Car Damage Analysis and Insurance Claim System

Gaurav Gunjal, Harshada Sapkal, Pooja Kulkarni

Abstract— *In this paper we consider the problem of car damage classification, where some of the categories can be fine-granular. We explore deep learning based techniques for this purpose. Initially, we try directly training a Convolute Neural Network. However, due to small set of labeled data, it does not work well. Then, we explore the effect of domain-specific pre-training followed by transfer learning. Experimental results show that transfer learning works better than convolutional neural network.*

Keywords— CNN,Transfer learning, Insurance.

1.CNN

Image classification is an important topic in artificial vision systems, and has drawn a significant amount of interest over the last decades. This field aims to classify an input image based on visual content. Currently, most researchers have relied on hand-crafted features, HoG[3] or SIFT[5]to describe an image in a discriminative way. After that, learnable classifiers, such as SVM, random forest and decision tree are applied to extracted features to make a final decision. However, when a lot of images are given, it is too difficult problem to find features from those. This is the one of reasons that deep neural network model is coming. A few years ago, Hinton et al. revealed the fascinating performance of deep belief nets, which use an effective deep learning algorithm, contrastive divergence (CD)[10], in which each layer is trained layer by layer. Owing to deep learning, it becomes feasible to represent the hierarchical nature of features using many layers and corresponding weights. However, when the input dimension is too large to use, the deep belief network takes a long time to train. At that time, CNN, sharing weights by convolution method, solved this problem and improved the classification performance for various datasets. It should be noted that all those studies mentioned above have only used features from the top layer to train the following fully-connected layers[7]. In contrast to this approach, Pierre et al. bridged between the lower layer's output and the classifier to take the global shape and local details into account. This use of multi -stage features improved the accuracy over systems that use single stage features on a number of tasks, such as in pedestrian detection and certain sorts of classification. Motivated by many advantages of the multi-layers features, we propose an alternative multistage strategy that can be applied to a standard one track CNN whose weight parameter is fixed after the training has been finished without the multi-stage strategy in mind. The experiment results show that our approach can further improve performance of a standard one track CNN.

1.1.OVERVIEW OF CNN ARCHITECTURE

CNNs are feedforward networks in that information flow takes place in one direction only, from their inputs to their outputs. Just as artificial neural networks (ANN) are biologically inspired, so are CNNs. The visual cortex in the brain, which consists of alternating layers of simple and complex cells motivates their architecture[6]. CNN architectures come in several variations; however, in general, they consist of convolutional and pooling (or subsampling) layers, which are grouped into modules. Either one or more fully connected layers, as in a standard feedforward neural network, follow these modules. Modules are often stacked on top of each other to form a deep model. Figure 1 illustrates typical CNN architecture for a toy image classification task. An image is input directly to the network, and this is followed by several stages of convolution and pooling. Thereafter, representations fromthese operations feed one or more fully connected layers. Finally, the last fully connected layer outputs the class label. Despite this being the most popular base architecture found in the literature, several architecture changes have been proposed in recent years with the objective of improving image classification accuracy or reducing computation costs.

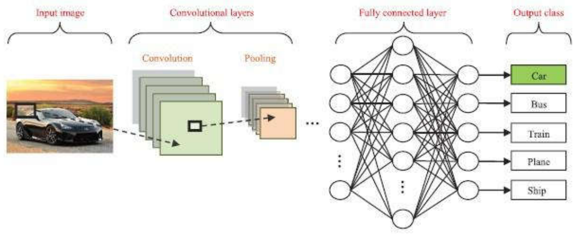

Fig 1: Overview of CNN architecture

1.2. CONVOLUTIONAL LAYERS

The convolutional layers serve as feature extractors, and thus they learn the feature representations of their input images. The neurons in the convolutional layers are arranged into feature maps[8]. Each neuron in a feature map has a receptive field, which is connected to a neighborhood of neurons in the previous layer via a set of trainable weights, sometimes referred to as a filter bank. Inputs are convolved with the learned weights in order to compute a new feature map, and the convolved results are sent through a nonlinear activation function. All neurons within a feature map have weights that are constrained to be equal; however, different feature maps within the same convolutional layer have different weights so that several features can be extracted at each location. More formally, the kth output feature map Yk can be computed as :

$Yk = f(Wk * x)$

where the input image is denoted by x; the convolutional filter related to the kth feature map is denoted by Wk; the multiplication sign in this context refers to the 2D convolutional operator, which is used to calculate the inner product of the filter model at each location of the input image; and f(·) represents the nonlinear activation function. Nonlinear activation functions allow for the extraction of nonlinearfeatures[3]. Traditionally, the sigmoid and hyperbolic tangent functions were used; recently, rectified linear units have become popular. Their popularity and success haveopened up an area of research that focuses on the development and application of novel DCNN[9] activation functions to improve several characteristics of DCNN performance.

1.3. POOLING LAYERS

The purpose of the pooling layers[1] is to reduce the spatial resolution of the feature maps and thus achieve spatial invariance to input distortions and translations. Initially, it was common practice to use average pooling aggregation layers to propagate the average of all the input values, of a small neighborhood of an image to the next layer. However, in more recent models, max pooling aggregation layers propagate the maximum value within a receptive field to the next layer. Formally, max pooling selects the largest element within each receptive field such that

$Ykij = \max \ xkpq$

$(p,q) \in i\ j$

where the output of the pooling operation, associated with the kth feature map, is denoted byYkij , xkpqdenotes the element at location (p, q) contained by the pooling region _i j, which embodies a receptive field around the position (i, j). Figure illustrates the difference between max pooling and average pooling. Given an input image of size 4 × 4, if a 2 × 2 filter and stride of two is applied, max pooling outputs the maximum value of each 2 × 2 region, while average pooling outputs the average rounded integer value of each subsampled region.

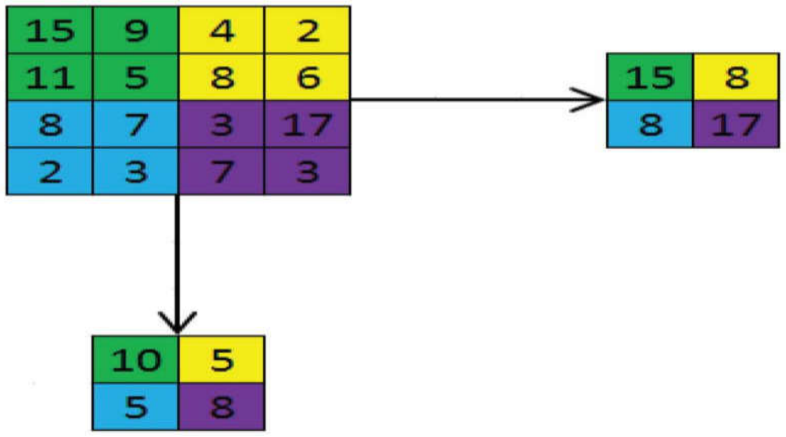

Figure 2 - Average versus max pooling.

1.4. FULLY CONNECTED LAYER

Several convolutional and pooling layers are usually stacked on top of each other to extract more abstract feature representations inmoving through the network. The fully connected layers[1]that follow these layers interpret these feature representations and perform the function of high-level reasoning. For classification problems, it is standard to use the softmax[5] operator on top of a DCNN.

While early success was enjoyed by using radial basis functions (RBFs), as the classifier on top of the convolutional towers, Tang found that replacing the softmax operator with a support vector machine (SVM) leads to improved classification accuracy. Moreover, given that computation in the fully connected layers is often challenged by their compute-to-data ratio, a global average -pooling layer, which feeds into a simple linear classifier, can be used as an alternative. Notwithstanding these attempts, comparing the performance of different classifiers on top of DCNNs still requires further investigation and thus makes for an interesting research direction.

1.5. TRAINING

CNNs, and ANNs in general use learning algorithms to adjust their free parameters (i.e., the biases and weights) in order to attain the desired network output. The most common algorithm used for this purpose is backpropagation. Backpropagation computes the gradient of an objective (also referred to as a cost/loss/performance) function to determine how to adjust a network’s parameters in order to minimize errors that affect performance. A commonly experienced problem with training CNNs, and in particular DCNNs, is overfitting,which is poor performance on a held-out test set after the network is trained on a small or even large training set. This affects the model’s ability to generalize on unseen data and is a major challenge for DCNNs that canbe assuaged by regularization.

2. TRANSFER LEARNING

Data mining and machine learning[1] technologies have already achieved significant success in many knowledge engineering areas including classification, regression and clustering. However, many machine learning methods work well under a common assumption: the training data and test data are drawn from the same distribution and the same feature space. When the distribution changes, most machine learning methods need to start from scratch, requiring users to collect a lot of training data again. In many real world applications, it is expensive to re- collect the needed training data and re-train models. It would be nice to reduce the need to recollect and reliable the training data. In such cases, knowledge transfer or transfer learning between task domains would be necessary.

Transfer learning[10] involves the approach in which knowledge learned in one or more source tasks is transferred and used to improve the learning of a related target task. While most machine learning algorithms are designed to address single tasks, the development of algorithms that facilitate transfer learning is a topic of ongoing interest in the machine-learning community.

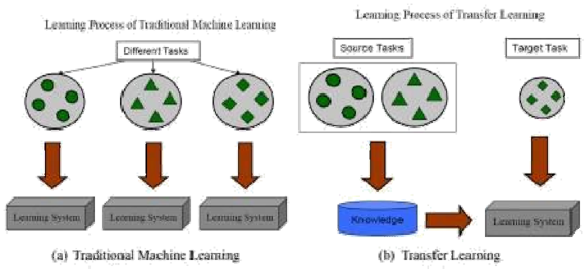

Figure 3 – Transfer learning

2.1. WHY TRANSFER LEARNING ?

Many deep neural networks trained on natural images exhibit a curious phenomenon in common: on the first layer they learn features similar to Gabor filters and color blobs[12]. Such first-layer features appear not to specific to a particular dataset or task but are general in that they are applicable to many datasets and tasks. As finding these standard features on the first layer seems to occur regardless of the exact cost function and natural image dataset, we call these first-layer features general. For example, in a network with an N-dimensional softmax output layer that has been successfully trained towards a supervised classification objective, each output unit will be specific to a particular class. We thus call the last-layer features specific.

In transfer learning we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, that is, suitable to both base and target tasks, instead of being specific to the base task.

In practice, very few people train an entire Convolutional Network from scratch because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pre- train a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet[10] either as an initialization or a fixed feature extractor for the task of interest.

2.2. TRANSFER LEARNING SCENARIOS

Depending on both the size of the new dataset and the similarity of the new dataset to the original dataset, the approach for using transfer learning will be different. Keeping in mind that ConvNet features are more generic in the early layers and more original-dataset specific in the later layers, here are some common rules of thumb for navigating the four major scenarios:

1. The target dataset is small and similar to the base trainidataset.

Since the target dataset is small, it is not a good idea to fine-tune the ConvNet due to the risk of overfitting. Since the target data is similar to the base data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, we:

◦ Remove the fully connected layers near the end of the pretrained base ConvNet

◦ Add a new fully connected layer that matches the number of classes in the target dataset

◦ Randomize the weights of the new fully connected layer and freeze all the weights from the pre-trained network

◦ Train the network to update the weights of the new fully connected layers

2. The target dataset is large and similar to the base training dataset. Since the target dataset is large, we have more confidence that we won't overfit if we try to fine-tune through the full network. Therefore, we:

◦ Remove the last fully connected layer and replace with the layer matching the number of classes in the target dataset

◦ Randomly initialize the weights in the new fully connected layer

◦ Initialize the rest of the weights using the pre-trained weights, i.e., unfreeze the layers of the pre-trained network

◦ Retrain the entire neural network

3. The target dataset is small and different from the base training dataset.

Since the data is small, overfitting is a concern. Hence, we train only the linear layers. But as the target dataset is very different from the base dataset, the higher level features in the ConvNet would not be of any relevance to the target dataset. So, the new network will only use the lower level features of the base ConvNet. To implement this scheme, we:

◦ Remove most of the pre-trained layers near the beginning of the ConvNet

◦ Add to the remaining pre-trained layers new fully connected layers that match the number of classes in the new dataset

◦ Randomize the weights of the new fully connected layers and freeze all the weights from the pre-trained network

◦ Train the network to update the weights of the new fully connected layers

4. The target dataset is large and different from the base training dataset. As the target dataset is large and different from the base dataset, we can train the ConvNet from scratch. However, in practice, it is beneficial to initialize the weights from the pre-trained network and fine-tune them as it might make the training faster. In this condition, the implementation is the same as in case 3.

CONCLUSION:

We have trained and tested more than 3000 images on both the approaches i.e. Convolutional Neural Network and Transfer Learning. Using CNN, trained model gave accuracy approximately 79%, where model trained with transfer Learning approach gave accuracy approximately 93%. Accuracy has improved drastically in Transfer Learning approach. Also, the number of trainable parameters in the transfer model is low as compared to our scratch model. Apart from this, the CNN scratch model took around 18 hours to train on CPU, while the transfer model took less than 6 hours to train the model on the same dataset. We can conclude that the use of transfer learning not only improves the performance of the model but also is computationally efficient.

REFERENCES

1. Waseem Rawat , Zenghui Wang, "Deep Convolutional Neural Networks ffor Image Classification," Neural Computation,Volume 29, Issue 9 , September 2017 ,p.2352-2449
2. Manuel Lagunas, Elena Garces , "Transfer Learning for Illustration Classification", W Rawat, Z Wang – Neural Computation, 2017.
3. Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, "Learning activation functions to improve deep neural networks" Submitted on 21 Dec 2014 (v1), last revised 21 Apr 2015.
4. Bachman,"An architecture for deep, hierarchical generative models", Philip Bachman submitted on 8 Dec 2016.
5. Belkin, M., Niyogi, P., & Sindhwani, "Manifold regularization: A geometric framework for learning from labeled and unlabeled examples". Journal of Machine Learning Research on 12 Jan 2006 .
6. Bulo, S., & Kontschieder, "Neural decision forests for semantic image labelling" – Procedings of IEEE 2014.
7. Cheng, Z., Soudry, D., Mao, Z., & Lan, " Training binary multilayer neural networks for image classification using expectation back propagation"(2015).
8. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... Wang, "Recent advances in convolutional neural networks"(2015)
9. HE K., ZHANG X., REN S., SUN J."Deep residual learning for image recognition"
10. OQUAB M., BOTTOU L., LAPTEV I., SIVIC J. "Learning and transferring mid-level image representations using convolutional neural networks". In Computer Vision and Pattern Recognition IEEE 2014.
11. LIN Y., LV F., ZHU S., YANG M., COUR T., YU K." Largescale image classification: Fast feature extraction and svm training NeuralComputation,Volume 29, Issue 9 , September 2017 .