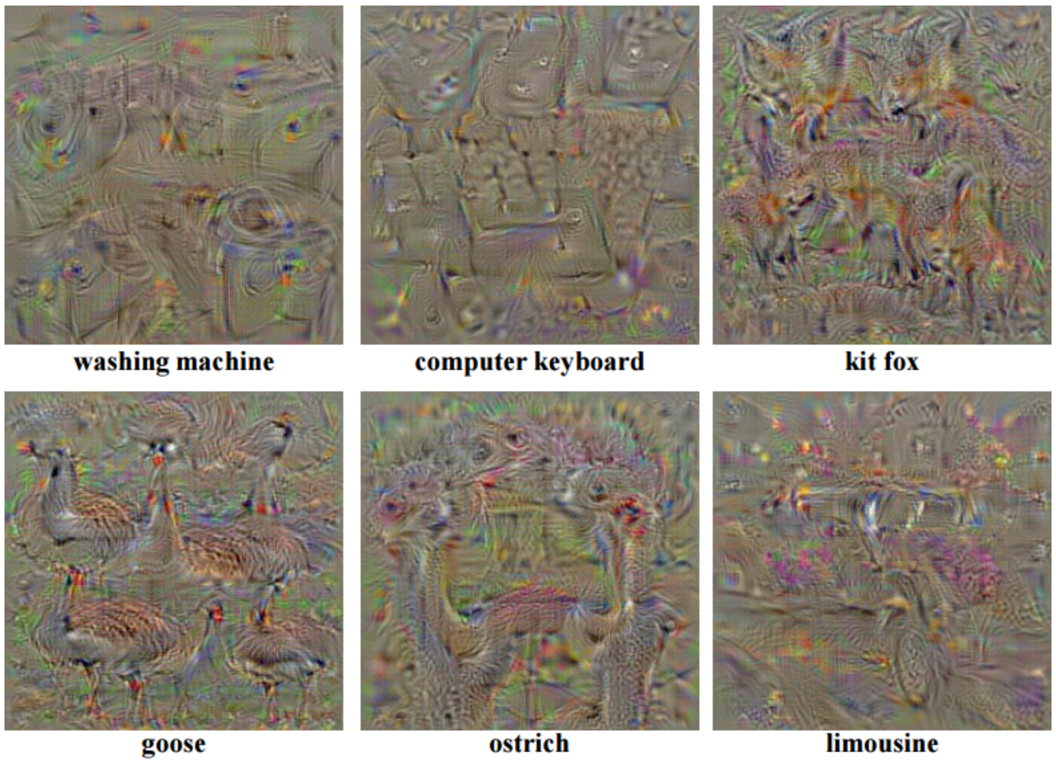


Figure 3.2: $\nabla_x y$ for several images and classes from ImageNet [RDS15]. Figure from [SVZ14].

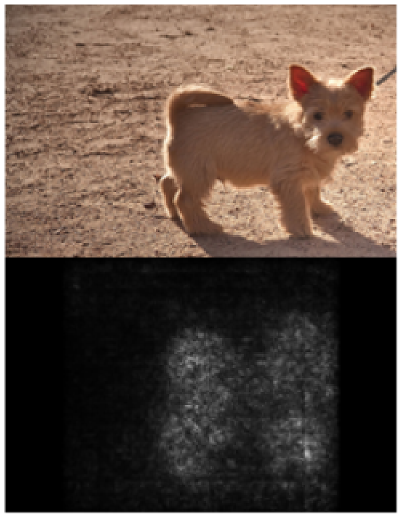


Figure 3.3: Image saliency, calculated using $\nabla_x y$. Figure from [SVZ14].

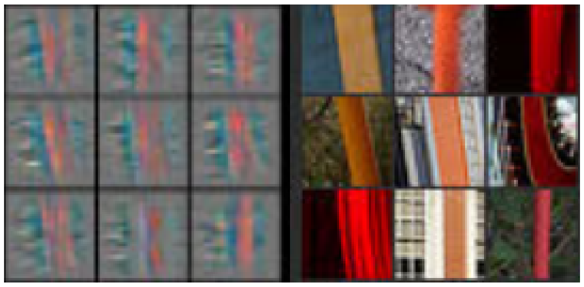


Figure 3.4: Images generated from the deconvnet (left) which demonstrate the shapes which activate most a given filter. On the right are real image patches which also maximize the activation for the same filter. [ZF13].

unit's activation [EBC09]. Unlike Zeiler and Simonyan's approaches, this technique does not require an image as input.

# CHAPTER 4

# EXPERIMENTS

We first want to determine whether or not PG actually quantifies impact on prediction. We focus on convolutional filters, which we aggregate using the Frobenius norm.

Second, we want to know what PG tells us about units in a neural network. How does PG vary between units? How does PG vary between classes and images? What general conclusions do these patterns suggest?

Finally, we demonstrate several uses of PGs, to find important regions and filters for face recognition, and thus extract features of faces most relevant to prediction.

## 4.1 Setup

In order to get meaningful PGs, we first train a convnet. We use a similar architecture to LeNet [LBB01], but with the following parameters:

- 3 convolutional layers, with 5, 8, and 10 feature maps.
- One hidden layer with 20 units.
- Hyperbolic tangent activation functions.
- $5 \times 5$ convolutional filters for all convolutional layers.

- $2 \times 2$ max pooling for all convolutional layers.

Input images are resized to $60 \times 60$ pixels. The learning parameter is set to 0.1.

To facilitate learning on our network, we use the Yale face database[1], which consists of face images of 15 identities in 11 different conditions [BHK97]. 75% of the data set is used for training, and 25% for testing. The network was trained on 15 classes, one for each identity.

Training was done for 250 epochs, after which the network achieved an error rate of 4.88% on the data set. Weights for the network were frozen after this, and no more training was done.

## 4.2 Utility of PG

This set of experiments verifies PG as a useful measure.

### 4.2.1 Does Averaging PG Make Sense?

PG must be calculated with respect to a given image. For it to measure unit relevance wrt a class, we average the PG for all samples with the same class label. This is justified by the following:

- As shown in Figure 4.1, PG patterns for units in a neural net stay consistent across different images of the same class.
- Learning in a neural net averages weight gradients for different samples. For this to work, the average PG must be a meaningful measure of relevance to

[1]http://vision.ucsd.edu/content/yale-face-database

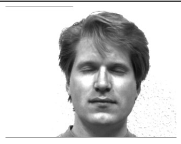
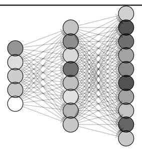
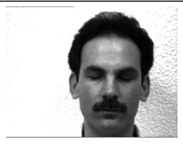
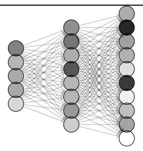
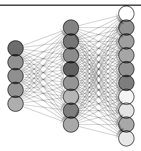
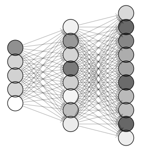
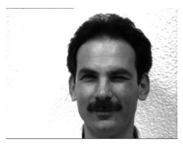
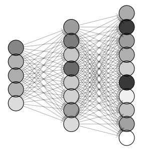
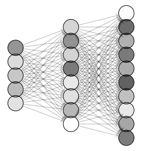
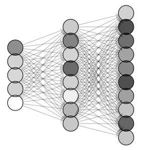
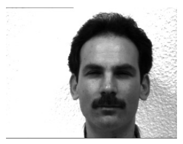
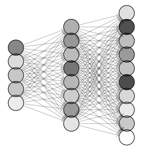
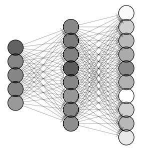
| Identity 1 | | Identity 2 | | Identity 3 | |
|---|---|---|---|---|---|
| Image | Network | Image | Network | Image | Network |
|  |  |  |  | |  |
| |  |  |  | |  |
| |  |  |  | |  |

Table 4.1: Prediction gradients for three images each from three identities. The diagram in the network column the three convolutional layers of the neural network. Each circle is a feature map. Brighter circles indicate a higher prediction gradient for that feature map. Note the patterns in prediction gradients within each identity.

a class (if it were not, the average weight gradient would not lead to lower classification error).

### 4.2.2 Does PG Correlate with Performance?

We hypothesize that PG is correlated to performance for a given class. In order to test this, we:

- Given a class label, calculate average PG.
- Score convolutional filters using the L2 norm of the average PG for that filter.

- This score should indicate “importance” of a filter. To test:
  - Remove highest scoring filters sequentially. Note performance.
  - Remove lowest scoring filters sequentially. Compare with above results.

In Table 4.2, we show the top 3 highest and lowest scores for five identities. These five identities are used in subsequent experiments; for each, we find their top 10 highest-scoring (strongest) filter. First, we set the highest scoring filter's output to 0, then evaluate the network on distinguishing images of that identity. We then proceed to “knock out” the rest of the filters in order of decreasing score and note performance at each step.

We compare this with the same procedure, but knocking out the top 20 lowest-scoring filters instead.

Figure 4.1 shows that the ability of the network is affected much sooner when we remove highest-scoring filters first, compared to lowest-scoring filters. This indicates that the PG-derived score that we used correlates with a filter's “importance”.

### 4.2.3 Other Findings

#### 4.2.3.1 Class vs. Overall Importance

Interestingly, the same filters which are important for class performance are also important for overall performance in the network (Figure 4.2). This indicates that the features that high-scoring filters use to distinguish between classes are useful for multiple classes.

These results support the hypothesis that PG measures the importance of convolutional filters in a convnet.

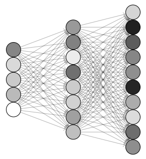
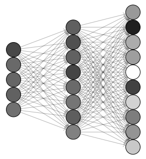
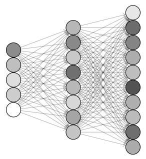
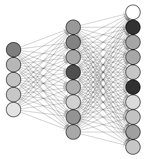
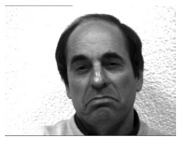
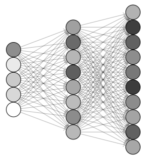
| ID | Image | Network | Highest Scores | | | Lowest Scores | | |
|---|---|---|---|---|---|---|---|---|
| | | | Layer | Unit | PG | Layer | Unit | PG |
| 15 | |  | 1 | 5 | 0.25153 | 3 | 2 | 0.03227 |
| | | | 2 | 3 | 0.23269 | 3 | 6 | 0.03676 |
| | | | 3 | 8 | 0.21675 | 3 | 3 | 0.09039 |
| 14 | |  | 3 | 5 | 0.00202 | 3 | 2 | 0.00025 |
| | | | 3 | 7 | 0.00167 | 3 | 6 | 0.00052 |
| | | | 3 | 10 | 0.00158 | 2 | 4 | 0.00055 |
| 7 | |  | 1 | 5 | 0.02247 | 3 | 6 | 0.00728 |
| | | | 3 | 1 | 0.02029 | 3 | 2 | 0.00951 |
| | | | 1 | 3 | 0.01951 | 3 | 9 | 0.00977 |
| 10 | |  | 3 | 1 | 0.00369 | 3 | 6 | 0.00071 |
| | | | 1 | 5 | 0.00328 | 3 | 2 | 0.00072 |
| | | | 3 | 7 | 0.00316 | 2 | 4 | 0.00114 |
| 12 |  |  | 1 | 5 | 0.08089 | 3 | 6 | 0.01925 |
| | | | 1 | 2 | 0.07403 | 3 | 2 | 0.01964 |
| | | | 1 | 4 | 0.06849 | 2 | 4 | 0.03013 |

Table 4.2: Five identities with their images, class gradients across the whole network, and top 3 strongest and weakest features, as ranked by the class gradient. These five identities were used in the knockout experiments.

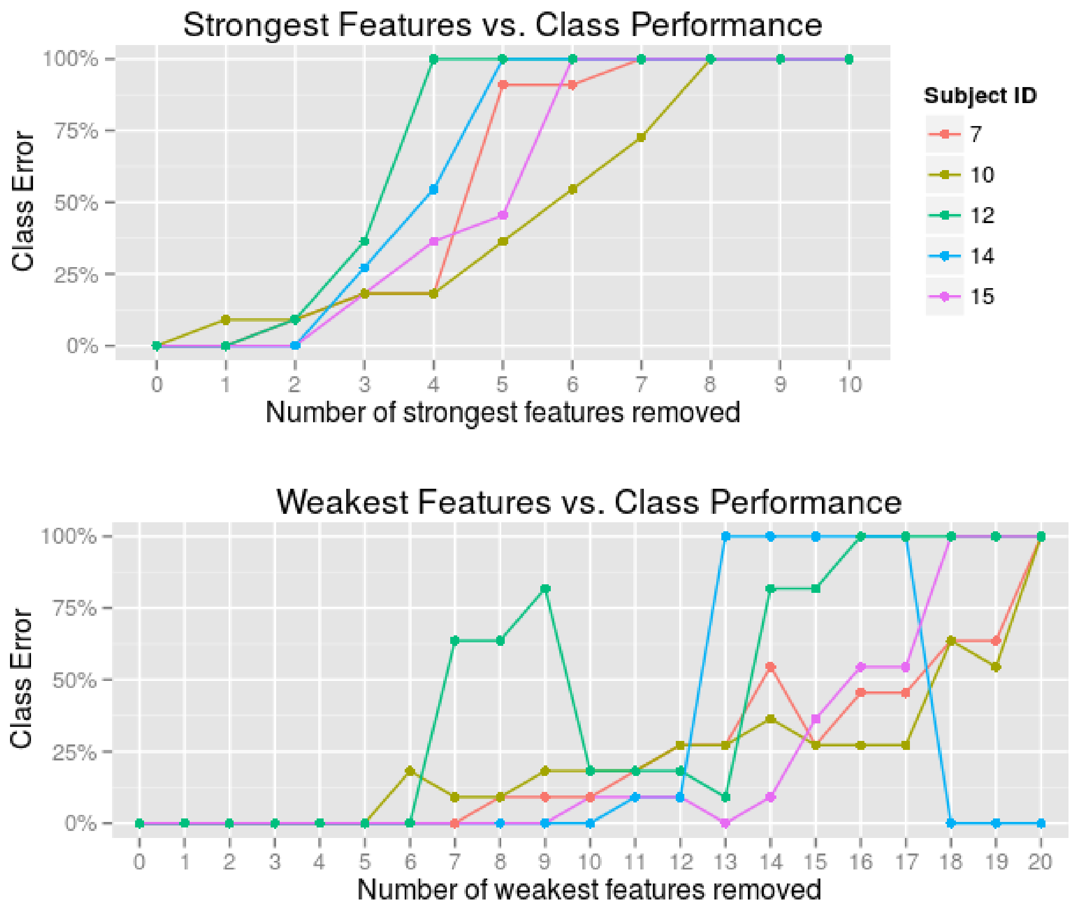


Figure 4.1: Network performance in recognizing a class after removing highest (strongest) and lowest (weakest) scoring filters. Error is calculated as the percentage of images with the given subject ID which are misclassified.

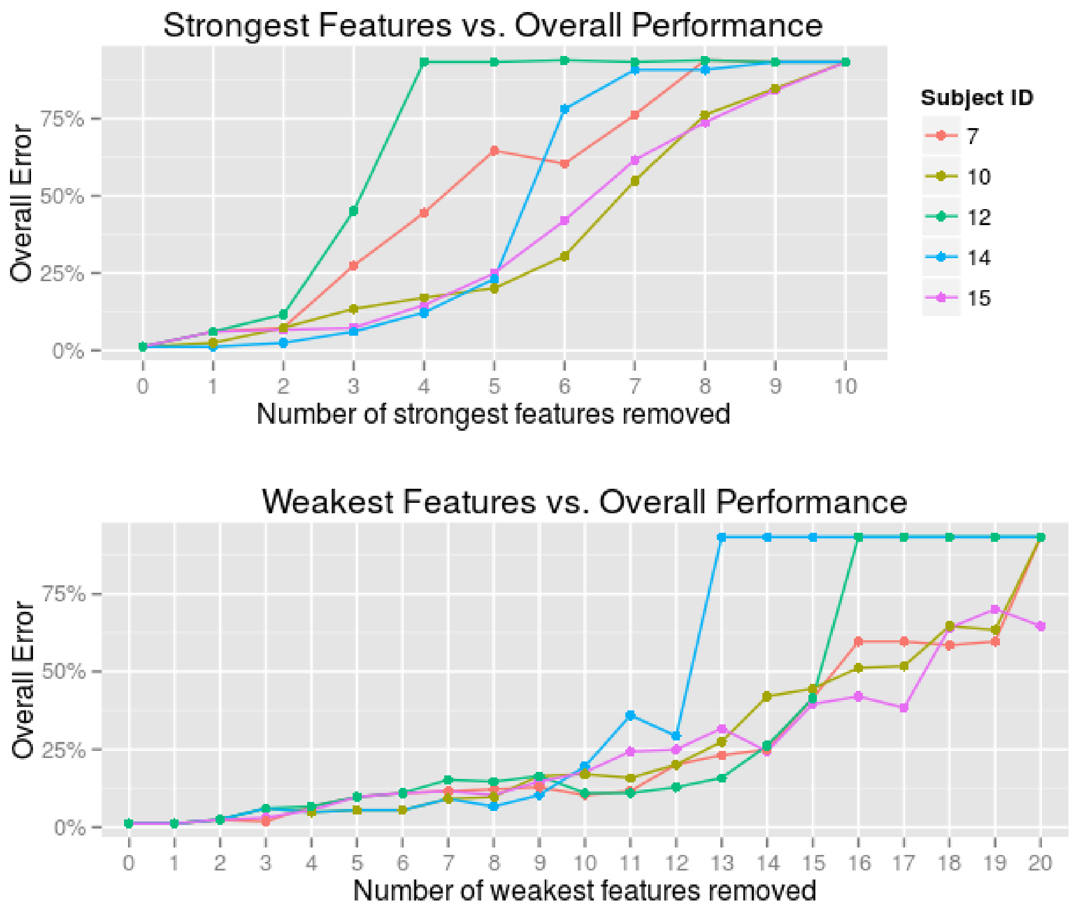


Figure 4.2: Same experiment as Figure 4.1, except error is calculated as percengtage of all images in the test set which are misclassified.

#### 4.2.4 Robustness of Representation

Even for high-scoring filters, removing one or two does not seem to have a large impact on performance. It is only after removing more filters that a significant decrease in performance is noted. This supports the notion that the neural network "distributes" its concept of a class among multiple (though apparently not many) units.

#### 4.2.5 Most Units Matter Little

Most units have a low PG. Overall performance degrades very slowly when removing them. The network almost completely fails after removing the top 10 strong features for any identity. In contrast, after removing 10 weak features, the network still achieves less than 25% error. Even after removing 15 weak features, overall error for many of the weak feature sets is below 50%. This is remarkable, considering that there are only 23 convolutional filters.

## 4.3 Using PG

Here we demonstrate some use cases of the prediction gradient.

#### 4.3.1 What is Your Most Characteristic Feature?

In order to find the most characteristic feature of someone's face, we need the following:

- Filter outputs, which demonstrate the convnet's internal representation of a person's face.

- PG, which will rate the importance of the pixels in this internal representation for the correct prediction.

The highest PG indicates which filter and region contributes most to classification. Combined together, this can be used to visually ascertain the most important feature. The example in Figure 4.3 shows that for one subject in our experiments, this was his widow's peak.

### 4.3.2 What Makes You Seem Most Like Someone Else?

If we take the PG of some other label $y \neq t$, we can see what makes the input seem most like some other class $y$.

In this and the previous example, the highest PG filter was the middle one, which is roughly a diagonal line detector. Filter 1 did not seem to learn anything useful. Filter 2 seems to be a vertical line detector, and filters 4 and 5 horizontal lines.

## 4.4 Adversarial Images

The activations $a^1$ for the first layer correspond to the input $x$. Thus, $\nabla_x y$ can be seen as a variation of the prediction gradient.

$\nabla_x y$, when $y \neq t$, is a *perturbation vector* which makes the image $x$ seem like class $t$. It has been shown that using $\nabla_x L$, it is possible to generate adversarial images with high confidence misclassifications [GSS15, SZS15].

Here, we try to generate adversarial images using $\nabla_x y$, which would allow us to *target* the misclassification of the neural net to a class $y$ of our choosing.

In our experiments, we attempt to perturb every image to be every class $y$. We

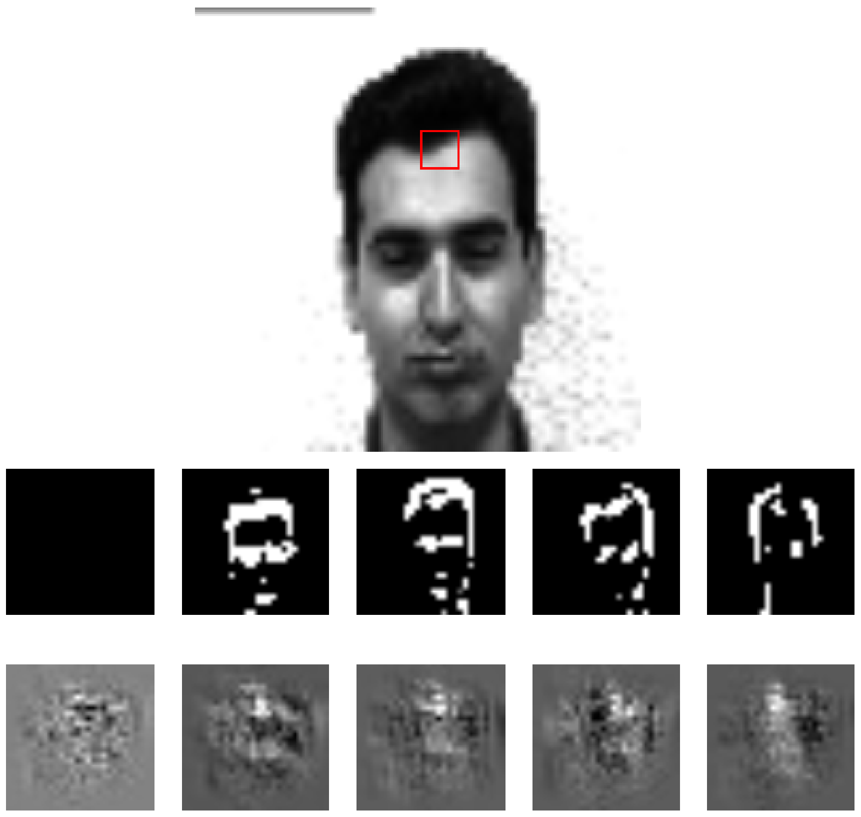


Figure 4.3: Top: a person and his most discriminating feature, his widow's peak. Middle: the first layer convolutional output. Filters roughly detect different types of edges. Bottom: the prediction gradients for each of the first layer units. White indicates greater positive impact on performance. The middle filter has the highest PG at the region of the widow's peak.

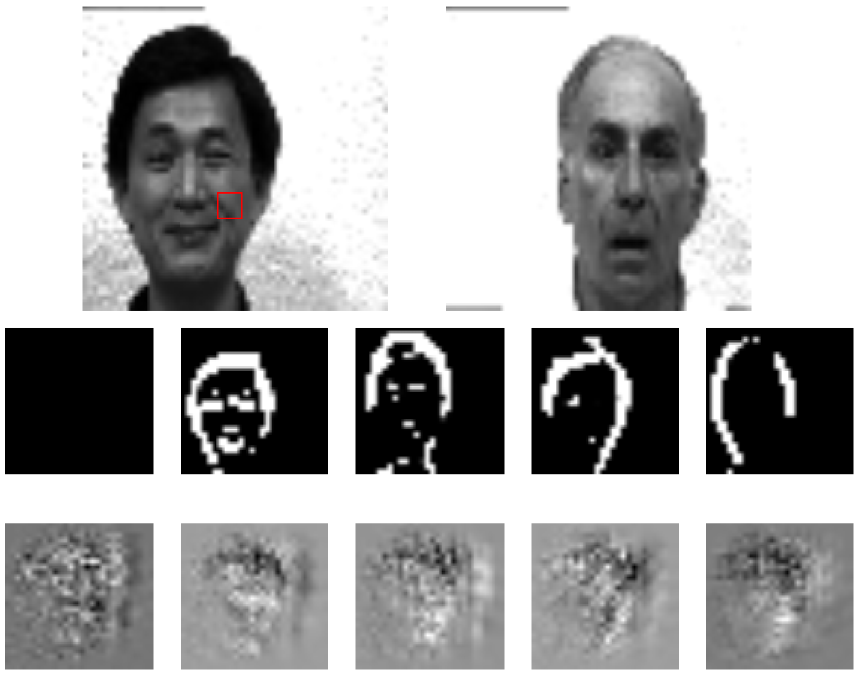


Figure 4.4: Top: the feature of the left face which most makes it seem like the right face. Middle: the first layer convolutional output. Bottom: the corresponding PGs. The middle filter has the highest PG at the region of the right smile line.

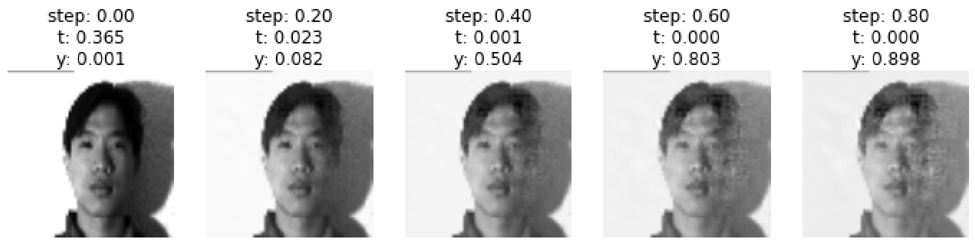


Figure 4.5: Face images and their perturbations for various step sizes. $t$ indicates confidence in original class and $y$ indicates confidence in target.

do this by calculating $\nabla_x y$ for all classes $y$ and all images $x$. Then we vary the step size; the larger the step, the more an image is perturbed (see Figure 4.5).

Results are shown in Figure 4.6. Some targets are easier to fool the classifier into believing than others. The prevailing explanation of adversarial images indicates that more of the extreme regions of image space are classified as these easier targets [NYC15, GSS15].

Figure 4.7 shows the classifier error induced. Previous works on adversarial images only focus on inducing this error, not targeting classes. These results show that the former is easier than the latter.

In general, it is difficult to perturb images to fool classifiers while maintaining some kind of imperceptible difference. It is believed that this is due to the low-dimensional input of the images in our data set (low-resolution, black and white).

## 4.5 Conclusion

In order to study the concepts learned in a convolutional neural network, we introduced the prediction gradient. The PG scores units according to their importance in predicting a class. Experiments verify that this "importance" corresponds di-

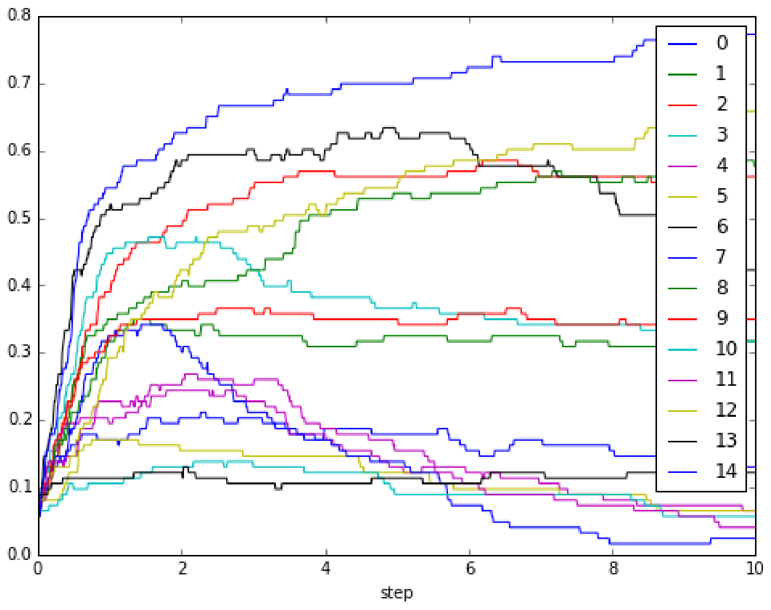


Figure 4.6: Effectiveness of perturbations at convincing classifier. $y$-axis measures success in perturbing image to be of target class. Color indicates target classes.

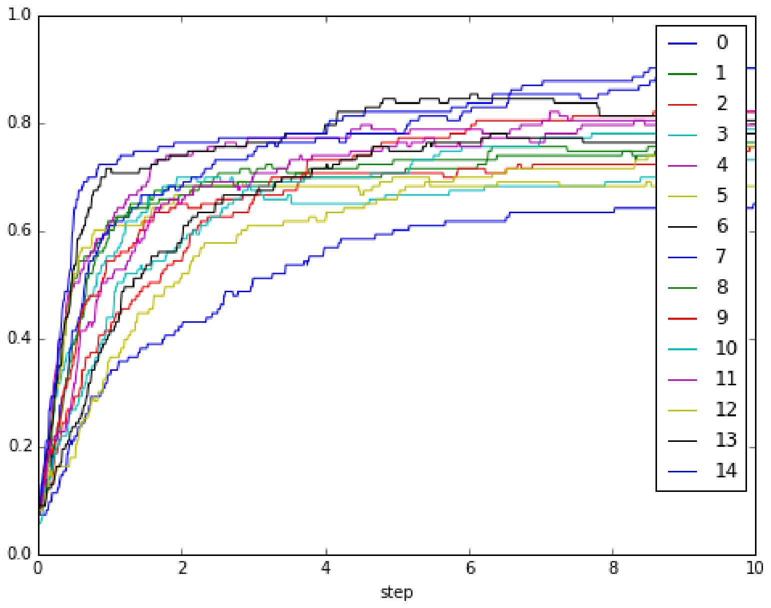


Figure 4.7: Effectiveness of perturbations at inducing classifier error.

rectly to class and overall performance in a convnet. We then show that the PG is useful in several regards: it can be used to find distinguishing features of individuals, find features of individuals which most resemble each other, and find manipulations of images which can best fool the classifier. There are many other use cases for PG, for example in pruning neural networks in order to speed up training, which will be explored in future work.

# References

[BHK97] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. "Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection." *IEEE TPAMI*, **19**(7):711–720, July 1997.

[Bis95] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.

[BK88] H. Bourlard and Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition." *Biological Cybernetics*, **59**(4-5):291–294, 1988.

[Cyb89] George Cybenko. "Approximations by superpositions of sigmoidal functions." 1989.

[EBC09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Visualizing Higher-Layer Features of a Deep Network." Technical Report 1341, University of Montreal, June 2009.

[GSS15] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples." In *ICLR*, 2015.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In *NIPS*. 2012.

[LBB01] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." In *Intelligent Signal Processing*, pp. 306–351. IEEE Press, 2001.

[LCD15] Henry Z. Lo, Joseph Paul Cohen, and Wei Ding. "Prediction gradients for feature extraction and analysis from convolutional neural networks." In *IEEE FG*, pp. 1–6, 2015.

[LRM12] Quoc Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. "Building high-level features using large scale unsupervised learning." In *ICML*, 2012.

[NYC15] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images." In *CVPR*, 2015.

[RDS15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. “ImageNet Large Scale Visual Recognition Challenge.” *IJCV*, **115**(3):211–252, 2015.

[SLJ15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. “Going Deeper with Convolutions.” In *CVPR*, 2015.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps.” In *ICLR Workshops*, 2014.

[SZS15] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. “Intriguing properties of neural networks.” In *ICLR*, 2015.

[ZF13] Matthew D. Zeiler and Rob Fergus. “Visualizing and Understanding Convolutional Networks.” 2013.

[ZKT10] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and Rob Fergus. “Deconvolutional networks.” In *CVPR*, 2010.