CNN architectures for regression?

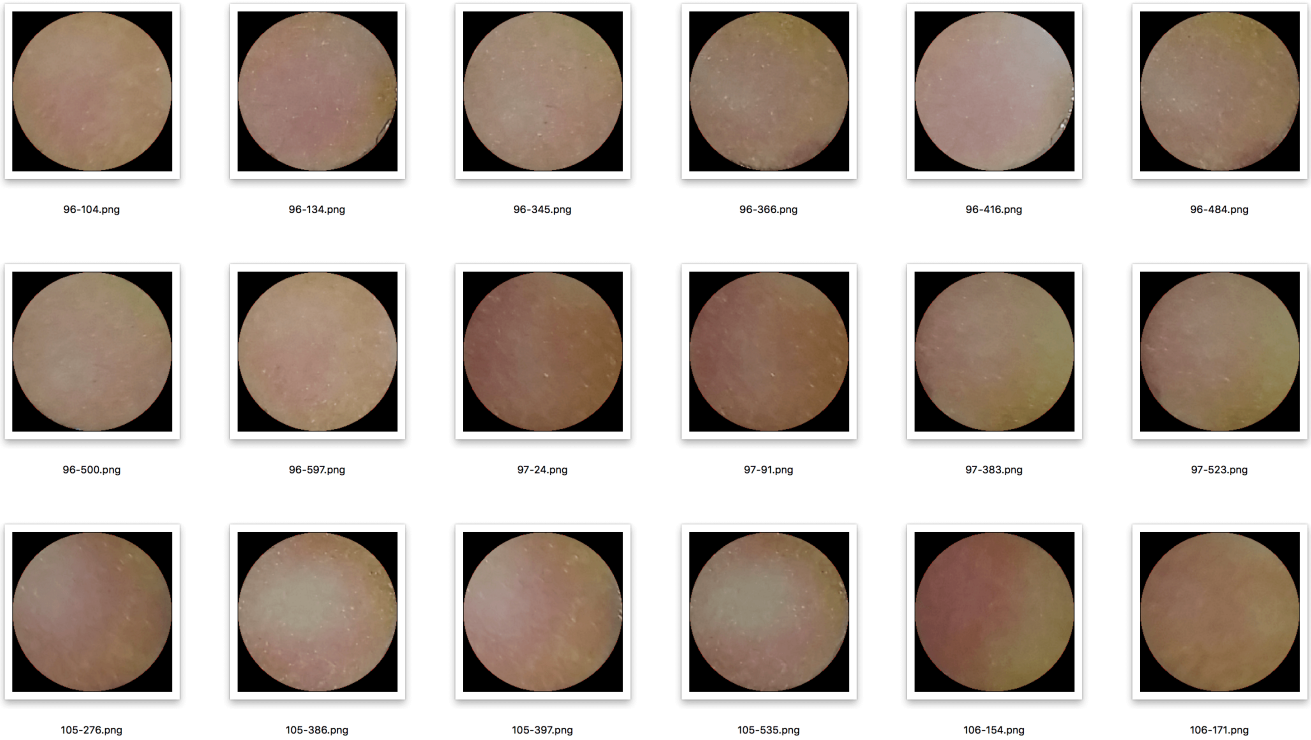
25

15

I've been working on a regression problem where the input is an image, and the label is a continuous value between 80 and 350. The images are of some chemicals after a reaction takes place. The color that turns out indicates the concentration of another chemical that's left over, and that's what the model is to output - the concentration of that chemical. The images can be rotated, flipped, mirrored, and the expected output should still be the same. This sort of analysis is done in real labs (very specialized machines output the concentration of the chemicals using color analysis just like I'm training this model to do).

So far I've only experimented with models roughly based off VGG (multiple sequences of conv-conv-conv-pool blocks). Before experimenting with more recent architectures (Inception, ResNets, etc.), I thought I'd research if there are other architectures more commonly used for regression using images.

The dataset looks like this:



The dataset contains about 5,000 250x250 samples, which I've resized to 64x64 so training is easier. Once I find a promising architecture, I'll experiment with larger resolution images.

So far, my best models have a mean squared error on both training and validation sets of about 0.3, which is far from acceptable in my use case.

My best model so far looks like this:

```
// pseudo code
x = conv2d(x, filters=32, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=32, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=32, kernel=[3,3])->batch_norm()->relu()
x = maxpool(x, size=[2,2], stride=[2,2])

x = conv2d(x, filters=64, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=64, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=64, kernel=[3,3])->batch_norm()->relu()
x = maxpool(x, size=[2,2], stride=[2,2])

x = conv2d(x, filters=128, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=128, kernel=[3,3])->batch_norm()->relu()
x = conv2d(x, filters=128, kernel=[3,3])->batch_norm()->relu()
x = maxpool(x, size=[2,2], stride=[2,2])

x = dropout()->conv2d(x, filters=128, kernel=[1, 1])->batch_norm()->relu()
x = dropout()->conv2d(x, filters=32, kernel=[1, 1])->batch_norm()->relu()

y = dense(x, units=1)

// loss = mean_squared_error(y, labels)
```

Edit

I've rephrased my explanation and removed mentions of accuracy.

Edit 2

I've restructured my question so hopefully it's clear what I'm after

regression | machine-learning | neural-networks | conv-neural-network | tensorflow

edited Mar 21 '18 at 19:53

asked Mar 21 '18 at 12:53

rodrigo-silveira

377 1 5 15

4 Accuracy isn't a measure that can be directly applied to regression problems. What do you mean when you say your accuracy is 30%? Accuracy really only applies to classification tasks, not regression. – Nuclear Wang Mar 21 '18 at 13:31

1 What do you mean by *"predicts correctly 30% of the time"*? Are you really doing regression? – Firebug Mar 21 '18 at 13:39

1 Why do you call this problem regression? Aren't you trying to classify into labels? are the labels cardinal? – Aksakal Mar 21 '18 at 13:40

1 I don't want the exact same thing as vgg. I'm doing something vgg-like, meaning a series of convs followed by max pooling, followed by fully connected. Seems like a generic approach for working with images. But then again, that's the whole point of my original question. Seems like all of these comments, although insightful to me, completely miss the point of what I'm asking in the first place. – rodrigo-silveira Mar 21 '18 at 16:46

1 Also, we might be able to provide better help if you gave a better description of the problem. 1) What are the images? What's their resolution? What relationship is there between the images and your response, $y \in [80, 350]$? Is this relationship rotation-invariant, i.e., if I rotate your circular image by an arbitrary angle θ, do I expect y to change? 2) Are you aware that 5000 images to train a VGG-net architecture are a misery? Have you computed the number of parameters of your architecture? Is there any way you could get more images? If you can't, then maybe you need... – DeltaIV Mar 21 '18 at 17:48

1 Answer

33

First of all a general suggestion: do a literature search before you start making experiments on a topic you're not familiar with. You'll save yourself a lot of time.

In this case, looking at existing papers you may have noticed that

1. CNNs have been used multiple times for regression: this is a classic but it's old (yes, 3 years is old in DL). A more modern paper wouldn't have used AlexNet for this task. This is more recent, but it's for a vastly more complicated problem (3D rotation), and anyway I'm not familiar with it.
2. Regression with CNNs is not a trivial problem. Looking again at the first paper, you'll see that they have a problem where they can basically generate infinite data. Their objective is to

bigger training set. Thus the problem seems relatively simple, as far as Deep Learning problems go. By the way, note the other data augmentation tricks they use:

> We use translations (up to 5% of the image width), brightness adjustment in the range [−0.2, 0.2], gamma adjustment with γ ∈ [−0.5, 0.1] and Gaussian pixel noise with a standard deviation in the range [0, 0.02].

I don't know your problem well enough to say if it makes sense to consider variations in position, brightness and gamma noise for your pictures, carefully shot in a lab. But you can always try, and remove it if it doesn't improve your test set loss. Actually, you should really use a validation set or $k-$fold cross-validation for these kinds of experiments, and don't look at the test set until you have defined your setup, if you want the test set loss to be representative of the generalization error.

Anyway, even in their ideal conditions, the naive approach didn't work that well (section **4.2**). They stripped out the output layer (the softmax layer) and substituted it with a layer with two units which would predict the sine y and cosine x of the rotation angle. The actual angle would then be computed as $\alpha = \operatorname{atan2}(y, x)$. The neural network was also pretrained on ImageNet (this is called *transfer learning*). Of course the training on ImageNet had been for a different task (classification), but still training the neural network from scratch must have given such horrible results that they decided not to publish them. So you had all ingredients to make a good omelette: potentially infinite training data, a pretrained network and an apparently simple regression problem (predict two numbers between -1 and 1). Yet, the best they could get with this approach was a 21° error. It's not clear if this is an RMSE error, a MAD error or what, but still it's not great: since the maximum error you can make is 180°, the average error is $> 11\%$ of the maximum possible error. They did slightly better by using two networks in series: the first one would perform classification (predict whether the angle would be in the $[-180^\circ, -90^\circ], [-90^\circ, 0^\circ], [0^\circ, 90^\circ]$ or $[90^\circ, 180^\circ]$ class), then the image, rotated by the amount predicted by the first network, would be feed to another neural network (for regression, this time), which would predict the final additional rotation in the $[-45^\circ, 45^\circ]$ range.

On a much simpler (rotated MNIST) problem, you can get something better, but still you don't go below an RMSE error which is 2.6% of the maximum possible error.

So, what can we learn from this? First of all, that 5000 images is a small data set for your task. The first paper used a network which was pretrained on images similar to that for which they wanted to learn the regression task: not only you need to learn a different task from that for which the architecture was designed (classification), but your training set doesn't look anything at all like the training sets on which these networks are usually trained (CIFAR-10/100 or ImageNet). So you probably won't get any benefits from transfer learning. The MATLAB example had 5000 images, but they were black and white and semantically all very similar (well, this could be your case too).

Then, how realistic is doing better than 0.3? We must first of all understand what do you mean by 0.3 average loss. Do you mean that the RMSE error is 0.3,

$$\frac{1}{N}\sum_{i=1}^{N}(h(\mathbf{x}_i) - y_i)^2$$

assuming that you clip the predictions of your CNN between 80 and 350 (or you just use a logit to make them fit in that interval), you're getting less than 0.12% error. Seriously, what do you expect? it doesn't seem to me a big error at all.

Also, just try to compute the number of parameters in your network: I'm in a hurry and I may be making silly mistakes, so by all means double check my computations with some `summary` function from whatever framework you may be using. However, roughly I would say you have

$$9 \times (3 \times 32 + 2 \times 32 \times 32 + 32 \times 64 + 2 \times 64 \times 64 + 64 \times 128 + 2 \times 128 \times 128) + 128 \times 128 + 128 \times 32 + 32 \times 32 \times 32 = 533344$$

(note I skipped the parameters of the batch norm layers, but they're just 4 parameters for layer so they don't make a difference). You have half a million parameters and 5000 examples...what would you expect? Sure, the number of parameters is not a good indicator for the capacity of a neural network (it's a non-identifiable model), but still...I don't think you can do much better than this, but you can try a few things:

- normalize all inputs (for example, rescale the RGB intensities of each pixel between -1 and 1, or use standardization) and all outputs. This will especially help if you have convergence issues.
- go to grayscale: this would reduce your input channels from 3 to 1. All your images seem (to my highly untrained eye) to be of relatively similar colors. Are you sure it's the color that it's needed to predict y, and not the existence of darker or brighter areas? Maybe you're sure (I'm not an expert): in this case skip this suggestion.
- data augmentation: since you said that flipping, rotating by an arbitrary angle or mirroring your images should result in the same output, you can increase the size of your data set *a lot*. Note that with a bigger dataset the error on the training set will go up: what we're looking for here is a smaller gap between training set loss and test set loss. Also, if the training set loss increases a lot, this could be good news: it may mean that you can train a deeper network on this bigger training set without the risk of overfitting. Try adding more layers and see if now you get a smaller training set **and** test set loss. Finally, you could try also the other data augmentation tricks I quoted above, **if** they make sense in the context of your application.
- use the classification-then-regression trick: a first network only determines if y should be in one of, say, 10 bins, such as $[80, 97]$, $[97, 124]$,etc. A second network then computes a $[0, 27]$ correction: centering and normalizing may help here too. Can't say without trying.
- try using a modern architecture (Inception or ResNet) instead than a vintage one. ResNet has actually *less* parameters than VGG-net. Of course, you want to use the small ResNets here - I don't think ResNet-101 could help on a 5000 images data set. You can augment the data set a lot, though....
- Since your output is invariant to rotation, another great idea would be to use either group equivariant CNNs, whose output (when used as classifiers) is invariant to **discrete** rotations, or steerable CNNs whose output is invariant to continuous rotations. The invariance property would allow you to get good results with much less data augmentation, or ideally none at all (for what it concerns rotations: of course you still need the other types of d. a.). Group equivariant CNNs are more mature than steerable CNNs from an implementation point of view, so I'd try group CNNs first. You can try the classification-then-regression, using the G-CNN for

- experiment with the batch size (yeah, yeah, I know hyperparameters-hacking is not cool, but this is the best I could come with in a limited time frame & for free :-)
- finally, there are architectures which have been especially developed to make accurate predictions with small data sets. Most of them used *dilated convolutions*: one famous example is the mixed-scale dense convolutional neural network. The implementation is not trivial, though.

edited Sep 18 '18 at 14:35

answered Mar 22 '18 at 14:08

DeltaIV
8,744 1 33 74

2 Thank you for the detailed answer. I'd already been doing significant data augmentation. Tried a couple of variants of the inception model (where a variation means that the number of filters is scaled equally across the entire model). Saw incredible improvements. Still have a ways to go. I'll try a few of your suggestions. Thanks again. – rodrigo-silveira Mar 22 '18 at 15:50

@rodrigo-silveira you're welcome, let me know how it goes. Maybe we can talk in chat once you have results. – DeltaIV Mar 23 '18 at 13:59

1 Great answer, deserves more ^ – Gilly Jun 25 '18 at 2:11

1 Very well composed! – Karthik Thiagarajan Dec 12 '18 at 12:48

protected by Community ♦ Nov 14 '18 at 13:55

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?