embedded VISION SUMMIT 2018

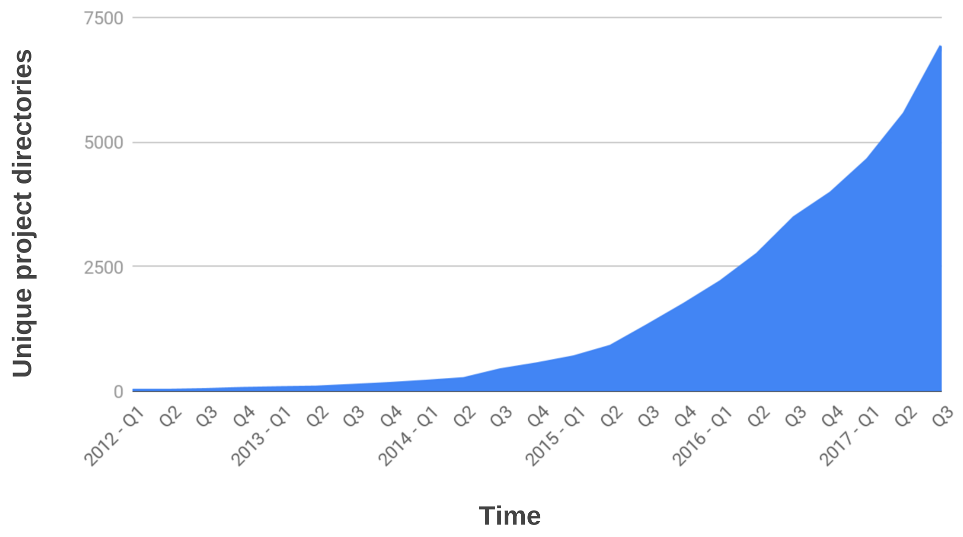
Solving Vision Tasks Using Deep Learning: An Introduction

Google

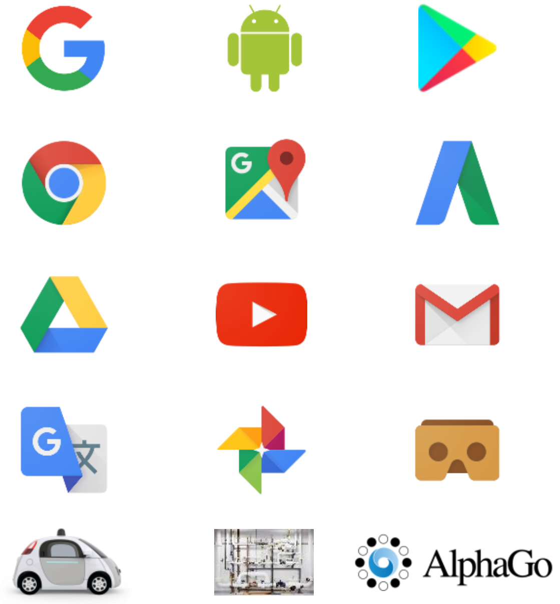
Pete Warden
May 2018

Google is an AI company

Google 3 directories containing Brain Model



Used across products:



What will I be covering?

- What is deep learning?
- How does it apply to vision problems?
- How can you use it on embedded and mobile devices?

What will I not be covering?

- In-depth coding questions
- Data acquisition and training

What is deep learning?

Traditional programming is like writing recipes

Deep learning is like teaching from examples

What is deep learning?

For example, detecting red-eye from camera flash in photos.

Before deep learning, I might write pseudo-code like:

- Scan image for pixels with RGB between (127, 0, 0) and (255, 127, 127)
- Look for dense circular clusters of those pixels
- Look for similar nearby pairs of these clusters

The work of writing these rules is known as feature engineering.

What is deep learning?

What’s wrong with this approach?

- Scan image for pixels with RGB between (127, 0, 0) and (255, 127, 127)
  - What colors should I be looking for exactly?
- Look for dense circular clusters of those pixels
  - How dense and how circular do these clusters need to be to match?
- Look for similar nearby pairs of these clusters
  - How similar, how nearby?

There are lots of rules to write with arbitrary cut-offs that are hard to figure out.

The cut-offs are also hard, there’s no fuzziness or probability.

What is deep learning?

What’s wrong with this approach, continued?

The decisions required make writing these kind of algorithms very hard.

They require a lot of experimentation to get right, but there’s no clear guidance about how to make changes when you get bad results. You’re often left tweaking parameters almost at random to get good results.

All this hard work doesn’t translate to other similar tasks, since just a change of camera or lighting conditions make break the assumptions it relies on.

Run-time will depend on number of detections - a crowd of faces may be very slow!

What is deep learning?

How would deep learning tackle the same problem?

- Take hundreds of thousands of typical photos
- Add bounding boxes and labels for any occurrences of red-eye
- Run that data through a deep learning model repeatedly to train it
- Predicted bounding boxes and probabilities are output by the trained model

For now, ignore how the training process works! Treat it as a black box that takes labeled data in and eventually outputs a predictive model.

What is deep learning?

How is this approach better?

- The arbitrary choices (once you've chosen a model architecture) are learned from the data, not chosen by an engineer.
- Feature engineering is handled by the model, not the developer.
- You get probabilities out for predictions, not hard cutoffs.
- Similar problems can be solved with no code changes, just by supplying different training data
- The run-time doesn't depend on the image contents.
- The results are almost always better!
- Degrades gracefully, so you can trade off accuracy for latency or memory.

What is deep learning?

What are the challenges?

- You need lots of data
  - My rule of thumb is 1,000 images per class if you're categorizing: https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/
  - Transfer learning, starting with a pre-trained model, reduces this dramatically: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/
- Training is still a dark art
  - Start with examples like the code lab above, or Keras
  - Don't worry if you don't understand how it works - nobody really does!

How does it apply to vision problems?

Many common vision problems fall into a few broad categories:

- Classification - Is this picture …
  - taken indoors or outdoors? - http://cs231n.stanford.edu/reports/2017/pdfs/102.pdf
  - pornographic? - https://github.com/mdietrichstein/tensorflow-open_nsfw
  - of a kind of object? - https://github.com/tensorflow/models/tree/master/research/inception
  - of a particular person's face? https://github.com/davidsandberg/facenet
- Localization - In this picture, where are …
  - the objects? https://github.com/tensorflow/models/tree/master/research/object_detection
  - the faces? https://github.com/yeephycho/tensorflow-face-detection
- Pose estimation for people - https://github.com/eldar/pose-tensorflow
- Segmentation - https://github.com/mrgloom/awesome-semantic-segmentation

How does it apply to vision problems?

Figure out what the closest broad category to your problem is.

If you can, find a pretrained model for that category, and then use transfer learning so you don’t have to do full retraining:

- Much faster to train
- Needs much less data
- Easier to set up
- Slightly less accurate than training from scratch with a massive data set (but a good small set of data with transfer learning will beat a lower-quality large set)

How does it apply to vision problems?

Transfer learning starting points:

- For classification, TensorFlow for Poets: https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#0
  - Example takes less than an hour to run on a laptop, with no GPU required.
- For localization, Object Localization API: https://towardsdatascience.com/building-a-toy-detector-with-tensorflow-object-detection-api-63c0fdf2ac95
- Other tasks may require more research. Look for “fine tuning” or “transfer learning” together with a model name.

How does it apply to vision problems?

If you really have to train a model from scratch:

- Do you ***really***? Transfer learning on a pretrained model lets you quickly identify gaps and problems with your training data, which is going to give you much bigger improvements in quality than full training.
- Good data matters much more than models or parameter choices.
- Improving and expanding data is almost always the best way to invest your time.
- There's an ancient technique called "Looking at your data" that's seldom used, but very valuable!
  - https://lukeoakdenrayner.wordpress.com/2017/12/18/the-chestxray14-dataset-problems/

How does it apply to vision problems?

For this stage, ignore the constraints of your target platform, and just try to build a model that works well enough on test data.

It’s likely that this will be the highest risk part of the project, so try to front load the risk.

Deploying models on embedded and mobile platforms is an engineering challenge, but it’s a linear process.

Training models to acceptable performance for your application is a lot less predictable.

Why is training data so important?

Make sure that you have test and training data that accurately reflects the application environment:

- Similar lighting conditions
- Similar framing
  - For example, ImageNet is full of well-composed photos taken by people, a robot will have lots of images where objects are half cropped out since it doesn't know to center them.
- Geometry, sensors
  - A security camera may have a fisheye FoV, shoot in black and white or under IR illumination, and be positioned at ceiling height.
- Content
  - A wildlife camera in a jungle shouldn't be trained on photos of penguins, since we know from context that any such detection is very likely to be an error. Make sure you're training on the kind of images you'll see, and if not you may need to post-calibrate your results.

How can you use it on mobile and embedded?

One big difference between cloud-based deployments and edge/on-device is that you have hard resource limits for storage, network transfer, and computation.

Cloud gives you a lot more flexibility for resources. You can always buy more memory or compute, so the decision becomes about what you can afford.

Embedded has much harder limits... Physical limits on memory and compute, and often battery life or thermal limits that restrict these even further.

As your first planning step, understand your compute and memory budget, so you can target your model deployment to fit that.

How can you use it on mobile and embedded?

What are your platform's resource constraints?

- Are you downloading the application (for example from a mobile store)?
  - Figure out how much extra space the feature can add to the application size.
  - It may only be a few hundred KB. Don't despair, we can work with this!
  - The two main sources of application size bloat are the extra code needed to run the model, and the model weights themselves.
    - TensorFlow with no changes adds 11MB of code on Android and iOS! This shrinks to 1.5MB using "selective registration" for a typical model, since this removes operators that aren't used.
    - TensorFlow Lite only adds 300KB of code for a typical model, and just 70KB for the core.

How can you use it on mobile and embedded?

What are your platform's resource constraints?

- Do you have storage and RAM limits?
  - Same code size estimates apply
    - (for example minimum of 1.5MB for Tensorflow, minimum of 70KB for TensorFlow Lite)
  - A model's size is usually dominated by the number of weight parameters it contains.
    - You can estimate this by running tensorflow/tools/graph_transforms /summarize_graph in TensorFlow on a frozen graph.
    - See http://www.oreilly.com/data/free/building-mobile-applications-with-tensorflow.csp for technical details on running these steps.
  - Multiply number of weights by four to get the bytes needed for default float weights.
  - Often you can use eight-bit weights (see guide above), which reduces size by 75%!

How can you use it on mobile and embedded?

Do you have processing limits?

- Understand what processing power a model requires:
  - Computation needed can be estimated from the math ops inside the model.
  - Run benchmark_model to estimate FLOPs (see guide on previous page).
- Estimate what your platform can do:
  - If you can, run a model with known FLOPs and measure latency to get FLOPs per second.
  - If GoogLeNet v1 (which uses 1.5 billion FLOPs) takes 1.1 seconds to run, then platform can do (1.5 billion/1.1) = 1.4 GFLOPs per second.
  - If you can't run a model yet, get a rough estimate from the platform spec.
  - If you have a device that runs at 1GHz, and a multiply-add instruction takes 2 cycles, and you have 1 core, then theoretical peak is (1 billion / 2) * 1 = 500 million FLOPs per second.
  - In practice, divide peak performance by five or ten to get realistic estimate of actual performance!

How can you use it on mobile and embedded?

Know your platform constraints, write them down as a budget

For example:

- SoC X can run 500 MFLOPs/second, has 1,000 KB storage free.

Then look at the models that are available:

- MobileNet v1 224 1.0 requires 569 MFLOPs and has 4.24 million parameters.
- Therefore it can run at about one frame per second, and would require 4.24 MB of memory and download bandwidth.

How can you use it on mobile and embedded?

Then look at what your application needs:

- Has to run image classification at 5 FPS, with at least 50% top-1 accuracy.

MobileNet v1 224 only runs at 1 FPS, and takes up too much storage space (4.24MB when only 1.0MB is available).

But, this version has 71% accuracy. We can trade off some of that accuracy for reduced processing and memory requirements.

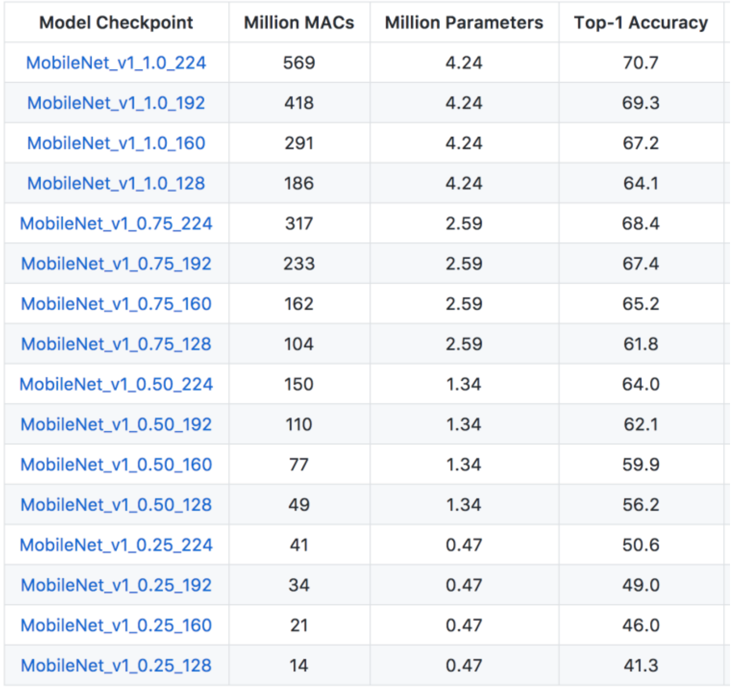
How can you use it on mobile and embedded?

https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

MobileNet has a variable architecture.

You can trade off size and compute for accuracy by picking a model from the menu that meets your requirements.

In our case, we need a model that consumes less than 1,000 KB of memory, and less than 100 MFLOPs. One MAC (multiply-accumulate) is two FLOPs, so mobilenet_v1_0.25_224 fits.



Model Checkpoint	Million MACs	Million Parameters	Top-1 Accuracy
MobileNet_v1_1.0_224	569	4.24	70.7
MobileNet_v1_1.0_192	418	4.24	69.3
MobileNet_v1_1.0_160	291	4.24	67.2
MobileNet_v1_1.0_128	186	4.24	64.1
MobileNet_v1_0.75_224	317	2.59	68.4
MobileNet_v1_0.75_192	233	2.59	67.4
MobileNet_v1_0.75_160	162	2.59	65.2
MobileNet_v1_0.75_128	104	2.59	61.8
MobileNet_v1_0.50_224	150	1.34	64.0
MobileNet_v1_0.50_192	110	1.34	62.1
MobileNet_v1_0.50_160	77	1.34	59.9
MobileNet_v1_0.50_128	49	1.34	56.2
MobileNet_v1_0.25_224	41	0.47	50.6
MobileNet_v1_0.25_192	34	0.47	49.0
MobileNet_v1_0.25_160	21	0.47	46.0
MobileNet_v1_0.25_128	14	0.47	41.3

How can you use it on mobile and embedded?

In general follow this process:

- Train a model to as high accuracy as you can, ignoring resource constraints.
  - Focus on gathering more data, making it more relevant, and improving labeling.
  - Don't worry too much about the architecture.
- Once you have a model working on the desktop, scale it down to fit.
  - TensorFlow offers scalable models for image classification and localization.
  - You should be able to apply similar approaches to other problems.
  - Once you have high-quality training data, it's likely most models will train well.

Thanks!

- https://twitter.com/petewarden
- petewarden@google.com

References and Resources

- https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/
- https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/
- PlantVillage crop disease detection: https://www.wired.com/story/plant-ai/
- MeterMaid Monitor: https://www.raspberrypi.org/blog/meter-maid-monitor-parking-protection-pi/
- Train spotting: https://svds.com/introduction-to-trainspotting/

References and Resources

My favorite resources to learn more:

- Jason's Machine Learning 101 - https://docs.google.com/presentation/d/1kSuQyW5DTnkVaZEjGYCkfOxvzCqGEFzWBy4e9Uedd9k/edit
- Kaggle's deep learning reading list - https://www.kaggle.com/getting-started/37999
- Rachel and Jeremy's Fast AI course - http://www.fast.ai/
- Ian Goodfellow's Deep Learning book - http://www.deeplearningbook.org/
- My blog! https://petewarden.com/

References and Resources

https://www.tensorflow.org/mobile/tflite/ - Introduction to TensorFlow Lite

http://www.oreilly.com/data/free/building-mobile-applications-with-tensorflow.csp - Building Mobile Applications with TensorFlow

Post on StackOverflow with the ‘tensorflow’ tag, we’ll see it!