Emblem Detection Using Convolutional Neural Networks

Dinesh Maria Antony, Kanika Nama, Inchara Niyanta
{ packianathanjerome.d , nama.k, niyanta.i }@husky.neu.edu
INFO 7390, Summer 2018, Northeastern University

1. INTRODUCTION

Machine Learning has been a trending research and experimentation topic over the past few decades. It has been implemented to solve most day-to-day problems. However, it is still challenging when it comes to computer vision. To address this challenge Neural Networks were introduced. Neural network is a system of computer software/hardware that is patterned after the working of neurons in the human brain. Deep learning refers to a subdivision of Machine Learning, wherein a system of neural networks is used for learning from data that is unstructured or unlabeled. One of the most popular deep learning technique is a Convolutional Neural Network which is a class of deep, feed-forward artificial neural networks [1]. CNNs are commonly used for solving problems related to Computer Vision. This project involves creating a Convolutional Neural Network to identify the breed of a dog from its image. The dataset used for this project is the Stanford Dogs dataset [2].

2. RELATED WORK

Classifying a breed of a dog is an example of a fine-grained image classification problem. Fine grained classification is challenging due to subtle differences among the images pertaining to each class. Among the various approaches for performing fine-grained image classification, one approach stood apart. This approach has been consumed in this project and has been discussed briefly below:

Image classification plays an important role in computer vision, it has a very important significance in our study, work and life. Image classification is process including image preprocessing, image segmentation, key feature extraction and matching identification.

With the latest figures image classification techniques, we not only get the picture information faster than before, we apply it to scientific experiments, traffic identification, security, medical equipment, face recognition and other fields. During the rise of deep learning, feature extraction and classifier has been integrated to a learning framework which overcomes the traditional method of feature selection difficulties. The idea of deep learning is to discover multiple levels of representation, with the hope that high-level features represent more abstract semantics of the data. One key ingredient of deep learning in image classification is the use of Convolutional architectures.

[1] Propose an effective and scalable framework for recognizing logos in images which is based on a method for encoding and indexing the relative spatial layout of local features detected in the images with logos. Based on the analysis of the local features and other spatial structures we use an automatic method for constructing a model for each logo-class out of multiple training images. This allows them to detect logos under varying conditions such as perspective tilt.

[2] View the problem of logo recognition as an instantiation of the broader problem of object recognition. We divide the overall problem into four sub-problems: logo classification, logo localization, logo detection without localization, and logo detection with localization. Using this we are able to achieve 90% accuracy and establish the state of the art in logo recognition. This problem is also very similar to reading text in the wild like house numbers, for example. Thus, we can expect that approaches from that domain with a little adaptation would work very well for the task of logo recognition

[3] Use an approach as follows: word bounding box proposal generation, proposal filtering and adjustments, text recognition and final merging. The detection stage is done using the CNN object detection framework in which a region proposal is converted into a fixed size to which we can apply a CNN. This helps avoid the computationally expensive task of scanning the full image for text. Before applying the CNN, we use various classifiers to filter out the large number of false-positive region proposals. Once we have filtered out some of the region proposals and only retain ones that are highly likely to contain text, we apply whole-word approach for recognition by providing the entire region proposal to a deep convolutional neural network.

3. DATASET

The Flickr Logos 27 dataset is an annotated logo dataset downloaded from Flickr and contains more than four thousand classes in total. It consists of three image collections/sets.

The training set contains 810 annotated images, corresponding to 27 logo classes/brands (30 images for each class). All images are annotated with bounding boxes of the logo instances in the image. We allow multiple logo instances per class image. The training set is randomly split in six subsets, each one containing five images per class.

The distractor set contains 4207 logo images/classes, that depict, in most cases, clean logos. All images come from the Flickr group Identity + Logo Design. Each one of the distractor set images defines its own logo class and we regard the whole image as bounding box.

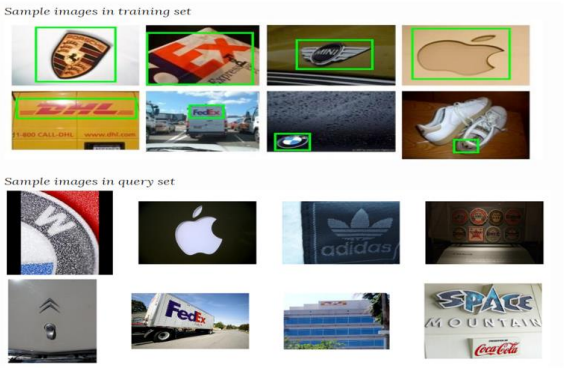


Figure 3.1 – Random Sample Images from the dataset

4. MODEL SELECTION

4.1 Pre-trained model:

The dataset is a subset of images from ImageNet. There are lot of models with pre-trained weights that are trained with all the images in ImageNet. Hence, these pretrained models can be used to obtain the bottleneck features from the images which are then used as input for the custom made fully-connected dense layer which classifies the image into respective classes.

The following are the pre-trained models that are available in KERAS library.

1. ResNet50
2. VGG16
3. Xception
4. Inception

In this project we have used all the above-mentioned models to classify the images.

The pre-trained models are used to obtain the bottle-neck features and then a densely connected layer is added to classify the images from the higher dimensional feature vector obtained from the pre-trained models. The model details are given below:

Pretrained Model	No. of Layers
ResNet50	50
VGG16	16
Xception	36
Inception	48

4.2 Our own ConvNet:

For this project, we constructed our own Convolution Neural Network to classify the images and compared the results obtained with those obtained from the pre-trained models.

The architecture of our Convolutional Neural Network is shown in figure 4.1.

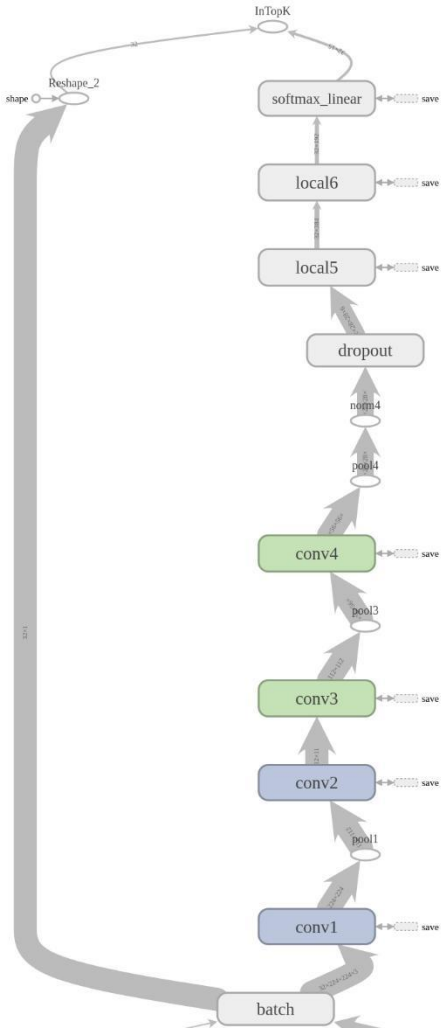


Figure 4.1 – CNN architecture

4.2.1 CNN Architecture:

Our CNN is stacked with the following layers. The functionality of each layer is briefly explained below:

- **Convolutional Layer:** Core building block of a Convolutional Neural Network that does most of the computational heavy lifting
- **Pooling Layer:** Progressively reduces the spatial size of the representation to reduce the number of parameters and computation in the network, and hence to also control overfitting.
- **Dense Layer:** comprises of the traditional fully connected neural network which classifies the input to their respective classes

The CNN is designed with the following parameters:

Input layer: (224,224,3)

224 refers to no. of pixels in each dimension and 3 refers to the RGB values (channels) for each pixel.

Conv-1: 64 filters of size 5 x 5
tanH activated

Pool-1: 2 x2 filter of stride 2
Max pool

Conv-2: 64 filters of size 5 x 5
tanH activated

Dropout: 60% of the neurons

SoftMax-Linear: fully-connected 120 neurons output

4.2.2 Salient features of the CNN model:

The layers are designed in the following way in our model:

1. The decaying weights are used for certain conv layers to prevent the weights from over shooting
2. The layers conv1 and conv2 use ordinary weights while the remaining conv layers use decaying weights
3. Less number of Pooling layers are used as too many pooling layers would result in feature loss
4. The layers except conv1 and conv2 uses decaying weights, due to which sparsity is introduced in these layers
5. Sparsity results in only few neurons in the layer being active, which effectively increases the speed and helps the CNN to consume less resources while training and evaluating the model
6. Biases are added to the convolution layers
7. tanH activation function is used for the conv layers and ReLU is used for the dense layers as tanH increased the accuracy of the model by 10% when trained with 6 classes.

5. METHODOLOGY

5.1 Processing images for the input:

For this project, we have designed our classifiers using both pre-trained models and our own ConvNet.
Note: The pre-trained models are high-level and have as many as 30 layers. They are also trained with millions of training samples before they are used in this project. Hence, the model using these pre-trained models will have very high accuracy when compared with our custom-made CNN model.

The following steps discuss about how the inputs(images) are processed and the model is trained.
The images of the emblem are placed in the folder containing its brand name. The whole list is populated in a data dictionary along with a field that specifies whether it is a training sample or test sample.

Step 1:
The images are cropped based on the bounding box specified in their corresponding annotations. This process replaces identifying the logo in each image to implement Part Localization.
Step 2:
These cropped images will have different dimensions which cannot be used for training the model.
Hence, all the images are reshaped to 64*64 pixels using crop-or-pad operation.

Step 3:
In order read the images quickly and to store them efficiently, we then encode the logos and their labels into binary format. The images are converted into their (r, g, b) values and each value is converted into binary. The labels (brand names) are one-hot encoded and then converted into binary value. The data is then stored in the following format in a binary file:

5.2 Preprocessing the images for the training:

In other words, the first byte will represent the label of the image which represent the pixel values of the image (64x64x3). The pixel values are placed in such a way that first 50,176 bytes represent the R values, followed by G values and then B values.
Distortions are introduced into the training set to avoid the model from getting overfitted. The images are randomly flipped horizontally. Then the brightness and contrast of the images are randomly increased or decreased to add more distortion.

This script loads the pre-trained module and trains a new classifier on top for the logo photos that was pre-processed. The transfer learning is that the lower layers that have been trained to distinguish between some objects can be reused for many recognition tasks without any alteration.

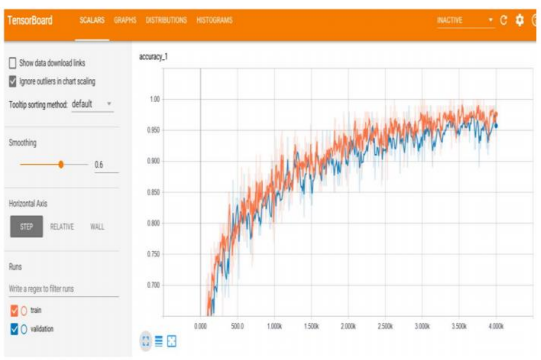
The first phase analyses all the images on disk and calculates and caches the bottleneck values for each of them. 'Bottleneck' is an informal term we often use for the layer just before the final output layer that does the classification. Because every image is reused multiple times during training and calculating each bottleneck takes a significant amount of time, it speeds things up to cache these bottleneck values on disk, so they don't have to be repeatedly recalculated.
The script will write out the new model trained on your categories to /tmp/output_graph.pb, and a text file containing the labels to /tmp / output_lables.txt. In-order to test our retrained model, we used lable_image.py provided by Inception V3. But the code required lot of changes before it started running.

6. RESULTS

We used keras as a wrapper for tensorflow for creating networks with pre-trained models. Hence, once the model is evaluated the output is printed which contains the results for that model.

In case of our CNN model, we used tensorflow directly to construct the model and train them. Hence, we created checkpoints frequently which contained all the data related to the model at that instant. Then we used tensorboard – a visualization tool for tensorflow to visualize various performance characteristics of our model along with the accuracy scores.

The model created a “tmp” on the root directory where the saved model and required files for supporting the training of the images generated. From the “retrain_logs”, we were able to visualize the accuracy.



Tensor Board graph for flickr-27 dataset after training

We used keras as a wrapper for TensorFlow for creating networks with pre-trained models. Hence, once the model is evaluated the output is printed which contains the results for that model.

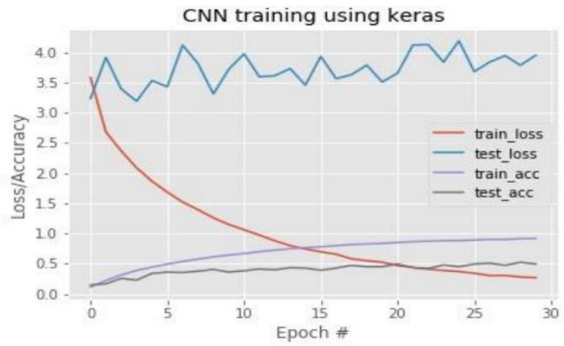
In case of our CNN model, we used TensorFlow directly to construct the model and train them. Hence, we created checkpoints frequently which contained all the data related to the model at that instant.

Then we used tensor board – a visualization tool for TensorFlow to visualize various performance characteristics of our model along with the accuracy scores.

```
def createModel():
    model = Sequential()
    model.add(Conv2D(64, 3, padding='same', activation='relu', input_shape=(64, 64, 3)))
    model.add(Conv2D(64, 3, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3),strides=2 ))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, 3, padding='same', activation='relu'))
    model.add(Conv2D(128, 3, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Dropout(0.25))

    model.add(Conv2D(256, 3, padding='same', activation='relu'))
    model.add(Conv2D(256, 3, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Dropout(0.25))

    model.add(Conv2D(512, 3, padding='same', activation='relu'))
    model.add(Conv2D(512, 3, activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=2))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(27, activation='softmax'))
    return model
```

Sample image: CNN model

Training all 16000 images the result produced by the model is below:



Accuracy and loss graph

7. CONCLUSION

We implemented a Convolution Neural Network to solve a fine-grained classification problem. We achieved 35% Top 3 accuracy among 15 classes and 15.4 % Top 10 accuracy among 120 classes. Using tensorboard, various hyper-parameters of the model and its effects on the performance of the model are studied. However, having just over 100 images per class for a fine-grained image classification problem is not enough for us to train our model completely to get a

good accuracy. Even fine tuning the hyper-parameters is not helping beyond a certain extent as the model is getting over-fitted to the training dataset. Having more number of training samples will help the model to get more generalized and hence will improve its performance.

In future, we plan to implement multiple CNNs to extract features from multiple localized part like eyes, eyes and mouth and then the bottleneck features of each CNN are combined with a neural network to finally classify the image. This would improve the accuracy of the model by a greater amount for the fine-grained image classification problem. We also plan on collecting more training samples from ImageNet and other sources, such that the model can be trained properly.

REFERENCES

[1] LecunY, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11):2278-2324.

[2] Cun Y L, Boser B, Denker J S, et al. Handwritten digit recognition with a back-propagation network[C]// Advances in Neural Information Processing Systems. Morgan Kaufmann Publishers Inc. 1990:465.

[3] Hecht-Nielsen R. Theory of the backpropagation neural network[M]// Neural networks for perception (Vol. 2). Harcourt Brace & Co. 1992:593-605 vol.1.

[4] Krizhevsky A, Sutskever I, Hinton G E. ImageNet Classification with Deep Convolutional Neural Networks[J]. Advances in Neural Information Processing Systems, 2012, 25(2):2012.

[5] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in ECCV, 2014.

[6] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in ICLR, 2015.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with Convolutionals,” Co RR, vol. abs/1409.4842, 2014.