Notes for Week 2 of CNN by Andrew. Ng @deeplearning.ai

Week 2 : Deep convolutional models: case studies
V1.0
17/02/2020

Case Studies

Why look at case studies?

- See someone others' codes can helps us understand better. We can perhaps use some ideas of others to build our own Neural Network.
- Some model of NN works well for certain project may work well for your projects, too.
- Help us to know how to read some **research papers**

Classic Networks

LeNet-5

First introduced by LeCun et al., 1998.

> [LeCun et al., 1998. Gradient-based learning applied to document recognition]

Structure of Model

- input layer: digital picture of size $(32 * 32 * 1)$
- 1st Convoluntional layer: $\text{size} \longrightarrow (28 * 28 * 6)$
  - $\text{size} = 5 * 5$
  - $\text{stride} = 1$
  - $\text{num\_of\_channels} = 6$
- 1st AvePooling layer: $\text{size} \longrightarrow (14 * 14 * 6)$
  - $f = 2$
  - $\text{stride} = 2$
- 2nd Convoluntional layer: $\text{size} \longrightarrow (10 * 10 * 16)$
  - $\text{size} = 5 * 5$
  - $\text{stride} = 1$
  - $\text{num\_of\_channels} = 16$
- 2nd AvePooling layer: $\text{size} \longrightarrow (5 * 5 * 16)$
  - $f = 2$
  - $\text{stride} = 2$
- 1st Full Connected layer: $\text{size} = (5 * 5 * 16) = 400 \longrightarrow 120$
- 2nd Full Connected layer: $\text{size} = 120 \longrightarrow 84$
- output layer: $\text{output\_size} = 1$

Pattern of Models

- As the network going deeper and deeper:
  - $n_H, n_W \downarrow$
  - $n_C \uparrow$
- Sequentials:
  - Conv + Pool
  - Conv + Pool
  - Fc
  - Fc
  - ...
  - Output

Guideline for reading the original paper

It's the most difficult paper introduced in this week, and it has used many techniques to reduce the commputations. We will focus on the `part ii` and `part iii`.

AlexNet

First introduced by Alex Krizhevsky et al., 2012.

> [Alex Krizhevsky et al,. 2012. ImageNet classification with deep convolutional neural networks]

Structure of Model

- input layer: digital picture of size $(227 * 227 * 3)$
- 1st Convoluntional layer: $\text{size} \longrightarrow (55 * 55 * 96)$
  - $\text{size} = 11 * 11$
  - $\text{stride} = 4$
  - $\text{num\_of\_channels} = 96$
- 1st MaxPooling layer: $\quad \text{size} \longrightarrow (27 * 27 * 96)$
  - $\text{size} = 3 * 3$
  - $\text{stride} = 2$
- 1st `Same` Padding layer: $\text{size} \longrightarrow (27 * 27 * 256)$
  - $\text{size} = 5 * 5$
  - $\text{num\_of\_channels} = 256$
- 2nd MaxPooling layer: $\quad \text{size} \longrightarrow (13 * 13 * 256)$
  - $\text{size} = 3 * 3$
  - $\text{stride} = 2$
- 2nd `Same` Padding layer: $\text{size} \longrightarrow (13 * 13 * 384)$
  - $\text{size} = 3 * 3$
  - $\text{num\_of\_channels} = 384$
- 3rd `Same` Padding layer: $\text{size} \longrightarrow (13 * 13 * 384)$
  - $\text{size} = 3 * 3$
  - $\text{num\_of\_channels} = 384$
- 4th `Same` Padding layer: $\text{size} \longrightarrow (13 * 13 * 256)$

  - $\text{size} = 3 * 3$
  - $\text{num\_of\_channels} = 256$
- 3rd MaxPooling layer: $\text{size} \longrightarrow (6 * 6 * 256) = 9216$
  - $\text{size} = 3 * 3$
  - $\text{stride} = 2$
- 1st Full Connected layer: $\text{size} = 9216 \longrightarrow 4096$
- 2nd Full Connected layer: $\text{size} = 4096 \longrightarrow 4096$
- output layer: $\text{size\_of\_Softmax} = 1000$

Patterns of Model & Paper

- Similarity to LeNet, but much more bigger
- Use Relu as an activation fonction
- Mentionned for the paper:
  - Multiple GPUs for Conv layers to gain a higher speed
  - Local Response Normolization(LRN)
    Normalize at each position of the layer with depth of all the chanels.
  - *Easiest paper to read introduced this week*

VGG-16

Introduced by Simonyan&Zisserman 2015. Using much more simple layers

- CONV : $3 * 3 \quad \text{filter}, \quad s = 1, \quad \text{padding} = \text{same}$
- MAX-POOL : $2 * 2 \qquad s = 2$

Structure of Model

With [CONV] and [POOL] layers defined above:

$$
\begin{aligned}
224 * 224 * 3_{\text{ Input size}} &\xrightarrow[\text{[CONV 64] * 2}]{} 224 * 224 * 64 \xrightarrow[\text{POOL}]{} 112 * 112 * 64 \\
&\xrightarrow[\text{[CONV 128] * 2}]{} 112 * 112 * 128 \xrightarrow[\text{POOL}]{} 56 * 56 * 128 \\
&\xrightarrow[\text{[CONV 256] * 3}]{} 56 * 56 * 256 \xrightarrow[\text{POOL}]{} 28 * 28 * 256 \\
&\xrightarrow[\text{[CONV 512] * 3}]{} 28 * 28 * 512 \xrightarrow[\text{POOL}]{} 14 * 14 * 512 \\
&\xrightarrow[\text{[CONV 512] * 3}]{} 14 * 14 * 512 \xrightarrow[\text{POOL}]{} 7 * 7 * 512 \\
&\longrightarrow \underset{4096}{FC} \longrightarrow \underset{4096}{FC} \\
&\longrightarrow \underset{\text{Softmax(1000)}}{\text{Output layer}}
\end{aligned}
$$

Patterns of Model

- It's really a huge Neural Network
  ~ 138M variables
- architechture uniform
- roughly double num of channels and divide with pooling
  - $n_H, n_W \downarrow \qquad * = \frac{1}{2}$

- $n_C \uparrow$ $* = 2$
- which is very intresting to reserve this relationship

Residual Network

Introduced by He et al., 2015

> He et al., 2015. Deep residual networks for image recognition

- Very, very deep neural networks are difficult to train because of vanishing and exploding gradient types of problems.
- Residual block
  - skip connections which allows you to take the `activation` from one layer and **suddenly** feed it to another layer even much deeper in the neural network
  - after the `linear` part but before the `activation` part
  - use a *skip cut* or "*skip connection*"
  - $a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$
- in theory deep nn gives better result but in reality no
- resNet can give better result with deep NN

Why ResNets Works?

- classicly, when neural network goes deeper and deeper its ability will go down in contrast to our wishes.

- after big nn ,we use residual block, which:

  - at least do no harm to the forword deep NN:
    We use `Relu` here as activation function, so there will be no-negative activation.
    As we known that the residual block works as

$$\begin{aligned} a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) \\ &= g(w^{[l+2]}a^{[l+1]} + b^{[l+1]} + a^{[l]}) \end{aligned}$$

  `IF` there is nothings to learn, then suppose that

$$w^{[l+2]} = b^{[l+1]} = 0$$

  we'll still get

$$\begin{aligned} a^{[l+2]} &= g(z^{[l+2]} + a^{[l]}) \\ &= g(w^{[l+2]}a^{[l+1]} + b^{[l+1]} + a^{[l]}) \\ \xrightarrow{w^{[l+2]}=b^{[l+1]}=0} &= g(a^{[l]}) \\ &= a^{[l]} \end{aligned}$$

  `ELSE` we can perhaps learn something in these residual layers

- In fact, in traditional deep network, those deep layers have huge difficult to choose parameters in order to get better performance

Networks in Networks and 1*1 Convolutions

First introduced by Lin et al., 2013. Network in Network what does a 1*1 convolution do?

- multiply?
- In the case of multiple channels,
  - in reality, these namely `1*1` Convolution layers have the demention of $(1 \times 1 \times n_C^{[l-1]})$.
  - Thus, if we have the number of `1*1` filters is $n_C^{[l]}$, we will get the layer $[l]$ of original height & weigth dimentions and new channel dimention.
- depend en channels + relu
- change num of channels
  - shrink
  - keep
  - or even increse
  - *by modifing num of filters of these* 1×1 *Convolutions*

Inception Network

- why should i chose all those size of conv pool...
- First introduced by Szegedy et al., 2014. Going deeper with convolutions

Motivation for inception network

- stack the results of different conv layers (Padding = Same)
  - 1×1 convolution
  - 3×3 same convolution
  - 5×5 same convolution
  - same Max-Pooling
  - ...

    These will result in different kind of channels with the `same size`

    So that we can esaily `stack them up` to form a new layer of multiple channels come from different `convolutions` and `poolings`
  - in this way we do not need to choose size of convolutions
- Problem of computational cost
  - ex. 5*5 filters

$$
\begin{aligned}
28 \times 28 \times 192 \xrightarrow[\text{same},32]{\text{CONV } 5\times 5} 28 \times 28 \times 32 \\
\text{Will takes} \\
28 * 28 * 32 \times 5 * 5 * 192 \simeq 120M \quad \text{computations}
\end{aligned}
$$

It's very expansize in fact

- in revanch, if we use a 1×1 to first shrink the size to

$$28 \times 28 \times 16$$

can definitely reduce this cost:

$$28 \times 28 \times 192 \xrightarrow[16]{\substack{\text{CONV}\\1\times1}} 28 \times 28 \times 16 \xrightarrow[\text{same,32}]{\substack{\text{CONV}\\5\times5}} 28 \times 28 \times 32$$

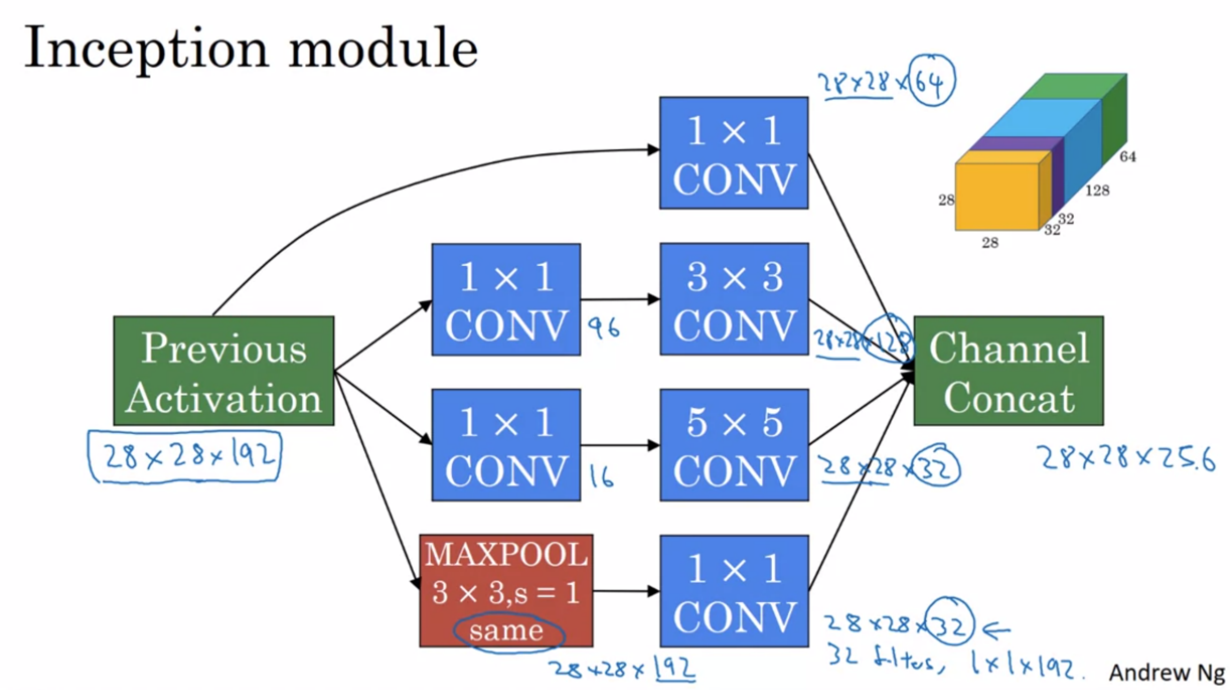
$$\text{Will takes}$$

$$28 * 28 * 16 \times 1 * 1 * 192 + 28 * 28 * 32 \times 5 * 5 * 16$$

$$\simeq 2.4M + 10.0M = 12.4M$$

$$\text{computations}$$

Build an Incerption Module



- In the paper:
  - what do side branches do?
    Side branches just try to predict the result as the output layer
  - first developped by google
    so we call it `gooLeNet`
  - Where comes from the name Inception:
    we need to go deeper
    from film Inception

Practical advices for using ConvNets

Using Open-Source Implementation

- Hard to replicate the result of jobs of papers
- Online open sourse!
- Find them in Github
- Advantages:
  - Normally good neural networks take long time and many GPUs to train. So we can take advantage of results of those open source to do `transfer learning`

   - We can also `Contribute` to the world
   - Much more easy to `Get start!`

Transfer Learning

- Someone else has alreafy trainned very deep neural networks thich may take some weeks and also many GPUs computation
- Transfer knowledge is very useful here!

How to do it?

- download some network implementataion

  not only the `code`
  but also the `parameters/weights` that have been beautifully trained

- get ride of some final output decision layers and use our own layers to give an output which corresponds to our own problems

- Freeze the foward layers by setting:

   - trainable parameter = 0
   - freeze = 1
     that is to say, these forward layers will be `frozen` so that we **don't** train their parameters but just use the result of ancien training.

We can also:

- `map` the input to the **last** frozen layer
  beacause these layers are already trained, we just do some `pre-compute` so that we can save the result to disk for later training.
- freeze fewer layers when we have a big dataset to learn from
   - we can just add more **our own hiden layers**
   - General pattern:
     more date --> fewer layers frozen

Data Augmentation

- Computer vision is very complex task so we need as much as possible datas
- Often data augmentation will help

Common augmentation method

1. Mirroring
2. Random Cropping
3. Rotation
4. Shearing
5. Local warping
6. ...

Color shifting

- R +=20
- G -=20
- B +=20
  More robust to the color changements
- PCA(Principle Component Analysis)
- ml-class.org
- AlexNet paper PCA color augmentation

Implementing distortions during training

- one thread and other thread
- CPU thread works to form a mini-batch of data
- While GPU or other CPU will work for other training procedure
- These can be done in parallel

State of Computer Vision

Data vs. hand-engineering

larger and larger dataset

- object detection
- Image recognition
- Speech recognotion

In history, when we do not have so much data, people tends to use a lot of `Hacks` (more `hand-engineering`). But now since there's larger and larger dataset, people tends to use some very simple architechture of algorithme.

Two sources of knowledge

- Labeled data
- hand engineered features/network architecture/other components

Often we donnot have as much data as we want

- when we have little data, there will be more techniques

Tips fpr doing well on benchmarks/winning comprtitions

Ensembling

- Train several networks independently and average their outputs
  outputs but not the weights

Multi-crop at test time

- Run classifier on multiple versions of test images and average results

Use open source code

- Use architecture of networks published in the literature
- Use open source implementations if possible
- Use pretrained models and fine-tune on your dataset