Who's my President?

Building an image classification model with ~~Apache Spark~~ Keras

University of California, Santa Barbara
PSTAT 194, Winter Quarter 2018
Professor Adam Tashman
March 19, 2018

Team Spotted Dogfish
Jason Freeberg, Samantha Lee, Timothy Nguyen

Table of Contents

Abstract

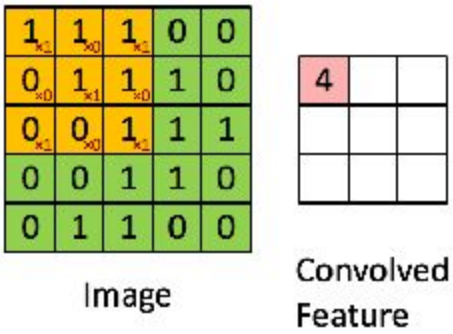
Our team sought to build an image classifier for the past four United States Presidents (Bill Clinton, George W. Bush, Barack Obama, and Donald Trump). Using the industry-standard technique known as transfer learning, we compared the performance of three Convolutional Neural Networks, Oxford University's VGG-16, Microsoft's ResNet 50, and Google's Inception V3 after fine tuning. We also investigated various data preparation methods, including training on multiple training sets and the effects of augmenting the training photos, to determine their effects on model performance.

The results determined VGG-16 as the most efficient model, and we completed the final steps from this source model. We discovered that having a larger training set will lead to better results, while augmenting the data can lead to better accuracy. Our models are evaluated to show the improvements of data preprocessing over the corresponding baseline neural network on image classification.

Introduction

Image classification is the task of assigning an input image a label from a fixed set of possible labels. This process includes systematically extracting information from the pixels of a labeled image, learning from the patterns across all labeled images, and using those patterns to compute a probability that a new, unlabeled image belongs to the labels. This field of research has far reaching applications in consumer software, manufacturing, defense, and many other industries.

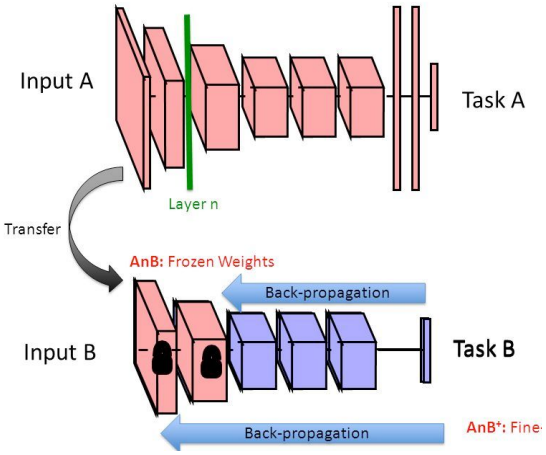
Convolutional Neural Networks



A common technique for building these machine learning models is to use a Convolutional Neural Network (CNN). These models have proven to be excellent at extracting complex patterns from training sets of images. The model's power comes from recursively passing convolution functions, such as Gaussian blurs or Sobel edge detectors, over the images. Every application of these functions reduces the output data by one pixel around the entire perimeter. So by recursively applying these functions until we are left with single-pixel outputs, we have abstracted away the location and context of the data. This leaves us with a powerful but un-interpretable representation of the image data. Please see the image showing a convolution function passing over an image. Note that the input image is 5x5 pixels, while the output is 3x3 pixels.

Transfer Learning

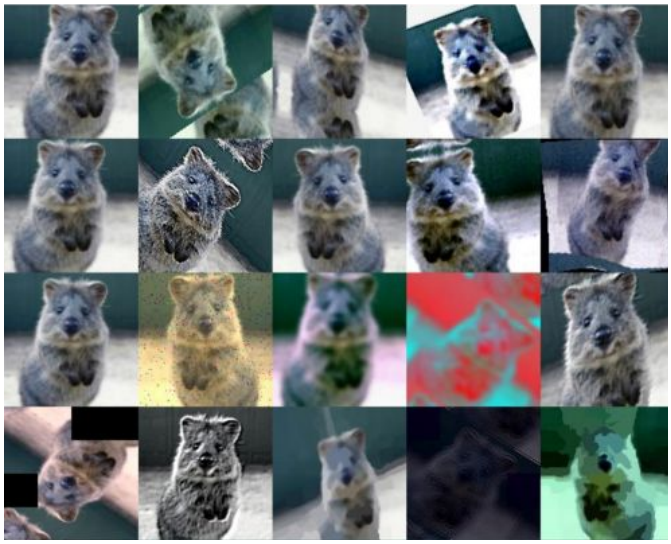
The process of training a convolutional neural network from scratch is a time consuming and computationally intensive process. A common work-around for this hurdle is the process of transfer learning. This technique leverages a CNN that has already been trained to classify a wide array of image labels. Since the CNN has already been tuned to efficiently extract features from an image, we simply replace the last layer of the network with a new decision-maker that is specific to the developer's classification problem. This "decision layer" need not be a neural network! The decision layer can be a typical classification model such as Logistic Regression, a SVM, etc.



These pre-trained models used in transfer learning are typically contributed to open source projects by large organizations, companies, and universities. Each of these neural networks come in a variety of depth, so the developer can select the most appropriate for their specific requirements.

Image Augmentation

Neural Networks are far more performant than comparable machine learning algorithms… assuming that training data is plentiful. However in image classification problems, training data can be a rare resource. A common workaround for this problem is to generate more training images from the existing images. The compilation of photos on the right shows how multiple images can be generated from the same, base image.



Common transformations are to rotate, sheer, blur, and flip the images. Other operations can act on the color scheme, by brightening, darkening, or inverting the pixels. All these operations can be applied randomly together.

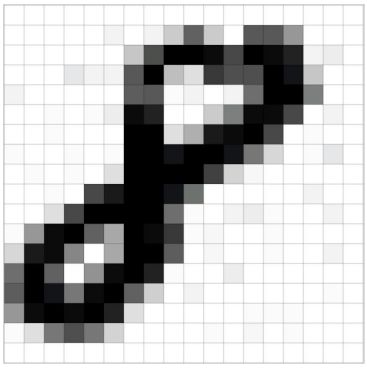
Training a CNN on extra, augmented data exposes the model to a more varied dataset. In our project, we explored the effect these transformations have on the accuracy of the Neural Network.

Data Summary

Data source

Our dataset is curated from Google, with a total of roughly 800 images and 4 classes. We collected the images using a Firefox browser plugin to download the image results for each president. We then combed through the image files to remove any obvious duplicates or corrupted files. Images from the same event (press conferences, campaign rallies) were left in the dataset. The images are of varying color schemes, perspectives, and focal points. In the sample below, one can gauge the high variance of the dataset.

Data attributes



The observations of our dataset are individual JPEG image files. Each image is represented as a three-dimensional array of integers ranging from 0 to 255, where each integer represents an individual pixel's intensity. The color channels of red, blue, and green correspond to the three dimensions. The values in each of the three arrays correspond to the intensity of each of the colors (0 meaning not present, and 255 meaning fully present). The animation shown to the right visualizes the process of mapping the pixels to integers. The process shown is repeated for each of the three color channels. Any non-JPEG images in our dataset were converted to JPEG format using the native file converter in Mac OS.

Methods

Transfer Learning from Various Source Models

Typically, image recognition through deep learning requires a great deal of data points. Because we manually gathered our data, it was not feasible to amass enough images to train a decent neural network from scratch. We turned to transfer learning to resolve this issue. We chose to compare results from the three most popular pretrained image recognition models: ResNet50 by Microsoft, VGG-16 by Oxford University, and InceptionV3 by Google. These models have all been trained on ImageNet, the largest publicly available dataset comprised of labeled images.

Our transfer learning process starts with loading a source model via the Keras *Applications* API and freezing its layers, making the parameters for those layers untrainable. Then, we added a fully connected layer with ReLu activation followed by a dropout layer to curb overfitting. Finally, we added a fully connected output layer with softmax activation and 4 nodes to accommodate the 4 presidents we wish to predict among.

We employed stratified sampling to split our data into 600 and 200 images for the train and test sets, respectively. We fit the models over the course of 30 epochs before evaluating them. The comparison between the models is displayed in Table 1.

Table 1: Source Model Comparison

Source Model	Parameters in source model	Trainable Parameters in complete model	Source model load time (s)	Complete model train time (minutes)	Validation Accuracy
VGG-16	14,714,688	6,423,812	1.01	12.276	0.760
Inception V3	21,802,784	13,108,484	12.41	12.089	0.440
ResNet-50	23,587,712	525,572	11.64	12.040	0.260

The factors we can considered were the time to load into the environment, time to train on the data, and validation accuracy on the final epoch. Please note that we trained our models using a GPU-configured EC2 instance provided by AWS, so our runtimes are low relative to the intensive computations being performed. We can see that all of the source models resulted in similar training times, so we focused on the other metrics. VGG-16 was the best fit model, having achieved a validation accuracy much higher than its more complicated counterparts. Additionally, it took the least amount of time to load due to its relatively simple architecture. We continued the rest of our project using only VGG-16.

Training with Various Dataset Sizes and Augmentations

Our next step was investigating the effects of image augmentation and training set size on model performance. We trained the models on sets of 100, 200, 400, and 600 images, all of which had balanced amounts of classes. We also trained each model with and without image augmentations applied to the training sets to give us a total of 8 models to compare. Table 2 shows the training time, validation accuracy, and validation loss of all the tested training set sizes. Our loss function for model training was classification cross-entropy.

Table 2: Performance of VGG-16

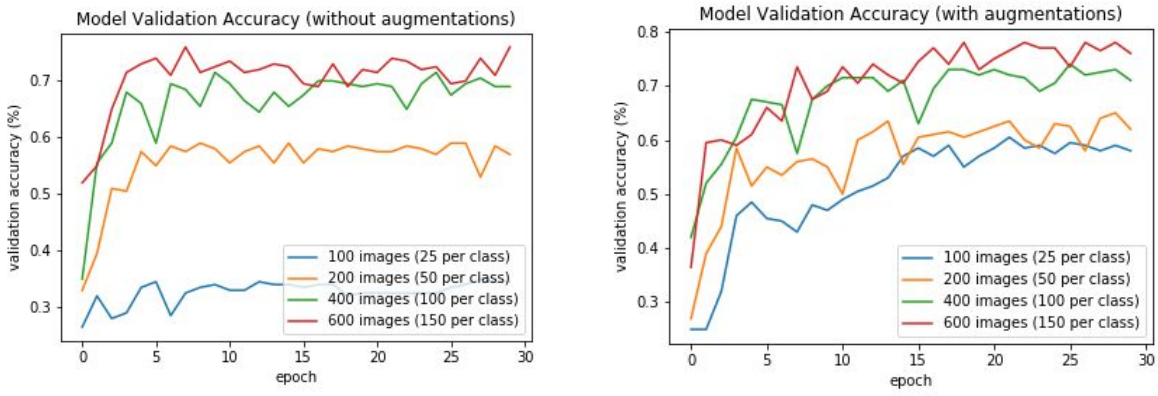
	Training Images (per class)	Train time (minutes)	Validation Accuracy	Validation Loss
Augmented	25	4.58	0.580	1.280
	50	5.30	0.620	1.306
	100	8.32	0.710	1.003
	150	12.4	0.760	0.761
Not Augmented	25	4.47	0.345	8.801
	50	5.44	0.570	1.343
	100	6.82	0.690	1.067
	150	9.76	0.760	0.8935

There is a clear improvement in accuracy with a larger training set, for both image cases, even with the longer training time. The augmented images also performed better in the testing, with a significantly lower validation loss.

Conclusions

The process of deep learning with images excels due to the extensive possibilities with data preprocessing and modeling. VGG-16, despite its drawbacks in training time, has shown to be a powerful source model using in transfer learning in the industry today. In our model comparison, VGG-16 was the quickest to load and had the highest validation accuracy. This is as expected, since VGG-16 is characterized by lesser layers and greater depth in the neural network to prove better results. Validation accuracy is also discernibly better with augmentations to the data and more points in the training data. Figure 3 is a visual comparison of the different models we implemented with these variant cases. We have reason to conclude that training set size and certain augmentations are important in creating a quality model in image classification.

Figure 3: Model Accuracy



Past Difficulties and Future Work

Difficulties

The majority of our difficulties stemmed from development and operations work: principally from our attempts to integrate standard deep-learning Python libraries such as Keras or Tensorflow with Apache Spark. It is possible to combine the two tools through the tensorframes and spark-deep-learning libraries, but correctly configuring those libraries is a very time-consuming process. The bulk of our efforts in the first two weeks were spent configuring our local and remote environments.

Once the libraries were configured, our team was limited by the computing power of our hardware. Although the CNN featurizer was trained, training our classifiers was still a long process due to the fact that each observation contains a massive amount of data. Unlike a typical classification data with simple tabular data, every observation in this project was a large JPEG image file. Spark is typically used on a cluster... however we were using Spark's local compute setting, so the communication costs between the master and the executors was not worthwhile.

Finally, our team abandoned Spark altogether in favor of using Keras on a GPU-configured EC2 instance on Amazon Web Services. The results and methods detailed in this report are all from that final attempt.

Future Work

If time had allowed, we would have investigated the performance of a model transfer learned from a source model trained on a facial images rather than the less focused scope of ImageNet.

References

1. Apache Spark v. 2.2.0
    a. https://spark.apache.org/
2. Tensorframes v. 0.2.9
    a. https://github.com/databricks/tensorframes
3. spark-deep-learning v. 0.2.0
    a. https://github.com/databricks/spark-deep-learning
4. Mozilla Firefox Plugin
    a. https://addons.mozilla.org/en-US/firefox/addon/google-images-downloader/
5. Keras
    a. https://keras.io/applications/
6. Tensorflow
    a. https://www.tensorflow.org/
7. Domino Data Labs Project
    a. https://trial.dominodatalab.com/u/jasonfreeberg/spotted_dogfish_dev/browse
8. Pre-trained Model Architectures
    a. https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
9. Facial Expression Recognition using Convolutional Neural Networks: State of the Art
    a. https://arxiv.org/pdf/1612.02903.pdf

Thank Yous

1. The man and legend that is Adam Tashman.
2. UC Santa Barbara's Statistic and Applied Probability Department for allowing our team to pilot this course.
3. Lukas Biewald of CrowdFlower for hosting a deep learning workshop from which we learned how to improve our computer vision models.
4. Domino Data Labs for letting us wrack up an enormous AWS bill.