A Report on

Image Classification using CNN

Dual Degree

COMPUTER SCIENCE AND ENGINEERING

By

Rajnish Kumar Ranjan

!5JE001619

SESSION: 2017-2018

UNDER THE GUIDANCE OF

Dr. Rajendra Pamula

Assistant Professor

DEPARTMENT OF PETROLEUM ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY

(INDIAN SCHOOL OF MINES)

DHANBAD-826004

Acknowledgements

It gives me immense pleasure to extend my thanks to those who provided inestimable support for the completion of my work.

In the first place, I would like to record my gratitude to Dr. Rajendra Pamula for his supervision, advice, guidance from the very early stage of this project and for patiently guiding me. I gratefully acknowledge him for his crucial contributions, which made him a backbone of this project. I am also thankful to Mr. Praphula Kumar Jain for his invaluable support and encouragement and providing time to time guidance during this period.

I would like my parents, for their faith in me and allowing me to be as ambitious as I wanted. Further, I would like to thank all the people who have directly or indirectly helped me in completing this project.

Rajnish Kumar Ranjan
Computer Science & Engineering.
IIT(ISM) Dhanbad

Abstract

There is a strong resurging interest in the neural-network-based learning because of its superior performance in many speech and image/video understanding applications nowadays.

In this project, an attempt is made to classify the images of two or more kinds with best accuracies using Convolutional Neural Network. In just 2 epochs of training 4000 data, we get the accuracy of 93%. The data-set consists of 4000 dogs' images and 4000 cats' images.

The implementation has been done in python using tensorflow library in Annaconda.

Table of Contents

Introduction

There are two common neural network architectures: the convolutional neural networks (CNNs) and the recurrent neural networks (RNNs). CNNs are used to recognize visual patterns directly from pixel images with variability. RNNs are designed to recognize patterns in time series composed by symbols or audio/speech waveforms. Both CNNs and RNNs are special types of multilayer neural networks. They are trained with the back-propagation algorithm. We will focus on CNNs in this work.

It is worthwhile to point out that the CNN is a special form of the feed-forward neural network (FNN), also known as the multi-layer perceptron (MLP), trained with back-propagation. Convolutional neural networks (CNN) have been widely used in automatic image classification systems. In most cases, features from the top layer of the CNN are utilized for classification; however, those features may not contain enough useful information to predict an image correctly. In some cases, features from the lower layer carry more discriminative power than those from the top.

The Architecture

A CNN usually takes an order 3 as its input, e.g., an image with H rows, W columns and 3 channels (R, G, B color channels). High order tensor inputs, however, can be handled by CNN in a similar fashion. The input then sequentially goes through a series of processing. One processing step is usually called a layer, which could be a convolutional layer, a loss layer, a fully connected layer etc.

Let us give an abstract description of the CNN structure first.

$$x^1 \rightarrow |w^1| \rightarrow x^2 \rightarrow \ \dots\dots \rightarrow x^{L-1} \rightarrow |w^L| \rightarrow z$$

The input is x^1, usually an image (order 3 tensor). It goes through the processing in the parameters involved in the first layer's processing collectively as a tensor w^1. The output of the first layer is x^2 , which also acts as the input to the second layer processing.

This processing proceeds till all the layers in the CNN has been finished, which outputs x^{L}. One additional layer z, is added for backward error propagation, a method that learns good parameter values in the CNN. A commonly used strategy is to output x^L a probability mass function, we can set the processing in the (L-1)-th layer as a softmax transformation of x^{L-1} (the distance metric and data transformation note). In other applications, the output x^L may have other forms and interpretations.

Let's suppose t is the corresponding target value for the input x^L, then a cost function can be used to measure the discrepancy between the CNN prediction x^L and the target **t.**

A simple loss function could be

$$z = \frac{1}{2}||t - x^L||^2$$

Layers of a CNN

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function. A few distinct types of layers are commonly used.

Convolutional layer

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.

Pooling layer

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which *max pooling* is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance.

The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

ReLU layer

ReLU is the abbreviation of Rectified Linear Units. This layer applies the non-saturating activation function $\boldsymbol{f}(\mathbf{x})=\mathbf{max}(\mathbf{0},\mathbf{x})$. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent $\boldsymbol{f}(\mathbf{x}) = \mathbf{tanh}(\mathbf{x})$, $(\boldsymbol{x}) = |\mathbf{tanh}(\boldsymbol{x})|$, and the *sigmoid function* $\boldsymbol{f}(\mathbf{x}) = (\mathbf{1} + \boldsymbol{e}^{-\boldsymbol{x}})^{-\mathbf{1}}$.

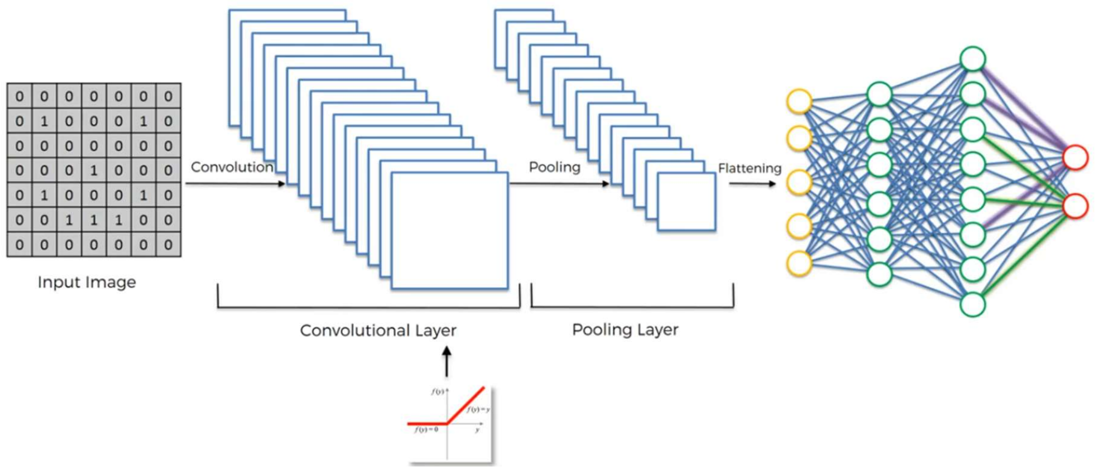
ReLU is often preferred to other functions, because it trains the neural network several times faster without a significant penalty to generalisation accuracy.

Fully connected layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Loss layer

The loss layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there. Softmax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in[0,1]. Euclidean loss is used for regressing to real-valued labels$(-\infty, \infty)$.



Motivation

Over the past few years, deep learning techniques have enabled rapid progress in this competition, even surpassing human performance. Today we're going to review that progress to gain insight into how these advances came about with deep learning, what we can learn from them, and where we can go from here.

The challenge of Convo-Net

So what's so hard about the Convo-Net challenge? Lets start by taking a look at the data. The data for the Convo-Net classification task was collected from Flickr and other search engines, manually labeled by humans with each image belonging to one of 1000 object categories/classes. The distribution of the data set is shown below in the table.

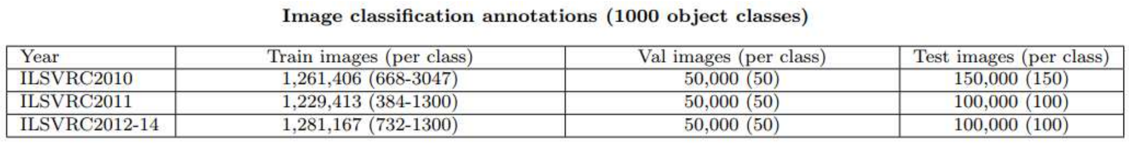
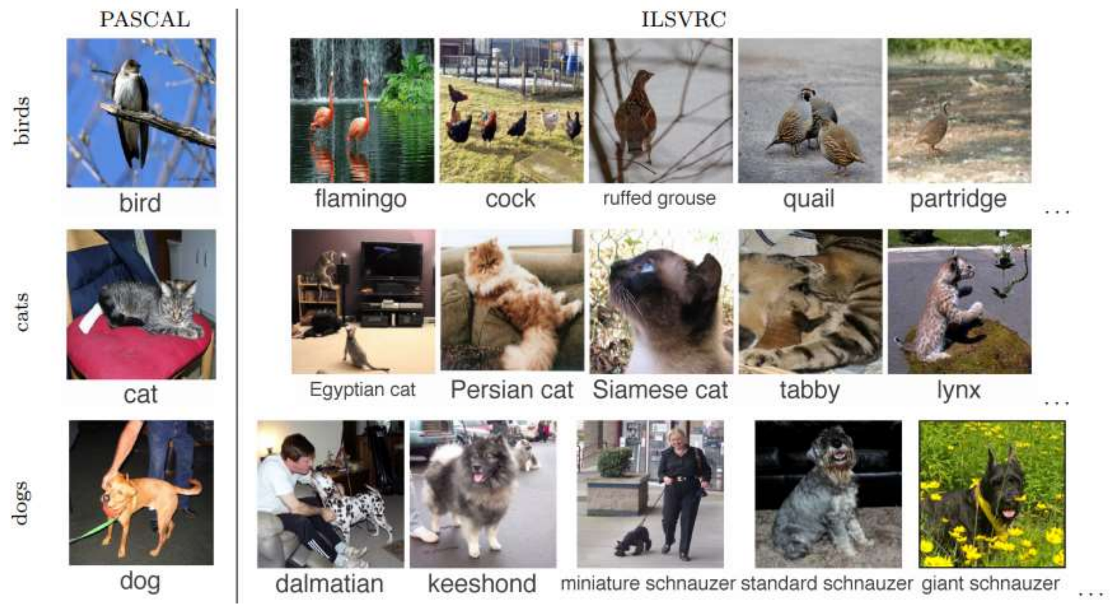


Image classification annotations (1000 object classes)

Year	Train images (per class)	Val images (per class)	Test images (per class)
ILSVRC2010	1,261,406 (668-3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384-1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732-1300)	50,000 (50)	100,000 (100)

In more technical terms, we want to maximize the **inter-class variability.**This means that we want two images each containing a different kind of bird to look very different to our model, since even though they are both birds, in our data set they are in different categories.

Check out the image below. On the left we see some example images from another image classification challenge: **PASCAL**. In the PASCAL challenge, there were only about 20,000 training images and 20 object categories. That challenge had quite generic class categories like "bird", "dog", and "cat" as depicted below. Shift over to the Convo-Net challenge and it's a whole new ball game. Instead of having a general class called "dog" that encompasses all kinds of dog, Convo-Net has classes for each dog species. In fact, instead of the PASCAL "dog" category, Convo-Net has 120 categories for the different breeds of dogs! Thus, any model/algorithm that we use for this task must be able to handle these very **fine-grained** and **specific classes**, even though they may look very similar and are hard to distinguish.



In more technical terms, we want to maximize the **inter-class variability.** This means that we want two images each containing a different kind of bird to look very different to our model, since even though they are both birds, in our data set they are in different categories.

Related Work

Nearly every year since 2012 has given us big breakthroughs in developing deep learning models for the task of image classification. Due to, it is large scale and challenging data, the Convo-Net challenge has been the main benchmark for measuring progress. Here we're going to take a look at the progress of deep learning on this task and some of the major architectures that made that progress possible.

The one that started it all: AlexNet

The paper proposed to use a deep Convolutional Neural Network (CNN) for the task of image classification. It was relatively simple compared to those that are being used today. The main contributions that came from this paper were:

- The first to successfully use a deep for large scale image classification. This was made possible because of the **large amounts** of **labelled data** from ImageNet, as well as training the model using parallel computations on two GPUs.

- They used **ReLU** for the **non-linearity activation functions**, finding that they performed better and decreased training time relative to the tanh function. The ReLU non-linearity now tends to be the default activation function for deep networks.

VGGNet

The VGGNet paper "Very Deep Convolutional Neural Networks for Large-Scale Image Recognition" came out in 2014, further extending the ideas of using a deep networking with many convolutions and ReLUs. Their main idea was that you didn't really need any fancy tricks to get high accuracy. Just a deep network with lots of small 3x3 convolutions and non-linearities will do the trick! The main contributions of VGGNets are:

- The use of **only** 3x3 sized filters instead of the 11x11 used in AlextNet. They show that two successive 3x3 convolutions has the equivalent **receptive field** or "field of view" (i.e the pixels it sees) as a single 5x5 convolution; similarily, three successive 3x3 convolutions is equivalent to a single 7x7. The advantage of this is that it simulates a larger filter while keeping the benefits of smaller filter sizes. The first benefit of smaller filters is a decrease in the number of parameters. The second is being able to use a ReLU function inbetween each convolution, that introducing more **non-linearity** into the network which makes the **decision function** more **discriminative**.

- As the spatial size of the input volumes at each layer decrease (as a result of the pooling layers), the depth of the volumes increase. The idea behind this is that as the spatial information decreases (from the downsampling down by max pooling), it should be encoded as more **discriminativefeatures** to use for accurate and highly discriminative classification. Thus the number of feature maps is increased with depth to be able to capture these features to be used for classification.

- It introduced a new kind of data augmentation: scale jittering.

- Built model with the Caffe toolbox. At this point deep learning libraries are becoming more and more popular.

DATA-SET

Data-set consists of training set and test set where there are 3000 images of dogs as well as cats in training set and 1000 dogs as well as cats in test set.

The link where, the data set can be found is http://www.superdatascience.com/wp-content/uploads/2017/04/Convolutional_Neural_Networks.zip

Model Formulation

Different Stages of Implementation

Building the CNN

Step 1 – Convolution

Step 2 – Pooling

Step 3 – Flattening

Step 4 - Full connection

Compiling the CNN (Fitting the CNN to the images)

Python Code for training the CNN using data

```
# Convolutional Neural Network


# Installing Theano

# pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git


# Installing Tensorflow

# pip install tensorflow


# Installing Keras

# pip install --upgrade keras


# Part 1 - Building the CNN


# Importing the Keras libraries and packages
```

```
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense

# Initialising the CNN
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(32, (3, 3), input_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Adding a second convolutional layer
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))
classifier.add(MaxPooling2D(pool_size = (2, 2)))

# Step 3 - Flattening
classifier.add(Flatten())

# Step 4 - Full connection
classifier.add(Dense(units = 128, activation = 'relu'))
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/training_set',
                                                 target_size = (64, 64),
                                                 batch_size = 32,
                                                 class_mode = 'binary')
classifier.fit_generator(training_set,
                         steps_per_epoch = 8000,
                         epochs = 3,
                         validation_data = test_set,
                         validation_steps = 2000)
```

Python Code for predicting whether an image is of cat or dog

```
# Part 3 - Making new predictions

import numpy as np
from keras.preprocessing import image
test_image = image.load_img('dataset/single_prediction/cat_or_dog_1.jpg', target_size = (64,
64))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis = 0)
result = classifier.predict(test_image)
training_set.class_indices
if result[0][0] == 1:
    prediction = 'dog'
```

```
else:
    prediction = 'cat'
```

REFERENCES

- Yann LeCun et al., 1998, ***Gradient-Based Learning Applied to Document Recognition***
- Jianxin Wu, 2017, **Introduction to Convolutional Neural Networks**
- C.-C. Jay Kuo, 2016, **Understanding Convolutional Neural Networks with A Mathematical Model**
- Kaiming He et al., 2015, **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**
- Dominik Scherer et al., 2010, **Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition**
- Adit Deshpande, 2016, **The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3)**
- Rob DiPietro, 2016, **A Friendly Introduction to Cross-Entropy Loss**
- Peter Roelants, 2016, **How to implement a neural network Intermezzo 2**
- Soumith benchmarks for CONV performanceConvNetJS CIFAR-10 demo allows you to play with ConvNet architectures and see the results and computations in real time, in the browser.
- Caffe, one of the popular ConvNet libraries.
- State of the art ResNets in Torch7