

Extending Convolution into the Z-Dimension

Experimenting with Uniting Convolution over Scale and Multi-Scale Features

A CS231A Project Report

Christopher Sauer
Stanford University
450 Serra Mall, Stanford, CA 94305
cpsauer@stanford.edu

Abstract

Convolutional neural networks represent the state of the art in a wide variety of computer vision tasks, from image classification, to object recognition, to image captioning. The convolutional architecture succeeds because it captures the translationally invariant structure of images in the structure of the computational graph, freeing up the network to learn the task at hand, rather than first having to also learn this structure from the data.

Unfortunately, the same connection restrictions that allow traditional convolutional networks to better learn translationally invariant structure, severely hamstringing them in their ability to handle two other key structural aspects of images from a camera: feature scale invariance and composition of multiple scales—or at least not without requiring dramatically increasing depth which is problematic for learning.

In this project, I explore a novel means of extending convolution such that it cleanly and readily capture those two structural aspects together. The method amounts to extending filters into an additional dimension—that of scale—and can be described as convolving over a pyramid shaped input tensor. Like traditional convolution, the resulting architecture is not dependent on the size of the input image, heavily shares parameters, and, in my tests, provides better performance with fewer parameters than larger networks that take far longer to train. It lays the foundation for future thesis work exploring this method as a replacement for traditional convolution.

1. Introduction

With this project, I investigate a new architectural idea for extending convolutional neural networks to better handle the structure inherent to real-world images taken by a perspective camera. In particular, the structural idea works to allow features to be composed of sub-features at multiple relative scales, without fixing the absolute scale of the feature. Since a strength of deep learning is its ability to learn higher level features from data, rather than require them to be hand designed, it follows that the network structure should be amenable to the types of features we would expect to be relevant in real-world, perspective images. Here, I will argue that current, spatial convolutional architectures are inherently limited in their handling of scale and propose a promising alternative that addresses these limitations.

The idea hit me in class while I was considering the reasons for the success of traditional spatial convolution in terms of the camera's position in world space. Convolution's efficacy is usually explained purely in terms of processing on an image, but there are considerable insights to be gained by taking a CS231A approach rather than a purely CS231N one and thinking instead about convolution's interpretation in world space: running filters over the projection of real objects onto the image plane. Excited by the idea and the insight of thinking in world-terms, I decided to begin exploring the idea as my class project. The project as proposed and executed aims to prospect the idea for potential continued work as part of my thesis next year. Hence my pursuing this as a solo project.

The structure of this report, then, will follow a similar, though more streamlined, path to the one that led to the idea and its testing. In the Problem Statement section, I be-

gin by analyzing convolution in world terms. This leads to finding that traditional convolutions does not lend itself to capturing certain aspects we would expect of data from objects projected onto an image plane.

I follow this with the Technical Approach section, which proposes my solution to the weaknesses identified in spatial convolution. Hopefully, the solution will feel natural and obvious given the explanation of the problem, though coming up with its formulation was actually a very substantial portion of the thought involved in this project. Non-obvious ideas usually seem obvious in hindsight.

Rather than explain the two approaches that were rejected in favor of the current, I will instead review the literature around building features that apply at multiple scales and that are composed of sub-features at multiple scales. As opposed to my approach, these two problems are usually seen as separate problems elsewhere. This gives equivalent insight into other promising areas and potential pitfalls.

Finally, I describe the experimental setup and the results found so far in testing the new architecture.

Having explained that this report and project are a beginning exploration of a substantial new idea, stemming from those in CS231A, I will close this section by noting what the project is not. This project is not about applying a set of established techniques to an existing problem and comparing their efficacy. Nor is it about reimplementing a published paper or iterating on it. Instead, it is a partly theoretical project and an exploration of that new theoretical territory, a prospect to see if the vein discovered is likely to be a valuable one. Exploring everything in full is a thesis length project, not a one-class individual project, as is hopefully clear from the Experimental Setup and Results section. The rest will—and was intended to be—explored next year.

I will now explain the theory, the intuition, and the experimental validation of the idea's promise.

2. Problem Statement

Let us begin, then, by exploring and explaining the effectiveness of convolutional neural networks in world terms. At the same time, we will explore the relationship between parameter sharing and data augmentation, both for its own sake and as a means of arguing for the importance of building the structure associated with scale into network architecture.

The benefits of building structure into the network to accommodate the properties of a projective camera might already be familiar to the reader from Spatial Transformer Networks, which, among other things, build naturally into a network the inverse of projecting a planar region. Baking in this 3D structure lead to practically perfect performance on sign recognition, which is an example of a task that involves the projection of planar regions.[8]

For the purposes of this report, however, we will start

from first principles, building up a justification for convolving filters from our intuitions about world space.

2.1. Convolution in World Space: The Power of Parameter Sharing

The core insight of convolutional neural networks is that the structure of images is largely translationally invariant. This is true for a deeper reason than simply stating that 2D images tend to have shared structure throughout: The position of the camera relative to the world and the objects of interest in it is somewhat arbitrary. Further, similar patterns, i.e. features worth learning, appear in different objects because the natural and man-made objects we photograph fit into categories with substantial similarities. Because position in the input image matrix is a projection of real world position, this is true in the image as well.

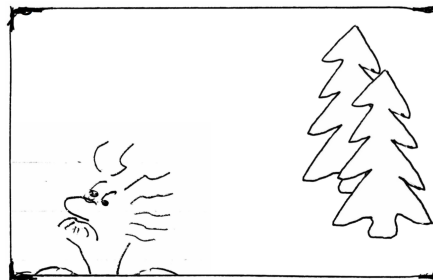
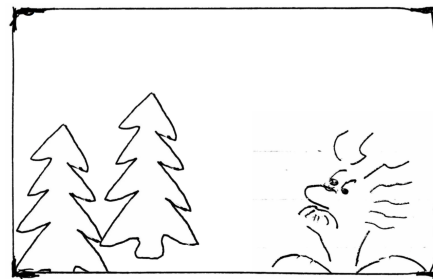


Figure 1. Cartoon intuition behind translational invariance in 2D. Because images are a projection of real world objects, which can be arranged arbitrarily, the same features that are useful in one image are useful at different position when the scene is arranged differently or when the camera is at a different position relative to the same scene.

Of course, objects of interest are more likely to be in the center of the frame, but nonetheless the similarity in structure across the image allows for a dramatic sharing of parameters that enables networks to capture the structure of images with far fewer training examples. From the perspective of each convolutional filter, this is akin to massive data augmentation whereby the filter effectively sees many more images than it would otherwise, each being a related sub-region of a smaller number of original images. At the

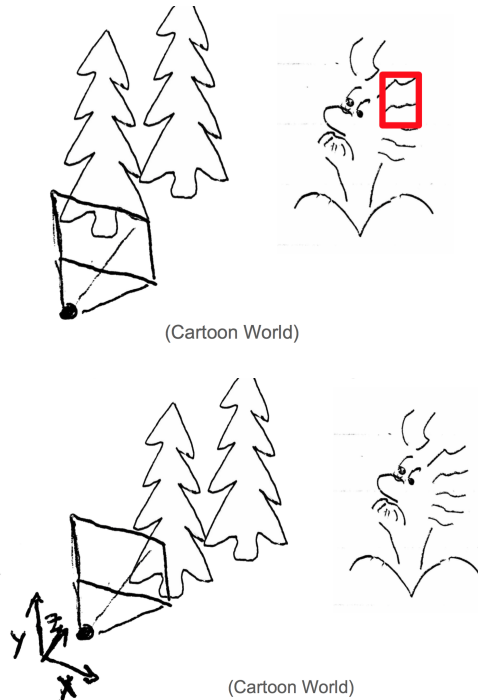


Figure 2. The same scene as in Figure 1 represented in world space. The camera is in the lower left, and the red box represents a the area corresponding to a hair-texture filter of use in both images. Note that this filter is useful in a different position in image space when the camera is translated along its x or y axes. This is addressed by spatial convolution over the image. But what about the arbitrariness of the camera's z -position?

level of the network, rather than the parameters, however, we can view it instead as sharing parameters with shared intermediate computation, allowing for fast, efficient training. Parameter sharing is thus like data augmentation in that it allows us to impart our understanding of the structure of the problem, but it is to be preferred because it allows us to share computation.

2.1.1 Z-Dimension Camera Translation

From this world-frame perspective it is hard to not be struck by the fact that there is a third axis being ignored. This leads to another type of invariance over which we would like to capture similarities in structure: scale. Camera position is not only largely arbitrary in the direction of the image plane; closeness, i.e., z -position, of the camera is arbitrary as well. For most objects, which have relatively small depth relative to the scene, changing the distance between the object and the camera manifests itself as simple scaling in the image. We know that we want our networks to capture scale invariance at the level of their output layer, e.g., we would like a ship to be recognized as such whether it takes up an eighth

of the frame or fills the image. Since sub-features scale with the image, we ought to strive for this also at every intermediate layer.

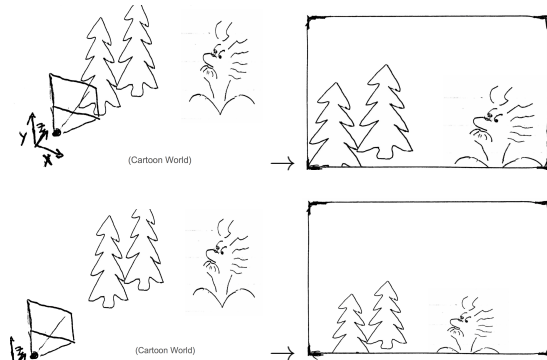


Figure 3. z -translation mostly manifests itself as changes in object scale.

We know that networks with such scale invariant properties perform better than those without, because training of modern networks explicitly involves enforcing this property with enormous, scale-based data augmentation. One of the earlier notes about this was in the All Convolutional Network paper, which notes the large accuracy gains from large-scale, scale-based data augmentation.[10] Similarly, one of the oft copied items from the paper outlining Residual Networks is their scale-smoothing data augmentation scheme, in addition to their new network architecture.[7]

2.1.2 Problems in Purely Spatial Convolution Capturing Scale

Said differently, if we would like our network's neurons to capture scale-invariant concepts, including output neurons, that build on one another in complexity, we ought to have a structure in which that relationship can be easily represented. Currently, our structure, with its reliance on small filters, ties layer depth heavily to receptive field size, meaning that only higher-layer neurons can capture larger concepts. However, the larger concepts are not necessarily more complex ones, and vice versa. We would like our network to be able to capture lower level concepts in lower layers and higher ones in higher layers, without instead having to skip many layers to build up a larger effective receptive field. One cannot help but wonder if this contributes to the great need for depth—which, among other things, allows us to represent functions not capturable in individual convolution layers—and to the effectiveness of skip connections in residual networks and highway networks.[7] [11] And while depth is often regarded with pride, we should remember that it comes at great cost to training time because of vanishing gradients among other things. Far better if we

could enable convolutional neural networks to capture scale without needing huge depth.

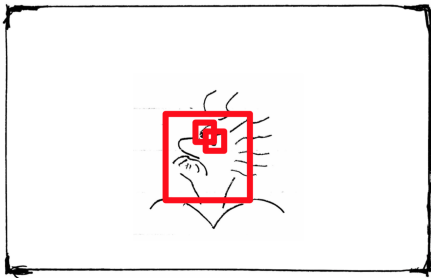


Figure 4. Features may be naturally composed of sub features of radically different sizes. Simple, “low-level” features are not necessarily small, nor do they necessarily fit in the receptive fields of low layer neurons. Red squares again represent filter features.

There is another scale-related concept we’d like our networks to capture, which is drawing on features at multiple scales. Consider how you might describe your process for recognizing a face. I might describe it in terms of an elastic collection of sub-features: a round head, hair texture, eyes, eyebrows, nose, and mouth. People have sometimes described CNNs as related to deformable part models, but we should expect the structure that I have described to be quite hard to represent with standard convolution layers.[5] Note that the aspects of the hypothetical face are radically differently sized, especially when you consider the round head compared to, say, an eyebrow. The need to build features out of multi-scale sub-features is explicitly part of the inspiration behind Google’s Inception Module, which does this by creating filters of different sizes with different parameters.[12]

To summarize where we are so far: Parameter sharing in convolutional neural networks works fundamentally not just because structure is shared across the image, but because the camera’s position is somewhat arbitrary in a similarly-structured world. This suggests we might also want to convolve over the third translation dimension of the camera, which manifests itself as the scale of objects in the image. We would like our networks to be able to capture this shared structure at whatever size it appears without blowing up the number of parameters, and to compose features out of sub-features of different sizes.

Instead of trying to solve these problems separately, I have been working on extending convolution into the camera’s z-dimension by expanding filters into a fourth dimension, over scale, and by sliding those filters over another dimension, up a scale pyramid. This approach should create scale-invariant layers capable of efficiently detecting multi-scale features with relatively few additional parameters. I will explain the structure in more detail in the following section.

3. Technical Approach

What follows are the technical details for the plan on convolution over scale and how I am breaking the problem down into two quasi-independent parts, with the first being the testable checkpoint investigated here before the second builds on it. This allows me to validate and explore assumptions early.

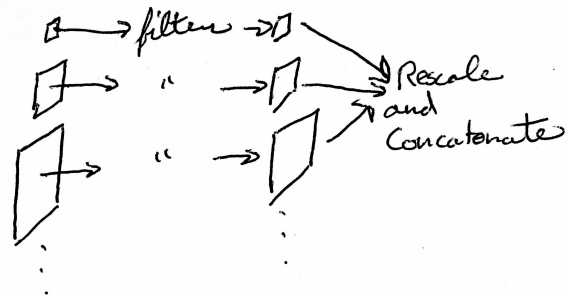


Figure 5. Extending filters up the pyramid into the scale dimension.

In the first step toward building scale convolution into the network, I’m working on expanding filters into the scale dimension but not sliding them linearly up the pyramid. This gives multi-scale features, and is equivalent to adding a scale-pyramid dimension to the convolutional filters. To simplify and be able to build this functionality out of standard scaling and convolution operations, I reformulate the addition of the scale dimension as appending to the channel dimension. This avoids introducing hard-to-visualize 4D tensors. To get a sufficiently tall scale pyramid, I need images where 3x3 filters are still sensible at 1/4th or 1/8th scale for this. I chose Tiny ImageNet, which is a test dataset made by scaling the ImageNet Classification and Localization dataset to 64x64 and reducing to 100,000 images for 200 classes. [9][3]. This dataset is a good choice, given it is the smallest scale that offers sufficient resolution, its objects are more heavily deformed and scaled, and it is not an easily solved task.

The second step is to begin sliding the filter over the scale dimension, with the intent of getting scale-invariant, multi-scale features (e.g., capture that a face is composed of elements of multiple sizes, a face can be at multiple scales, and that what matters is the relative scale and position between elements). Here I need a dataset that still meaningful to convolve over when scaled down 16x or more. This means having an original resolution of 128x128 or more. Therefore, I need to use either ImageNet or, if necessary for speed or experimentation, create a custom Tiny ImageNet that is 128x128.

As an aside, we should note that the full approach here is equivalent to converting the input image into a pyramid-

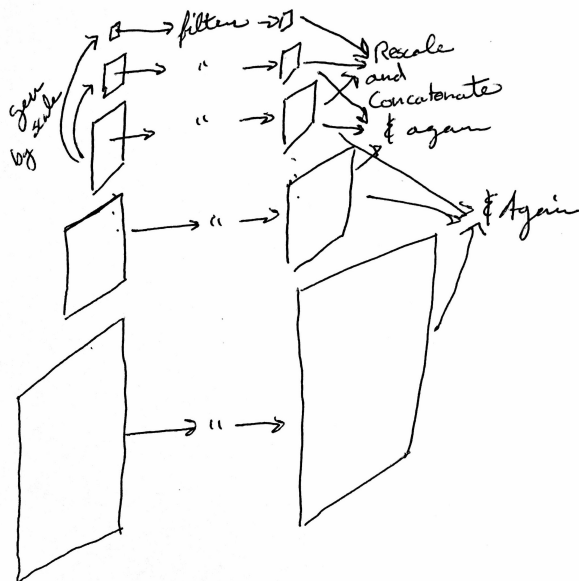


Figure 6. Sliding scale-convolution filters up the scale pyramid.

shaped tensor and convolving pyramid shaped filters over that. Imagine pushing the channels into a non-drawn 4th dimension, and instead drawing the scale pyramid as a solid. The first of the two steps above effectively adds the pyramid height dimension to our filters, which makes them smaller pyramids being slid laterally across the image pyramid. With the second step, the start being slid up and down they pyramid as well. Further, drawing a pyramid shaped tensor yields a figure strikingly similar to that of a three dimensional pinhole camera, and that is exactly what we would expect given our intuition behind the z -dimension manifesting as scale. The tensor's pyramid dimension is an approximation of what the image would have looked like for different positions of the image plane.



Figure 7. Pyramid shaped tensor has an interpretation as movement of the image plane.

4. Literature Review

There have been some previous attempts at the scale-invariant network and multi-scale feature problems separately. They achieve enough success in each to validate

that there is promise, but neither see the shared structure between the two, nor attempt to solve both together. When previous approaches do involve parameter sharing at all, the attempts tend to use an extremely computationally inefficient filter dilation approach. We will cover them briefly here:

On the scale-invariant network side, Farabet, et al, have an example of increased accuracy for scene parsing which got performance gains by running their entire network in parallel on each section of a pyramid.[4] This makes sense, because you'd expect the problem of objects appearing at different sizes to be exacerbated in a scene. It also validates that there's value in scale-invariance, and is similar to my approach if you cut the filter size in the z dimension to 1.

For multi-scale features, formulations either rely on filter dilation or on assembling independent filters of different sizes. Dilation increases the image size rather than reducing the image scale. [14] [13] This is extremely problematic for computational reasons, because not only do the spatial dimensions not shrink higher up the pyramid, but the larger filters demand more computation with their size squared. Thus, as the height of the scale pyramid increases, a sum of squares term emerges, leading to an $O(numScale^3)$ runtime increase, even without extending each filter into multiple scales. By contrast, the exponential decrease convolution approach I describe should lead to a $O(numScale)$ runtime increase, but as we will see, this is offset by needing fewer distinct filters. Of course, Inception modules provide a tested way of capturing features at different scales, but Inception does not provide for scale invariance, nor does it share parameters between the filter sizes.[12]

5. Experimental Setup and Results

5.1. Ablation Testing











The general approach I take in evaluating my approach for extending the filter dimension is termed ablation studies. Ablation studies involve running incrementally changed models against one another to determine the marginal effect, and this approach led Facebook researchers to quickly find substantial improvements to Microsoft Research's Residual Net architecture.[6] Usually one does so on the smallest datasets that still present a meaningful challenge to the architecture, which I do here with Tiny ImageNet.

5.2. Requisite Hardware

Successfully training and iterating quickly on these datasets requires working with a powerful GPU and in a library suited to machine learning experimentation. Of those two, the GPU is by far the harder part, since virtualized GPUs are unfortunately essentially crippled. In my benchmarks, I found I can train models about as fast on my four-

year-old laptop’s GPU as on the most powerful AWS GPU instance. As mentioned in the proposal, one of the first orders of business, then, was to build myself a more powerful machine. I can justify the upfront cost because I’ll continue to use it for my thesis work next year group, as well as for VR tasks—I’ll be at Oculus this summer.

I picked out the parts at below left, assembled them. After a few dozen hours of fiddling, I built a very stable system pictured in the figure below.

CPU	 Intel Core i7-6700K 4.0GHz Quad-Core Processor	\$343.99
CPU Cooler	 Zalman CNPS8000B CPU Cooler	\$34.99
Motherboard	 Gigabyte GA-Z170X-UD5 TH ATX LGA1151 Motherboard	\$189.99
Memory	 Crucial Ballistix Sport LT 32GB (4 x 8GB) DDR4-2400 Memory	\$123.99
	Add Additional Memory	
Storage	 Samsung 850 EVO-Series 250GB 2.5" Solid State Drive	\$85.79
	 Samsung 950 PRO 512GB M.2-2280 Solid State Drive	\$317.00
	 Seagate Barracuda 2TB 3.5" 7200RPM Internal Hard Drive	\$65.88
	Add Additional Storage	
Video Card	 EVGA GeForce GTX 980 Ti 6GB FTW ACX 2.0+ Video Card	\$614.22
	Add Another Video Card For 2-Way SLI	
Case	 NZXT H440 (Matte Black) ATX Mid Tower Case	\$99.99
Power Supply	 Corsair RM 850W 80+ Gold Certified Fully-Modular ATX Power Supply	\$129.99

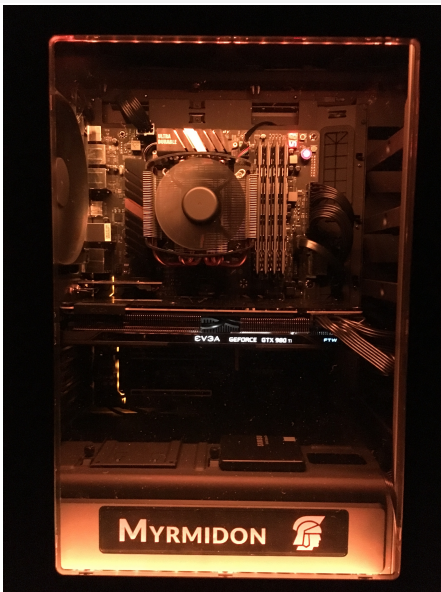


Figure 8. Deep learning machine built for the purposes of the project. The GPU is a NVIDIA GXT 980 Ti.

5.3. Framework and Writing Infrastructure

For the framework, I picked out Torch7 and CUDNN, so I applied for NVIDIA developer access and installed

both.[2] This is the fastest current setup and is great for experimentation, though TensorFlow is finally catching up, and Nirvana’s Neon remains a close second. [1] Unfortunately this means that writing the code requires installing both those libraries, so it will not run on Corn. This was cleared with Professor Savarese ahead of time.

If you do attempt to run the code, first install Torch7, and then I highly recommend placing the CUDNN libraries in Torch’s include folder rather than the system install NVIDIA recommends. Otherwise, Torch has trouble finding the libraries, especially on OS X.

What I did not fully appreciate when I chose Torch was just how rough of a language Lua is. Torch is an incredible package for pushing computational graphs through the GPU, but the training and dataset loading infrastructure required a large amount of effort to build from scratch in Lua, since libraries you would expect in every language—like string split, for example—are not present. All in all, I ended up writing more than 1500 lines of Lua to get a training suite set up. Existing libraries on the web, especially for ImageNet are often slow, and their means of multithreaded data loading executes a half dozen copies before finally feeding the data to the network. Instead, I wrote my own multithreaded data loader, extensible dataset reader, and model trainer that allowed me to quickly train one model after another. A periodic check-pointer that I wrote allows models to be freeze-dried to disk and resumed as necessary. This large amount of infrastructure code will likely serve me well in the future, but required quite a lot of up front time to write as an individual. I will try to release components of it on GitHub in the near future to speed others in their work.

For speed of prototyping, I implemented the extension of filters into the scale dimension using Torch table layers. This will be discussed more in the Comparison Results subsection.

5.4. Initial Training Difficulties: In Both Senses of the Term

I was surprised to find that, by default, the current version of Torch7 still uses Xavier initialization. This completely stalled training until I discovered it and replaced it with MSR initialization designed for ReLU units.

By contrast, other data preprocessing like mean subtraction and normalization that are usually considered of the utmost importance for training convergence led to little improvement. Eliminating these unnecessary steps will improve training infrastructure in the future.

5.5. Comparison Results

After significant experimentation comparing my step one models against traditional spatial convolutional networks, I found repeatedly that by replacing standard convolutional layers with those described I could train networks with the

same accuracies but more quickly and with many fewer parameters. Rather than describe one model after another, I will focus on a single comparison as a case study.

As an overview, I provide a picture of the computational graphs used in the head to head comparison.

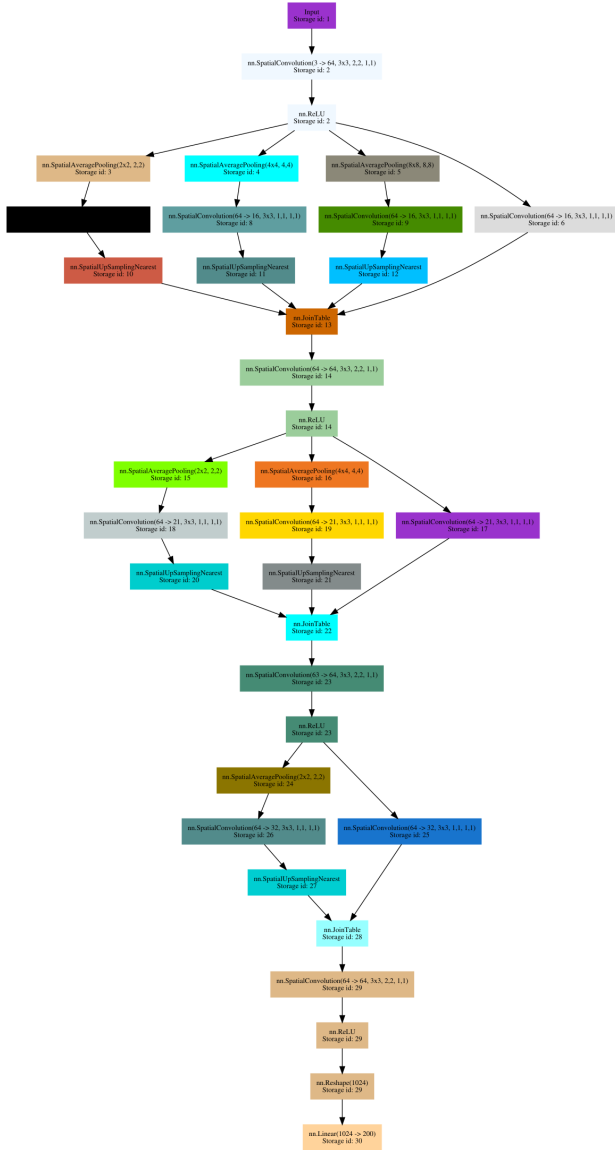


Figure 9. The computational graph for the multi scale architecture proposed in this paper.

The two networks are identical except that convolutional layers in the baseline network have been replaced with the layers described in this paper. These layers have 1/4, 1/3, and 1/2 the number of parameters as the originals they replace.

Nonetheless, the two networks are largely equivalent in their accuracies. The non-baseline model also trained sub-

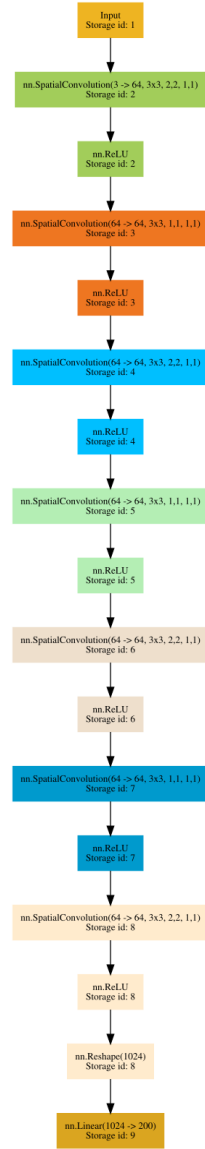


Figure 10. The baseline network benchmarked against.

stantially faster than the baseline, achieving its best validation error in only 3 epochs as opposed to 9.

In terms of runtime performance, the two were also largely equivalent, with batch step times of 26ms and 25ms respectively for batches of 64 images. Note that the new architecture could likely be optimized to run substantially faster if it were optimized to not reconstruct the pyramid from scratch at each layer.

6. Conclusions

The convolution over scale approach described in this report represents a mathematically and intuitively clean approach for extending the convolutional architecture to han-

Model	Train Error	Validation Error
Baseline	32%	77%
Pyramid	36%	76%

Table 1. Slightly better accuracies with many fewer parameters. Note that the goal is comparison to validate the approach, not low overall error. This result was not cherry-picked among the models tried.

dle scaling properties we would expect from images captured by a perspective camera in the real world. Traditional networks cannot easily capture these scale invariances, nor do they automatically generalize learning at one scale to learning at another. Instead, they rely on depth to capture scale invariances, which is costly in training and execution time. The approach presented here captures scale invariances within each layer by design.

The goal of this project was to find an approach for building convolutional layers that are scale-invariant and support multi-scale-features, to implement those layers, to validate that they can be trained by conventional means, and to begin to evaluate their promise against standard convolutional layers. I did all of those things, and the layers show exciting promise in terms of being able to capture image structure with far fewer parameters.

Excitingly, there are many avenues for future research here. What happens when these convolutional layers are imbedded in ResNets? How do they perform on larger image datasets with training times that would make testing on them during a project infeasible? How does their effectiveness change as we change the amount of steps the filters slide along the scale pyramid? I look forward to exploring all these and more in my thesis next year.

7. Appendix: Project Code Access

The code written for this project can be accessed through the following short link:

<http://bit.ly/1ZsGKbF>

This link leads to a directory on Dropbox, since the repository is not otherwise public. All the Lua code written is present and reasonably well documented and commented, with the exception of the testing suite, which has been omitted for ease of your reading. The link will remain active through grading. If somehow it gets broken, please send me an email, and I will fix it immediately.

References

- [1] Convnet benchmarks. <https://github.com/soumith/convnet-benchmarks>, 2016.
- [2] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning.
- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [4] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, August 2013.
- [5] R. B. Girshick, F. N. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. *CoRR*, abs/1409.5403, 2014.
- [6] S. Gross and M. Wilber. Training and investigating residual nets.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [8] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.
- [9] J. Johnson. Tiny imagenet. <https://github.com/jcjohnson/tiny-imagenet>, 2016.
- [10] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014.
- [11] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015.
- [12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.
- [13] Y. Xu, T. Xiao, J. Zhang, K. Yang, and Z. Zhang. Scale-invariant convolutional neural networks. *CoRR*, abs/1411.6369, 2014.
- [14] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015.