October 17, 2016 - François Maillet

DeepTeach: the interactive deep image classifier builder

A picture is worth a thousand words. Everyone has heard that phrase, and as Wikipedia aptly puts it, it “refers to the notion that a complex idea can be conveyed with just a single still image”.

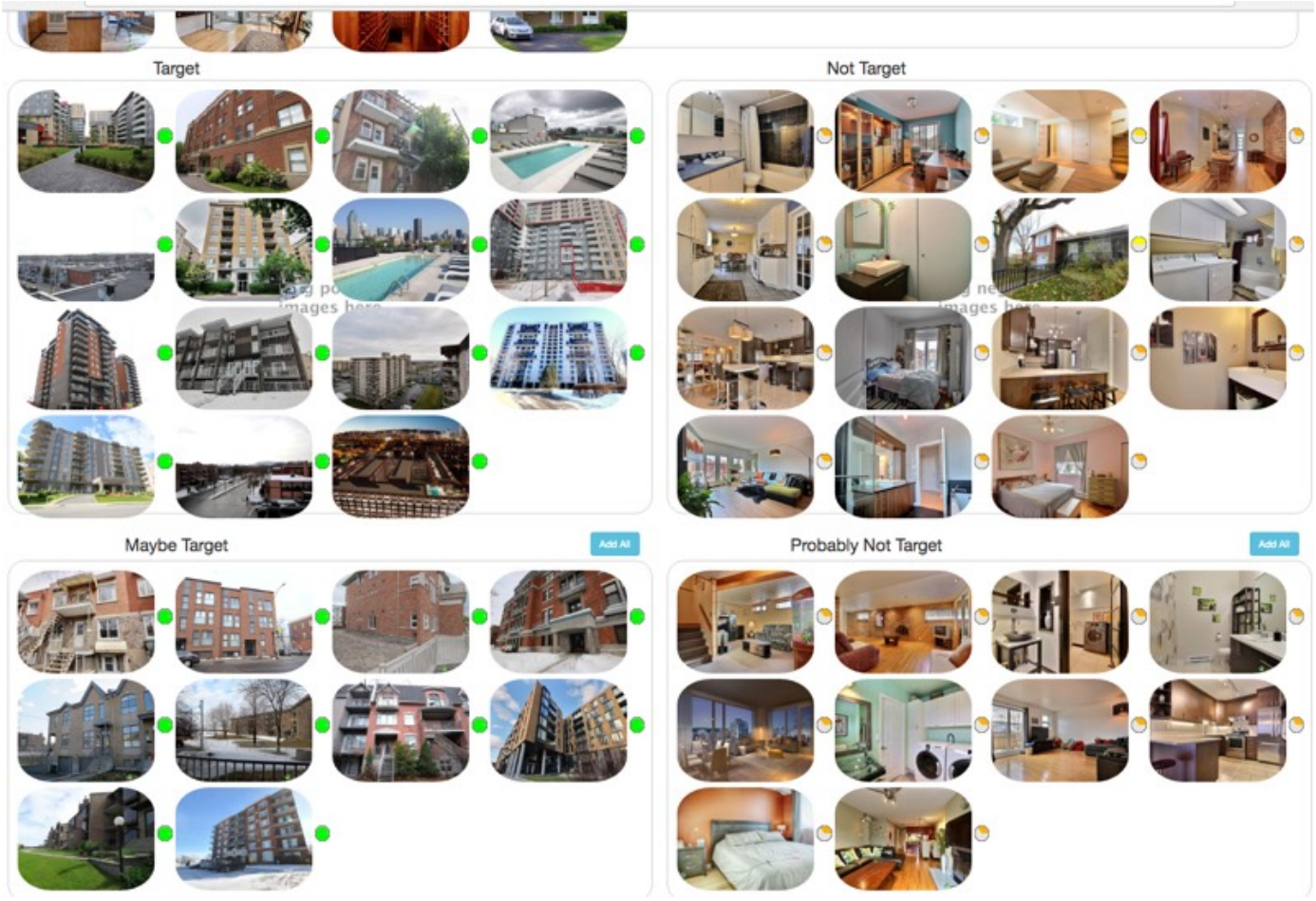
Think of real estate pictures. Everyone has their own taste and thinks some properties are nicer than others. But although we each know what we like in terms of style, colors and materials, it’s sometimes hard to clearly communicate that to someone else. Taking it a bit further, it would be even harder to quickly encode what we like in an algorithm, so it could predict

whether or not we would like a property based on its pictures. Imagine, if we had such a system, we could automatically go through real estate listings, filter them by our hard criterias like price and location, and then order them by the likelihood that we will find them pretty.

This is the type of problem that DeepTeach tackles, and as you'll see, it's is a great example of how machine learning can be used for human augmentation, which means making humans more efficient by assisting them with artificial intelligence.

The plugin

As the Youtube demo video above shows, DeepTeach is a web application developed on MLDB that allows a user to teach a machine learning model what types of images he or she is looking for through an iterative process. The user starts with a random sample of unlabelled images, labels a few by dragging them in the positive label zone (called Target in the UI) or the negative label zone (Not Target in the UI), and then trains a model that learns what each group looks like and suggests new images for each label. The user can then approve or correct the model's predictions, and redo the procedure. Using this workflow, we're able to train a binary image classifier from unlabelled data in about a minute.



The same idea naturally extends to multi-class classification. Instead of having only two groups of images like in DeepTeach, there could be many more. For example, imagine that we have an image dataset containing 5000 unlabelled images of 5 different breeds of dogs. The user could start to label images by dragging them in one of five different boxes, one for each dog breed. Since it's an iterative process, the model would quickly start to figure out what each breed looks like, and would suggest unlabelled images for each of them, just like in DeepTeach. The user can then approve or correct the suggestions, and would then get better and better suggestions.

We say it's human augmentation because the machine learning works in tandem with the user and makes him more productive. The user only needs to manually label a few examples before the model understands the difference between each class. For the dog breed example above, maybe only about 50 or 100 images would need to be manually looked at in total before the model learns how to label the remaining 4900. But since we initially started with unlabelled data, the model wouldn't have been able to function without the initial user input, except to do some unsupervised learning which might not have given the results the user wanted.

The human augmentation workflow that DeepTeach implements has countless useful applications.

The machine learning

The machine learning concepts behind DeepTeach are straightforward but powerful. We use a pretrained deep convolutional neural network (ConvNet) using MLDB's TensorFlow integration (https://docs.mldb.ai/ipy/notebooks/_tutorials/_latest/Tensorflow%20Image%20Recognition%20Tutorial.html) to embed pictures in a conceptual space, giving us a very powerful representation of the images. The ConvNet we use is the Inception-v3 model (https://arxiv.org/abs/1512.00567) that was trained on ImageNet. We then train a random forest using the output of the ConvNet as features and the user's input as labels. Training a random forest is lightning fast on MLDB (https://github.com/szilard/benchm-ml/tree/master/z-other-tools), which allows DeepTeach to be used interactively. Using what one model learned and using it in a second model is called transfer learning, while the overall strategy of having a model query the user to clarify labels iteratively is called active learning (https://en.wikipedia.org/wiki/Active_learning_(machine_learning)).

For more explanations on the techniques used here, I encourage you to check out our transfer learning demo notebook (https://docs.mldb.ai/ipy/notebooks/_demos/_latest/Transfer%20Learning%20with%20Tensorflow.html), or the KDNuggets guest post (http://www.kdnuggets.com/2016/10/mldb-machine-learning-database.html) I wrote on this subject and that shows how to do a lot of this using MLDB. The rest of this post assumes you are roughly familiar with those concepts and will focus on what is specific to DeepTeach.

The appeal of this plugin is that it allows you to build an image classifier quickly, meaning you don't have to invest a lot of time, having to look at a lot of examples. The flip side of this is that it's very easy to create a model that will overfit and perform poorly when using it in the wild on new examples. Below I explain what I did to help make sure the model generalizes to unseen data.

Limit the number of allowed features

The first thing I did was control how many input dimensions the classifier was allowed to use. Remember that we're using a ConvNet, Inception-v3 to be precise, as our feature generator. This means that when we run an image through it and use the second to last layer, it gets transformed into a 2048 dimensional vector. To help with generalization, I played a bit with the number of random features (the `random_feature_propn` configuration key (https://docs.mldb.ai/doc/#builtin/ClassifierConf.md.html#dt)) that the random forest was allowed to use at each step. The default value used in DeepTeach is 10%, which is a lot lower than what I usually use. My intuition behind why this makes sense is the ConvNet features are extremely good, and so with only a handful of examples, the random forest was able to get an almost perfect accuracy with only a few decisions. By removing most of the features, it forces

the model to use more indirect features that have a better chance of working with images that are a bit different to those in the positive examples training set. We're essentially making the training task harder for the model.

Augment the user's labeled examples

If you think about the workflow a user goes through when using the plugin, there are at first no examples specified for either the positive or negative labels. We can expect the user to drag a few in both labels, meaning we'll have maybe 10 examples at most for our first training. That's not a lot of examples.

I did two things to improve this.

For positive examples, I used the embedding.neighbors function (https://docs.mldb.ai/doc/#builtin/functions/NearestNeighborsFunction.md.html) to quickly retrieve the list of the 10 closest images in the 2048 dimensional space we get from running images though Inception for each positive examples, and added them as positive to the training set at a reduced training weight. Giving an example a reduced weight means the classifier will spend less effort trying to get it right, but it still counts a bit. The weight of an example is used when calculating the error function of a classifier. We can get the neighbors of an image quickly because the images' representation are stored in a dataset of type embedding (https://docs.mldb.ai/doc/#builtin/datasets/EmbeddingDataset.md.html) that stores an index in a Vantage Point Tree, making nearest neighbour searches very efficient. This is where the concept of similarity search ties in.

For the negatives, I simply added all the remaining examples as negatives with a very low weight. This exposes the model to all the variety that exists in the dataset. It doesn't cause a problem to add everything as negative for two reasons. First, since those examples are set to have a very low training weight, you can think of image that really should be positive as noise, and machine learning excels at working with noisy data. Second, if some images are clearly positive, they'll pop up as bad suggestions in the UI and the user will be able to make the correction. We benefit from having a human augmentation workflow where the user and the AI work together.

Conclusion

We have presented DeepTeach, an MLDB plugin that implements a human augmentation workflow allowing a user to quickly build an image classifier from unlabelled data within minutes. The same concepts apply to any type of data (text, sound, etc) and it really shows how humans can be made more efficient by orders of magnitude when integrating artificial intelligence in their workflow.

This plugin uses many of MLDB's unique features, such as its TensorFlow integration, its ability to host entire web applications and its lightning fast random forest training.

You can try DeepTeach by launching a free hosted instance of MLDB, or running it on your own machine. Once you're in MLDB, check the instructions on DeepTeach's repo (http://github.com/mldbai/deepteach).

If you have any questions or comments, come and chat with us on Gitter (https://gitter.im/mldbai/mldb)!

RSS (/rss.xml) - Contact (/doc/#builtin/help.md.html)