Master Thesis

Gabriele Angeletti

April 6, 2017

Acknowledgements

Acknowledgements section.

Abstract

Abstract Saenko et al. 2010, yeah Yoo et al. 2016, moar cite Selvaraju et al. 2016
spp

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Background

This chapter is a brief overview In this section, we'll review some basic concepts about deep learning

2.1 Domain Adaptation

Problem of dealing with situations in which the training distribution is different from the test distribution. We assume that the source (train) data is abundant and labeled, while the target (test) data is only partially labeled. The joint distribution of data and labels is, of course, unknown. $P_s(X, Y)$ is the joint distribution of source data, while $P_t(X, Y)$ is the joint distribution of the target data. In a standard classification problem, we assume $P_s(X, Y) = P_t(X, Y) = P(X, Y)$. In Domain Adaptation instead, we have $P_s(X, Y) \neq P_t(X, Y)$. Marginal Distributions: $P_s(X)$, $P_s(Y)$, $P_t(X)$, $P_t(Y)$. Conditional Distributions: $P_s(X|Y)$, $P_s(Y|X)$, $P_t(X|Y)$, $P_t(Y|X)$. If target has labels, we call the problem Supervised Domain Adaptation. If target is unlabeled, we refer to it as Unsupervised Domain Adaptation. One Approach to domain adaptation is that of representation learning, i.e. build a mapping from source representations to target representations so that $P_s(X)$ approaches $P_t(X)$. Many algorithms work with fixed length representations, i.e. they extract (say fc7) features from S and T and then they learn a mapping from one vector to the other in order to near the two representations. Other algorithms instead performs both representation learning and domain adaptation in the same end-to-end learning architecture. Different Approaches to this matching of the feature space distributions are: reweighing the sample in the source domain, find an explicit geometric transformation, modifying the feature representation itself.

2.2 Deep Learning

2.2.1 Feed-Forward Neural Networks

2.2.2 Convolutional Neural Networks

Chapter 3

Related Work

3.1 Domain Adaptation approaches

3.1.1 Geometric transformations

3.1.2 Representation learning

3.2 DANN

3.3 AutoDIAL

Chapter 4

Approach

4.1 Grad-CAM

Gradient-weighted Class Activation Mapping Selvaraju et al. 2016. Technique for producing visual explanations about why a Convolutional Neural Network opted for a certain class. The technique is applicable to a wide variety of tasks, ranging from image captioning to visual question answering to reinforcement learning, and also to a wide variety of CNN architectures. In the context of this work, we'll use this technique to produce visuals for an object classification task. This technique was designed as a visualization tool to better understand how CNNs reason and to improve their interpretability, which is a fundamental property to have for AI systems that wants to be deployed in the real world. We extracted the Grad-CAM visualization from the last convolutional layer because, as also the authors or the technique note, the last conv. layer is the best compromise between high-level semantics and spatial localization. The former is a general property of deep learning models: the further the layer from the input, the more high-level and meaningful features one can expect to be extracted. The latter is a property of Convolutional Feature Maps, which are designed to retain as much spatial information as possible about the data. The high-level architecture of the method can be seen in the following figure.

Basically, the input image is forward propagated through the network in order to compute the softmax scores of the last layer, which we call $y^c = f(x)$, where $f()$ is the input-output network mapping. If we were training the network, at this point we would have computed the gradients of the last layer neurons with respect to some loss function, like cross-entropy. Instead, in Grad-CAM, we manually create these gradients. In particular, we set them to one for the neuron corresponding to the class of interest, while all

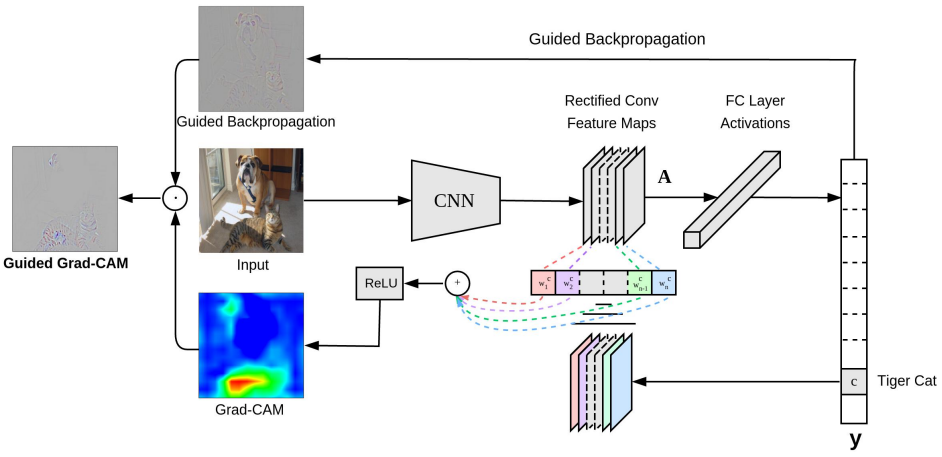

Figure 4.1: Grad-CAM Architecture, taken from Selvaraju et al. 2016

the other neurons are set to zero.

$$\frac{\partial y^c}{\partial \theta_{output}} = [0, 0, \ldots 1, 0, 0, \ldots]$$

Then, we backward propagate these gradients through the network untill the last convolutional layer, where the Grad-CAM map is created. Assume that the last conv. layer feature maps are in $\mathbb{R}^{K \times H \times W}$, where H and W are respectively the height and the width of each feature map, and K is the number of feature maps. The map that will be generated is thus in $\mathbb{R}^{H \times W}$. The generation procedure has multiple steps:

1. Compute the weight of each feature maps. We want a number measuring how much each filter contributed to the final score. It is therefore natural to compute this number as the derivative of the output gradients with respect to the weights of the filter. These derivatives are then global average pooled in order to get a single number:

$$w_k^c = \frac{1}{Z} \sum_{i=1}^{h} \sum_{j=1}^{w} \frac{\partial y^c}{\partial F_{i,j}^k} \tag{4.1}$$

   w_k^c is the weight of feature map k with respect to target class c, while F is the feature map itself. We can see that the derivative is positive if an increase of the value of the pixel $F_{i,j}^k$ yields an increase of the value of y^c.

2. The Grad-CAM map is obtained by performing a rectified weighted linear combination of the forward feature maps:

$$M^c = ReLU\left(\sum_{k=1}^{K} w_k^c F^k\right) \tag{4.2}$$

The $ReLU()$ function simply set to zero all the negative values. The Grad-CAM technique uses this function to retain only the values that have a *positive* influence on the class decision. Once the Grad-CAM map is generate, it is then upsampled to the pixel space using bi-linear interpolation, then a heatmap is generated from the values and the heatmap is point-wise multiplied with the output of the Guided-BackPropagation procedure Springenberg et al. 2014, to produce the final result Guided-GradCAM.

4.1.1 Grad-CAM for domain localization

4.2 Spatial Pyramid Pooling

Convolutional Networks can be thought of as composed by two parts, a feature extractor and a classifier. The former is composed by convolutional layers, while the latter by fully-connected layers, that is simply an affine mapping followed by some non-linearity. Each convolutional layer has three sequential stages: a convolution operation, a pooling operation, and a non-linearity operation. The pooling operation is needed to reduce the dimensionality of the data and to improve robustness with respect to input distortion. Usually, Max Pooling is used. Max pooling is a sliding window in which for each window, the pixel with the max value is taken as output. Spatial Pyramid Pooling is a method that was used extensively in computer vision algorithms before the deep learning revolution. This paper He et al. 2014 introduced the technique in the context of deep networks. Basically, a Spatial Pyramid Pooling operation it's nothing but multiple Max Pooling ops performed at different window sizes and then concatenated together. This seemingly simple alteration of the standard pooling layer has profound implications, such that:

- CNNs with standard Max Pooling layers must take in input images of fixed size (say 224×224). With SPP instead, the network can take in input images of arbitrary sizes, scales and aspect ratios.
- The network is much more robust to distortions and alterations of the input, because the pooling is done at multiple levels.

- Classification accuracy is increased.

4.3 Domain-Multiplicative Fusion

4.3.1 Architecture Design

Chapter 5

Experiments

5.1 Office 31

5.1.1 Dataset Statistics

5.1.2 Training methodology

5.1.3 Testing methodology

5.1.4 Results

5.2 ICW

5.2.1 Dataset Statistics

5.2.2 Training methodology

5.2.3 Testing methodology

5.2.4 Results

5.3 LookBook

5.3.1 Dataset Statistics

5.3.2 Training methodology

5.3.3 Testing methodology

5.3.4 Results

Chapter 6

Comparison

Chapter 7

Conclusions

7.1 Lessons learned

7.2 Future work