Car Make and Model Recognition using Convolutional Neural Networks

Atanas Poibrenski*(2554135)*, Marimuthu Kalimuthu *(2557695)*

May 8, 2016

Abstract

In this project, we use convolutional neural network to predict the make and the model of a car from a front-view image. We adopt the transfer learning technique on the famous VGG16 network with pretrained weights on the "ImageNet" dataset. We achieve an impressively high accuracy of (>90%) on the test surveillance data of the "CompCars" dataset.

1) Introduction and Problem statement

Car recognition is an important part of the object recognition domain. It can have various applications such as security and law enforcement.

An example of an interesting application could be to mount a camera on the entrance of a parking lot of an office/company. It can then build a database of all the Make and Models of the cars of the employees through their entering and leaving. The data can then be analyzed to see who are the more frequent visitors as well as identify the unfamiliar ones. This can then be used to find suspicious activity.

Car recognition is not an easy task as face recognition, for example. Cars are more difficult to detect and recognize due to their varying structure *(color, shape, size etc.)* from different views of the car as well as between different Makes and Models.

The next section attempts to summarize what people have done in the field of car recognition in the years so far.

2) Related work

One of the earliest works on this topic was by *Thiang,Guntoro and Lim (2001)* [1]. They developed a method in which an image of car was compared to several image templates (with the same dimensions). The image will then have a similarity value for each template image. The type of the car is determined by the highest similarity value.

Cheung and Chu [2] propose a more advanced method which uses the idea of interest point detection (i.e. SIFT). They extract interesting points from all the images in the database. Then, when a query image comes, its interesting points are compared to the points of all the images in the dataset. The paired interest points are used to find a subset of inliers (with the help of RANSAC) which best fit to a given transformation model. The image with the highest inlier count will be labeled as being the best match to the query image.

Microsoft Research [3] propose a car recognition system combining global and local cues. They extract an edge map of each image and for each selected edge a global shape descriptor is computed. Another local shape descriptor is computed for edge points belonging to manually-annotated local parts. In the query phase, an edge map is first computed using a probabilistic boundary detector. The global shape descriptors are used to perform a registration of the query image to each template image, resulting into a dissimilarity measure. Local shape descriptors are also matched. The

query is then assigned to the class with the minimum weighted sum of global and local dissimilarity measures.

A more recent approach based on deep learning is proposed by *Gao and Lee* [4]. They use a three layer Restricted Boltzmann machines (RBM) for the car model recognition. A binary image of the frontal part of the car is used as input. The binary image is unrolled into a vector with dimension 2000. The three RBM layers are used as pretraining to obtain the initial weight. After that, they use a traditional back-propagation approach to fine-tune the deep network.

3) Convolutional Neural Networks

This section describes the classical architecture and training of a convolutional neural network. The next figure (LeNet-5 Yann LeCun [5]) shows the basic structure of such a network.

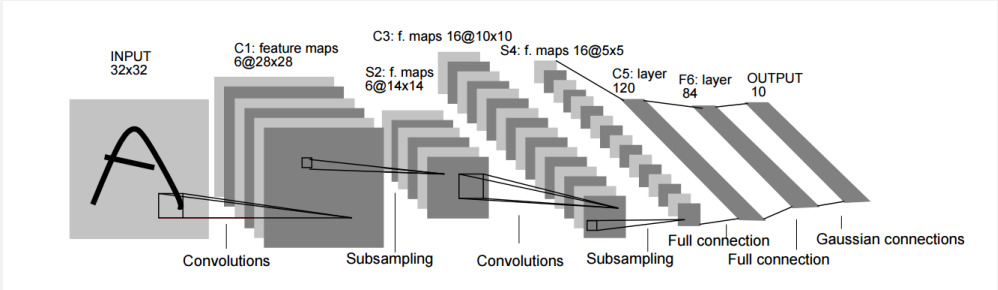

Figure 1: Architecture of LeNet-5, a Convolutional Neural Network

There are several blocks which should be discussed:

Input to the network: The input layer will hold the raw pixel values of the image. In this case it is of size 32x32. It can also hold color information by stacking different color channels (such as RGB).

Convolutional layer: This is the core building block of a convolutional neural network. This layer will compute the output of neurons which are connected to local regions in the previous input layer. The convolution is just a dot product between the weights of the convolutional kernel and the region it is connected to in the input. Usual kernel sizes are 3x3,5x5 and 7x7. The idea can be seen visually in the following figure: [6].

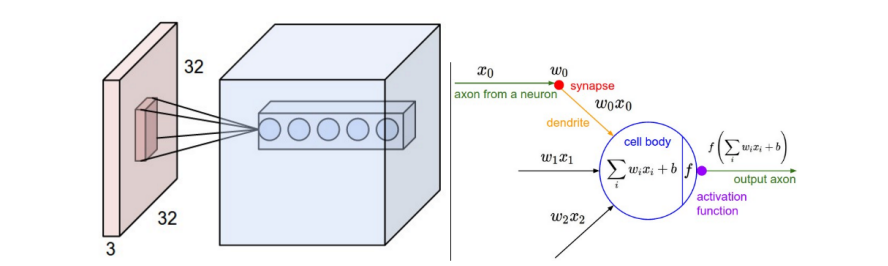

Figure 2: An example volume of neurons in Convolutional Layer

Usually, several such kernels are used in order to extract different features from the input. In figure 1, it can be seen that there are 6 such feature maps. A small detail about this convolutional layer is the so-called padding (e.g adding zeroes to the border of the input). If we don't use any padding then the size of the feature maps is smaller than the input (28x28 instead of 32x32,see figure 1). The convolutional kernels will be learned during the training (backpropagation stage).

Nonlinearity: The convolutional operation so far is a linear operation. In order to be able to represent more complex patterns we need to break this linearity. The most common non-linearity used after the convolutional layer is the Rectified Linear Unit *(ReLU)* f(x) = max(0,x) which is applied elementwise to the feature maps and leaves the output size unchanged. The ReLU has many nice properties, one of which is that it deals with the vanishing gradient problem.

Pooling: The next layer after the convolutional layer is the pooling layer. Its function is to reduce the spatial size of the input. This is done to reduce the trainable parameters of the network as well as to control overfitting. The pooling operation works independently on each convolutional feature map by applying the so-called MAX operation in a 2x2 region (usually). This operation results in decreasing the input by half, as it can be seen in figure 1.

Fully-connected layer: After a successive stacking of CONV and POOL layers, a fully-connected layer is usually used. It applies an affine transformation to its inputs as in a classical neural network. Mathematically, from 'n' inputs to 'h' outputs it works as the following: $f(X) = WX + b$, where W $\in \mathbb{R}^{n \times h}$ and b $\in \mathbb{R}^{h}$. The output therefore depends on the learnable matrix of parameters W as well as on the learnable bias b.

Dropout: Dropout is non-deterministic operation which is used in the training of modern deep neural networks. A dropout layer takes a number of input connections and sets them to zero with probability 'p' and leaves the others unchanged with probability '1-p'. The value 'p' is a hyperparameter for the network. The general idea of dropout can be interpreted as a form of regularizer. By dropping some connections randomly, the network finds a way to compute a result in many different ways rather than just relying heavily on a single connection.

Softmax layer: This is the last layer of the network which computes class scores which will be fed into the loss function. There are no learnable parameters in this layer, it is just a convenient way to compute class probabilities. Mathematically, the i^{th} class probability $f(x)_i$ is computed as follows:

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \qquad \textit{for i=1, ..., J}$$

Training the network: For the training of the network, the *Categorical Cross-Entropy* loss is usually used. It is defined as: -log(X_{y_i}), where X is the output of the softmax layer and y_i is the correct class label. Using the gradient of this loss function, all the parameters of the network mentioned earlier are updated using successive applications of the chain rule. Ultimately, we are interested in the gradient of the output of our network 'f' with respect to its input 'x'. The chain rule tells us that $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x}$ for some intermediate value q. Of course, for a convolutional neural network this expression is much more involved because we have to find derivatives of the non-linearities, pooling and the convolutions. For simplicity, these derivations are skipped as in practice people usually use some software packages for the computation of these derivatives (e.g Theano, Tensorflow). For the actual update of the parameters, Stochastic Gradient Descent is the most commonly used optimization method.

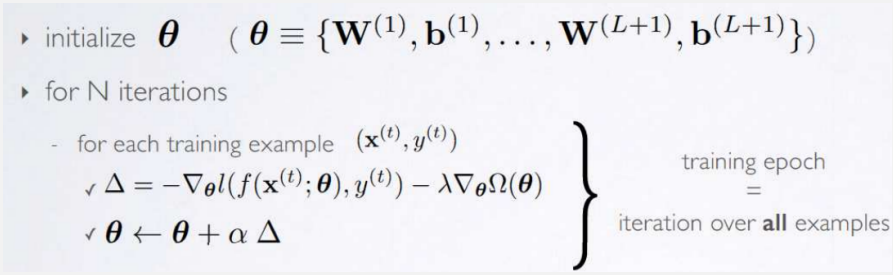

Figure 3: Pseudocode for SGD [from DSP lecture-16]

In practice, *SGD + Nesterov momentum* is used since it converges quicker than using only SGD. The idea of Nesterov momentum is that it keeps track of a variable 'v' that is a function of the magnitude of previous updates, allowing the learning rate to depend on this variable. This type of update is more tolerant to different values of learning rate and can adapt over time based on how quickly the parameters are changing.

4) Transfer Learning

Transfer learning is defined as the following: *"It is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned."* [7]. Training a convolutional neural network from scratch is rarely done in practice. Deeper networks almost always perform better on a given task. But in order to train such a deep network, a lot of training data, time, and resources are required. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g ImageNet which has 1.2 million images) and then use the network for an initialization for the task of interest. For most reasonable image classification tasks, the features learned in the convolutional layers in such a network will be roughly the same, regardless of the image dataset. This means that the features can be reused with some fine-tuning. Of course the classifier (e.g the fully-connected layer) has to be retrained for the specific task.

5) Our Method

The first thing we tried was to train a not so deep ConvNet from scratch with the following architecture and *SGD + Nesterov momentum* for the parameters update:

Input (224x224x3) → CONV(5x5, 32 filters) → POOL(2x2) → CONV(5x5, 32 filters) → POOL(2x2) → FC(256 units, p=0.5 dropout) → Softmax → Categorical Cross-Entropy

The training was relatively fast but the accuracy on the test set was not satisfactory (less than 75%). We then tried to increase the number of convolutional maps as well as the number of fully-connected(FC) units. The accuracy improved a little but the training time increased a lot. Then we switched to the more promising idea of transfer learning. The architecture we chose is *VGG16* because of its simplicity and publicly available pretrained weights for Theano. The network was trained on the ImageNet dataset. The network is shown in the next figure [8].

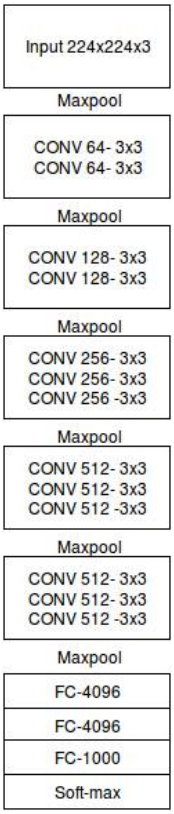

Figure 4: Architecture of VGG16

We then proceeded as follows:
1. Since the input to the network is 224x224x3 we had to resize all the images such that the minimum dimension is of size 256 and then we did a central crop of 224x224. We kept the original aspect ratio.
2. Subtract the mean channel values from all the training/testing images in our dataset.
3. We created the VGG16 architecture in our code and initialized the convolutional layers' weights with the pretrained weights. The fully-connected layers were initialized randomly. The number of classes were changed from 1000 to 281 to fit the number of classes of our dataset.
4. We used Stochastic Gradient Descent with Nesterov Momentum for the fine-tuning of the convolutional weights and training of the fully-connected layers. The learning rate was set to 0.0001 so that we don't change the pre-trained weights too much. The decay was set to $1e^{-6}$ and the momentum to 0.9. The biggest batch size we were able to fit into the GPU memory was 12.

6) Implementation details

Dataset: We downloaded the surveillance data from the CompCars dataset from `http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/index.html` . It is split into 70% training images and 30% testing images. There are a total of 44,481 images. Here are some sample images:

Figure 5: Some Car Models from the dataset

Software frameworks: Theano + Keras For installation, please see the README file
VGG16 pretrained weights: We downloaded the weights from here
`https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3`

Hardware Used: The training was done on a XMG p505 laptop with CUDA and CuDNN enabled. The laptop has nVidia GTX 970 and 3GB of GPU memory, 16 GB of RAM and i7 4720 CPU. The training took around 1 hour per epoch.

7) Results

The following figures show the results we got through the training process.

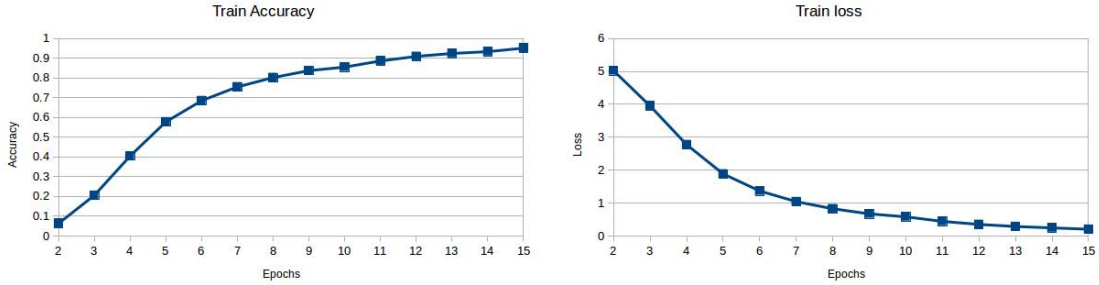

Figure 6: *Train Accuracy and Train Loss*

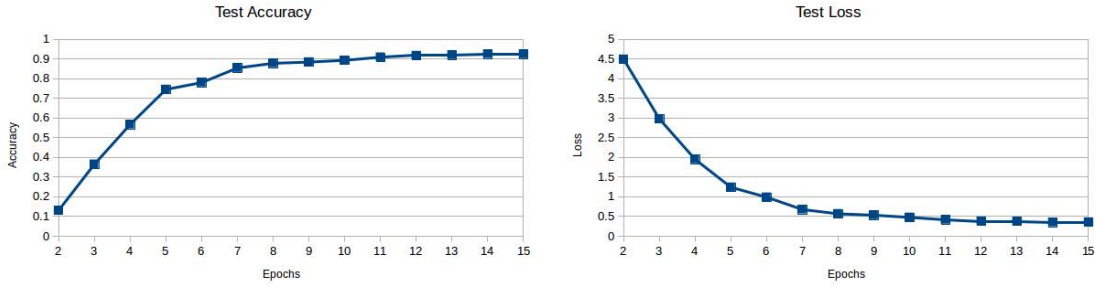

Figure 7: *Test Accuracy and Test Loss*

The training accuracy could increase even further but we stopped the training after the 15-th epoch since the network started to overfit, resulting into decrease in the test accuracy. We could squeeze a little extra test accuracy by adding a stronger regularizer which is discussed in the next section. The best test accuracy we were able to achieve was **92.5%**.

8) Conclusion and Future work

We achieved >90% accuracy on the surveillance data of the CompCars dataset by using transfer learning and the VGG16 ConvNet. The authors of the dataset were able to achieve an accuracy of 98.4% [9]. We believe that the result we obtained is reasonable for the limited amount of time and hardware we had. We propose three different methods which can help improve the accuracy for future work:

- Several models can be trained (e.g 5) and then the final prediction of the class label can be done with majority voting. Model ensembles usually improve the performance.

- Gather more training data through data augmentation. Augment the data through different transformations such as horizontal/vertical flip,shear,color shift, ZCA whitening, etc.
- Try bigger/deeper models such as VGG19 or an even more recent network such as ResNet [10]
- Try L2 or L1 regularization in order to penalize the learned weights which might prevent overfitting.

9) References

1)citeseerx.ist.psu.edu/viewdoc/download
2)https://cseweb.ucsd.edu/classes/wi08/cse190-a/reports/scheung.pdf
3)http://research.microsoft.com/pubs/168859/icpr12_0733_fi.pdf
4)http://onlinepresent.org/proceedings/vol90_2015/13.pdf
5)http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf
6)http://cs231n.github.io/convolutional-networks/
7)ftp://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf
8)http://arxiv.org/pdf/1409.1556.pdf
9)http://arxiv.org/pdf/1506.08959v2.pdf
10)http://arxiv.org/pdf/1512.03385v1.pdf