IMAGE CLASSIFICATION USING PYTORCH

ABHISHEK PANDEY

Final year student, Department of Computer Science and Engineering, Inderprastha Engineering College, Dr.A.P.J Abdul Kalam Technical University.

---***---

Abstract - *This paper presents image classification using pytorch in 2020. Pytorch is the newest tool in python for image classifying with a high accurate results.*

PyTorch is currently the hottest Deep Learning library out there. In terms of popularity, it has even taken over Tensorflow. Tensorflow came before PyTorch and is backed by the engineering and marketing might of Google.

Key Words: Classification; Clustering; Learning; MLP; SOM; Supervised learning; unsupervised learning

1 INTRODUCTION

Image classification refers to the task of extracting information classes from a multiband raster image. The resulting raster from image classification can be used to create thematic maps. Depending on the interaction between the analyst and the computer during classification, there are two types of classification: supervised and unsupervised. With the ArcGIS Spatial Analyst extension, there is a full suite of tools in the Multivariate tools to perform supervised and unsupervised classification. The classification process is a multi-step workflow, therefore, the Image Classification toolbar has been developed to provide an integrated environment to perform classifications with the tools. Not only does the toolbar help with the workflow for performing unsupervised and supervised classification, it also contain additional functionality for analysing input data, creating training samples and signature files, and determining the quality of the training samples and signature files. The recommended way to perform classification and multivariate analysis is through the Image classification toolbar.

1.1 Build a pytorch CNN model

Convolution neural networks(CNNs) is the most popular neural network model being used for image classification problems. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It's like reading a book by using a magnifying glass, eventually, you read the whole page, but you look at any a small patch of the page at any given time.

1.2 CNN ARCHITECTURE

Self-Organizing neural networks learn to use unsupervised learning algorithms to identify hidden patterns in unlabelled input data. This unsupervised refers to the ability to learn and organize information without providing an error signal to evaluate the potential solution. The lack of direction for the learning algorithm in unsupervised learning can sometime be advantageous, since it lets the algorithm to look back for patterns that have not been previously considered. The main characteristics of Self-Organizing Maps (SOM) are:

1. It transforms an incoming signal pattern of arbitrary dimension into one or 2 dimensional maps and perform this transformation adaptively
2. The network represents feed forward structure with a single computational layer consisting of neurons arranged in rows and columns.
3. At each stage of representation, each input signal is kept in its proper context and,
4. Neurons dealing with closely related pieces of information are close together, and they communicate through synaptic connections.

The computational layer is also called a competitive layer since the neurons in the layer compete with each other to become active. Hence, this learning algorithm is called a competitive algorithm. Unsupervised algorithm in SOM works in three phases:

Competition phase: for each input pattern x, presented to the network, inner product with synaptic weight w is calculated and the neurons in the competitive layer finds a discriminant function that induce competition among the neurons and the synaptic weight vector that is close to the

input vector in the Euclidean distance is announced as winner in the competition. That neuron is called best matching neuron, i.e. $x = \arg\min \|x - w\|$.
Cooperative phase: the winning neuron determines the center of a topological neighborhood h of cooperating neurons. This is performed by the lateral interaction d among the cooperative neurons. This topological neighborhood reduces its size over a time period.

Adaptive phase: enables the winning neuron and its neighborhood neurons to increase their individual values of the discriminant function in relation to the input pattern through suitable synaptic weight adjustments, $\Delta w = \eta h(x)(x - w)$.

Upon repeated presentation of the training patterns, the synaptic weight vectors tend to follow the distribution of the input patterns due to the neighborhood updating and thus ANN learns without supervision.

Self-Organizing Model naturally represents the neuro-biological behavior, and hence is used in many real world applications such as clustering, speech recognition, texture segmentation, vector coding etc.

A typical cnn architecture look like below:

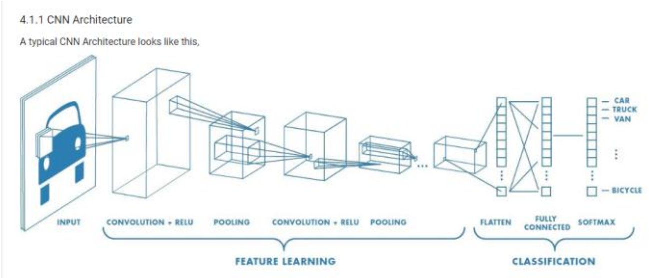


Fig 1: cnn architecture

CNN models may differ based on how they are trained with each and every different case.

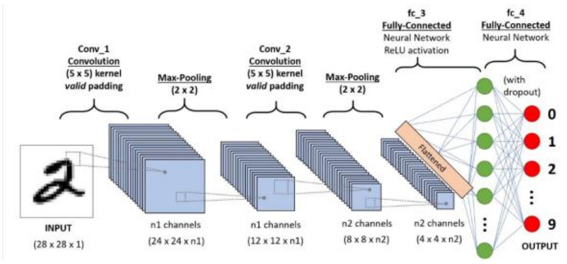


Fig 2: cnn architecture

1.3 Basic cnn layer

And in order to build a "deep" neural network, we can stack several layers like the one built in the previous section. To show the reader how to do it in our example, we will create a second group of layers that will have 64 filters with a 5×5 window in the convolutional layer and a 2×2 window in the pooling layer. In this case, the number of input channels will take the value of the 32 features that we have obtained from the previous layer, although, as we have seen previously, it is not necessary to specify it because Keras deduces it:

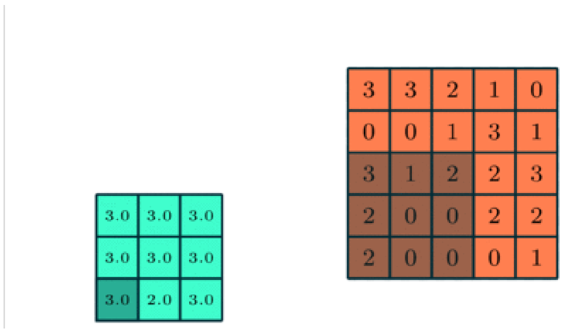


Fig 3: basic cnn model

2. STRIDES

The amount of the movement between applications of the filter to the input image is referred to as the strides, and it is almost always symmetrical in height and width dimensions the default strides or stride in two dimensions is (1, 1) for the height and the width movement, performed when needed. And this default works well in most cases. The strides can be changed which has an effect both on how the filter is applied to the image and, in turn, the size of the resulting feature map.

for example, the stride can be changed to (2,2).this has the effect of moving the filter two pixels right for each horizontal movement of the filter and two pixels down for each vertical movement of the filter when creating the feature map.

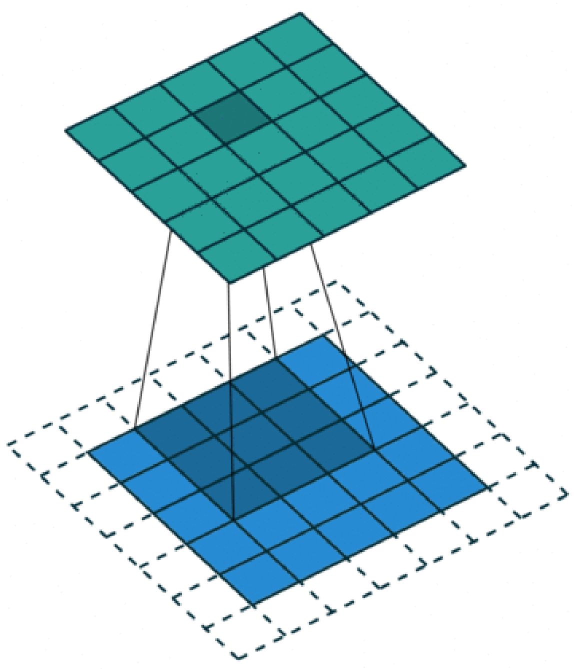
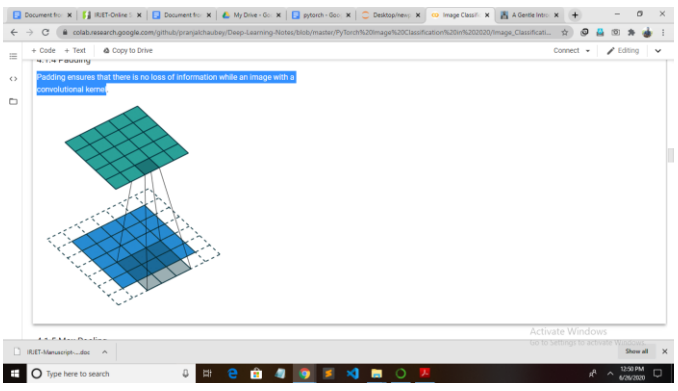


fig 4. strides

3. Padding

Padding ensures that there is no loss of information while an image with a convolution kernel



Padding ensures that there is no loss of information while an image with a convolutional kernel.

4. MAX POOLING

Max pooling layer primarily reduces the dimensionality of the input.

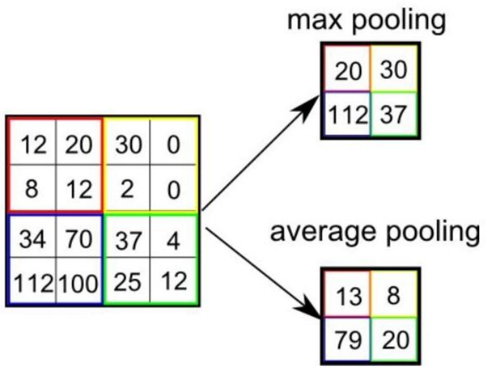
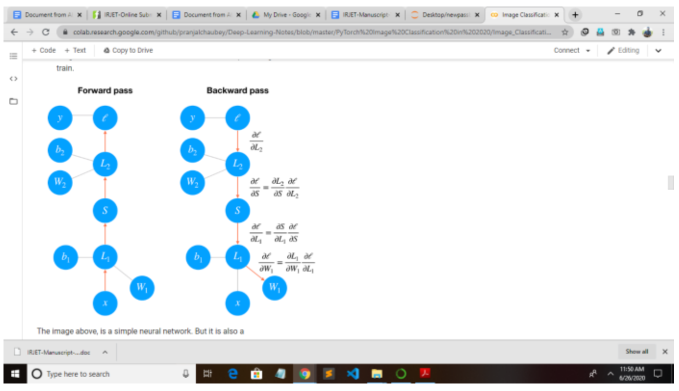


Fig 5 max pooling

5. UNDERSTANDING COMPUTATIONAL GRAPHS

At the bottom of every Deep Neural Network training, there are only two things taking place,

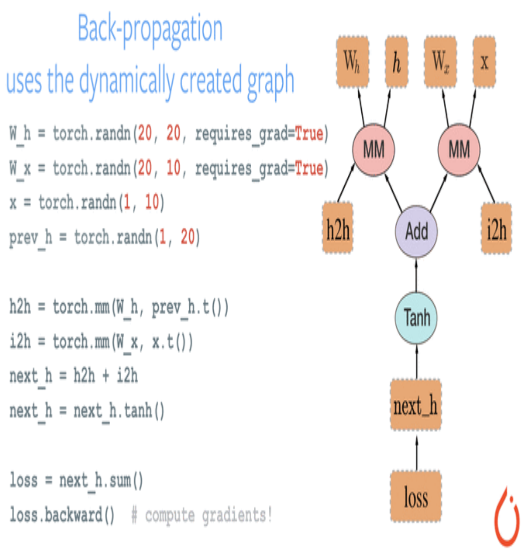
1. A forward pass - pushing images/data from the start of the network and generating an output (and a loss/error).
2. Backpropagation - essentially a backward pass where we calculate gradients using partial derivatives with respect to the loss, and make changes to the weights of the network. In a nutshell, this is how deep learning networks train.



train.

The image above, is a simple neural network. But it is also a

The image above, is a simple neural network. But it is also a computational graph. We first make a forward pass through our network and then a backward pass to calculate how much loss was being contributed by ***W1*** weight in particular. Every neural network you define, PyTorch *sees* it as a computational graph similar to what we see above and keeps a track of all the operations performed by every node. This ensures that it calculates accurate gradients when making a backward pass. Good thing about PyTorch is that it creates these computational graphs on the fly! And this aspect makes PyTorch and extremely flexible (and pythonic) deep learning library.



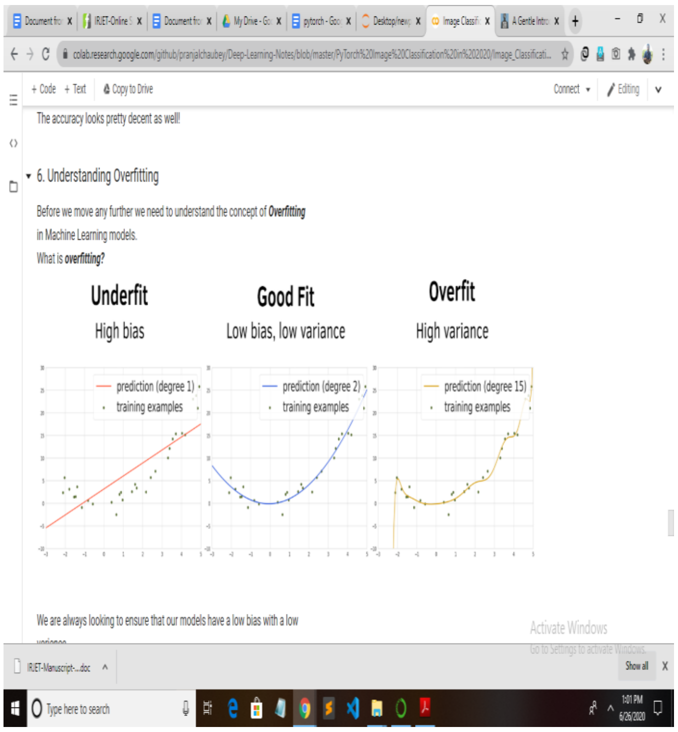
```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)

h2h = torch.mm(W_h, prev_h.t())
i2h = torch.mm(W_x, x.t())
next_h = h2h + i2h
next_h = next_h.tanh()

loss = next_h.sum()
loss.backward()  # compute gradients!
```

5. UNDERSTANDING OVERFITTING

One of the problems that occur during neural network training is called overfitting. The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations.



The accuracy looks pretty decent as well!

6. Understanding Overfitting

Before we move any further we need to understand the concept of **Overfitting** in Machine Learning models.

What is ***overfitting?***

We are always looking to ensure that our models have a low bias with a low variance.

6. RESULT AND CONCLUSION

Pytorch has changed the image classification and made the image resolution to be deeper resolved than earlier libraries.

7. REFERENCES

https://colab.research.google.com/github/pranjalchaubey/Deep-Learning-Notes/blob/master/PyTorch%20Image%20Classification%20in%202020/Image_Classification_practice.ipynb#scrollTo=hakTX49k5wnW