Heirarchical feature Learning in deep convolutional networks

December 11, 2019

Abstract

(100-200 words) Convolutional networks are able to learn abstract features in an image. However, the features that the network learns is still not known very well. We attempt to explain feature learning by explicitly training the network on input features. In a novel approach, called patch-training, we generate multiple patches of the region-of-interest and modify well-known architectures like VGG and ResNet to accomodate learning of these new features. This experiment shows that learning on the patches indeed improves the recall rate for the multi-label classification task with significant reduction in training time due to reduced feature space.

1 Introduction

(0.5-1 pg)

Deep Convulutional Neural networks (DCNNs) have state-of-the-art performance in image classification and object detection tasks. Classically, when training a neural network, we present the network with a sample of the dataset, usually in a random order. The model performs a forward pass on this sub-sampled data, subsequently a gradient is computed and propagated in the backward pass based on a well-defined criterion.

However, the decision-making process of Deep Neural Networks are still largely unclear and non-deterministic.

A neural network model is essentially a universal function approximator, that tries to model the intricate dependencies between the input features and output labels. In the case of image tasks, such as object detection, classification and segmentation, we see that the output features are influenced by contiguous regions of input pixels, by plotting the saliency map. That is, features at higher levels are composed of lower level features.

This phenomena is further explored in recent research. By visualizing the feature representations in the hidden layers, we understand that they are indeed learning progressively higher level features. As a concrete example, the output activations of the first few layers react to various types of edges, while deeper layers are sensitive to faces and body shapes.

Humans learn better when examples are presented in a meaningful order according to Bengio et al. Bengio proposes a technique knows as Curriculum Learning, which gives better results on the speed of convergence as well as obtaining a better local optima.

2 Background/Related Work

(1-2 pgs)

Curriculum Learning

Imposing a curriculum is a method to guide the training of neural networks. As children, we learn from a fixed curriculum, where concepts are introduced to us from easy to difficult. Each new concept builds on prior concepts to help us grasp them. (Bengio) hypothesizes choosing the examples and their order, not only leads to faster convergence but also better quality of local minima. This strongly suggests that the choice of training strategy has a greater role to play than previously thought

Similar training procedures have been applied to Natural Language processing tasks (TODO), where the model was initially trained on basic language syntax. Multi-stage curriculum learning have been shown to give improved generalization in vision and language tasks.

We develop on this idea, to enable a model to learn the rudimentary features of the region of interest explicitly

rather than having to learn them implicity.

Restricted Boltzmann machines

Boltzmann machines are a class of stochastic and generative neural networks, introducted by (Hinton). Restricted Boltzmann Machines (RBMs) imposes an additional constraint that there are no connections between units of the same layer. Essentially, they are two-layered neural networks with no output layer, allowing for more efficient training algorithms. A stack of RBMs allows us to learn more complex relations between inputs and outputs. The training procedure is similar multi-stage hierarchical learning.

Spatial Pyramid Pooling

The inputs to a neural network models are usually inputs of fixed square sizes (eg 224 x 224). The square sizes of image also limits aspect ratio. Generally, this is achieved either center cropping the image or resizing the image till the desired shape is achieved (through bilinear interpolation). Doing so, we lose important spatial information, either through downsampling the image or cropping, compromising the recognition accuracy. Convolutional architectures comprise of two parts, the convolutional layers and dense layers deeper in the model. Since convolutional layers behave like a sliding window protocol, they have no restriction on the input size. This restriction is imposed by the fully connected layers. (Kaiming He et al) proposes a Spatial Pyramid Pooling method to generate a fixed-size representation of features, regardless of input size. The procedure is as follows, the Spatial Pyramid Pooling is inserted after the final convolutional layers. Pooling layers (MaxPool) of different kernel sizes (1, 3, 5, etc) are applied on the intermediate activations. The results of each of the pooling operations are flattened into a 1-d vector and concatenated (end-to-end), to yield a fixed-sized vector (Figure). The intuition is that, the different-sized filters capture features heirarchies at different scales.

(Liang-Chieh Chen et al) builds on top of this architecture by using dilated (or atrous) convolutions over pooling operations. Like the SPP module, after the last convolutional layer, they use multiple filters of the same kernel sizes (3), but with different dilation rates. The results are concatenated and a 1x1 convoultion is applied. This effectively increases the receptive field, capturing larger spatial dependencies, and does not destroy context like the pooling operations.

We use the pyramid pooling to account for the different patch sizes to obtain a fixed sized representation.

Bag-of-words

Bag-of-words is a technique used in NLP

3 Datasets

(0.5-1 pgs)

Learning features of inputs necessiates a high-quality dataset. We explore COCO and Pascal VOC.

The Pascal VOC has 11k examples, split into roughly 5.5k train set and 5.5k test set.

It comprises of 20 classes: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tv

There can be multiple objects in each image, making this a multi-label classification task.

For model training we use two datasets — `FullSet` comprises of the full resolution train set.

The `PatchSet` is derived as follows. Using the bounding boxes of the images, we derive a new dataset comprising of the extracted objects. From 5k train images, we obtain 15k patch dataset.

To ensure that each patch has adequately representable features, we filter objects of smaller patch sizes. In our initial expreriments, we only consider patch sizes of more than $64px$ resolutions.

For both the datasets, we convert the PIL Images to Tensors and normalize the distribution using the ImageNet mean and standard deviation.

The PatchSet contains data with only a single label (single-label classification task).

4 Methodology

(1-6 pgs)

The models were built using `PyTorch 1.3`, and trained on Google Cloud AI Platform on an NVIDIA Tesla K80 GPU (1GB). Input images are represented in PIL (Python Imaging Library) format, using the Pillow Library. The `torchvision` package consists of popular datasets, model architectures, and common image transformations for computer vision. We primarily use the Normalize, RandomResizedCrop, ToTensor transforms

from the torchvision package. Further, torchvision provides a wrapper over the Pascal VOC dataset. We choose the dataset corresponding to the 2012 (latest) detection challenge.

We implement the VGG-16 architecture as a baseline model. VGG is chosen for its simplicity, since it doesn't have skip connections. We explore other architectures further. VGG-16 comprises of 6 blocks.

In each of the first 5 blocks, there are 2-3 convolutional layers of $kernel\_size = 3$, $stride = 1$ and $padding = 1$. The padding ensures that the spatial dimensions are intact through the Convolutional layers. Each Convolutional layer is followed by a ReLU nonlinearity, and finally a MaxPool layer with $kernel_size = 2$ and $stride = 2$. After the convolutional layers, VGG adds a `AdaptiveAvgPool` layer, to ensure that the output size will adhere to 7x7 The last block comprises of 3 Fully Connected (FC) layers (4096, 4096, 20, respectively), with ReLU and Dropout inserted between.

We modify the VGG-16 architecture by adding an additional branch (control flow). We diverge post the convolutional layer, adding a `AdaptiveAvgPool` layer of output size (4x4). We also add a new classifier head, consisting of 3 Fully Connected (FC layers) (Figure).

We are essentially merging two different tasks — single-label and multi-label classification into one. Therefore, we define two loss functions for each of them. It is helpful to note that, we only forward pass through one branch at a given epoch, and compute the gradient for that branch only. To simplify training, we define a $switch_epoch$. Training when the $epoch < swtich_epoch$ is done on the `PatchSet`, while training is done on the `FullSet` otherwise.

TODO - Training Algorithm

The patch-training uses cross entropy loss, `nn.CrossEntropyLoss`, defined as $loss(x, class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right)$ The full-training uses binary cross entropy (`nn.BCEWithLogitsLoss`, which posits the existence of a given class.

It is defined as $loss(x, y) = -y_n \log(\sigma(x_n)) - (1 - y_n)\log(1 - \sigma(x_n))$

$nn.BCEWithLogitsLoss$ generalizes to the multi-label case, by keeping the output and input dimensions the same.

The output from a forward pass of the model is a 20-d Tensor giving logits for each class. For the PatchSet data, we one-hot encode the labels and use cross entropy loss. For the FullSet data, we multi-hot encode the labels and sum over the binary cross entropy loss over each class.

The Pascal VOC dataset is inherently imbalanced (as seen in the figure). The 'person', 'chair' and 'car' classes are present in higher proportion as compared to other classes. Moreover, the multi-hot encoded labels results in a many more negative examples compared to positive examples.

From our expreriments with training the model as is, the model pushes all predicions to 0, (class absent in the image). Clearly, this leads to lowest average loss across all datapoints. However this generalizes poorly and the predictions made are not useful. To accomodate for this imbalance, we assign higher weights to positive samples. These weights are calculated based on the distribution of the training data.

The loss value is not an accurate indicator of the model performance, since the loss for training on patches is significantly different from the model trained on full images. Canonically, for classification we use precision and recall metric.

$precision = \frac{TP}{TP+FP}$

$recall = \frac{TP}{TP+FN}$

where TP are the number of True Positives, FP are False Positives and FN are False Negatives.

5 Experiments

(2-3 pgs)

We compute the `PatchSet`, which involves extracting the pixels from the bounding box, converting them to tensors and normalizing them. We save them as tensors and for all further expreriments we create the dataset using torchvision's `TensorDataset`.

In all experiments, we train the model for 10 epochs. Further, we use a $swtich_epoch \in [0, 10]$. $swtich_epoch = 0$, corresponds to the model being trained completely on the $FullSet$. This serves as the baseline model to compare to.

Since this is a multi-label classification, we compute the Confusion Matrix for each class (20 x 2 x 2) on the validation set after training for each switch epoch.

The complete model is trained on a single GPU, while the metrics are computed on the CPU. We use `sklearn.metrics.multilabel_confusion_matrix` for the Confusion matrix computation.

To calculate a single metric, we sum the confusion matrix over all classes to yield a $2x2$ matrix. Next, we compute the precision and recall for this using the given formula. We plot the precision and recall rates as a function of $swtich_epoch$.

From the plot, we see that the recall rate improves with $swtich_epoch$. There is 3% increase in the recall rate (from 0.5 to 0.53) over all classes. However, there is a decline in the precision, a 1% decrease over all classes. One seemingly obvious explaination is that, since the model is trained on the patches, which are the positive examples, they have less exposure to the negative examples (one where the class is not present). This causes the False Positives and True Positives to increase proportionally. This results in the precison rate $\alpha \frac{1}{FP}$ to reduce.

The same explaination can be extended to the $recall$ rate. The higher training on positive examples, makes the True Negatives and False Negatives to reduce. So the $recall$ rate imporoves $\alpha \frac{1}{FN}$.

Notice however, that the percentage improvement or degradation in the recall and precision is not the same. The recall rate improves significantly more. The choice of $switch_epoch$ also plays a huge factor in the performance of the model. Our model, has a sharp decline in the $precision$ rate after $switch_epoch = 5$. By simple cost-benefit analysis, this $switch_epoch$ provides the best trade-off between the metrics. $+2\%$ recall rate and -0.4% precision.

Hyper-parameter tuning

We perform an ablation study on the model. The initial hyper-parameters were chosen from the set that gave the best performance on the baseline model (VGG-16) We use the `Stochastic Gradient Descent` optimizer with an initial learning rate of $1e-3$ and $momentum = 0.9$

We train our model on different learning rates to verify if convergence is faster or a better quality of local optima is obtained. From the experiments (figure), we see that a $learningrate < 0.001$ provides roughly the same result, and higher learning rates give poor generalization.

TODO Learning rate table.

Batch sizes:

From relevant work, we learn that a larger batch size gives a marginal improvement over performance metrics, while significantly increasing training time. We choose an intial $batchsize = 16$ and compare its performance to $batch_size = 8, 32$.

TODO Batch size

6 Results

(2-5 pgs) Confusion matrix (Across classes) Train time

7 Conclusion

(0.5-1) Better Recall, Faster training. Lower Precision.

8 Future

[?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?] [?]

References

[1] Artem Babenko and Victor S. Lempitsky. Aggregating deep convolutional features for image retrieval. *ArXiv*, abs/1510.07493, 2015.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. ACM.

[3] Wieland Brendel and Matthias Bethge. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. *ArXiv*, abs/1904.00760, 2019.

[4] Chaofan Chen, Oscar Li, Alina Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: deep learning for interpretable image recognition. *CoRR*, abs/1806.10574, 2018.

[5] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[7] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[8] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels, 2017.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[10] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2:2169–2178, 2006.

[11] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[12] Eva Mohedano, Amaia Salvador, Kevin McGuinness, Ferran Marqués, Noel E. O'Connor, and Xavier Giró. Bags of local convolutional features for scalable instance search. In *ICMR '16*, 2016.

[13] Stéfan van der Walt, Johannes L. Schonberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.