IMaterialist Challenge: Fashion classifier
Final Report

Ryan Thangavelu - rthangav@ucsc.edu
Diego Magdaleno - dmmagdal@ucsc.edu
Emmett Greenberg - emjgreen@ucsc.edu
Mattheo Ioannou - mioannou@ucsc.edu

June 9, 2018

Introduction

AI for games is understandable, but why would we need AI for fashion? As the digital marketplace is increasing at an ever growing rate, organization for this online space is incredibly important. The Kaggle competition description also notes that, "Many of today's general-purpose recognition machines simply cannot perceive such subtle differences between photos, yet these differences could be important for shopping decisions." This is why our group chose to participate in the 2018 Immaterialist Challenge (fashion), which tasked us of giving images labels describing the piece of clothing that is pictured. Is the garment for a man or woman? Is it a shirt, a pair of pants, a novelty unicorn onesie? What colors are present? And so on and so on. There were 228 different labels that we were allowed to assign to an image. Each image that our AI model labeled would have some subset of those 228 different labels.

The data set that the competition gave us came in the form of several large JSON files (for a total of 375MB of plain text) which contained more than a million URLs of images on Wish.com that we had to download.

We also changed the input to allow for multiple labels to be associated with each training image.

Other Approaches

From the Kaggle discussion boards, we have observed that other teams are stacking many CNNs on top of eachother. Other teams have mentioned that they have been re-formatting output from each model and feeding it into the next model. Popular models that are being used and stacked include: Desenet201, Inception Net V3, Inception ResNet V2, ResNet50, NasNet, Xception, and XgBoost. The higher scoring teams also seem to be using F1 score optimization.

Huge fashion companies are also using AI for many reasons.They themselves are training models to label clothing. Amazon is using this technology for creating its own AI fashion designer. VF Corporation is redefining the conventional shopping experience by introducing more interaction and suggesting items that it predicts you will like. There are even teams developing

models that will determine whether an image is fashionable or not, and models that learn about particular styles.

Challenges

Image recognition for fashion is a difficult process in and of itself, simply because of the amount of granularity possible in fashion. Fashion is also all about pushing the boundaries of what we think of as clothes, so how would an AI model know what label to assign some pieces of clothing - let alone a human being? Designers are always creating crazy new styles that might not fit the conventional categories like bottoms, shirts, jackets, etc. They might even blend these categories in some interesting way, so the correct labels might not always be clear.

As for more typical pieces of clothing, once you get past the basics (what body part the clothing goes on, and the base color), you also have to to determine the subclass of the object (i.e. dress shirt vs blouse), what parts of an object counts as a single piece of clothing (do detached sleeves still count as part of a shirt?) and the precise color (navy,cobalt,and robin egg are all technically blue). In addition to this, any classifier also has to deal with learning clothing both with and without a model, learn how to separate clothing both from other pieces of clothing and from the background of the image, and how to adjust the actual color based on the perceived color and the lighting conditions.

For this specific project we had a couple major problems. The data set that we were given to use was enormous coming to a total of over a million images at more than 65 gigabytes, even after changing the resolution to 299x299. The data set was also imbalanced with 74% of all images having the labels 66 and 105 with 33% and label 153 with 26%. Additionally, our classifier needed to predict an arbitrary number of labels from a minimum of 1 to a maximum of 23, so we not only needed to figure out what labels to apply to an image, but also how many. Since we were classifying images, our model needed to be relatively large resulting in extremely long training times - at the worst, we had a train time of close to a week.

Model and Infrastructure

For this project, we used the image recognition neural net architecture Inception(V3) based on the Tensorflow API. Inception Net (V3) is a convolutional neural network that uses 105 Convolutional layers, 4 MaxPool layers, 11 AvgPool layers, 15 Concat layers, one Dropout layer, two Fully Connected layers and two Softmax layers, which we changed to output sigmoids.

We changed the output of the neural network (final_tensor) from a Softmax to a sigmoid which instead of outputting a single label outputs a percentage for each label that we can use to determine which labels to apply to each image. This also required us to also change the input to accept multiple labels for each image.

The Inception network is a complex convolutional neural network. Convolutional neural networks are composed of a series of layers each of which performs an operation on the data passing through it.

The first layer of a Convolutional network is Input, which converts an image into a three dimensional array (for our current purpose 299x299x3) containing the RGB value of each pixel of the original image.

The most common layer is the Convolution layer, which looks at a small portion of the image and takes the dot product of the values of each of the pixels in the area and the weights and biases that have been trained into the neuron (in Tensorflow represented as "tensors"). The width and height of the layer is ideally as small as reasonable (larger vectors provide more data at a much larger computational cost, Inception net usex 1x1 layers feeding into higher dimension layers to keep computation costs down) but must have a depth equal to that of the input. Inception Net's convolutional layers look at 5x5x3 arrays of RGB values.

Between Convolutional layers, Pooling layers are used to decrease the size of the array input into it using downsampling. The Pooling layer will take an array of x,y,z and return a layer of $((x - extent/stride + 1), (y - extent/stride + 1)), z$. Pooling layers generally use extent = 2 and stride = 2, since pooling as a concept destroys data, and larger pools tend to be overly destructive. What data is output into the smaller vector depends on

the type of pool. Inception Net uses MaxPools which outputs the maximum of everything in its extent and AvgPools which output the mean of numbers in the input vector. Inception net also uses some Pooling layers with stride = 1, which do not decrease the size of the array, and basically act as convolutional layers that use the operation (max for MaxPool etc.) instead of the weights and biases of the convolutional layers. Pooling layers with a stride of 1 exist only for the chance that a Pooling layer may be more effective than a convolutional layer.

Concatenation layers are used to combine different branches after branching. They work by finding the largest value for height and width, then zero padding all other height and width values, and finally concatenating along the depth. Therefore, if two elements have values (x,y,z) and (a,b,c), a Concatenation layer will result in (Max(x,a),Max(y,b),(c+z)) if x ¿a and b ¿ y the result will be (x,b,(c+z)). The reason for branching is that for each problem there are a variety of methods that may work (one 1x1 Conv layer, A 1x1 Conv layer, then a 3x3 Conv layer, a 3x3 max pool with a stride of 1 then a 1x1 Conv layer, etc.), so the inception net tries them all then concatenates the result so later layers can decide which worked better. The parallel layers between Concatenation layers are referred to as Inception Modules.

The Dropout layer is rather interesting. What it does is selectively ignores certain neurons in each path, in the short term reducing the effectiveness of the model, but in the long term producing redundancies in the network that make the network less sensitive to the weights of individual neurons, and by extension less susceptible to overfitting. Without a dropout layer it is possible for a single neuron to have an excessive impact on the result, but if that neuron is "dropped" then the network can work around the missing neuron and improve the final solution.

Softmax layers provide the output, (though we took a sigmoid rather than a softmax) after the fully connected layers, and provide the data for back propagation. Inception has two sofmax layers each of which are used to calculate the accuracy of the model. Unmodified Softmax layers take the output of the neural network and create a vector of values that add up to 1, which can be used to adjust the weights and biases of the convolutional layers.

There is an output layer in the middle of Inception Net meant to prevent

the vanishing gradient. The vanishing gradient is a machine learning problem where the weights of the layers closest to the output become more important to the results and thus more "learned" than the neurons further from the output. By taking an output in the middle the effect of the weights of the earlier neurons can be determined and thus adjusted to improve the output graph

Results and Error Analysis

Kaggle used the F1 score to measure the accuracy of graphs created by the challengers, which takes into account the precision and the recall of the model. Precision is the number of correct positive results divided by the number of all positive results that should have been returned, and recall is the number of correct positive results divided by the number of positive results that should have been returned. The mean F1 score is calculated such that:

$$F1 = 2(precisionrecall)/(precision + recall)$$

As of now, the top score on kaggle is a F1 score of 71Several graphs were trained with different amounts of the training data. At the lowest, a graph was trained with 10,000 images with a pixel quality of 40 by 40 pixels. We then trained graphs with 10,000 images, 100,000 images, 200,000 images, 300,000 images, 400,000 images, 500,000 images, and 1,000,000 images, all at a image quality of 299 by 299 pixels. The lowest quality graph (10,000 at 40 by 40) yielded an F1 score of 34% while the highest quality graph (1,000,000 at 299 by 299) gave an F1 of 36%. This puts our team in the top half of the competitors in the challenge with a difference of 2 points between our graphs.

There are a few conclusions we can draw from these results. The first conclusion is that the inception net is able to pick up on general multi-labeling patterns with very low amounts of data. An alternative conclusion is that the distribution of the labels is too unbalanced, as we know for a fact that certain labels were oversampled. The ratio of oversampled or high distribution to labels to the number of labels for an image was probably high enough to yield such a mediocre F1 score. We can also play around with the sigmoid threshold we used to determine if a label should be applied to

an image, as we calculated that about 60% of the time, the correct labels existed in the top 10% of sigmoids per image. Either way, these results were not as high as we expected given our complicated model and the size of the training data, but further study could have been done on the model or on a different model to find better ways to improve accuracy if we had enough time.

Individual Contributions

Ryan

I found the model we eventually ended up using and determined how to use it and ran the dataset with downsized images. I also wrote the proposal and progress report. I also wrote the scripts that retrieved the labels from the JSON files and wrote them into the expected filetype for inception net.

Diego

I ran the full training set on my machine and wrote the testing and accuracy script.

Emmet

I ran the smallest training set on my machine because I have very little space on my computer. Researched what other peoples' approaches have been. I also wrote a script to web scrape images from google, in the hopes to eventually rebalance our dataset. I also wrote a lot of the final report.

Mattheo

After joining the team in Week 6, I used the model and trained it 7 times with an increasing number of training images. I created a model trained on 10,000 images, 100,000, 200,000, 300,000, 400,000 and 500,000, which took about a day each to train. I then wrote the script for testing these models with the F1 means score based on the testing script Diego wrote, and tested each model which also took about a day each.

Sources

https://towardsdatascience.com/multi-label-image-classification-with-inception-net-cbb2ee538e30

https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

https://stats.stackexchange.com/questions/184823/how-does-the-depthconcat-operation-in-going-deeper-with-convolutions-work?utm_medium=organic\&utm_source=google_rich_qa\&utm_campaign=google_rich_qa

https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

http://cs231n.github.io/convolutional-networks/#pool

https://arxiv.org/pdf/1409.4842.pdf

http://neuralnetworksanddeeplearning.com/chap5.html

https://www.technologyreview.com/s/608668/amazon-has-developed-an-ai-fashion-designer/

https://www.techemergence.com/ai-in-fashion-applications/