Object Identification using Deep Learning Framework TensorFlow

K. V. Prema*, Ananta Arora and Ankit Kumar

1. Abstract

In this paper, the aim is to implement a Dog Breed Classifier using Convolution Neural Network (CNN). Object recognition is one of the fundamental field of computer vision. Dog's breed detection is a tough task in itself. There are many breeds which look similar to each other. Example: Brittany and Welsh Springer Spaniel share identical features. We downloaded the dataset from the site of Stanford University and classified it into 133 different categories. This era of deep learning is data hungry and the more the number of neural network layers the more the requirement of the data.

Once we are into training more amount of data, we need a deeper neural network for it. To run such deep neural network on a machine with CPU is next to impossible. So, we used the concept of transfer learning in which we imported a pre-trained model, ResNet-50 and trained our data using that and got a test accuracy of 84%. While training our data using a CNN model made from scratch we got test accuracy of 49%. We used TensorFlow as backend and keras library on top of it.

Keywords *— Convolutional Neural Network (CNN), ImageNet, ResNet-50, Transfer Learning, Dog Breed, Image Classification, Feature detection.*

2. Introduction

Object detection is one of the most important task to be performed by deep neural networks. This field can be used in the advances of medical research, fraud detection, self driving cars and to fight terrorism. The results of recognising objects from an image or any other source is getting better day by day. Since the introduction of Deep Learning, there have been rapid advances in the field of Image Classification[9]. Convolutional Neural Networks (CNN)[1] have been used to a great effect in these applications such as object classification, scene recognition mainly due to its high accuracy. Many machine learning methods work well only under a common assumption: the training and test data are drawn from the same feature space and the same distribution. When the distribution changes, most statistical models need to be rebuilt from scratch using newly collected training data. In many real world applications, it is expensive or impossible to re-collect the needed training data and rebuild the models. It would be nice to reduce the need and effort to re-collect the training data. In such cases, knowledge transfer or transfer learning between task domains would be desirable. In this paper, we demonstrate the use of Transfer Learning[7] to classify the images of 133 different classes of dog's breed. We used a pre-trained model, ResNet-50[10], designed by a team of Microsoft engineers to train on the ImageNet[2] dataset. The paper firstly explains the concepts of Convolutional Neural Network then we have described how we made a model from scratch and trained it with our dataset. Later we have described the methodology to implement the concepts of transfer learning using the ResNet-50 model. The original ResNet-50 model is trained by feeding an image to the input, and at each layer of the model it will perform some computations on the data until it outputs a label and classification accuracy.

3. Problem Setup

A. Convolutional Neural Networks

Deep Convolutional Neural Networks (CNNs) have been at the heart of spectacular advances in deep learning. Although CNNs have been used as early as the nineties to solve character recognition tasks[1], their current widespread application is due to much more recent work, when a deep CNN was used to beat state-of-the-art in the ImageNet[6] image classification challenge[2]. CNN uses variation of multilayer perceptron designed to require minimal pre-processing. It consists of multiple layers of receptive fields. These are small neuron collections which process portions of the input image. The outputs of these collections are then tiled so that their input regions overlap, to obtain a better representation of the original image; this is repeated for every such layer.

One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance. Different layers in CNN include Convolution, Pooling and Fully Connected layers.

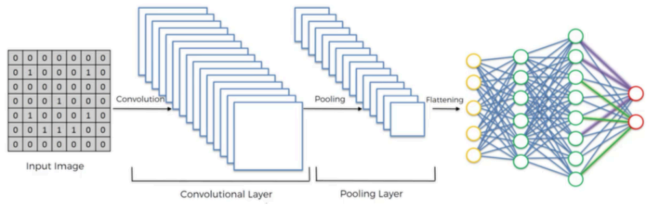


Figure 1. A Fully connected CNN architecture

Convolution: The convolution layer is applied and convolution operation is done for the input, passing the results to the next layer. The main function of convolution is to extract features from an image. It combines and integrates two functions to generate the third function. A feature map is generated based on the feature detector which acts as the filters in CNN.

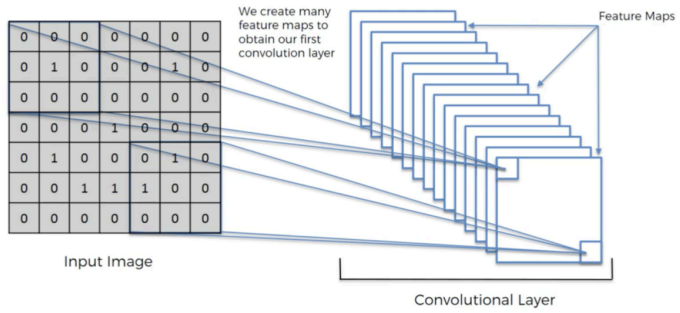


Figure 2. Convolution step

Pooling: Convolutional neural networks may include one or more local or global pooling layers. For example, max pooling[4] uses the maximum value from each of the cluster of neurons formed. So, we don't lose any important information. Pooling layers are used to shrink the height and width without losing features or edges of the images.

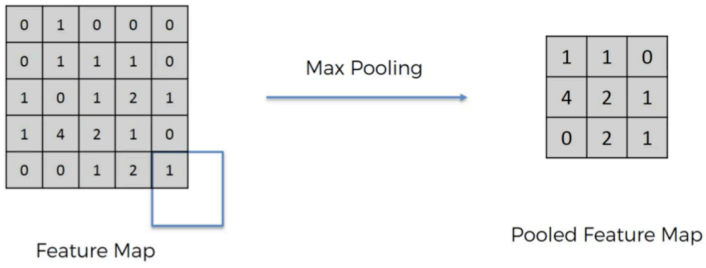


Figure 3. Pooling step

Flattening: In Flattening we convert the matrix of features into single columns and pass it through a neural network. We practically do this as we take them as input for our convolutional neural network.

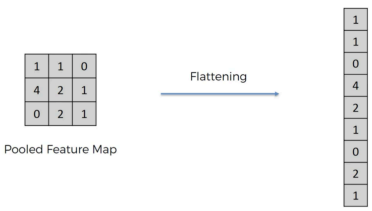


Figure 4. Flattening

Fully Connected: Fully Connected Layers connect every neuron in one layer to every neuron in another layer. It is in principle the same as traditional multi-layer perceptron neural network.

B. Transfer Learning

Transfer Learning is also know as inductive learning. It is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different problem which is a related problem. It is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. It is related to problems such as multi-tasking and concept drift and is not exclusively an area of study for deep learning. Over the years there is a trend to go more and more deeper to solve more complex tasks and to increase or improve the classification /recognition accuracy. But as we go deeper, the training of neural network becomes difficult and also the accuracy starts saturating and then degrades also. Transfer Learning tries to solve these problems. When to use transfer learning:

- **Higher start:** The initial skill (before refining the model) on the source model is higher than it otherwise would be.
- **Higher slope:** The rate of improvement of skill during training of the source model is steeper than it otherwise would be.

Some of the pre-trained models are as follows:

- VGG-16 and VGG-19 Model
- Inception Model
- Res-Net Model
- Alex-Net Model
- Word2vec Model

We used ResNet-50 pre-trained model to train our data.

C. ResNet-50 Model

In recent years, neural networks have become deeper, with state-of-the-art networks going from just a few layers (e.g., Alex-Net) to over a hundred layers. The main benefit of a very deep network is that it can represent very complex functions. It can also learn features at many different levels of abstraction, from edges (at the lower layers) to very complex features (at the deeper layers). However, using a deeper network doesn't always help. A huge barrier to training them is vanishing gradients: very deep networks often have a gradient signal that goes to zero quickly, thus making gradient descent unbearably slow. More specifically, during gradient descent, as you backprop[5] from the final layer back to the first layer, you are multiplying by the weight matrix on each step, and thus the gradient can decrease exponentially quickly to zero (or, in rare cases, grow exponentially quickly and "explode" to take very large values).

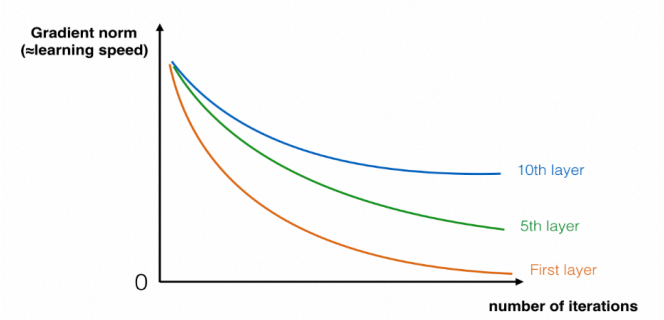
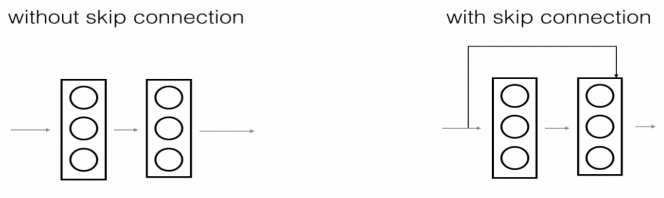


Figure 5. Problem of deep neural network

This problem can be solved by using Residual Neural Networks. In Res-Nets, a "shortcut" or a "skip connection" allows the gradient to be directly backpropagated to earlier layers:



Building of a ResNet-50 model:

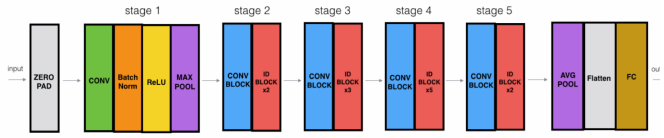


Figure 6. Bulding stages of a ResNet-50 model

4. Training Process

A. Data Collection

We collected the dog[8] breed dataset from the data source of Stanford website. There are 133 categorised dog's breed available in which there are 100-150 pictures of dogs available. The dog pictures are not labeled but they are stored in the categorised dog's breed folders.

B. Data Pre-processing

Data pre-processing is an essential part of prediction models in Neural Network.
We labeled the dog images according to their breed. We have used TensorFlow[3] as backend by using Keras API available for tensorflow. When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape:

(nb_samples, rows, columns, channels)

where nb_samples correspond to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively. The *path_to_tensor* function takes a string-valued file path to a color image as input and returns a 4D tensor suitable for supplying to a Keras CNN. The function first loads the image and resizes it to a square image that is 224×224 pixels.

Next, the image is converted to an array, which is then resized to a 4D tensor. In this case, since we are working with color images, each image has three channels. Likewise, since we are processing a single image (or sample), the returned tensor will always have shape (1, 224, 224, 3)

(nb_samples, 224, 224, 3)

The *paths_to_tensor* function takes a numpy array of string-valued image paths as input and returns a 4D tensor with shape

Here, *nb_samples* are the number of samples, or number of images, in the supplied array of image paths. It is best to think of nb_samples as the number of 3D tensors (where each 3D tensor corresponds to a different image) in your dataset.

C. Making a dog detector using ResNet-50

We are using pre-trained ResNet-50 model to detect the dogs in the images.

Our first goal is to download the ResNet-50 model, along with the weights that have been trained on ImageNet a very large and popular dataset used for computer vision and image classification tasks. The dog images are parsed and detected using the **landmark detection** algorithm i.e. their features are saved.

To make predictions with ResNet-50:

- First, the RGB image is converted to BGR by reordering the channels.
- All pre-trained models have the additional normalization step that the mean pixel (expressed in RGB as [103.939,116.779,123.68] [103.939,116.779,123.68] and calculated from all pixels in all images in ImageNet) must be subtracted from every pixel in each image.

Normalising inputs speeds up the learning:

$$\mu = \frac{1}{m}\sum_i x^{(i)}$$

$$\sigma^2 = \frac{1}{m}\sum_i x^{(i)2}$$

D. CNN Architecture

First, we started with augmenting the images so that we can increase the number of images for test and training data. There are many ways to do data augmentation, such as the popular horizontally flipping, random crops and color jittering. We flipped the images horizontally and increased the size of images for few more data incrementation. We made a 5 layer CNN architecture using keras library. Subsequently increasing the number of filters from 16 to 32 until 256. We kept the kernel size fixed to 3 and used gaussian kernel intializer which truncates normal distribution from the data.

$$\sigma = \sqrt{\frac{2}{fan_{in}}}$$

One of the drawbacks of non-regularized neural networks is that they are extremely flexible: they learn both features and noise equally well, increasing the potential for overfitting. Thus, we applied batch normalization[12] regularisation to scale the mini batches going through the neural network. Each mini batch is scaled by the mean/ variance computed on just that mini batch.

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

We used **ReLU** (Rectified Linear Unit) activation function within our hidden layers. ReLU doesn't suffer from vanishing gradient problem. It doesn't update the weights when the output of a neutron is negative as the output of ReLU for a negative value is zero. To overcome this problem we generally use Leaky ReLU.

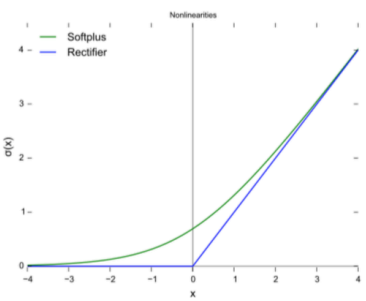


Figure 7. ReLU

We applied Max Pooling of size 2 at each hidden layer and at the output layer we kept the density to 133. Since, we have a multi class classification problem we used **softmax** activation function at the output layer. Since, softmax activation function is responsible for providing different number of values between 0 and 1. It gives the best probability as an output. Thus, using **softmax** activation function for multi-class classification problem is best way to go.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}}$$

We used **Adam**[11] optimisation algorithm for our deep learning model. A good optimiser makes the difference in how many days, hours or minutes our data can be trained using deep learning concepts. In adam a learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.

Adam (Adaptive momentum estimation) = RMS Prop + Momentum

$$V_{dw}^{corrected} = \frac{V_{dw}}{(1-\beta_1^t)}$$

$$V_{db}^{corrected} = \frac{V_{db}}{(1-\beta_1^t)}$$

$$S_{dw}^{corrected} = \frac{S_{dw}}{(1-\beta_2^t)}$$

$$S_{db}^{corrected} = \frac{S_{db}}{(1-\beta_2^t)}$$

$$W := W - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected} + \epsilon}}$$

$$b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected} + \epsilon}}$$

Hyperparameter choices:

α : needs to be true

β_1 : 0.9

β_2 : 0.999

ϵ : $1e-8$

We used cross-entropy loss function as our model is giving output in terms of probabilistic classification. If we think of a distribution as the tool we use to encode symbols, then entropy measures the number of bits we'll need if we use the *correct* tool y.

$$J(y, \hat{y}) = \sum_i y_i log \frac{1}{\hat{y}^i} = - \sum_i y_i log \frac{1}{\hat{y}^i}$$

E. Applying Transfer Learning to our dataset

Deep Learning models are growing with layers day by day and they are becoming more data hungry. Training a model with 5k to 6k dataset is not enough nowadays. We can't define a too deep neural network for them. Here, we can use the concepts of transfer learning i.e. to use pre-trained models which were trained on millions of images or some other form of data. As the founder of coursera, Andrew Ng believes the future of deep learning may rely on concepts of transfer learning. Companies like Google, Microsoft, Apple, Facebook, Amazon can train models on millions of data and then they can provide those models based on subscriptions. As we decided to use a pre-trained network, slight constraints in terms of the model architecture needed to be handled. For example, we couldn't arbitrarily take out convolutional layers from the pre-trained network. However, due to parameter sharing, we could without much of a stretch, run a pre-trained network on images of different spatial size. This is obviously apparent in the case of Convolutional and Pool layers because their forward function is independent of the input volume spatial size. In case of Fully Connected (FC) layers, this still remains constant in light of the fact that FC layers can be changed over to a Convolutional Layer.

Learning Rate: We used the same optimizer that we had used in our CNN model made from scratch. The learning rate was kept to 0.002 so that there isn't any distortion in the weights while training the data with this new model.

Bottleneck features: The ResNet-50 model comprises of many layers stacked on top of each other. These layers are pre-trained and are already very valuable at finding and summarising information that will help classify most images. We intended on training only the last layer; the previous layers retain their already-trained state. The first phase analyses all the images on disk and calculates the bottleneck values for each of them. The layer just before the final output layer that actually does the classification is informally termed as 'Bottleneck'. This penultimate layer has been trained to output a set of values that's good enough for the classifier to use to distinguish between all the classes it's been asked to recognise. That means it has to be a meaningful and compact summary of the images, since it has to contain enough information for the classifier to make a good choice in a very small set of values. The reason our final layer retraining can work on new classes is that it turns out the kind of information needed to distinguish between all the 1,000 classes in ImageNet is often also useful to distinguish between new kinds of objects. The dictionary which consist the dog categories in the Image net dataset belongs from 151 to 268.

ResNet-50: We imported ResNet-50 model using the keras library and trained our data using this pre-trained model keeping every other parameters same so that we don't deviate our weights much. We wrote a python script to retrain the data and predict the dog breed based on new classifier's results.

$$epochs = 30$$

$$batch - size = 64$$

We used **adam** optimizer for optimization and the loss function used was **cross entropy**.

The tools we used for implementation are:

- Python 3.6
- Keras API
- TensorFlow as backend
- Jupyter Notebook

5. Results

We obtained a test accuracy of 49.4% on training our dataset with the model we create using CNN architecture.

Test accuracy = 49.4% , Number of epochs = 25 Batch size = 20, Training accuracy = 84% Validation loss = 2.28. On improving the architecture i.e. by using Res-Net50 architecture for training the dataset and creating a model we got an accuracy of 84.09% , Test accuracy = 84.09% Number of epochs = 30 , Batch size = 64, Training accuracy = 99.8% , Validation loss = 0.54

The following graph can illustrate the details:

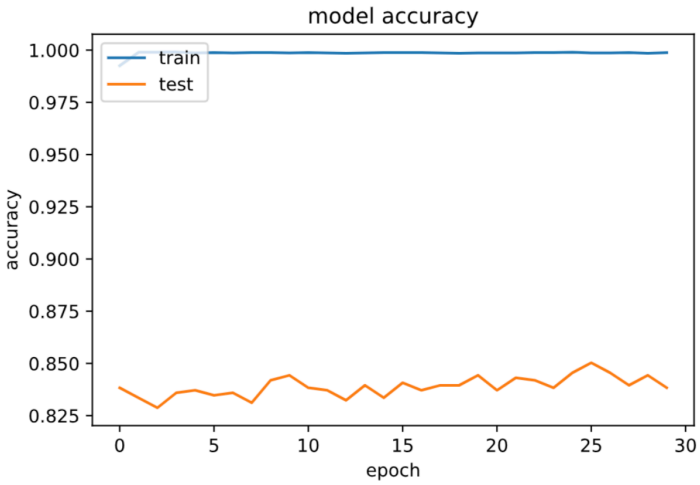


Figure 8. Model accuracy

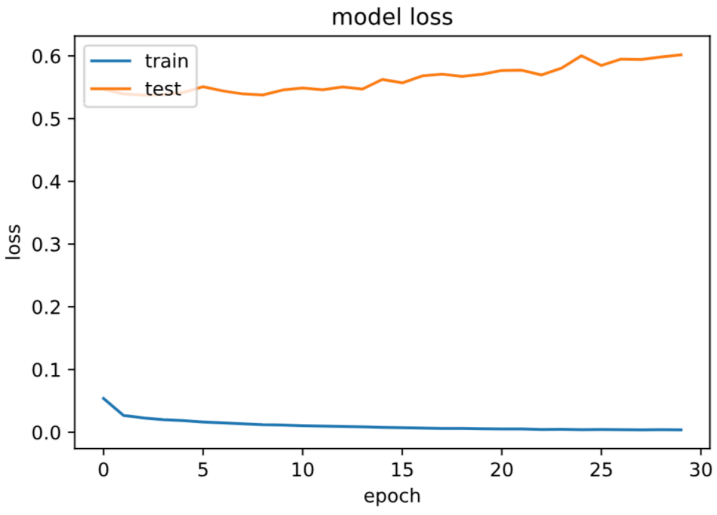


Figure 9. Model loss

Testing the model:

Figure 10. Test Image

6. Conclusion

This paper proposes a method to classify dog breeds from various dog images using the concepts of transfer learning. We demonanstrated an application of transfer learning in the context of dog breed classification task and introduced the concept of using pre-trained models like ResNet-50 in this experiment, 8351 dog images belonging to 133 different categories. Using our CNN architecture which we made from scratch we got a test accuracy of 49% which is too low for a classification problem as it is always a 50-50 chance if a dog belongs to a particular breed or not. Thus, we applied transfer learning and got a increase in test accuracy to 84% which is pretty good for a classifier.

In future work, we hope to explore some more classification models and try to make them from scratch. We want to do the task of feature extraction, edge detection, filter values more

efficiently. The model can be transformed and can be made into account for use in mobile applications. We can also make other object detection, scene detection task and improve our results based on concepts used in this deep learning model.

7. References

[1] Le Cun, Y., Bottou, L., and Bengio, Y. (1997). Reading checks with multilayer graph transformer networks. In Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on, volume 1, pages 151–154. IEEE.

[2] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105.

[3] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., et al. (2015). Tensorflow: Large- scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[4] Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B, and Ng, A. (2011). On random weights and unsupervised feature learning. In L. Getoor and T. Scheffer, editors, Proceedings of the 28th International Conference on Ma- chine Learning (ICML-11), ICML '11, pages 1089–1096, New York, NY, USA. ACM.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to hand-written zip code recognition. *Neural computation*, 1989.

[6] A. Krizhevsky, I. Sutskever, and G. Hinton. "Image-Net classification with deep convolutional neural networks", Advances in neural information processing systems. 2012.

[7] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," in IEEE Transactions on Knowledge and Data Engineering, vol. 22, no. 10, pp. 1345-1359, Oct. 2010.

[8] O. M. Parkhi, A. Vedaldi, A. Zisserman and C. V. Jawahar, "Cats and dogs," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, 2012, pp. 3498- 3505. doi: 10.1109/CVPR. 2012.6248092

[9] Kang, Tim, "Using Neural Networks for Image Classification" (2015). Master's Projects. Paper 395.

[10] He et al, 2015. "Deep Residual Networks for Image Recognition"

[11] Kingma, Durk P. and Welling, Max. Auto-encoding variational bayes. In Proceedings of the International Conference on Learning Representations (ICLR), 2014.

[12] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.