Stephanie Angulo, Emily Tran
Machine Learning
Professor Leonard
11 December 2017

Bread or Corgi?

Introduction

The internet has been obsessed with a new trend of collages that contain images of certain animals and inanimate objects that look like each other. What about bread and corgi butts?

Because humans have the seamless ability to classify objects without huge trouble, they can inevitably tell which is which. Humans can recognize features and use that knowledge to form a conclusion. What makes human thought process so special? The field of Artificial Intelligence has been researching on Convolutional Neural Networks (CNNs) because this type of network closely resembles the biological process in human brains. In essence, CNNs are a type of neural network that specializes in recognizing features through supervised learning as a way of generating output. Given that CNNs replicate the way humans classify objects, what are its true strengths and weaknesses compared to true human cognition? The chosen task is to recognize whether a given picture is a corgi or bread.

Over 2,000 images were used to train the CNN from scratch. Experimenting different layer types, combinations, and parameter settings (such as strides and padding) created neural networks that were roughly 80% correct — highest being 87%. Although the error percentage was relatively low, it was unconvincing that the neural network truly understood what features to look for.

Background

Because we were handling images, we decided to use convolutional neural networks (CNN). CNN is a type of deep neural network, which replicates the physiological process of the human brain: neurons passing information to other neurons until a result occurs. CNNs uses that underlying concept for image processing by analyzing subsets of pixels. Each convolutional layer looks at a subset of pixels, figures out if that group of pixels have anything that stands out, puts its conclusion in an activation map for that convolutional layer. Then, the subset will shift (or "stride") over by a defined amount, and repeats the same steps until there are no more new subsets. When the layer finishes one sweep (or "filter") of an image, the convolution layer will have a blueprint (or "activation map") of a feature that seemed prevalent during training (eg. a convolutional layer might recognize that corgis have stubs that resembles legs and learn that bread does not share that feature).

A neural network can have multiple convolutional layers, at which point the neural network becomes a deep learning algorithm. Having multiple layers will disassemble each image even further for deeper analysis as the neurons (or "nodes") feed their outputs to the next neuron. This process of forward feeding is key to CNNs to finding features. Perhaps the first layer will recognize hair textures in each corgi while the second layer learns that the shape of bread is quite rectangular compared to that of corgis. Once all the layers have been established, the last layer then link together in a fully connected layer, which combines all the features that the net recognize. Then, the final layer, called the classification layer, calculates the final output of the neural network, which in our case is the probability that a given image is a corgi or a bread.

Recognizing feature is key to classifying objects, which is why CNNs were chosen as our method. Although CNNs are complex and powerful, they still fall faulty in comparison to human cognition. The reasons are that

1) CNNs, just like neural networks, require heaps of training data. Even Google's neural network has error, and it has millions of training data at its disposal. Meanwhile, humans can most likely learn the difference between a bread and corgi within 0 to 3 encounters.
2) CNNs, just like neural networks, tend to overfit the training data. CNNs look at specifically placements of pixels in images, so often times the CNN will be confused when it has to classify a mirrored image of the original training image. Humans don't have this problem; a dog is still a dog even if it moves 10 pixels to the right. Fortunately, there are hidden layers that attempts to prevent the issue of overfitting, which will be discussed later.
3) CNNs might not learn about features that are intuitive to humans. This can be due to noisy training data, or badly defined parameters. Regardless, the theme is that CNNs read pixels, while human reads the photo as a whole. We can tell that a corgi is a corgi given the face or the paws, but CNNs might find different patterns that is undetectable and nonsense to us. Finding the right combination of layers and parameters to lead the neural net towards learning desired features is very difficult.

4) CNNs, just like any neural network, require a great deal of computational power because of their nature to read images pixel by pixel. Since each pixel equates to a neuron, if each image is 100x100, then the fully connected layer will have 100,000 neurons. The computational time increases more when the neural network is trained with multiple epochs, which are essentially trials or generations of the training process.

Methods

Preparing the data

We began our project by collecting as many training and testing data as possible to obtain accuracy in our neural networks. We downloaded a Firefox browser extension that downloads photos from our Google Search. To filter out all irrelevant data, we manually sifted through the downloads and deleted corrupted files, memes, text-heavy images, and photos that had nothing to do with breads or corgis. We also deleted all black and white photos in order to avoid image size errors (RGB photos uses 3 channels whereas black and white photos only need 1). We then prepared the images for training by resizing them to be 100x100 through MATLAB.

Main Layers

Our neural network contained a combination of several different layers. The first layer was the image input layer, which was the foundation layer that initializes each input image for usage. The input size was 100x100x3, 3 being the RGB channels for the colored images.

Next was the convolutional layer. Essentially, each convolutional layer looks at a certain dimension of pixels, figures out if that group of pixels has anything that stands out, and puts its conclusion on an activation map for that convolutional layer. As a result, each convolutional layer will have a vague blueprint of a feature that seemed prevalent during training (eg. a convolutional layer might recognize that corgis have stubs that resemble legs, or learn that bread has a different overall shape).

Hidden Layers

After the convolutional layer, there were hidden layers used to improve space and time complexity and performance quality. The hidden layers we used were

Batch Normalization Layer, which MATLAB describes as, “The layer first normalizes the activations of each channel by subtracting the mini-batch mean and dividing by the mini-batch standard deviation. Then, the layer shifts the input by an offset β and scales it by a scale factor γ. β and γ are themselves learnable parameters that are updated during network training.” Batch normalization can also improve performance as it provides some regularization.

Rectified Linear Unit (ReLU) Layer, which is a nonlinear activation function that sets all negative elements to zero. This layer was necessary to avoid keeping the all layers solely linear.

Max Pooling Layer, which downsampled or compressed the inputs by keeping the largest value in every subset of an image. This layer was useful for reducing time and space complexity.

Output Layers

Once all the convolutional and hidden layers were initialized, the output layers work together to generate conclusions about the input. The layers we used were

Fully Connected Layer, which connected to all the nodes from the previous layer, essentially connecting all the feature maps back together.

Softmax Layer, which read the output of the previous layer as input and created a probability map for the neural network output. In our case, this layer generated 2 probabilities: likelihood that a given image is a corgi versus a bread.

Dropout Layer, which prevents overfitting by turning off random neurons or nodes.

Classification Layer, which looks at the probability outputs from the softmax layer and assigns the final classification as output. In this case, the size of this layer is two because we only want to classify two objects: bread and corgi.

Other Parameters

Both the convolutional and max pooling layers had these two parameters:

Padding, which controls how far away the layers should stay from the image borders.

Stride, which determines the length in which each subset moves across the images.

The convolutional layer had two more parameters:

Filter size, which is the size of the subsets that sweeps across the image.

Number of Filters, which determines how many times the convolutional layer performs a complete sweep of the image.

Composition of Layers

Here is an example of the layer composition we tested:

Image Input Layer
Convolutional Layer
Batch Normalization Layer
ReLU Layer

Max Pooling Layer

Convolutional Layer

Batch Normalization Layer
ReLU Layer

Max Pooling Layer

Fully Connected Layer
Softmax Layer
Classification Layer

Results

We spent most of our time experimenting with parameters and number of layers. We kept a test log:

Event Log

Steps	Event	Accuracy (%)
1	Initialize layers: Image Input Layer (1) Convolutional Layer (Filter size = 5, Number of Filters = 15) Batch Normalization Layer ReLU Layer Max Pooling Layer (Size = 2, Stride = 2) (2) Convolutional Layer(Filter size = 3, Number of Filters = 15) Batch Normalization Layer ReLU Layer Fully Connected Layer Softmax Layer Classification Layer	71.84
2	Added a Dropout Layer	68.23
3	Increase Number of Filters for first Convolutional Layer 15 → 50	76.39
4	Swapped Filter Size of first and second convolutional layers	70.27
5	Undid previous event. Increased Learning Rate 0.001 → 0.01	44.79
6	Decreased Learning Rate 0.01 → 0.0001	70.75
7	Added another set of Convolutional Layer Increased Epoch 1→ 3	78.12
8	Increased Epoch 3 → 5	81.23
9	Increased Epoch 5 → 10	86.45
10	Increased Epoch 10 → 20	85.53

11	Experimented with Layers and parameters	N/A
12	Copied AlexNet's layer composition and parameters	77.97

We used Matlab's deepDreamImage function to visualize the features of categories at each layer. Here is an example of features of corgis at the fully connected layer:

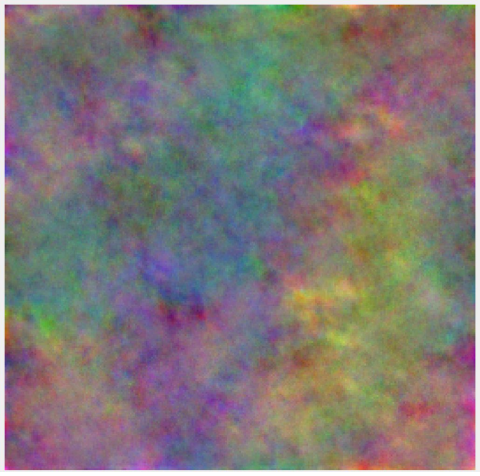

Image 1

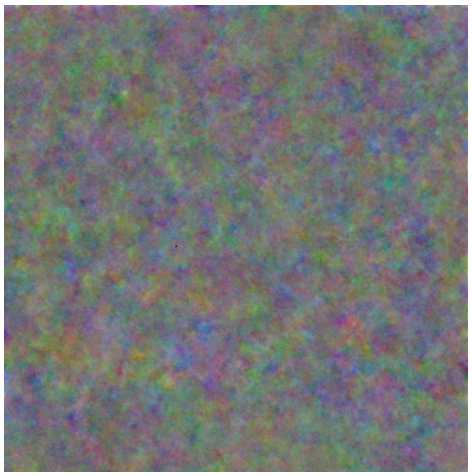

image 2

Discussion

When we first created our neural network from scratch, we did not know what to expect. As we experimented with the layers, we learned several lessons.

The first lesson was that the first convolutional layer should have the largest filter size in order for the neural net to analyze larger features, such as edges or color schemes. Every convolutional layer after the first should be smaller in order for the CNN to target smaller features. As can be seen in step 4 of the Event Log, starting the net with a smaller filter size decreased its accuracy.

The next lesson was that decreasing learning rate increased accuracy (see Step 6 in Event Log). This result makes sense because a higher learning rate equates to higher bias, which might overfit the neural network towards the wrong direction.

We then learned that increasing epochs increased accuracy (see Step 7 to 10 from Event Log) because the neural network was able to develop and evolve what its learning. We figured that there was a limit to this phenomenon before the accuracy drops again, and we confirmed it in Step 10.

The highest accuracy we were able to obtain was 86.45%. Although this was number was high, we were skeptical about the neural network learning the right features. When we printed out the fully connected layer (see Image 1), we saw no concrete pattern of feature. We experimented with more combinations of layers and parameter measure without worrying about accuracy (see

Step 11). We also removed more noisy images that we knew would confuse the neural network. At this point, we simply prioritized teaching the CNN to recognize key features rather than focus on the accuracy.

We decided to copy AlexNet's composition of layers to see if that would improve performance. As can be seen from Step 12, this decision lowered the accuracy and generated incomprehensive feature maps as well (see image 2).

Experimenting with parameters and layer types was a challenge in itself because training requires a great amount of computational power. Even with parallel processing and a high amount of GPU, each training would take at least 10 minutes.

Future Work

Because of limited time yet long computational durations, we were not able to experiment as much as we wanted to. Therefore, we wish to continue to improve our neural net so that we can achieve a more intuitive feature map such as these:

Examples of flower pots and hens; Taken from www.mathworks.com/help/nnet/examples/deep-dream-images-using-alexnet

We expect that to achieve this level of accuracy, we surely need a larger and cleaner dataset of corgis and bread so that the neural network knows what to specifically look for. We want to avoid confusing the neural network in the initial training phase in order for it to recognize coherent features. Perhaps working with black and white images might improve accuracy and allow the CNNs to focus on the characteristics of edges and shapes.

We also plan to experiment with pre trained neural networks (specifically GoogleNet and AlexNet) to increase future accuracy.

We also would like to create a website, similarly to deepdoggo.com, for Occidental students to input their own corgi/bread photos and return the CNN's classification and percentage of classification accuracy.

Conclusion

Overall, we concluded that although neural nets and CNNs are powerful, they can be complicated to program in a way that executes in a fashion similar to human cognition. Humans can easily look at the dataset we have and know which photo has a corgi and which photo has a bread. Indeed, the colors are similar between the two items, but the task of classification goes well beyond the colors. We figured that the CNNs did not know how to handle so much color noise. As impressive as it is for neural networks to be able to read thousands of images within minutes, neural networks are very complicated to execute well. Even a simple task such as classifying a hairy animal and a blob of carb is challenging to our neural network. Granted, we are not professionals. We know that with greater understanding, computational power, resources, and time, neural networks can be very powerful.