Image Classification using Transfer Learning from Siamese Networks based on Text Metadata Similarity

Dan Iter
Stanford University
daniter@stanford.edu

Abstract

Convolutional neural networks learn about underlying image representations just by optimizing to a supervised classification. This project attempts to learn better features from images by training a network based on the similarity of pairs of images. Similarity of images will be computed based on the similarity of the text associated with the images as metadata, specifically captions for MSCOCO but the text used is extendable for other datasets such as Flickr 100M tags and descriptions. While these similarity measures will not be as precise as manual classification, thus being noisy labels, it will allow for training on much larger datasets since many such image/ text pairs are readily available on the internet. To learn similarity between images, we will implement a Siamese network architecture where two separate towers share weights and combine the features representation for a pair of images only at the final fully connected layers. We will evaluate the quality of the features learned by using them as the initial weights for the image classification task and then finetuning to learn the final layer that translates from the 4096 element vector representation to a classification.

1. Introduction

Two major challenges facing the computer vision community are (1) building large corpora of annotated images that can be used for training models and (2) integrating non-image data into visual models to allow for more sophisticated application. Building large corpora such as ImageNet are prohibitively expensive because they require human workers to classify images. This task can even grow to be challenging for humans because of the large number of classes and complexity of the images. Furthermore, while images still have much data that we are not able to extract effectively, most of the internet is a combination of images and text which are related. There are also many industry applications that collect both image and text data together, such as medical records and insurance claims. This project attempts to make progress on both of these fronts. Specifically, this experiment will use text metadata as the noisy classifications for images. The MSCOCO dataset provides 5 captions for each image that will be used as the metadata. Rather than trying to classify images, images that have similar metadata are assumed to have similar feature representations and vis versa. This lifts the requirement of annotated corpora because images with some kind of user metadata is readily found on the internet. The main tradeoff is handling noisy labels in exchange for significantly expanding the set of available training data.

Similarity of the metadata is measured with cosine similarity. However there is a clear tradeoff between the complexity of the similarity function and the quality of the feature representations learned by the system. A major challenge with this approach is that there is significant noise in the metadata because the data was not created for the purpose of image classification but rather for general sharing on the internet. They may be simply names of people, location or short comments that need not mention all the items in the image. However, there is still a shared context which this project tries to exploit in order to reach a better representation of the data in the image. While out of the scope of this project, there are many applications that contain related images to large amounts of text, such as medical records with x-rays and insurance claims with images of car accidents, that may be able to leverage models that include both images and text data.

While a goal of this work is to provide an opportunity to scale training data, building such large networks and datasets is very expensive in resources and training time. Furthermore, the noisier the labels, the larger the training dataset must be to compensate. Therefore, in an attempt to prove some value in this approach before significant investment, the experiments are run using the MSCOCO 2014 dataset [3] because it is much smaller and has high quality text data associated with each image. It should be clear that the MSCOCO captions can be replaced by Flickr 100M [4] tags and descriptions to scale the training set up at the cost of a noisier signal.

This work assumes that transfer learning can be leveraged to save time and resources training networks from scratch. Specifically, all the networks are initialized with some weights that have been learned from previous larger scale training, specifically by CaffeNet [5]. The high accuracy of the baseline classification task of MSCOCO images indicates that CaffeNet weights provide a quality feature representation even when run on a different dataset of images.

The baseline for this experiment is to build an image classifier for MSCOCO by fine tuning weights from CaffeNet to save on compute time of training the network from scratch. The Siamese network which learns similarity between images will be run on a large sampling of pairs of images and will also fine tune over the CaffeNet weights to provide a slightly different feature representation but not pay the price of training from scratch. The middle layers of the Siamese network are the same as the baseline architecture so this work hopes to show that retraining the baseline using the weights from the Siamese network rather than the CaffeNet weights will improve the image classification accuracy.

2. Related Work

This works expands on the work done by Han *et al.* [1] and Doersch *et al.* [2]. MatchNet describes a two-tower architecture that is similar to the architecture used in this study. The main idea is that the two towers share weights but then concatenate the resulting feature representations of pairs of patches that are then fed into a fully connected metric network and softmax loss function. The metric network measures the similarity of two feature representations and calculates the loss to the ground truth similarity between the pair of patches. This is congruent to this projects calculation of the similarity of two feature representations, but our ground truth is based on the text metadata that accompanies the the images rather than a computed function over the two images.

Doersch *et al.* [2] focuses on learning features on an unsupervised dataset by splitting a single image into 9 patches and training a network than can guess the correct orientation between patches. The intuition is that if the network can learn the orientation of patches, it must have learned some underlying features of the objects in the images. Using these learned representations, Doersch *et al.* show that the features learned from this network can then be used for unsupervised object discovery. This project hopes to find similar results of being able to find underlying object features from “similar” images.

3. Dataset

This experiment was conducted using the MSCOCO 2014 dataset [3]. The training set has 82081 images. Each image has 5 captions which are full sentences that were written by humans for the purpose of building this dataset. These captions are used as the text metadata in computing the similarity between images. There are also 90 categories, with each image being labeled by a single category. These categories are used as the image classes for the baseline and evaluation networks that are described in the experiments section of this paper.

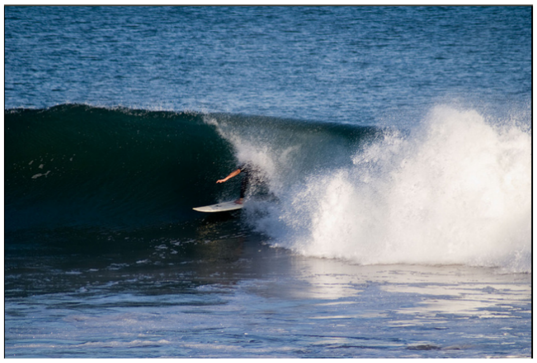
The long term goal is to exploit the large amount of available images with text metadata to see if growing the size of the dataset continues to improve the model as more “noisily” labeled images are added. A potential dataset for this task would be the Flickr100M [4]. This dataset has 100 million images along with titles, tags and descriptions set by the uploader. However, this dataset presents two major challenges. The text is very noisy. Titles are often default names such as “IMG_XXX”, tags often describe the camera used, such as “Nikon”, rather than the content of the image and the descriptions occasionally contain advertisements and copyright statements. This could potentially be handled by either naively feeding the data into the network and hoping it learns what is and isn’t relevant or by doing some kind of data cleaning during sampling such as stripping out certain titles and tags that are known to not add information. This task can quickly grow in scope however and measuring its effectiveness can be challenging. Secondly, the dataset is extremely large providing a challenge for even storing the data on disks or loading data efficiently. A working neural network over Flickr 100M would require significant engineering and resource investment to make the data manageable.

For these reasons, this project focuses on the smaller and less noisy MSCOCO dataset to try to show some indication of value before trying to scale up to large and noisier data. Two sample images that were used as a similar pair are shown in Figure 1, including their labels and captions.

4. Computing Similarity

The similarity measure takes the text metadata associated with a pair of images and returns a score between 0 and 1 of how similar the images are expected to be based on the text. Obviously similar words should indicate similarity but many repeated words such as “a” or “the” should not indicate similarity. Different sizes of text data should be comparable, so the function must be robust to comparing between small and large instances. Finally, since there is a large number of pairs that can be generated even from a small dataset, the operation should not be prohibitively expensive as it is run many times, depending on the sampling

(a) Label: Person
Captions:
A man riding a wave on top of a surfboard.
A man on a surfboard riding an ocean wave.
A man in a black wetsuit riding a wave on a white surfboard.
a man surfing in the water with a body suit
The man in the black wet suit is surfing on a board.



(b) Label: Surfboard
Captions:
A man riding a wave on top of a surfboard.
A surfer is in the middle of an ocean wave.
Surfer emerging from water tunnel in the ocean.
A person is riding a wave on a surfboard.
a person on a surf board riding through the waves

Figure 1: Two images from the MSCOCO [3] dataset.
Cosine Similarity: 0.861295

approach.

Algorithm 1 Cosine Similarity

```
1: procedure GetSimilarityOfPair
2:     caps1 ← Counter( split ( image 1 captions))
3:     caps2 ← Counter( split ( image 2 captions))
4:     intersection ← keys( caps1) ∩ keys( caps2)
5:     numerator ← $\sum$ caps1[x] * caps2[x]
           for x ∈ intersection
6:     sum1 ← $\sum$ caps1[x]$^2$ for x ∈ keys( caps1)
7:     sum2 ← $\sum$ caps2[x]$^2$ for x ∈ keys( caps2)
8:     denominator ← $\sqrt{sum1 * sum2}$
9:     return numerator / denominator
```

As the initial approach, this paper uses the cosine similarity between two bags of words, created by taking the set of words associated with a given image and the counts for each word. Cosine similarity provides a reasonable similarity measure and allows for comparing different sized sets. Furthermore, it is a fast operation that can be easily computed during sampling as described later in this paper. Algorithm 1 outline the pseudo-code for the similarity measure. Counter refers to the python class that returns a dictionary of words as the keys and their respective counts as their values.

There may be better measures for similarity of text metadata and this work does not attempt to measure the quality of this measure or compare it to other measures. There is also a likely trade off between the computational complexity of the similarity measure and the noisiness of the results. However, this simple measure was used for this work leaving further exploration in this space out of the scope of this paper.

5. Sampling

As Doersch *et al*. [2] reports, using a naive sampling of images, the model will be biased toward dissimilar images because dissimilarity is much more common in practice. This project will include building a preprocessing engine that computes similarities of images, creates balanced pairs of similar and dissimilar images and loads them into an LMDB database for easy loading during CaffeConTroll run time. As mentioned above, this component will also encapsulate the definition of similarity between text and can be arbitrarily expanded. If this project finds positive results with a simple similarity function, a future study could include the tradeoffs between more complex similarity models vs the runtime and effectiveness of the end to end system.

Since the MSCOCO 2014 dataset has only 82,081 images, all 3.3 billion pairs were explored but only pairs that had a similarity of greater than 0.85 or less than 0.05 were considered. The purpose of this purging was to provide the network with only examples of similar and non-similar images rather than also complicating the network with different grades of similarity. The thresholds were tuned to return roughly 2 million examples each. The final set of pairs were randomly sampled 1.1 million pairs of each similar and non-similar.

Note that while all combinations of 80K images is possible, this is no longer possible as the number of base images increases. Therefore a more scalable approach would be to do random sampling. This raises issues of what factors to balance in the sampling. While this was out of the scope of this work, some of these issues are mentioned here for future work. There is an open question of the value of choosing pairs with and without replacement. It is unclear how having the same images in multiple pairs may influence the network. Furthermore, it may be desirable to balance the number of pairs for each categories. For example, if there are an equal number of similar and non-similar pairs, but all the similar pairs are of a single type, such as cat, then this will likely produce biases in the network. Clearly there is not an even distribution of each category of images generally found on the internet so it may be necessary to artificially balance training sets.

As an implementation detail, the sampling was part of the preprocessing stage that prepared all the data for training a neural network. Since the Siamese network draws a pair of images at a time, the pairs were stored contiguously. Specifically, as described in the implementation, the pairs get split and then their feature vectors are concatenated. To support this structure, the pairs of images were stored in LMDB as a single image with the original height and width, but with a depth of 6 channels instead of 3, basically stacking the two images on top of each other. Note that if any image occurs in multiple pairs, it will be stored in the LMDB multiple times. This redundancy is very costly, increasing the dataset size from 16GB to 800GB for this experiment. This was done because the size was still manageable and it provided for faster performance since there was no delay in retrieving the images. However, for larger dataset, there should be a more clever sparse representation of the data.

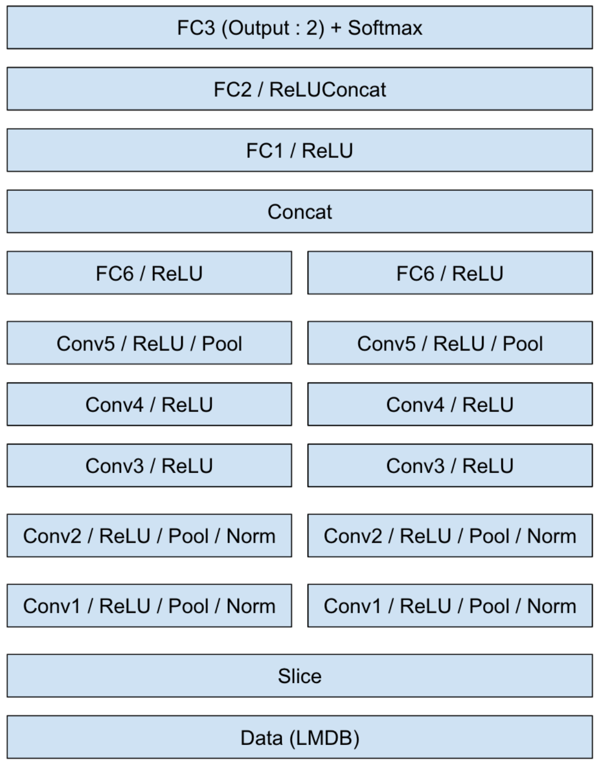


Figure 2: Architecture of the Siamese network inspired by MatchNet [1]. The split layer splits two images stacked on top of each other in the depth dimention and passes them into the two separate towers. The resulting vectors are concatenated.

6. Implementation

The experiments for this project were run on Caffe and used two network architectures. The basic classification task was run on CaffeNet [5] and a Siamese network was used for the pairs of similar images based on MatchNet. All the data was stored in LMDB and used Softmax for the loss layer.

The Siamese network is based on He *et al*. [1] and shown in Figure 2. The split section of the network which appears as two separate towers are themselves CaffeNet towers. There are two critical requirements for these towers. First, they must share weights. The way that the network learns about pairs of images at the same time is by actually running the pairs of images over the same set of weights at each layer. Second, each tower is itself exactly a CaffeNet. This allows using the weights learned from the Siamese network on the basic CaffeNet. The experiments cover in more detail how shared weights are used but the implementation required that the towers in the Siamese network and the CaffeNet had the same architecture and parameter names so that weights could be transferred between the two different architectures. Caffe provides a simple method of doing this. During training weights can be initialized with a *.caffemodel file and when training is complete the model can be saved to a *.caffemodel file that can be loaded into another instance of training or testing using the “weights” parameter.

The Siamese network is composed of a standard set of convolution, pool, ReLU and normalization layers. The unique aspect of the Siamese network is that when it loads an image, (which is actually 2 images stacked along the depth dimension as mentioned above), it must split the image and then recombine the features before the fully connected layers. Since the images are stored contiguously, the split layer simply splits the 6 channel image into two 3 channel images (which are in fact the original images). Each image is fed into one of the towers. The output of each tower is a 4,096 element vector representation of the image. These two vectors are simply concatenated and fed into a set of fully connected layers that are trained to convert the vector to scores for similarity and non-similarity. The final layer computes a Softmax loss over the predicted similarity versus the expected similarity as computed by the Cosine similarity over the text of the image.

7. Experiments Evaluation and Results

The goal of this project is to be able to learn visual features in an unsupervised system with noisy labels. This provides a challenge is evaluating the results because there isn't a clear ground truth to compare against. There are several tasks that can be evaluated using these features to compare to state of the art systems. [2] uses its learned representation in unsupervised object discovery and compares its results to Pascal VOC 2011 detection dataset. While there are many such tasks in the unsupervised learning space, to bound the difficulty of evaluating the results of this work, the first set of experiments will simply attempt to test if the weights learned in the Siamese network are a better initialization for fine-tuning an image classification task than the provided CaffeNet weights which were learned on ImageNet. The rest of this section will outline in detail how these experiments were built and how weights were transferred.

As a baseline, the stock CaffeNet model, using weights from Model Zoo that were learned on ImageNet, was fine tuned to classify images in MSCOCO based on 90 categories. After 50,000 iterations with a batch size of 50, the baseline network was able to reach 84% validation accuracy.

Next, the Siamese network was trained to classify pairs of images as similar or not similar. The towers of the network were initialized to the same CaffeNet weights as those used above in the baseline but the final fully connected layers were initialized randomly. The Siamese network was trained for 100,000 iteration with a batch size of 128 and reached 99.97% validation accuracy in classifying a pair of images as similar or not similar. The new weights learned during this training were stored for reuse in the evaluation experiment.

As the evaluation experiment, the weights from the Siamese network were used to initialize the same network as the baseline (in place of the CaffeNet weights), and the same baseline image classification task was evaluated. Note that the Siamese network weights are able to be loaded into the CaffeNet because each tower in the Siamese network is actually itself a CaffeNet.

These experiments operate on the assumption that transfer learning will adequately transfer useful image feature representations from one task to another. This is a significant assumption for two reasons. First, the larger scope goal is to show that using noisy labels on a large dataset can be used to learn a useful feature representation for arbitrary tasks and images. Second, this significantly decreases the resources and time required to train these networks. Rather than learning something like ImageNet which might take a week on great hardware, this project attempts to use those weights as the initialization and slightly tweak them using a novel training approach allowing the training dataset to be much smaller and the resources required to be much more limited.

Dataset	Weights	Validation Accuracy
MSCOCO	CaffeNet	84%
MSCOCO	Siamese Learned	77%
ImageNet subset	CaffeNet	96%
ImageNet subset	Siamese Learned	93%

Figure 3: Validation accuracies using baseline and learned weights

The hypothesis was that using newly learned weights from the Siamese netwokr would perform better at image classification than using the CaffeNet weights on the base MSCOCO dataset. The experimental results returned 77% validation accuracy after 50,000 iterations with a batch size of 50. With an identical setup and the same number of iterations, the resulting accuracy decreased by 7% suggesting that the hypothesis was false. The same experiment was run but using a subset of ImageNet as the baseline. The results were similar with validation accuracies of 96% for the baseline and 93% using the new weights. Note that the accuracies are higher for ImageNet than for MSCOCO because the networks were initialized with weights learned on ImageNet so they would clearly be more effective on that dataset. The discussion offers some intuitions as to why this may be the case and expands on some of the factors that contribute to the results that may offer some motivation for further experimentation.

8. Discussion

As shown in the results, the validation accuracy of the baselines were higher than the accuracies using the weights learned in the Siamese network. One intuition of why this may be the case can be found by taking a closer look at the example of the two surfing images in Figure 1. Notice that the images are both of a person surfing on a wave and the captions are very similar in describing this. In fact, there is a caption that is identical for both images. However, the labels for the two images are actually different, one being labeled as a “Person” and the other as a “Surfboard”. It may be the case that the weights actually did learn similarities between images but these similarities were not reflected in the labels. Reviewing the input data showed a large number of images classified as similar actually had different category labels in MSCOCO.

Another potential factor in the negative results was the scale of the data. While the Siamese network was trained on over 2 million image pairs and ran for over 16 hours on a GPU, current state of the art models are trained on much more hardware resources and are run for a week or longer. It may be the case that to have a significant impact on the image representation, the training time must be scaled up or perhaps done from scratch.

The motivation for this work's setup was to attempt to show a positive result on a small dataset with a short training time in hopes to save effort on a larger experiment. Since this work was not able to accomplish this, the measurement of success for this task needs to be reevaluated. It is likely still too large of an investment to build and run this experiment with the full Flickr 100M dataset distributed on many machines. An alternative approach would be to find tasks that could measure incremental improvements in image feature representation to show a trend of improving before scaling up, if possible.

9. Conclusion

This work attempted to enable training on larger and noisier datasets for image classification. It explored the space of using similar images based on text metadata with Siamese networks to learn better image feature representations. Ultimately, using similar images to learn feature representations did not improve image classification. While the captions used in these experiments were not very noisy, perhaps image classification was not the correct approach for evaluating the quality of the results because there wasn't a strong enough correlation between similar images and matching labels. Future work may include attempting other tasks such as unsupervised object detection using the new learned weights from these results.

Also this work took advantage of a small dataset to minimize system complexity. Nevertheless, even with a smaller dataset, sampling good distributions of similar pairs of images, computing their scores and storing them for efficient loading into Caffe provided to be challenging. This indicates that any further scaling would require a more disciplined approach to sampling and storing data. In particular, there is a necessity in removing redundant storage and balancing sampling across similarity and classes.

References

[1] X. Han, T. Leung, Y. Jia, R. Sukthankar and A. Berg. *MatchNet: Unifying Feature and Metric Learning for Patch-Based Matching*. Proceedings of Computer Vision and Pattern Recognition, 2015.

[2] C. Doersch, A. Gupta and A. Efros. *Unsupervised Visual Representation Learning by Context Prediction*. In ICCV'15.

[3] T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, P. Dollr. *Microsoft COCO: Common Objects in Context.* ECCV (5) 2014

[4] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, L. Li. *The New Data and New Challenges in Multimedia Research.* arXiv 2015

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell. *Caffe: Convolutional Architecture for Fast Feature Embedding* arXiv 2014

A. solver.prototxt

```
net: ``./train_val.prototxt"
test_iter: 100
test_interval: 1000
base_lr: 0.001
lr_policy: ``step"
gamma: 0.1
stepsize: 20000
display: 100
max_iter: 100000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
snapshot_prefix:``siamese"
solver_mode: GPU
```

B. train_val.prototxt

```
name: "SiameseNet"
layer {
  name: "data"
  type: "Data"
  top: "pair_data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    mirror: true
    crop_size: 227
    mean_file: "mscoco-mean.binaryproto"
  }
  data_param {
    source: "../mscoco_pairs_lmdb"
    batch_size: 128
    backend: LMDB
  }
}
layer {
  name: "data"
  type: "Data"
  top: "pair_data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
    crop_size: 227
    mean_file: "mscoco-mean.binaryproto"
  }
  data_param {
    source: "../mscoco_val_pairs_lmdb"
    batch_size: 64
    backend: LMDB
  }
}
layer {
  name: "slice_pair"
  type: "Slice"
  bottom: "pair_data"
  top: "data"
  top: "data_p"
  slice_param {
    slice_dim: 1
    slice_point: 3
  }
}

#1st siamese tower
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
   name: "conv1_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv1_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm1"
  type: "LRN"
  bottom: "pool1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "norm1"
  top: "conv2"
  param {
   name: "conv2_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv2_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm2"
  type: "LRN"
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "norm2"
  top: "conv3"
  param {
   name: "conv3_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv3_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  param {
   name: "conv4_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv4_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu4"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
}
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "conv4"
  top: "conv5"
  param {
   name: "conv5_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv5_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "conv5"
  top: "conv5"
}
layer {
  name: "pool5"
  type: "Pooling"
  bottom: "conv5"
  top: "pool5"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "tower_fc6"
  type: "InnerProduct"
  bottom: "pool5"
  top: "tower_fc6"
  param {
   name: "fc_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "fc_b"
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "tower_fc6"
  top: "tower_fc6"
}

#can also try with the dropout layer
#layer {
#  name: "drop6"
#  type: "Dropout"
#  bottom: "tower_fc6"
#  top: "tower_fc6"
#  dropout_param {
#    dropout_ratio: 0.5
#  }
#}


#2nd siamese tower
layer {
  name: "conv1_p"
  type: "Convolution"
  bottom: "data_p"
  top: "conv1_p"
  param {
   name: "conv1_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv1_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu1_p"
  type: "ReLU"
  bottom: "conv1_p"
  top: "conv1_p"
}
layer {
  name: "pool1_p"
  type: "Pooling"
  bottom: "conv1_p"
  top: "pool1_p"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm1_p"
  type: "LRN"
  bottom: "pool1_p"
  top: "norm1_p"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv2_p"
  type: "Convolution"
  bottom: "norm1_p"
  top: "conv2_p"
  param {
   name: "conv2_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv2_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu2_p"
  type: "ReLU"
  bottom: "conv2_p"
  top: "conv2_p"
}
layer {
  name: "pool2_p"
  type: "Pooling"
  bottom: "conv2_p"
  top: "pool2_p"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "norm2_p"
  type: "LRN"
  bottom: "pool2_p"
  top: "norm2_p"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv3_p"
  type: "Convolution"
  bottom: "norm2_p"
  top: "conv3_p"
  param {
   name: "conv3_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv3_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layer {
  name: "relu3_p"
  type: "ReLU"
  bottom: "conv3_p"
  top: "conv3_p"
}
layer {
  name: "conv4_p"
  type: "Convolution"
  bottom: "conv3_p"
  top: "conv4_p"
  param {
   name: "conv4_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv4_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
```

```
layer {
  name: "relu4_p"
  type: "ReLU"
  bottom: "conv4_p"
  top: "conv4_p"
}
layer {
  name: "conv5_p"
  type: "Convolution"
  bottom: "conv4_p"
  top: "conv5_p"
  param {
   name: "conv5_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "conv5_b"
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu5_p"
  type: "ReLU"
  bottom: "conv5_p"
  top: "conv5_p"
}
layer {
  name: "pool5_p"
  type: "Pooling"
  bottom: "conv5_p"
  top: "pool5_p"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "tower_fc6_p"
  type: "InnerProduct"
  bottom: "pool5_p"
  top: "tower_fc6_p"
  param {
   name: "fc_w"
    lr_mult: 1
    decay_mult: 1
  }
  param {
   name: "fc_b"
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "relu6_p"
  type: "ReLU"
  bottom: "tower_fc6_p"
  top: "tower_fc6_p"
}

#can also try with the dropout layer
#layer {
#  name: "drop6_p"
#  type: "Dropout"
#  bottom: "tower_fc6_p"
#  top: "tower_fc6_p"
#  dropout_param {
#    dropout_ratio: 0.5
#  }
#}


#feature vector classification network
layer {
  name: "concat"
  bottom: "tower_fc6"
  bottom: "tower_fc6_p"
  top: "concat"
  type: "Concat"
  concat_param {
    axis: 1
  }
}

layer {
  bottom: "concat"
  top: "feature_fc1"
  name: "feature_fc1"
  type: "InnerProduct"
  inner_product_param {
    weight_filler {
        type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0
    }
    num_output: 4096
  }
}
layer {
  bottom: "feature_fc1"
  top: "feature_fc1"
  name: "feature_fc1_relu"
  type: "ReLU"
}
layer {
  bottom: "feature_fc1"
  top: "feature_fc2"
  name: "feature_fc2"
  type: "InnerProduct"
  inner_product_param {
    weight_filler {
        type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0
    }
    num_output: 4096
  }
}
layer {
  bottom: "feature_fc2"
  top: "feature_fc2"
  name: "feature_fc2_relu"
  type: "ReLU"
}
layer {
  bottom: "feature_fc2"
  top: "feature_fc3"
  name: "feature_fc3"
  type: "InnerProduct"
  inner_product_param {
    weight_filler {
        type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0
    }
    num_output: 2
  }
}

layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "feature_fc3"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "feature_fc3"
  bottom: "label"
  top: "loss"
}
```