NORGES TEKNISK-NATURVITENSKAPELIGE UNIVERSITET

Semantic Image Retrieval and Clustering

Alfredo Clemente

November 29, 2016

1 PART I

For this section we were asked to deliver an end to end deep learning system that takes as its input a 256x192 RGB image y, and is able to return a set of images X that are semantically similar to the input image. We are given a set of 100000 image, label pairs $\{(x_0, \boldsymbol{l_0}), (x_1, \boldsymbol{l_1}), ..., (x_n, \boldsymbol{l_n})\}$ where the labels are in the form

$$\boldsymbol{l_i} = \begin{pmatrix} 0.6 \\ \vdots \\ \vdots \\ \vdots \\ 0.9 \end{pmatrix} \tag{1}$$

where each row represents a given label, for example *house, village, building*.

More concretely we are asked to maximize the score

$$S(X, y) = \frac{2}{|X| + n} \cdot \sum_{x_i \in X} \frac{\boldsymbol{l}_{x_i} \cdot \boldsymbol{l}_y}{\sum_j l_{y,j}} \tag{2}$$

where $\boldsymbol{l}_y$ is the vector containing the values of the labels for the image y, and $n = 50$.

Additionally a set of constraints were set. We are not able to perform any kind of search on the labels themselves, the system must be able to be tested on 10000 images within a *reasonable* timeframe, and it must run solely on CPU. The above constraints, in addition to the large amount of training data present three large difficulties that must be solved:

1. Efficient training and evaluation

2. Indexing of a large amount of images

3. Unsupervised learning of semantic similarity

I will now present my solution, and how I address the difficulties of the problem.

1.1 Training and Evaluation

The dataset we were presented is a subset of the Open Image Dataset (Krasin et al., 2016). For my solution I use an Inception v3 (Szegedy et al., 2015) neural network pre-trained on the Open Images Dataset. The network is able to tag images with upto 6012 human generated labels.

For my solution I use most of the network as is, but remove the final layer leaving the second to last layer as its output. This layer has 2048 outputs with a ReLu non-linearity. The output of this layer is resized to the range $[0,1]$ and fed to a fully connected layer with 512 output layers and a ReLu non-linearity, the final layer has a sigmoid non-linearity and k output neurons. The choice of k will be explained in the following section.

The labels in the dataset were originally generated by running each image through the Inception v3 network taking the second to last layer, which is of size 2048, this output is then transformed it into a vector of size 6012 and a sigmoid non-linearity is applied. After the non-linearity all classes with probability greater than 0.5 were used as labels. I chose to use the layer of size 2048 as my input as it contains all the information necessary to generate the labels, more compressed, and without the arbitrary limit of 0.5. I believe that these "soft labels" will aid in training, much like transfer learning as presented in Hinton et al. (2015).

1.2 Indexing of Large Amount of Images

The task at hand is one of classification, in which the system learns to choose the appropriate images depending on its input. The system can only choose images from the training set, which contains 100000 images or categories. Traditionally neural networks choose categories with a softmax output layer that gives the probability of choosing each category. The softmax function is given by

$$softmax(o_i) = \frac{e^{o_i}}{\sum_j e^{o_j}} \tag{3}$$

As shown in Equation 3, to calculate the probability of a single category, we must sum over the probabilities of all 100000 categories. This problem is commonly encountered with language models, in which each output category represents a word, and a language model may have a vocabulary of over a million words. Most approaches to resolve this issue are either sample based, or hierarchical.

A model that is commonly used is the hierarchical softmax, which replaces the global normalization present in Equation 3 with a series of local normalizations. For example, in the tree

shown in Figure 1 we have 8 classes $C_0, C_1, ..., C_7$ and 6 pseudo-classes $c_0, c_1, ..., c_5$. Then the probability for each class is calculated by multiplying together the probabilities of its parent pseudo-classes

$$p(C_i) = \prod_{c_j \in parents(C_i)} p(c_j) \tag{4}$$

which means that we can calculate the probability for a given class C_i in $log(n)$ time for n classes. To create a neural network that implements the hierarchical softmax we would need a network with $n-1$ output neurons, with an activation function with an output in the range $[0,1]$. To construct a hierarchical softmax, the classes must be structured in a meaningful hierarchy, for example for language models it is common to use WordNet (Fellbaum, 1998) which is a lexical database of English words.

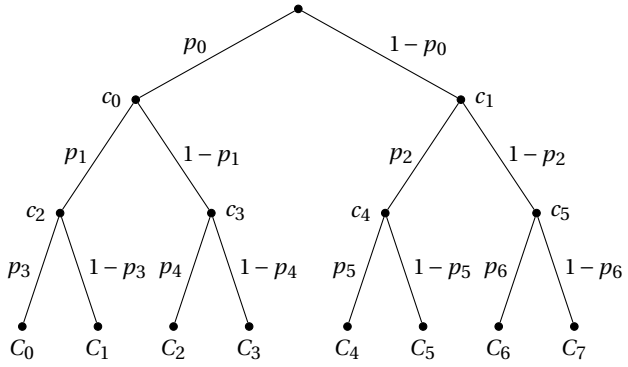

Figure 1: Standard implementation of hierachical softmax

I have chosen to represent my hierarchical softmax as a binary tree, in which each node of the tree attempts to divide the remaining images into exactly two sets. In order to do this I run a set of k_{imgs} images through a pretrained Inception v3 model and extract the output from the second to last layer. The output of this layer represents an embedding for the images that is meaningful for annotating them with labels. I then perform Principal Component Analysis (PCA) (Jolliffe, 2002) with k components on the generated embedding. PCA learns a mapping $\mathbb{R}^{2048} \rightarrow \mathbb{R}^k$ where the resulting vector represents the most variance observed in the provided data, with the limit of k output dimensions. Each of the k dimensions is constrained to be linearly independent of the others (diagonal covariance matrix) and have a mean of 0. The dimensions are sorted by how much variance they explain. Figure 1.2 shows the distribution of the first three dimensions of the training set transformed using PCA.

We can then construct the hierarchical softmax by interpreting each of the k dimensions as a pseudo-class, where each class (image) is a member of the pseudo-class c_i if its PCA transformed representation has a value greater than 0 on the ith dimension. As seen in Figure 1.2 each dimension of the data after PCA, has a median close to 0, effectively dividing the data into two at each dimension. We choose the the dimension that explains the most variance as

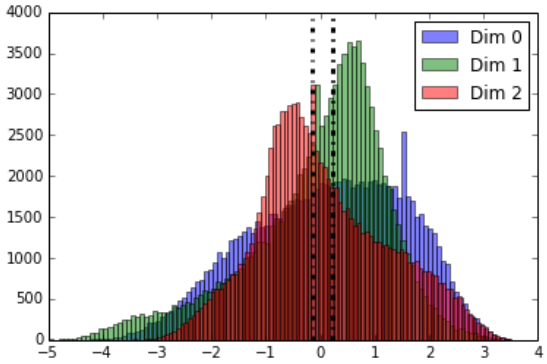

Figure 2: Distribution of first dimensions of the training data after PCA, dotted lines are the medians

the root node of the tree, and then choose dimensions for the following levels in descending order. Figure 3 shows the hierarchy generated by this representation.

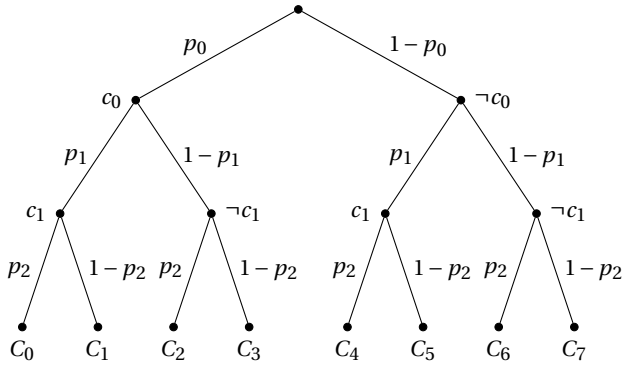

Figure 3: Modified hierachical softmax for my solution

PCA provides additional benefits over human generated hierarchies given that all its dimensions are independent of each other. We can represent the probability of an input image belonging to each of the n classes with only $n >> k > ln(n)$ probabilities, one for each level of the hierarchy. This means that we can have a neural network that can represent the probability of an image belonging to all classes with only k sigmoid output layers.

Each image is represented by its location on the binary tree. This location is represented as a binary vector $b = [b_0, b_1, ..., b_k]$ for $b_i \in \{0, 1\}$ where $b_i = 1$ represents choosing the right branch at level i.

1.3 Unsupervised Learning of Similarity

For this solution I have decided to not use any similarity measure to train the network on how to choose semantically relevant images. Label information is not used at all during training, only features extracted using the pretrained Inception v3 model are used.

The model is trained with input-target pairs, where its input are 299×299 RGB images, and the targets are the image's location in the binary tree. The training objective is then to maximize the probability choosing the path in the tree leading to the chosen image, given the image. More concretely, we minimize the loss function

$$L(x) = -\sum_{j=1}^{k} y_j log(o_j) + (1 - y_j) log(1 - o_j) \tag{5}$$

where x is an input image, y_j is the probability of taking the right branch at level j and o_j is the network's estimated probability of taking the right branch at level j. As I will show, the network is able to use these targets to effectively learn the location of every image it is given, in the tree.

1.4 Results

My solution achieves an average score of 0.383 on the validation set. Figure 1.4 shows the top 3 retrieved images for three queries, one with the best score on the validation set, one with the median score, and one with the lowest score.

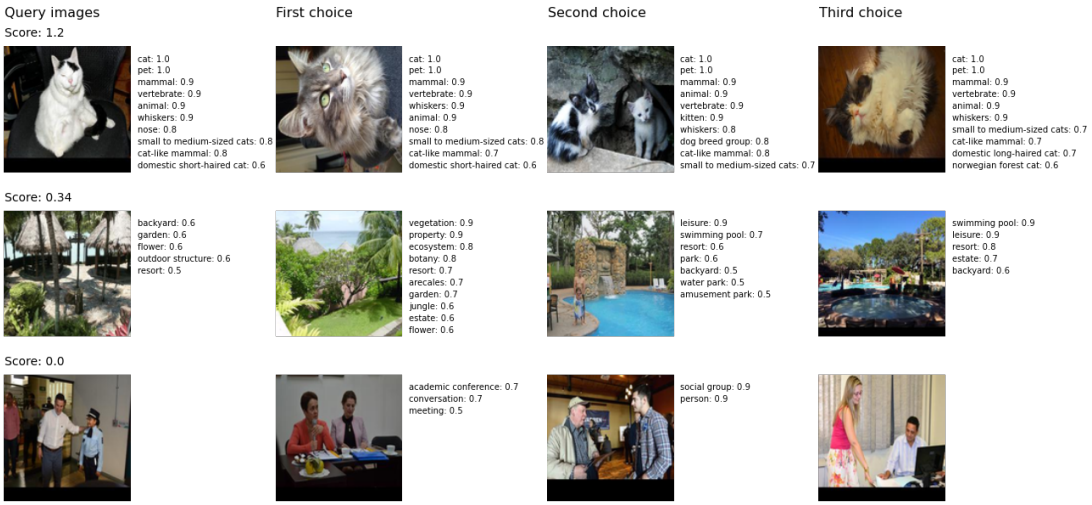

Figure 4: Shown query images on the left column, with the top 3 returned images to the right for 3 queries.

The network is able to to correctly return images that are semantically similar to the query given, even if they themselves do not have labels. The final query shown in Figure 1.4 results in a score of 0.0, however one may observe that the returned images are indeed semantically similar.
The results from the middle query shows that the network is able to correctly identify the content of an image, in this case a beach resort, and correctly return semantically similar alternatives regardless of the labels themselves. A human could probably imagine that the the scenes depicted in the query image and the first choice image could very possibly contain a swimming pool, even though it is not shown in the images. Figure 1.4 shows the distribution of scores on the validation set.

Note that for $|X| > n$ the score may give values greater than 1.0, even with per-image contributions under 1.0.

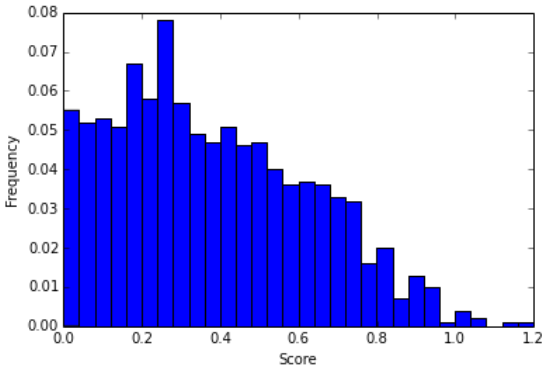

Figure 5: Normalized distribution of scores on the validation set.

The binary tree representation of the images shows to be quite robust. Running PCA on only 50 (out of 100000) images results in a representation good enough so that the network is able to achieve an average score on the validation set of 0.37. Additionally, after analyzing the generated tree, it shows that only 32 out of the 100000 images shared a leaf node with another image (excluding the place-holder images), meaning that the great majority of images could be indexed uniquely by their place in the tree.

1.5 Configuration and Training

Several hyperparameters must be tuned for this model. We must choose the amount of components for PCA k, and how many images n_{imgs} from the training set to use for PCA, additionally we must choose which optimizer to use and which learning rate α. I did a formal search for $n_{imgs} \in \{30, 50, 100, 500, 1000, 10000, 25000, 50000, 10000\}$, and for $k \in \{17, 20, 30, 40, 50, 100, 200\}$. The results show that the value of n_{imgs} does not have a statistically significant effect on the score achieved for $n_{imgs} \geq 50$. The value of k however does have an effect on the score achieved, as shown in Figure 1.5 and the best value I found was $k = 40$. I chose the RMSProp optimizer as I have observed good results with it across several domains and mod-

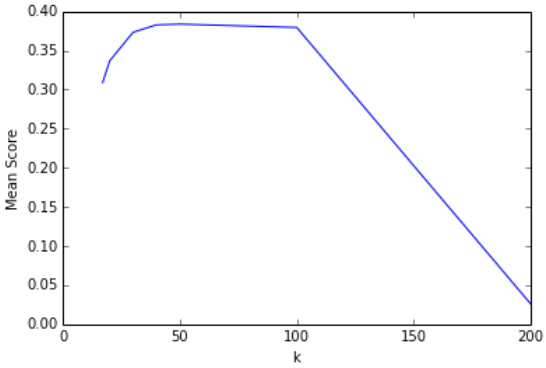

Figure 6: Choice of k plotted against mean score on the validation set

els in the past, the learning rate was found by performing an informal search in the set $\alpha \in \{1e^{-1}, 1e^{-2}, 1e^{-3}, 1e^{-4}, 1e^{-5}, 1e^{-6}\}$ where $\alpha = 1e^{-3}$ gave the best results. Finally we must decide how many images n_{ret} to return per query, I found that 200 gave the best results, after an informal search.

All my tests were run on an Intel i7-4970K CPU with 4 cores and an Nvidia GTX 980ti GPU. The bottleneck values, and other data necessary for training for all 110000 test and validation images can be obtained in about 20 minutes on the GPU. Then the hierarchy tree can be created in 15 seconds on the CPU. Training the lower part of the network, leaving the Inception v3 part untouched, takes 1 minute to train on the CPU.
With the transformed training data available it takes only 5 minutes in total to train the network, run 1000 queries and calculate their score, all on the CPU.

1.6 Computational complexity

The algorithm is very efficient, with $O(k)$ complexity at training time, and $O(n)$ complexity at inference time. At training time we must run PCA once to generate the hierarchy. The time complexity of PCA is $O(max(n_{imgs}, 2048)^2 \cdot k)$ (Halko et al., 2010) where n_{imgs} is independent on the size of the training set n resulting in $O(k)$. Additionally we use $O(k)$ to run a single example through the network giving a total of $O(k) + n_{train} \cdot O(k)$ for n_{train} training examples seen (note $n_{train} \neq n$). At inference time for each query image we must calculate the k probabilities of turning right on each node of the tree in $O(k)$ time. The time complexity of calculating the probability for all classes is $O(n)$ and retrieving the top n_{ret} classes has a complexity of $O(n_{ret} \cdot log(n_{ret}))$. Finally the time complexity at inference time is $O(n)$ per query. Additionally, the network must only learn $O(k)$ weights, as opposed to $O(n)$ for the hierarchical softmax.

The asymptotic runtime of this design is the same as a hierarchical softmax, however the constants involved are quite different. Assuming a single layer with 2048 outputs from Inception v3, fed directly to a hierarchical softmax it would need to learn 2048 $\times$ 99999 weights one for

each hidden node to output node combination. Additionally when calculating the probabilities of all classes at inference time it must multiply a 1×2048 matrix with a 2048×99999 matrix, performing $2048 \times 99999 \doteq 2 \cdot 10e^8$ operations. My implementation must only learn $2048 \times k$ weights and then at inference multiply matrices of size 1×2048 and $2048 \times k$ once and $1 \times k \cdot k \times 100000$ twice resulting in $2048 \times k + 2 \times k \times 100000$ computation. For $k = 40$ we have a reduction in the amount of computation of 96% and a reduction in the amount of weights to be learned of 99,96%.

REFERENCES

Fellbaum, C. (1998). *WordNet*. Wiley Online Library.

Halko, N., Martinsson, P.-G., Shkolnisky, Y., and Tygert, M. (2010). An algorithm for the principal component analysis of large data sets. *ArXiv e-prints*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *ArXiv e-prints*.

Jolliffe, I. (2002). *Principal component analysis*. Wiley Online Library.

Krasin, I., Duerig, T., Alldrin, N., Veit, A., Abu-El-Haija, S., Belongie, S., Cai, D., Feng, Z., Ferrari, V., Gomes, V., Gupta, A., Narayanan, D., Sun, C., Chechik, G., and Murphy, K. (2016). Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. *ArXiv e-prints*.