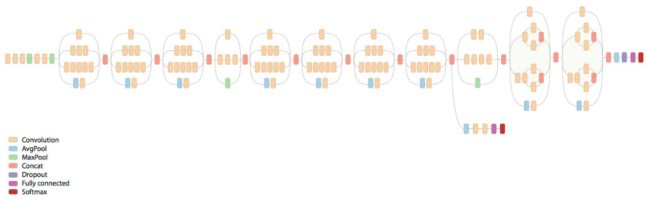
THE NEURAL PERSPECTIVE

DEEP LEARNING SIMPLIFIED

IMAGE RECOGNITION

Edit

In the **CNN tutorial**, I covered the basics of how image recognition/classification works. Now we will extend that beyond just classifying digits. However, we will be using some very powerful pre-trained models to help with our unique task. Our tutorial will be based on this **link** but I will extend to show how to train for a variety of different tasks and also talk a bit more about transfer learning and its nuances.



Benefit from trained intricate image processing models, such as the Inception model.

INSTALLATION:

For this implementation, we will be using some component not available in the current stable tensorflow version (r0.10). So if you are using this one or an older one, head over to this **link** and click on the right version for your computer and then download with something like:

```
sudo pip install --upgrade tensorflow-0.11.0rc0-py2-none-any.whl
```

Next get into your workspace and download the models from tensorflow with:

```
git clone https://github.com/tensorflow/models/
cd models/slim
```

FINE-TUNING A MODEL

DIRECTORIES

We will first set up the directories for the trained model (on original dataset), new dataset directory which has the data we wish to train on and a train directory to store new fine-tuned checkpoints.

```
PRETRAINED_CHECKPOINT_DIR=/tmp/checkpoints
DATASET_DIR=/tmp/flowers
TRAIN_DIR=/tmp/flowers-models/inception_v3
```

DATA

We will download the flowers dataset and train it on inceptionV3 which was originally trained on imagenet. The program below changes images into TFRecord format. Later on, we will take a look at how to do this with a dataset of image of our own.

```
python download_and_convert_data.py \
 --dataset_name=flowers \
 --dataset_dir=${DATASET_DIR}
```

MODEL

Next will download the Inception v3 model with the commands below. Note that Google is always releasing new models so keep up with the **research blog** to get the latest ones.

```
if [ ! -d "$PRETRAINED_CHECKPOINT_DIR" ]; then
 mkdir ${PRETRAINED_CHECKPOINT_DIR}
fi
```

- SEARCH -

Search …

- FOLLOW ME ON TWITTER -

Tweets by @GokuMohandas

Goku Mohandas
@GokuMohandas

Your model may be performing really well by incorrectly focusing on confounding features (extraneous influencers in the data that aren't accounted for). Check out @johnrzech's post where x-ray stickers unintentionally influenced the classifications: medium.com/@jrzech/what-a…

Embed View on Twitter

- RECENT POSTS -

Update on Embeddings (Spring 2017)

Exploring Sparsity in Recurrent Neural Networks

Question Answering from Unstructured Text by Retrieval and Comprehension

Overcoming Catastrophic Forgetting in Neural Networks

Opening the Black Box of Deep Neural Networks via Information

```
if [ ! -f ${PRETRAINED_CHECKPOINT_DIR}/inception_v3.ckpt ]; then
 wget http://download.tensorflow.org/models/inception_v3_2016_08_28.tar.
 tar -xvf inception_v3_2016_08_28.tar.gz
 mv inception_v3.ckpt ${PRETRAINED_CHECKPOINT_DIR}/inception_v3.ckpt
 rm inception_v3_2016_08_28.tar.gz
fi
```

Inception v3 was originally trained on Imagenet (1000 classes) but we will fine-tune the model to perform on the flowers dataset (5 classes). We can't just use the entire original model and expect results so we will keep the earlier layers from the model and implement new layers at the end to fine-tune for our task. The intuition behind this is simple; the earlier layers are able to learn general features that can be assumed to be ubiquitous features in all images. So we can keep the exact weights for these layers. But the later layers, especially the ones nearing the softmax classification are highly skewed towards the original task (Imagenet). They were trained to pick up concrete features based on the input images so we will remove these weights and just retrain these points with our new task. The early conv layers' weights will be frozen and will maintain their original weights while fine-tuning. I oversimplified this principle of transfer learning but below is a good set of rules to go off on from cs231. For a more formal study, I suggest this **paper** for more information on how transferable some features can be.

1. ***New dataset is small and similar to original dataset.*** Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.
2. ***New dataset is large and similar to the original dataset.*** Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.
3. ***New dataset is small but very different from the original dataset.*** Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
4. ***New dataset is large and very different from the original dataset.*** Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

It's also a good idea to keep some constraints in mind, such as using a small learning rate. Our final linear classifier layer(s) may be randomly initialized but if we choose to fine-tune some of the layer conv layers as well, then we need to use a small learning rate. The original conv layer kernel weights for these later layers are still pretty decent so we don't want to distort them by a lot. If we use a large learning rate, the effect from backpropagation will be high especially because our linear classifiers have been randomly initialized.

TRAIN (FINE-TUNE)

```
python train_image_classifier.py \
 --train_dir=${TRAIN_DIR} \
 --dataset_name=flowers \
 --dataset_split_name=train \
 --dataset_dir=${DATASET_DIR} \
 --model_name=inception_v3 \
 --checkpoint_path=${PRETRAINED_CHECKPOINT_DIR}/inception_v3.ckpt \
 --checkpoint_exclude_scopes=InceptionV3/Logits,InceptionV3/AuxLogits \
 --trainable_scopes=InceptionV3/Logits,InceptionV3/AuxLogits \
 --max_number_of_steps=1000 \
 --batch_size=32 \
 --learning_rate=0.01 \
 --learning_rate_decay_type=fixed \
 --save_interval_secs=60 \
 --save_summaries_secs=60 \
 --log_every_n_steps=100 \
 --optimizer=rmsprop \
 --weight_decay=0.00004
```

With the parameters that we are passing in above, we are fine-tuning our inceptionV3 model with the flowers dataset. Since the dataset is ***small and similar to original dataset*** we only have to train the logit producing (FC) layers. If we wanted to train previous layers we just have to append the appropriate layers by name to checkpoint_exclude_scopes and trainable_scopes as below. To get the names of layers for **inceptionV3** (or other pretrained models), head over to **this link**.

```
Old name          | New name
=====================================
conv0             | Conv2d_1a_3x3
conv1             | Conv2d_2a_3x3
conv2             | Conv2d_2b_3x3
pool1             | MaxPool_3a_3x3
```

```
conv3           | Conv2d_3b_1x1
conv4           | Conv2d_4a_3x3
pool2           | MaxPool_5a_3x3
mixed_35x35x256a | Mixed_5b
mixed_35x35x288a | Mixed_5c
mixed_35x35x288b | Mixed_5d
mixed_17x17x768a | Mixed_6a
mixed_17x17x768b | Mixed_6b
mixed_17x17x768c | Mixed_6c
mixed_17x17x768d | Mixed_6d
mixed_17x17x768e | Mixed_6e
mixed_8x8x1280a  | Mixed_7a
mixed_8x8x2048a  | Mixed_7b
mixed_8x8x2048b  | Mixed_7c
```

```
--checkpoint_exclude_scopes=InceptionV3/Logits,InceptionV3/AuxLogits,Inc
--trainable_scopes=InceptionV3/Logits,InceptionV3/AuxLogits,InceptionV3/
```

To train the entire network from scratch, just remove the two lines above.

EVALUATION

To see how our model is doing after training, we can evaluate of the validation set using the commands below. This will give us recall (top 5) and accuracy.

```
# Run evaluation.
python eval_image_classifier.py \
 --checkpoint_path=${TRAIN_DIR} \
 --eval_dir=${TRAIN_DIR} \
 --dataset_name=flowers \
 --dataset_split_name=validation \
 --dataset_dir=${DATASET_DIR} \
 --model_name=inception_v3
```

OUR DATA

So far we've seen how to fine-tune a pretrained model with a new dataset but some parts may still be unclear for when we want to train on any dataset. So now, we will take a closer look at creating a TFRecord formatted dataset for fine-tuning and we will also do some evaluation with qualitative results. More details available on the inception **page**. Let's move WORKSPACE and the inception directory from models/**inception**/ to models/slim/cat_classifier. We will also create TRAIN and VALIDATION directories to hold our image data (details below). Also create an empty TFRecord_data folder to store our formatted data in later.

Let's build an iconic cat classifier. We'll first get our images from Google image and we will use **Fatkun Batch** to download the images. Once you download the dogs and cats (they download into separate folders and at the location you specified), let's go ahead and convert them to TFRecord format. We will first need to arrange our data to look like below:

```
 $TRAIN_DIR/dog/image0.jpeg
 $TRAIN_DIR/dog/image1.jpg
 $TRAIN_DIR/dog/image2.png
 ...
 $TRAIN_DIR/cat/weird-image.jpeg
 $TRAIN_DIR/cat/my-image.jpeg
 $TRAIN_DIR/cat/my-image.JPG
 ...
 $VALIDATION_DIR/dog/imageA.jpeg
 $VALIDATION_DIR/dog/imageB.jpg
 $VALIDATION_DIR/dog/imageC.png
 ...
 $VALIDATION_DIR/cat/weird-image.PNG
 $VALIDATION_DIR/cat/that-image.jpg
 $VALIDATION_DIR/cat/cat.JPG
 ...
```

```
# location to where to save the TFRecord data.
TRAIN_DIR=TRAIN/
VALIDATION_DIR=VALIDATION/
OUTPUT_DIRECTORY=TFRecord_data/
LABELS_FILE=labels.txt

# build the preprocessing script.
bazel build inception/build_image_data
```

```
# convert the data.
bazel-bin/inception/build_image_data \
```

```
  --train_directory="${TRAIN_DIR}" \
  --validation_directory="${VALIDATION_DIR}" \
  --output_directory="${OUTPUT_DIRECTORY}" \
  --labels_file="${LABELS_FILE}" \
  --train_shards=2 \
  --validation_shards=2 \
  --num_threads=1
```

shards: "we aim for selecting the number of shards such that roughly 1024 images reside in each shard."

labels.txt: Needs to go inside: Note that the labels skips 0 since it is reserved for background.

```
OUTPUT_DIRECTORY=TFRecord_data/
```

1:dog
2:cat

Note: when executing the bazel command you may get some weird errors, just retry in a few seconds or delete the folder and restart. You may also get some errors from copy/paste from here or a notepad (esp. for the bazel-bin command, so make sure it's all cleanly copied)

So now, inside TFRecord_data, we will have our TFRecord formatted images like so:

Now we can train inceptionV3 on our new image dataset.

```
PRETRAINED_CHECKPOINT_DIR=/tmp/checkpoints
DATASET_DIR=cat_classifier/TFRecord_data/
TRAIN_DIR=/tmp/cat_model/inception_v3
```

```
python train_image_classifier.py \
 --train_dir=${TRAIN_DIR} \
 --dataset_name=flowers \
 --dataset_split_name=train \
 --dataset_dir=${DATASET_DIR} \
 --model_name=inception_v3 \
 --checkpoint_path=${PRETRAINED_CHECKPOINT_DIR}/inception_v3.ckpt \
 --checkpoint_exclude_scopes=InceptionV3/Logits,InceptionV3/AuxLogits \
 --trainable_scopes=InceptionV3/Logits,InceptionV3/AuxLogits \
 --max_number_of_steps=1000 \
 --batch_size=32 \
 --learning_rate=0.01 \
 --learning_rate_decay_type=fixed \
 --save_interval_secs=60 \
 --save_summaries_secs=60 \
 --log_every_n_steps=100 \
 --optimizer=rmsprop \
 --weight_decay=0.00004
```

You need to change the following lines from flowers.py from models/slim/datasets/flower.py to:

```
#_FILE_PATTERN = 'flowers_%s_*.tfrecord'
#SPLITS_TO_SIZES = {'train': 3320, 'validation': 350}
#_NUM_CLASSES = 5

_FILE_PATTERN = '%s-*'
SPLITS_TO_SIZES = {'train': 100, 'validation': 100}
_NUM_CLASSES = 2 # NOT INCLUDING BACKGROUND
```

When evaluating this fine-tuned model, you can run the same evaluation script as before. But before running, change the following line in eval_image_classifier.py:

```
1 | 'Recall@5': slim.metrics.streaming_recall_at_k(logits, labe
```

You may want to change k (5) or comment this out as this will cause an error if you have less than 5 classes in your task.

```
python eval_image_classifier.py \
```

```
--checkpoint_path=${TRAIN_DIR} \
--eval_dir=${TRAIN_DIR} \
--dataset_name=flowers \
--dataset_split_name=validation \
--dataset_dir=${DATASET_DIR} \
--model_name=inception_v3
```

QUALITATIVE RESULTS:

Let's take a look at some qualitative results. I will be extending on material from this **notebook**.

I made a python file called **qualitative_evaluation.py** that can process images from our training/validation batch or from a url using our new fine-tuned model.

Posted in: *Image Recognition*
Tagged: *Tutorials*

← *Generative Adversarial Networks (GAN)*

Gradients, Batch Normalization and Layer Normalization →

ONE THOUGHT ON "IMAGE RECOGNITION"

ANNE July 18, 2017 at 6:57 pm EDIT REPLY →

Hello can u give few insights on qualitative_evaluation.py that u have written? Can you share your notebook or github link ?

★ Like

LEAVE A REPLY

Enter your comment here...