Paper 8: Image Searching and Modelling Using Machine Learning

Handout 3: Deep Learning

Roberto Cipolla and Matthew Johnson
May 2016
CIFAR-10

We have seen some pretty impressive things so far with the artificial neural nets we have studied, but it is time to make things much harder. CIFAR-10, while it has many of the same characteristics as MNIST, is a considerable step up in difficulty. It is drawn from the “80 million tiny images” dataset [6] which consists of (almost) 80 million $32 \times 32$ RGB images downloaded from the internet and labeled with one of 75,062 non-abstract nouns in English.

CIFAR-10 (named after the Canadian Institute for Advanced Research where it was gathered) consists of 60,000 images gathered from this larger set by gathered by Alex Krizhevsky, Vinod Nair and Geoffrey Hinton [3]. These images belong to 10 classes: (in order from left to right above) airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. As can be seen, the dataset contains a lot of intra-class variance, in addition to complications brought about by color and the much larger variety inherent in non-curated natural images.
LeNet-5 on CIFAR-10

The first thing we will do with this new dataset is use the best tool we have seen so far: LeNet-5. It requires a small adjustment to take in 3-channel RGB images as input, but otherwise we can use it exactly as is and it will provide a good baseline moving forward.

LeNet-5 does considerably better than random, but it has considerably more difficulty with this new dataset than the one for which it was designed. An interesting reference point here is human performance on category recognition for small images. The resolution chosen for the “80 million tiny images” dataset was not arbitrary.

As can be seen, $32 \times 32$ is the “bend in the curve” for human recognition, below which our ability to recognised scenes deteriorates rapidly. We can thus count about 90% accuracy as good performance to achieve for this dataset. We have some way to go, and to get there we are going to need some new tools.
Demo

In this demo we will observe the behavior of LeNet-5 on this new dataset.

borfudin.github.io/anndemos/lenet_cifar.html
Rectified Linear Units

At the end of the previous lecture we discussed the problem of the vanishing gradient, where the repeated multiplication of the non-linearity derivatives resulted in the overall magnitude of the gradient decreasing as the network increased in depth, as can be seen below for a 20-layer fully-connected network:

To overcome this, a new form of non-linearity was developed: the Rectified Linear Unit, or ReLU [4]. ReLUs have a deceptively simple form:

\[ f(x) = \begin{cases} 
  x & x > 0 \\
  0 & x \leq 0 
\end{cases} \]

which has the corresponding derivative:

\[ f'(x) = \begin{cases} 
  1 & x > 0 \\
  0 & x \leq 0 
\end{cases} \]

While at first the discontinuity in the function and the lack of a “squashing” effect would appear to be a problem, in practice it does not have any negative effect on convergence of the network. However, it does have the very desirable effect of entirely reducing the vanishing gradient effect. It does this so nicely (and is so computationally efficient as well) that it is used in the majority of modern deep network architectures.
### Average Pooling

Another useful tool is pooling. The concept of pooling is similar to sub-sampling, whereby an image is reduced in size by representing a region in the input image via a single pixel. In standard sub-sampling this value is a single pixel in the input region, *e.g.* the center. In average pooling, the value is the mean value of the pixels in the input region.

As can be seen above a filter of size 3 is applied at a *stride* of 2, though the size and stride can be altered in practice as needed by the problem. This pooling acts as a *non-parameterised* layer in the neural network, much like non-linearities. An output value is computed as follows:

\[
o[r, c] = \frac{1}{9} \sum_{k=0}^{2} \sum_{l=0}^{2} i[2r + k - 1, 2c + l - 1]\]

This is equivalent to using the embedding function \(\mathcal{E}\) and then multiplying by a weight matrix \(\mathbf{W} = \{W_{i,j} | \forall i \forall j \ W_{i,j} = \frac{1}{9}\}\). When treated in this way the derivative is also this matrix (as in this case \(\mathbf{W}^R \equiv \mathbf{W}\) ).
Max Pooling

An alternative to average pooling is maximum, or max, pooling. As with average pooling, the max pooling operator has a set size and is applied at a predetermined stride. However, instead of taking the arithmetic mean of all of the pixel values it selects only the maximum value as its output value.

This creates a slight difficulty during backpropagation, as the partial derivatives passing backwards through the operator only apply to a single input pixel. We can achieve this by creating, during each forward pass, a $O \times I$ matrix $W$ in which each row has a single value of 1 at the column $i$ corresponding to the maximum input value used for the output row $o$. The derivative is then $W^T$. 
Anatomy of a Deep Net

Using these new building blocks, we can now fully examine the anatomy of a modern deep net for CIFAR-10.

We can compute the number of parameters per trainable layer: Channels × Size² × Nodes.

<table>
<thead>
<tr>
<th>NAME</th>
<th>CHANNELS</th>
<th>SIZE</th>
<th>NODES</th>
<th># PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3</td>
<td>5</td>
<td>32</td>
<td>2400</td>
</tr>
<tr>
<td>C3</td>
<td>32</td>
<td>5</td>
<td>32</td>
<td>25,600</td>
</tr>
<tr>
<td>C5</td>
<td>32</td>
<td>5</td>
<td>64</td>
<td>51,200</td>
</tr>
<tr>
<td>F7</td>
<td>1024</td>
<td>64</td>
<td>64</td>
<td>65,536</td>
</tr>
<tr>
<td>F8</td>
<td>64</td>
<td>10</td>
<td>10</td>
<td>640</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>145,376</strong></td>
</tr>
</tbody>
</table>

We can also calculate the input and output dimensions. For convolutional and pooling layers, output is a function of stride, size and padding: \( \text{dim}_o = \left\lfloor \frac{\text{dim}_i + 2 \times \text{padding} - \text{size}}{\text{stride}} \right\rfloor + 1 \).

<table>
<thead>
<tr>
<th>NAME</th>
<th>INPUT</th>
<th>SIZE</th>
<th>STRIDE</th>
<th>PAD</th>
<th>NODES</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>3x32x32</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>32</td>
<td>32x32x32</td>
</tr>
<tr>
<td>P2</td>
<td>32x32x32</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>32</td>
<td>32x16x16</td>
</tr>
<tr>
<td>C3</td>
<td>32x16x16</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>32</td>
<td>32x16x16</td>
</tr>
<tr>
<td>P4</td>
<td>32x16x16</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>64</td>
<td>32x8x8</td>
</tr>
<tr>
<td>C5</td>
<td>32x8x8</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>64</td>
<td>64x8x8</td>
</tr>
<tr>
<td>P6</td>
<td>64x8x8</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>64</td>
<td>64x4x4</td>
</tr>
<tr>
<td>F7</td>
<td>1024</td>
<td></td>
<td></td>
<td></td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>F8</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
Anatomy, cont.

Now that we understand the parameters and the dimensionality, let us look at this in more detail.

Each layer in the network plays an important role.

**C1** Extracts low-level features in the image, like edges, corners and blobs.

**P2** Provides some flexibility of location to the low-level features

**C3** Looks for parts that are combinations of lower-level features

**P4** Smoothes the part responses before subsampling

**C5** Finds structures that are built from parts

**P6** Smoothes and subsamples the structural responses. The output of this final layer acts as a CIFAR-10 specific feature vector of length 1024.

**F7** Sub-category classifiers

**F8** Final classifier
Demo

In this demo we will examine the behaviour of a standard modern-style DNN on the CIFAR10 dataset.

borfudin.github.io/anndemos/dnn.html
Deep Net Training

The following is a typical procedure for training a deep neural network on a dataset.

1. Divide the dataset into *training* $D$, *validation* $V$ and *test* $T$ partitions
2. Compute the “mean” image of $D$, and subtract it from all images
3. Scale all images so that pixel values are in the range $[-1, 1]$
4. Design a network architecture, or adapt a known-good architecture to the dataset
5. Select an optimisation algorithm (*i.e.* a version of SGD)
6. For a pre-determined set of epochs, do the following:
   (a) Randomly sample (without replacement) $B$ images from $D$
   (b) For each image $b \in B$:
      i. Compute $P(i|b)$ by showing $b$ to the network
      ii. Compute $\nabla w_b = \eta \frac{\partial L}{\partial w}$ using the label $i_b$ and add it to $\nabla w$
   (c) Update the parameters using $\frac{1}{|B|}\nabla w$
   (d) Once all images in $D$ have been seen, evaluate the network on $D$ and $V$ to monitor changes in the loss
7. Evaluate final performance on $V$, and repeat the above with different optimisation hyperparameters as necessary
8. Retrain the network using $D \cup V$ with the final hyperparameters, and evaluate on $T$
Data Augmentation

When it comes to deep networks, the only thing better than a lot of data is a whole lot of data. However, gathering supervised data is very expensive. One way to overcome this problem is via data augmentation, whereby on each epoch each training image is turned into several additional images by way of translated cropping and horizontal reflection:

In this way one training image can become 10. Another common usage of data augmentation is during the testing phase, whereby instead of simply computing $P(i|x)$ from a single image, $x$ is altered as shown above and then each version of $x$ is shown to the network. In this scenario, the predicted label is then:

$$i_x = \arg\max_j \left\{ j_a | \forall a : j_a = \arg\max_k P(k|x_a) \right\}$$

This can result in significant improvement regardless of the network model used.
Batch Normalisation

Batch Normalisation [2] is a technique that is essential to the deepest neural net architectures, but also useful in general for stabilising and accelerating the training process in all circumstances. The concept is straightforward: if we knew the statistics of each layer’s output over the entire dataset, we could normalise those outputs so that all output vectors had zero mean and unit variance. In mathematical terms, we want to do the following:

$$
\mu_D = \frac{1}{D} \sum_{i=0}^{D} x_i
$$

$$
\sigma^2_D = \frac{1}{D} \sum_{i=0}^{D} (x_i - \mu_D)^2
$$

$$
\hat{x}_i = \frac{x_i - \mu_D}{\sqrt{\sigma^2_D + \epsilon}}
$$

The problem is that we do not know $\mu_D$ or $\sigma^2_D$, nor can we compute them as they will change as the network itself changes during training. We overcome this by using the same batch trick to estimate these values i.e. by computing them for each batch during training. Once the network is trained, we can then compute the true values and use those during testing. We can even learn training parameters for scaling and shifting the normalised vectors:

$$
y_i = \gamma \hat{x}_i + \beta
$$

Backpropagation through this operation is outside of the scope of these lectures, but is detailed in the referenced paper.
ImageNet

The final dataset we will look at is the ImageNet dataset. It consists of over 14 million images mapped to around 20,000 labels. The images are extremely varied, coming from a variety of different sources all over the internet, and have each been hand-labeled by a human.

![ImageNet dataset images]

Every year since 2010 there has been a challenge organised around subsets of these images. In 2015, it was dominated by Microsoft Research Asia [5].

**Detection** The algorithm has to specify for each image multiple classes (from a choice of 200) and their locations in the image.

<table>
<thead>
<tr>
<th>Team Name</th>
<th># Categories</th>
<th>Mean AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSRA</td>
<td>194</td>
<td>0.621</td>
</tr>
<tr>
<td>Qualcomm Research</td>
<td>4</td>
<td>0.536</td>
</tr>
<tr>
<td>CUIImage</td>
<td>2</td>
<td>0.527</td>
</tr>
</tbody>
</table>

**Classification** The algorithm has to specify for each image to which class it belongs (from a choice of 1000)

<table>
<thead>
<tr>
<th>Team Name</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSRA</td>
<td>0.0357</td>
</tr>
<tr>
<td>ReCeption</td>
<td>0.0358</td>
</tr>
<tr>
<td>Trips-Soushen</td>
<td>0.0458</td>
</tr>
</tbody>
</table>

We will look at their network in our final case study.
Case Study: ResNet

ResNet [1] used modular construction to assemble the deepest successfully trained neural network to date. The overall structure of the network (in a 34-layer configuration) looks like this:

The bottom of each $R$ block in that diagram is a neural net in miniature of the following form:

Note that while ResNet can be extremely deep (with up to 152 layers) the building blocks are the elements we have already seen in these lectures. The essential techniques that make the training of this network possible are batch normalisation and the shortcut structures, which enable the network to selectively exclude blocks during training and then add them in as they become useful.
References


