Using Deep Convolutional Neural Networks and Knowledge Transfer for Image Recognition

By

Muayyad Alsadi

Supervisor

Prof. Arafat Awajan

Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Princess Sumaya University for Technology

King Abdullah I School of Graduate Studies and Scientific Research

December, 2018

Princess Sumaya University for Technology
King Abdullah I School of Graduate Studies and Scientific Research
Authorization Form

I, Muayyad Saleh Mahmoud Alsadi, authorize Princess Sumaya University for Technology to supply copies of my Thesis/Dissertation to libraries or establishments or individual on request, according to the Regulations of Princess Sumaya University for Technology.

Signature:

Date:

Examination Committee Decision

This Thesis (Using Deep Convolutional Neural Networks and Knowledge Transfer for Image Recognition) was Successfully Defended and Approved on 2018-06-25.

Examination Committee	**Signature**
Prof. Arafat Awajan, Supervisor, Chairman Professor of Computer Science	.....
Prof. "Muhammad Bilal" Al-Zoubi, Member Professor of Computer Science	.....
Prof. Nadeem Obaid, Member Professor of Computer Science	.....
Dr. Rawan Ghnemat, Member Associate Professor of Computer Science	.....
Dr. Nijad Amin Al-Najdawi, External Member Associate Professor of Computer Science Al-Balqa' Applied University	.....

Using Deep Convolutional Neural Networks and Knowledge Transfer for Image Recognition

By

Muayyad Alsadi

Supervisor

Prof. Arafat Awajan

Abstract

Image recognition is a topic of "Computer Vision" that aims to find and identify one or several specified objects, object classes, features or activities in a given input image or video frame, even if it's partially obstructed from view. Deep Convolutional Neural Networks is the-state-of-the-art technique for Image recognition, but it requires a lot of training time and computing power to converge.

This thesis approaches the problem of image recognition with constrained budget in terms of limited training time and limited computing power of typical commodity hardware. Another imposed constraint is to have an expandable solution.

"Knowledge Transfer" technique is used to reuse publicly available off-the-shelf trained models. Several techniques are introduced to make it feasible, accelerate the process and make it expandable.

There are several applications for image recognition in different domains, such as Face recognition, Optical Character Recognition, Manufacturing automatic inspection and Quality Control, Medical diagnosis, and Autonomous vehicle related tasks such as Pedestrian detection. The application discussed in this thesis is the task of identifying brand, model and year of an image of a used car uploaded by a user of an e-Commerce service. The importance of this application comes from the raising popularity of smart phones equipped with cameras, where snapping a picture is far more convenient than typing description.

Trainable parameters (weights) are transferred from pre-trained ImageNet model into a model that gets fine-tuned on a task of cleaning up noisy input dataset. Then another transfer on a second model is used to identify most appropriate car models based on their

market share, and then it can be expanded to include more car models using a special proposed procedure.

Accuracy of more than 81% was achieved identifying 229 different car models, by re-using off-the-shelf Inception V1 trained to solve ImageNet one thousand classes task having accuracy of 69.8% in that task.

To make sure the proposed method is generic and not domain specific, several known academic tasks are also evaluated.

Dedication

To my mother and father.

To my wife and my family.

I dedicate this work.

Muayyad Alsadi

Acknowledgments

I wish to express my deep sense of gratitude to my supervisor Prof. Arafat Awajan, for his outstanding guidance and support which helped me in completing my thesis work. I would also like to thank Dr. Rawan Ghnemat, for her valuable assistance and help to fulfill my work.

Muayyad Alsadi

Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Meaning
AI	Artificial Intelligence
ANN	Artificial Neural Networks
C2C	Customer to Customer
CIFAR	Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
ConvNet	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
ILSVRC	Large Scale Visual Recognition Challenge
MINST	Modified National Institute of Standards and Technology
ML	Machine Learning
NN	Neural Networks
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine

Chapter 1. Introduction

1.1 Problem Statement

Image recognition is an important topic of computer vision, that is becoming more relevant with the advent of smart phones equipped with cameras.

Deep Convolutional Neural Networks technique can be used to solve Image recognition tasks, it achieves high accuracy but requires running expensive specialized hardware (like high-end Graphics Processing Units (GPUs)) for long period of training time. When approaching the problem with constrained budget of limited time and limited computing power of typical commodity hardware, comes the importance of reusing existing trained networks using a technique called "Knowledge Transfer". Another constraint is to have an expandable solution that is capable to recognize new classes with incremental training time that is much less than starting training from scratch.

1.2 Application

Professionals stores, shops and agencies typically employ paid editors to maintain their listing of products or items properly described, tagged and categorized. Items are shown with stock quality photographs (sometimes too perfect and unrealistic). On the other hand Customer to Customer (C2C) e-Commerce platforms (like OpenSooq.com) depend on user-generated content, where some of the platform users (website visitors or mobile app users) intend to sell their used items. Different users describe their items differently with varying accuracy and quality, thus machine help is desired.

Most views in OpenSooq come from mobile phones, due to wider adoption of smart phones, but number of items added using mobile platforms is still lagging behind laptops and desktops, because describing products on mobile phones is very difficult and inconvenient.

Smart phones are equipped with microphones, and many are capable of "voice typing" but variable accuracy of different languages limit their adoption. They are also equipped with cameras and most people already post pictures of items to be sold. Thus image recognition techniques are highly desired to describe these items in an accurate way.

The proposed flow goes like this: a seller uses the mobile application of some C2C platform, taking pictures of items to be sold, pictures got uploaded to the platform, plat-

form servers use machine learning to tag those pictures with proper meta-data like item type and possibly product brand, model and generation (model year). The application of the problem addressed in this research is identifying car make, model and model year. Focusing on getting state-of-the-art accuracy and sub-second speed and reasonable amount of time to train and retrain using commodity hardware.

1.3 Scope

The importance of image detection speed came from two factors: smoother user experience as it should be faster than user typing product details on keyboard, and reducing the cost of hosting the service, slower detection means more expensive computers with higher specifications (scale up) or running more computers of the service to serve more users in the same period of time (scale out).

Another important factor is the time needed for the machine to learn about these images in order to build a model that is able to recognize these images. With more products, brands and models coming every year (or even smaller periods), the training time and cost should also be considered and reduced, for this purpose "transfer learning" is to be considered in this research.

This research focus on studying methods for reusing pre-trained state-of-the-art models using a technique called "knowledge transfer". Our approach has to be expandable to cover new classes, new products or models with least training time without sacrificing accuracy. This research tries to estimate or predict the cost of this process by studying factors that affect training time in different models or methods.

1.4 Methodology

The objective of this work is generic image recognition and not specific to the task of identifying car models. Moreover the researcher applied his approach to different well-known datasets and tasks in academia. Varying number of classes in those datasets (from two classes to two hundred classes) will also help studying how well does each procedure scale with more complex problems. For the sake of "Knowledge Transfer", different source pre-trained models are also studied.

Research-specific datasets are obtained from images uploaded by users of OpenSooq.com. It is a well-recognized classified and C2C platform in the country and the region. A dataset for the purpose of this research is generated by downloading images from items of specific popular categories knowing their brand and model and year. Those images are scaled to 300x300 to save space, no further pre-processing is done.

All datasets are split into 90% training and 10% for test, and those test images not to be presented during training phase, so that used accuracy metrics are not affected by over-fitting. Several metrics are used to assess performance in a way that cover all possible aspects of the task being solved. Details about metrics are discussed in later sections. Each proposed procedure is studied and assessed separately, to show the effect of that specific procedure, and then assess the whole proposed solution.

Instead of measuring training speed in number of steps, time is used because different models varies in complexity due to varying number of trainable parameters and number of arithmetic operations. For example two models reaching 90% accuracy in 1000 epochs are not the same, because one might reach that in two hours and the other in two days. That's why "Epoch Time" (seconds per epoch) should be measured.

1.5 Contribution

This research study many state-of-the-art Convolutional Neural Networks (ConvNets) and best ways on how to re-use them for different tasks. This research have many contributions including:

- a formula to evaluate which model designs are more suitable for transfer learning.
- adaptive batch size method to accelerate learning or transfer.
- gradual layer inclusion to accelerate reaching better accuracy.
- study different ways to get a model without having a sufficient training dataset.
- introduce the concept of "Jointly Activated Classes" using cosine similarity on the columns of the weights matrix in Artificial Neural Networks (ANN) and its applications.
- introduce a method for category adaptation using weight manipulation.
- introduce a method to inject "none" class using weight manipulation.
- introduce a method to clean a noisy dataset using a model trained to clean the dataset.
- introduce a method to extend a trained model to add a new class by crafting weights introducing confusion.
- accelerate resolving confusion using over-sampling/under-sampling.
- application of combined above methods to identify vehicle make-model-year for its image.

This research introduces the following terms:

Knowledge donor model: is a previously trained model on source task (using different dataset)

Knowledge acceptor model: A new model that is reconstructed or derived or extended to perform target task using knowledge transferred from the donor model. This model need to be fine-tuned or re-trained before being useful.

1.6 Thesis Organization

This thesis is divided into five chapters. After this introduction chapter, comes the **"Background"** chapter which discuss historical researches and introduce terminologies and concepts used through out this thesis and needed in order to understand it. Next comes **"Literature Review"**, where several related researches in the field are discussed including state-of-the-art models. The **"Implementation"** chapter lists trials, experiments and stages of implementing proposed solutions in chronological order. The last chapter **"Discussion and Recommendations"** formulate the proposed solution and researcher recommendations and discuss them and how good they fit the addressed task of this work.

Chapter 2. Background

This chapter discuss historical researches and describes main concepts and terminologies used through out this thesis. It covers mainly topic about Artificial Intelligence (AI) and Machine Learning (ML).

2.1 Artificial Intelligence and Machine Learning

Terms like “Intelligence”, “Knowledge”, “Thinking” and “Learning” have been controversial for long time. In 1950, Alan Turing proposed the question “Can machines think?”(Turing, 2009). and instead of trying to answer that question, he proposed using a behavioral test for intelligence in what he called “imitation game”.

Human intelligence is manifested in many abilities like the ability to plan, solve problems, make decisions, perceive information, retain knowledge, comprehend ideas, reason, form concepts, recognize patterns, involving cognition, motivation, feelings (emotional knowledge), and self-awareness. Learning, thinking, and communication using languages.

This research focus on one of those abilities which is the ability to recognize images.

AI is usually defined using the paradigm of intelligent agent

> “Any device that perceives its environment and takes actions that maximize its chance of success at some goal.” (Russell & Norvig, 2003)

And for such agent to be able to learn, here comes the need for ML

> “Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy. Learning capabilities can improve the performance of an intelligent system over time. Machine learning mechanisms form the basis for adaptive systems.”(Negnevitsky, 2005)

A computer program is said(Thrun & Pratt, 1998) to “learn” if its performance at the task improves with experience. This means we need to define a performance metric and a way to pass examples to it, giving it experience. This process is called training. The

metric to be minimized can be called error, loss, cost, ..etc or to be maximized like fitness, reward, profit,... etc..

Humans do multiple learning tasks (concepts, skills, ..etc.) simultaneously where knowledge flows as a stream. Beside learning such tasks, they learn how to learn and how to generalize even if only given a single example. On the other hand, known "Machine Learning" techniques require many examples (sometimes in thousands) and the domain(s) and task(s) are fixed (translate from English to Arabic, diagnose cancer or drive an autonomous car).

If the examples passed to the machine are labeled (those are examples of dogs, and those are cats), then the task is called a "supervised" task. If the examples are not labeled, then the task is called "unsupervised", (given those loads of cats and dogs pictures, let's put them apart, without knowing which is which) "Semi-supervised learning" is very similar to "supervised learning" having a small number of labeled examples, and loads of unlabeled examples.

2.2 Classification Tasks

This research discuss one of the common ML applications, which is "classification". Given a classification task τ belonging to some specific domain D in some specific feature space χ having instances like X in the form $\{x_1, x_2, ..., x_n\}$, having marginal probability distribution $P(X)$ for every $X \in \chi$, and label space Y

$$D = \{\chi, P(X); X \in \chi\} \tag{2.1}$$

and given a labeled training dataset $T = \{(X_1, y_1), (X_2, y_2), ...\}$, for $X_i \in \chi$, $y_i \in Y$.

The classification task is defined(Pan & Yang, 2010) to find a function $g(X)$ that predicts the label having maximum conditional probability $P(y|x)$ or predict the joint probability of each label $f(X, y) = P(X, y)$ in the domain

$$\tau = \{Y, P(y|X); y \in Y\} \tag{2.2}$$

There are two main types of supervised classification tasks: "Constructive models" (like "Generative Models") and "Discriminative Models". Constructive models tries to understand how the output is generated from the input, as in the case of Generative Models(Battiti & Brunato, 2017) try to model how a class y relates to input x in the form of

class-conditional density$p(y_0|x)$ as in by Bayes' theorem 2.3.

$$p(y_0|x) = \frac{p(x|y_0) \cdot p(y_0)}{\sum_y p(x|y) \cdot p(y)} \tag{2.3}$$

Where $p(y_0)$ is the prior probability, and $p(x|y_0)$ is the likelihood of the data. The denominator normalize the formula to make it sum into one. Which means that it needs to model $p(x|y_0)$.

Discriminative models, on the other hand, only care about the end result of the classification without knowing the relation that forms different instances x in a given class $y0$.

A binary classifier is a classifier that has two labels or classes (that is Y has two members), it's either a yes or no, a cat or dog, if it's not a dog then it's a cat (no none). the probability of one is complementary to the other $P(y2) = 1 - P(y1)$.

- "does the given input data indicate that this patient has Diabetes? yes or no"
- "is this a picture of a dog? yes or no" (or equivalently "is this a dog or a cat?")

There are different ways to implement a binary classifier like the original Support Vector Machine (SVM) paper(Cortes & Vapnik, 1995) or as a neuron with only one output (that either activated or not activated).

Multi-class classifier have more than two members in Y, let's say a cat, a dog or a bird. There are two strategies to extend a binary classifier like the original SVM into multi-class classifier(Vapnik, 1995)(Hsu & Lin, 2002)(Knerr, Personnaz, & Dreyfus, 1990)

"one against one" - performing pair-wise comparisons between all n classes

"one against all/rest" - perform comparisons against all at once, not pair-wise.

Here are some examples of classification tasks and their data sets

Fisher's Iris Flower data set - has three classes "Setosa", "Versicolour", and "Virginica" based on four features that is Petal length, Petal width, Sepal length, and Sepal length(Fisher, 1936).

Breast Cancer Wisconsin (Diagnostic) Database - has two labels (Malignant and Benign) and 30 numeric attributes(Street, Wolberg, & Mangasarian, 1993)(Wolberg, Street, & Mangasarian, 1994)(Mangasarian, Street, & Wolberg, 1995).

2.3 Knowledge Transfer

Given a trained model for some task (an accurate solved problem), and a another different but similar task that is to be addressed (unsolved problem). "Knowledge Transfer" is all about reusing the source task to solve the target task. There are many different approaches for doing this some (like Category Adaptation) does not involve training target task while others use transfer only as an initialization and require training target task.

2.4 Domain-specific Image Recognition

Domain-specific Image Recognition application as in "Face recognition" and "Fingerprint scanner", not only utilize specialized algorithms that are only valid in that domain but also typically work on pre-processed images so that images are centered, segmented and have fixed orientation.

2.5 Artificial Neural Networks

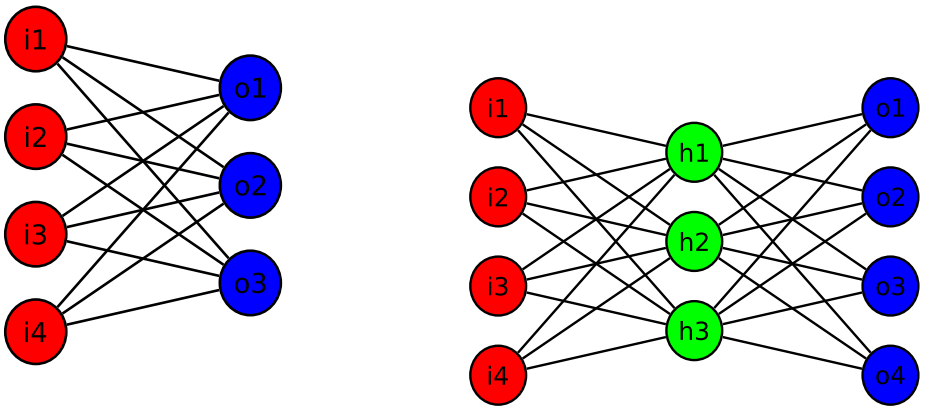
The most basic form of ANN or simply Neural Networks (NN) is "Perceptron", which is basically one output that is the result of weighted summations of all inputs plus some scalar value called "bias" then apply some function that is called "activation function" to achieve non-linearity.

Slightely more complex ANN have more than one output, having different weights and bias values for each output. For example, in figure 2.1a o_1 is the weighted summation of i_1, i_2, i_3, and i_4 that is $o_1 = F(w_{11} \cdot i_1 + w_{12} \cdot i_2 + w_{13} \cdot i_3 + w_{14} \cdot i_4)$.

The weighted summation can be expressed in the form of matrix multiplication. That is given input I of size n written as $I_{n\times1}$, output of size m written as $O_{m\times1}$, the weights $W_{m\times n}$, and bias values $B_{m\times1}$, and activation function F The output of the perceptron is defined by formula 2.4

$$O_{m\times1} = F(W_{m\times n} \times I_{n\times1} + B_{m\times1}) \tag{2.4}$$

The most basic form of activation function $F(x)$ is a simple threshold as in 2.5 denoted as $[x]^+$, if the weighted summation plus the bias is greater than zero, it's kept as is (acti-



(a) A simple neural network with single layer that is the output layer of size three output nodes formed from four input layers and no hidden layer

(b) A neural network with a single hidden layer of size three between four input nodes and four nodes outputs

Figure 2.1: Two simple neural networks, input in red, output in blue.

vated), otherwise it's zero (deactivated). The value of the bias controls the threshold. This form of activation function is called Rectified Linear Unit (ReLU)(Hahnloser & Seung, 2001)(Jarrett, Kavukcuoglu, LeCun, et al., 2009)(Nair & Hinton, 2010)

$$f(x) = [x]^+ = max(0, x) \tag{2.5}$$

There exist many other forms of activation functions like "Sigmoid functions"(Orr & Müller, 1998)(LECUN, 1998)

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$f(x) = \tanh(x) \tag{2.7}$$

$$f(x) = \tanh(x) + a \cdot x \tag{2.8}$$

A common setting for "Scaled hyperbolic tangent"(LeCun et al., 1989) in 2.9 is to have $A = 1.7159$ and $S = 2/3$, which would result $f(1) = 1$ and $f(-1) = -1$.

A multi-layer neural network is merely a concatenation of simple ones, the neural network first generate an intermediate state called hidden layer, then that hidden layer is then

$$f(x) = A \cdot \tanh(S \cdot x) \tag{2.9}$$

Equation 2.9: Scaled hyperbolic tangent. Source:(LeCun et al., 1989)

become an input of another stage as seen in figure 2.1b. One can add many intermediate hidden layers. “Deep Neural Network” is that with more than one hidden layer.

The training process is composed of two phases, forward pass (or forward propagation), and “Back propagation”. Parameters (weights and bias values) are randomly initialized, then in forward pass, each input instance is passed through the ANN, and compared against the desired output using some cost function like mean squared error E and used to adjust weights accordingly as in figure 2.2, that’s why training process is a form of optimization problems with objective to minimize the cost function.

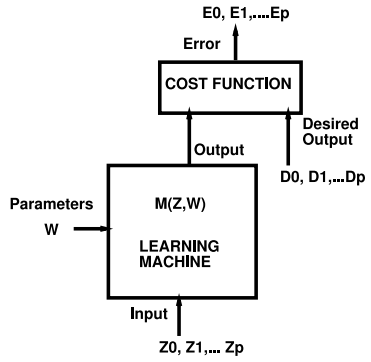


Figure 2.2: Gradient based learning.
Source: (LECUN, 1998)(Orr & Müller, 1998)

When having n layers, the input is denoted as X_0, the output of first hidden layer is X_1, the last output of the whole network is X_n. Output can be expressed as a recursive formula 2.10 as in(LECUN, 1998)(Orr & Müller, 1998)

$$X_n = F_n(W_n, X_{n-1}) \tag{2.10}$$

Typically F_n is the activation function applied to the matrix multiplication of weights to input.

$$X_n = sigmoid(W_n \times X_{n-1})) \tag{2.11}$$

$$y = X_n \tag{2.12}$$

$$E = \frac{1}{2}\sqrt{(y_d - y)^2} \tag{2.13}$$

A fraction of difference between the desired result and the obtained result is carried backward (thus the name "Back propagation") to adjust the weights, that fraction is called "Learning Rate" in 2.14 it's η which can be a constant like 0.01 or some faded value through out the process. This process is called "gradient descent".

$$W(t) = W(t-1) - \eta \cdot \frac{\partial E}{\partial W} \tag{2.14}$$

$$\Delta W = -\eta \cdot \frac{\partial E}{\partial W} \tag{2.15}$$

Taking a random instance (or a very small random subset of training data) at each step then this is called Stochastic Gradient Descent (SGD). Streaming instances and doing the update as they come is called "online learning".

2.6 Neural Networks for Classification

Taking the classical example of Iris database, which have four input features and a corresponding label that has three possible values. Each one of the three possible classes is made into an output node in the neural network, and "Hot One Encoding" is used to represent it, like this:

1. "Setosa" is represented as $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$
2. "Versicolour" is represented as $\begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$
3. "Virginica" is represented as $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$

Only one output node is activated that correspond to the desired class/label.

Normalized exponential function (Softmax)2.16 is used to convert the open-ended actual output into probability-like output, because softmax has to important properties(Nasrabadi, 2007)

- each output in the interval [0, 1]

- the summation of all output values is 1.

$$softmax(o_j) = \frac{e^{o_j}}{\sum_{i=1}^{n} e^{o_i}} \tag{2.16}$$

One problem of this, is that it would boost otherwise negligible values as it has to make the summation equal to 1, for example, given two output nodes having 0.01 and 0.001 the softmax will map them into 50.2% and 49.8%. When given a picture of a tree in a cat or dog task, the probability of being a cat or being a dog would always be high, it's either a cat or a dog, it would never return a number smaller than some threshold indicating none of them.

2.7 Generalization

If one made a ML learning model that memorize all the training dataset, then it would be wrongly considered accurate as it would fail to generalize and when presented with input that was not part of training data it would fail miserably this is called "Over-fitting". To measure this, part of the dataset is put aside into "test dataset" that is never presented during training but rather used later to evaluate the model.

In case of mini batches with SGD, each small batch is also shuffled and split into training and validating dataset.

Regularization(Ng, 2004) tend to prefer smaller weights like the green curve in figure 2.3, by adding a scalar multiple of weights magnitude summation to the loss function, either absolute value as in "L1 Regularization"2.17 or squared as in "L2 Regularization"2.18 or both2.19. The scaler λ is called "Regularization Penalty".

$$C_{l1} = C + \lambda \sum_i |w_i| \tag{2.17}$$

$$C_{l2} = C + \lambda \sum_i w_i^2 \tag{2.18}$$

$$C_{l12} = C + \lambda_1 \sum_i |w_i| + \lambda_2 \sum_i w_i^2 \tag{2.19}$$

Another method to force the neural network to generalize is to introduce noise in the

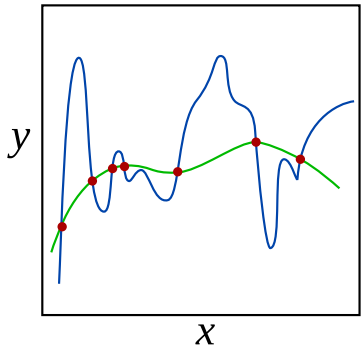


Figure 2.3: Two curves that have same loss of zero. Green one have smaller weights, blue one have higher weights.
Source: Wikipedia

training dataset. Another method is to randomly drop parts of the signal, which is called "dropout layer"

2.8 Convolutional Neural Networks

Convolutional Neural Network (CNN) or more commonly ConvNets was inspired by how visual cortex is assumed to work. CNN is similar to ANN but uses Convolutional Matrix operator. The 3D volume formed by multiple 2D matrices channels is called a "Tensor", Given an input image in the form of a Tensor formed by of multiple 2D matrices of pixels for each color channels, several filters can be written in the form of convolution matrix like edge detection (as in figure 2.4), sharpen, smoothing/blurring (as in figure 2.5), and pattern matching. ANN can be used to learn the values in many convolutional matrices resulting in image filters that activate when being exposed on specific visual features or structures.

End-to-End solutions feed unprocessed input directly without much prepossessing (if any at all) and get the end desired result and NN is good at that.

The size of each convolutional filter matrix which is also called the "kernel size" or the "neighborhood size" specifies the receptive field of the filter. For example, a kernel of size 3×3 means that bounding box of neighborhood starts one pixel to left, one pixel to top and ends on one pixel to the right and one pixel to the bottom while 5×5 kernel size means two pixels neighborhood from each side.

The convolution operator used in CNN is called 2D convolution (usually named "conv2d()" or "conv_2d()" depending on the used software package) and it operates on 3D volumes (taking a 3D volume as input and resulting a 3D volume as output) For example, an input can be a colored image of size 224×224 and depth of 3 RGB channels (written as

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & +4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Figure 2.4: Laplacian operator (edge detection) written as 3×3 convolutional matrix

$$\frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 2.5: A blur filter written as 3×3 convolutional matrix

224×224×3 or 3@224×224(LeCun, Bottou, Bengio, & Haffner, 1998)), when passed to a convolution on a neighborhood size 3×3 and depth of 96 channels (written as 3×3×96 or 96@3×3) result would be an output volume of size 224×224 (or 222×222 in case of no padding) and depth of 96. The size of the input is $width \times height \times depth$ or for short $W_i \times H_i \times D_i$. And the output size is $W_o \times H_o \times D_o$. The depth of the convolution kernel represent how many filters do we want to apply/learn.

Gray-scale images have input depth of one, but even in that case we still need to operate in 3D volumes, because we don't want to learn a single filter, after first convolution we would have an output of 3 axis.

Convolution depth varies from layer to layer but input depth of a layer must match output depth of the previous layer as they work in a pipeline.

So passing an image of size $W_i \times H_i$ and input depth D_i (ex. $D_i = 3$ for RGB channels) to a convolution layer of kernel size $W_k \times H_k$ and output depth D_o would have number of weights to be learned is $W_k \times H_k \times D_o \times D_i$ plus D_o bias values.

You can think of convolution layers as an operator that takes a 3D volume of $W_i \times H_i \times D_i$ and result a 3D volume of $W_i \times H_i \times D_i$ and this operator is to be applied to every possible pixel in the image by starting from top-left corner (in case of padding), where the neighborhood window is centered, and slide that window one pixel until all input image

is covered.

In summary a convolution layer has the following properties:

Input volume size: $W_i \times H_i \times D_i$

Kernel size: $W_k \times H_k \times D_o$

Output volume size: $W_o \times H_o \times D_o = W_i \times H_i \times D_o$

Number of parameters: are

- $W_k \times H_k \times D_o \times D_i$ weight values
- D_o bias values

Number of window positions: $P = W_o \times H_o$

Number of multiplications in a given position: $W_k \times H_k \times D_o \times D_i$

Number of addition operations in a given position: $W_k \times H_k \times D_i \times D_i$ (and another one for bias)

Total number of multiplication operations $W_o \times H_o \times W_k \times H_k \times D_o \times D_i$ and same value for addition operations

In case of no padding, convolution start at offset $= \frac{W_k - 1}{2}$ pixels (and a similar offset to the end) which would be make output smaller by $W_k - 1$ that is $W_i - W_k + 1$ so with height.

The final objective of all convective convolution operators after all layers of any CNN is to get us to the desired labels of the classification task, that is D_o of the last layer is either the labels or flat features that are being fed to usual ANN (called Fully Connected layers).

An important special case of no-padding convolution is the case when kernel size matches input size, that is $W_i = W_k$ then the output width would be $W_i - W_k + 1 = W_i - W_i + 1 = 1$, similarly with the height, then the size of the output would be 1×1. Which converts the 3D volume into one dimension along the depth, making it suitable to be considered to correspond to the desired labels in the classification task or to flat features to be fed to usual ANN.

Another important note that 1×1 convolution with depth D_o have the same effect of a hidden layer of size D_o in usual ANN.

- Input volume size: $1 \times 1 \times D_i$
- Output volume size: $1 \times 1 \times D_o$
- Number of parameters:
  - $D_i \times D_o$ weight values
  - D_o bias values
- Number of window positions: $P = 1 \times 1 = 1$
- Total number of multiplication operations = $D_i \times D_o$ and same value for addition operations

Some designs may involve sliding more than one pixel at a time, which is a hyper-parameter called "Stride" and usually written as $/S$, for example a kernel of $5{\times}5{\times}96/3$ means it has width and height of five, depth of 96 and stride of 3. And it's a technique used to reduce number of operators (but it does not reduce number of weights). With a stride value of two output width and height halves. With a Stride of one it's like having no stride.

Input volume size: $W_i \times H_i \times D_i$

Kernel size: $W_k \times H_k \times D_o/S$

Output volume size: $\frac{W_i}{S} \times \frac{H_i}{S} \times D_o$

Number of parameters: are

- $W_k \times H_k \times D_o \times D_i$ weight values
- D_o bias values

Number of window positions: $P = W_i \times H_i/S^2 = W_o \times H_o$

Number of multiplications in a given position: $W_k \times H_k \times D_o \times D_i$

Number of addition operations in a given position: $W_k \times H_k \times D_i \times D_o$ (including one for bias)

Total number of multiplication operations: $W_o \times H_o \times W_k \times H_k \times D_o \times D_i$ and same value for addition operations

The effect of stride is not limited to reducing number of multi-add operations but on the exponential effect of reducing the output volume of this filter becoming input filter of next layers, for example applying five layers of convolution with stride of two would reduce width and height of 224×224 into 7×7, because $224/(2^5) = 7$.

2.9 No weights layers

In CNN, not all layer have weights. Pooling layers have a neighborhood window size and a stride S written as $W_k \times H_k/S$. It uses some aggregate operator like maximum or average to summarize the signal in that window, that is moved S pixels each time, resulting on a down-sampled image. For example average pooling of size 2×2 and stride of 2 (written as 2×2/2) would have the effect of down-sampling the input image by factor 2. Pooling is possible with window size not matching the stride, and is also possible without stride at all ($S = 1$) that is to summarize surrounding input signals in the window.

The difference of applying stride in convolution or in a pooling layer after it, is that the first one ignores some input signals while the other one takes the strongest signal (in case of max-pooling).

Another form of no-weights layers is to apply some activation function on input like "Rectifier Layer" which applies ReLU(Hahnloser & Seung, 2001)(Jarrett et al., 2009)(Nair & Hinton, 2010) to input as in formula 2.5. Such layers does not affect the shape of its input volume.

2.10 Design of CNN models

A typical basic design of a CNN model starts with an input image of certain width and height $W_i \times H_i$ in case of colored images that is a volume of size $W_i \times H_i \times 3$. Then that volume is feed to a sequence of convolution layers of certain kernel size and depth (number of filters). Then comes a pooling layer (maximum pooling or average pooling), then that is repeated alternating convolution and pooling layers.

The objective of the design is to form a flat signal of no spacial dimension (width=1 and height=1) but rather along the depth which will be the signal of output classes. To reduce the width and the height of the signal volume, stride is used on convolution layer or pooling layers. Different designs of CNN models use stride at different stages, it can be placed early, down-sampling the signal too early will trade accuracy with speed.

Toward end of the model, either pooling or convolution of kernel size matching the size of the input will summarize the signal into the desired flat signal of size 1×1. When the signal is flat along the depth (having width and height equal to one), classical ANN can be used, this is called fully connected layers, this is equivalent of convolution of kernel size 1×1.

The last signal is processed with Softmax function as in equation 2.16 to make output signal form probabilistic-like values as previously discussed in section 2.6.

Figure 2.6 shows the whole process of such design.

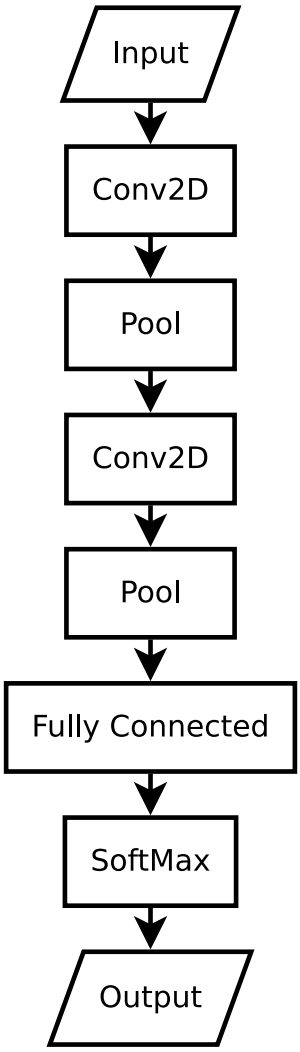


Figure 2.6: A simple CNN design.

2.11 Batch Normalization Layers

Just like it's important to normalize input signal (for example, by subtracting mean and dividing by variance), it's also important to regulate hidden layers signals. A batch normalization layer(Ioffe & Szegedy, 2015) is placed after some hidden layers, having two trainable parameters that are called "scale" and "shift", which is updated for each mini-batch in training phase, by finding batch mean and variance.

Some researches suggest using larger mini-batches(Goyal et al., 2017).

Table 2.1: How separable operators reduce number of weights and multiplications

Name	Input	depth	kernel	depth	weights	mults
original	224×224	3	3×3	96	3×3×3×96 =2,592	224×224×3×3×3×96 = 130.1M
Separable equivalent						
depth-wise	224×224	3	3×3	1	3×3×3×1 =27	224×224×3×3×3×1 =1.4M
point-wise	224×224	1	1×1	96	1×1×1×96 =96	224×224×1×1×1×96 =4.8M
				Total	27+96 =123	6.2M

2.12 Separable Operators

One of the methods used to reduce number of weights is to decompose the kernel from being of size $n \times m$ into two matrices of size $n \times d$ and $d \times m$ and typically $d = 1$, in this case the number of weights would be $(n \times 1) + (1 \times m) = n + m$ instead of $n \times m$ (or $2 \cdot n$ instead of n^2 when width and height are equal). For example one can write a 3×3 blur kernel using two matrix multiplication of size 3×1 and 1×3 as in (2.20), there are a total of six weights in two matrices (3×1 and 1×3) compared to nine in the equivalent 3×3 matrix.

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.20}$$

Equation 2.20: 3×3 blur operator written as two matrices each of three

Beside being used along the width and height of the convolution kernel it can be used along kernel depth where the convolution kernel of $W_k \times H_k$ that takes input of depth D_i and output depth of D_o using intermediate value called depth multiplayer D_M and which can be 1. One way of doing this is two consecutive convolutions one is called(Chollet, 2017)(Mamalet & Garcia, 2012) "depth-wise convolution" that operate on a kernel of 3×3 on an intermediate depth of D_M (typically 1) and the other is called "point-wise convolution" that operate on a kernel of 1×1 with output depth of D_o

As we can see table 2.1 when have an input image of size 224×224 and 3 RGB channels which is convolved with 3×3 kernel and 96 filters (output depth), then typically we have 2.5K weights and 130M multiplications. A separable counterpart of 3×3×1 and 1×1×96 with same input volume and output volume would have only 123 weights and 6M weights. In the second case as in table 2.2 when we take an input volume of size 224×224×64 to be convolved with a kernel of 3×3 and an output depth of 64, the usual convolution have 36K

Table 2.2: Separable operators reduction applied to hidden layer

Name	Input	depth	kernel	depth	weights	mults
original	224×224	64	3×3	64	64×3×3×64 =36,864	224×224×64×3×3×64 =1,849.7M
Separable equivalent of same hidden layer						
depth-wise	224×224	64	3×3	1	64×3×3×1 =576	224×224×64×3×3×1 =28.9M
point-wise	224×224	1	1×1	64	1×1×1×64 =64	224×224×1×1×1×64 =3.2M
				Total	640	32.1M

weights and 1,800M multiplications. On the other hand the separable version of $D_M = 1$ have only 640 weights and 32M multiplications.

2.13 Accuracy Metrics

Accuracy rate which is the ratio of true items, both true positives (T_P) and true negatives (T_N) over total (or its complement "Error rate" which is the ratio of false items both false positives F_P and false negatives F_N over total). Those two metrics are vulnerable if the likelihood of labels are not equal. For example, if the problem domain is diagnosing cancer, assuming that the probability of cancer is small like 0.1%, a classifier that randomly pick cancer positives with very small probability or even not report any cancer positive at all (classify every body as healthy) would have very high accuracy (99.9%) and low error rate (since it only mistaken people with cancer, so it have error rate of 0.1%). To overcome this, a metric (like "Precision" and "Recall" below) is needed that does not consider success rate or failure compared to total, but makes the denominator to be limited to smaller context.

$$Accuracy = \frac{T_P + T_N}{SampleSize} \tag{2.21}$$

Precision is the ratio of true positive items over positively diagnosed items (both T_P and F_P), which is a measure of Type I error caused by the F_P in the denominator 2.22.

$$Precision = \frac{T_P}{T_P + F_P} \tag{2.22}$$

Recall is the ratio of true positive items compared all positive items (both truly diagnosed and falsely missed), which is a measure of Type II error that cases the F_N in the

denominator 2.23.

$$Recall = \frac{T_P}{T_P + F_N} \tag{2.23}$$

In the above example of random cancer diagnose that picks 0.1%, in a sample of 10,000 person, it would randomly diagnose ten persons with cancer, but only one of them actually have cancer and it missed eight persons that diagnosed as healthy despite that they actually have cancer (the rest where accurately diagnosed healthy).

$$accuracy = \frac{1 + 9982}{10,000} = 99.83\%$$

$$precision = \frac{1}{10} = 10.00\%$$

$$recall = \frac{1}{1 + 8} = 11.11\%$$

2.14 Public Datasets and Benchmarks

Canadian Institute for Advanced Research (CIFAR) have published CIFAR-10(Krizhevsky & Hinton, 2009) dataset of small 32×32 labeled images of ten classes.

ImageNet(Deng et al., 2009) Large Scale Visual Recognition Challenge (ILSVRC)(Russakovsky et al., 2015)(Krizhevsky, Sutskever, & Hinton, 2012) is the benchmark used by state-of-the-art methods. ILSVRC has many tasks, most notably the classification task of 2012 (ILSVRC-2012-CLS) which has one thousand class of objects and large colored images (compared to CIFAR-10's 32×32, ILSVRC's 224×224 is 49 times more pixels).

There are other much simpler datasets like the Modified National Institute of Standards and Technology (MINST)(LeCun, Cortes, & Burges, 2010)(LeCun et al., 1998) which is a database of 70,000 handwritten digits of size 28×28 pixels that is centered and normalized, it's considered easy to achieve 99% accuracy on this dataset.

Chapter 3. Literature Review

3.1 CNN Model Design

Different papers define and draw their CNN design differently, this research analyses them then represent them into a unified table.

3.1.1 LeNet

LeNet(LeCun et al., 1998) was one of the first successful ConvNets. It was used to recognize characters (thus domain specific), the research was based on NIST Special Database 1 (SD-3)(Wilson & Garris, 1990) and NIST Special Database 3 (SD-3)(Garris & Wilkinson, 1992) and their research resulted in one of the most infamous databases; the Modified National Institute of Standards and Technology(MNIST) database(LeCun et al., 2010).

Their CNN operates on a gray-scale input of 32×32 pixels (or 28×28 surrounded with 4 padding pixels), input images were pre-processed to be centered, normalized, segmented ..etc. First convolution is applied without padding, it has kernel size 5×5 (two pixels from each side around center) which would output a volume of size 28×28. In the initial paper they used output depth of 6 for the first convolution layer as shown in figure 3.1 from same paper.

In other words, first layer would train six kernels each of size 5×5 transforming a volume of 32×32×1 input (depth is 1 not 3 because it's input is not colored) into an output of 28×28×6.

LeNet-5 has seven layers as seen in figure 3.1, five of them have trainable weights and two layers of them do not have weights.

Table 3.1 demonstrates operations done on each layer and the corresponding number of parameters to be trained (weight and bias values) and number of multiplications operations (same number of addition operations) in each layer

Despite having a simple design, shallow depth and specific simple task operating on a pre-processed normalized input, it has more than 61k parameters and about half million multiplication operation and half million addition operations. F6 is a fully connected hidden layer and F7 is the fully connected output layer, the output depth of last layer is the number of classes in this example that is the ten digits from 0-9. Another note that will

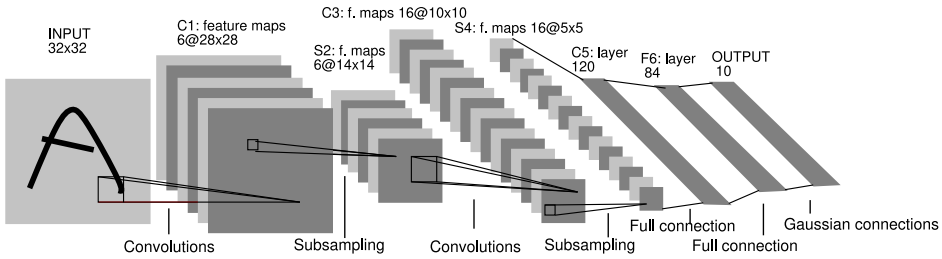


Figure 3.1: LeNet-5 CNN desgin
Source: (LeCun, Bottou, Bengio, & Haffner, 1998)

Table 3.1: Details of LeNet layers, their input size, kernel size, output size and number of parameters (weights and bias). W×H×D/S stands for width, height, depth and stride respectively

Layer Name	Input W×H×D	Kernel W×H×D/S	Output W×H×D	Params	Mults
C1: conv2d	32×32×1	5×5×6	28×28×6	1×5×5×6+6 =156	28×28×1×5×5×6 =117,600
S2: pool/2	28×28×6	2×2/2	14×14×6	0	0
C3: conv2d	14×14×6	5×5×16	10×10×16	6×5×5×16+16 =2,416	10×10×6×5×5×16 =240,000
S4: pool/2	10×10×16	2×2/2	5×5×16	0	0
C5: conv2d	5×5×16	5×5×120	1×1×120	16×5×5×120+120 =48,120	1×1×16×5×5×120 =48,000
F6: conv2d	1×1×120	1×1×84	1×1×84	120×1×1×84+84 =10,164	120×84 =10,080
F7: conv2d	1×1×84	1×1×10	1×1×10	84×1×1×10+10 =850	84×40 =840
			Total	61,706	416,520

be commonly seen in this research that most weights are in layer that flatten its input into 1×1 just before fully connected layers, in LeNet it's C5 with 78% of weights.

Note: Because bias values is very small compared to weights as we have seen above, we would only show weights.

3.1.2 AlexNet

AlexNet(Krizhevsky et al., 2012) was the winner of ILSVRC in 2012, the design of this ConvNet is demonstrated in figure 3.2 which shows half of it which is running on one GPU having a similar copy runs on another GPU, for example, on the left it starts with an input image and pass it through 11×11 kernel of depth 48 on one GPU and another same

sized depth on the other GPU, resulting total of 96 filters to be learned.

AlexNet main contribution is the use two GPUs in parallel with less communication between them and the reduction of over-fitting with data augmentation and dropout.

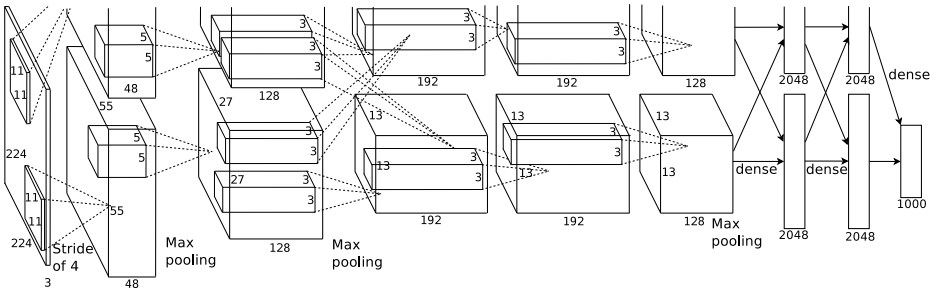


Figure 3.2: AlexNet running on two GPU cores
Source: (LeCun, Bottou, Bengio, & Haffner, 1998)

A slightly modified version of AlexNet (called AlexNet v2(Krizhevsky, 2014)) is shown in the table 3.2

Table 3.2: Analysis of AlexNet v2 layers, showing input size, kernel size, output size and number of weights and multiplications and their percent. Kernel size is width, height, depth and stride in the form of WxHxD/S

Layer	Input	Kernel	Output	Weights	%	Mults	%
c1: conv2d/4	224×224×3	11×11×64/4	54×54×64	3×11×11×64	0%	54×54×3×11×11×64	9.2%
pool/2	54×54×64	3×3/2	26×26×64	0	0	0	0
c2: conv2d	26×26×64	5×5×192	26×26×192	64×5×5×192	0.6%	26×26×64×5×5×192	**28.2%**
pool/2	26×26×192	3×3/2	12×12×192	0	0	0	0
c3: conv2d	12×22×192	3×3×384	12×12×384	192×3×3×384	1.3%	12×12×192×3×3×384	13.0%
c4: conv2d	12×12×384	3×3×384	12×12×384	384×3×3×384	2.6%	12×12×384×3×3×384	25.9%
c5: conv2d	12×12×384	3×3×256	12×12×256	384×3×3×384	1.8%	12×12×384×3×3×384	17.3%
pool/2	12×12×256	3×3/2	5×5×256	0	0	0	0
fc6	5×5×256	5×5×4096	1×1×4096	256×5×5×4096	**52.1%**	1×1×256×5×5×4096	3.6%
fc7	1×1×4096	1×1×4096	1×1×4096	4096×1×1×4096	33.4%	4096×4096	2.3%
fc8	1×1×4096	1×1×1000	1×1×1000	4096×1×1×1000	8.1%	4096×1000	0.6%
			total	50M	100%	737M	100%

3.1.3 ZFNet

ZFNet(Zeiler & Fergus, 2014) (also known as Zeiler & Fergus architecture) was the winner of ILSVRC 2013. ZFNet's main contribution is the concept of "DeConv", as they covered part of input image with a gray square to see which part of the image activates which part of the neural network.

This model is slightly modified Alexnet, same depth, same blocks with smaller stride of two at first convolution of four and smaller filter size of 7 instead of 11, slightly more weights and operations, for details refer to figure 3.3 and table 3.3.

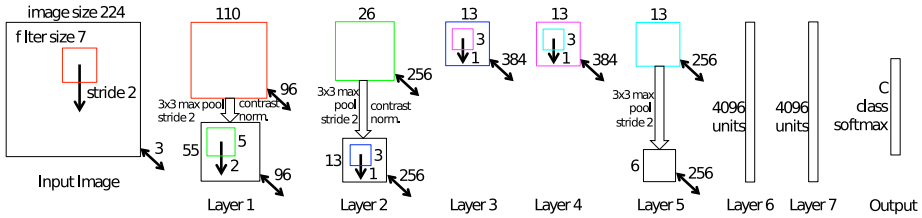


Figure 3.3: ZFNet design
Source: (Zeiler & Fergus, 2014)

Table 3.3: Analysis of ZFNet layers

Layer	Input	Kernel	Output	Weights	%	Mults	%
c1: conv2d/2	224×224×3	7×7×96/2	110×110×96	3×7×7×96	0.0%	110×110×3×7×7×96	14.6%
pool/2	110×110×96	3×3/2	55×55×96	0	0.0%	0	0.0%
c2: Conv2d/2	55×55×56	5×5×256	26×26×256	96×5×5×256	1.0%	26×26×96×5×5×256	**35.6%**
pool/2	26×26×256	3×3/2	13×13×256	0	0.0%	0	0.0%
c3: conv2d	13×13×256	3×3×384	13×13×384	256×3×3×384	1.4%	13×13×256×3×3×384	12.8%
c4: conv2d	13×13×384	3×3×384	13×13×384	384×3×3×384	2.1%	13×13×384×3×3×384	19.2%
c5: conv2d	13×13×384	3×3×256	13×13×256	384×3×3×256	1.4%	13×13×384×3×3×256	12.8%
pool/2	13×13×256	3×3/2	6×6×256	0	0.0%	0	0.0%
fc6	6×6×256	6×6×4096	1×1×4096	256×6×6×4096	**60.5%**	256×6×6×4096	3.2%
fc7	1×1×4096	1×1×4096	1×1×4096	4096×4096	26.9%	4096×4096	1.4%
fc8	1×1×4096	1×1×1000	1×1×1000	4096×1000	6.6%	4096×1000	0.4%
			Total	62M	100%	1,168M	100%

3.1.4 VGG

VGG(Simonyan & Zisserman, 2014) came second in ILSVRC 2014, The paper discussed many variations named after number of layers with trainable weights for example VGG16 (variation D in the paper) have has 16 layers with trainable weights, VGG-16 has 138M weights, 74% of which are in “fc6” layer as seen in table 3.4.

VGG-16 was the best performing variation of VGG, It was better than shallower variations like VGG-11 (variation A) and deeper variations like VGG-19 (variation E)

Table 3.4: Analysis of VGG-16

Layer	Input	Kernel/Stride	Output	Weights	%	Mults	%
Name	Wi×Hi×Di	Wk×Hk×Do/S	Wo×Wo×Do	Wk×Hk×Do×Di	-	Wo×Ho×Wk×Hk×Do×Di	-
conv1_1	224×224×3	3×3×64	224×224×64	3×3×64×3	0.0%	224×224×3×3×64×3	0.6%
conv1_2	224×224×64	3×3×64	224×224×64	3×3×64×64	0.0%	224×224×3×3×64×64	12.0%
pool/2	224×224×64	2×2/2	112×112×64	0	0	0	0
conv2_1	112×112×64	3×3×128	112×112×128	3×3×128×64	0.1%	112×112x3×3×128×64	6.0%
conv2_2	112×112×128	3×3×128	112×112×128	3×3×128×128	0.1%	112×112x3×3×128×128	12.0%
pool/2	112×112×128	2×2/2	56×56×128	0	0	0	0
conv3_1	56×56×128	3×3×256	56×56×256	3×3×256×128	0.2%	56×56×3×3×256×128	6.0%
conv3_2	56×56×256	3×3×256	56×56×256	3×3×256×256	0.4%	56×56×3×3×256×128	12.0%
conv3_3	56×56×256	3×3×256	56×56×256	3×3×256×256	0.4%	56×56×3×3×256×128	12.0%
pool/2	56×56×256	2×2/2	28×28×256	0	0	0	0
conv4_1	28×28×256	3×3×512	28×28×512	3×3×512×256	0.9%	28×28×3×3×512×256	6.0%
conv4_2	28×28×256	3×3×512	28×28×512	3×3×512×512	1.7%	28×28×3×3×512×512	12.0%
conv4_3	28×28×256	3×3×512	28×28×512	3×3×512×512	1.7%	28×28×3×3×512×512	12.0%
pool/2	28×28×512	2×2/2	14×14×512	0	0	0	0
conv5_1	14×14×512	3×3×512	14×14×412	3×3×512×512	1.7%	14×14×3×3×512×512	3.0%
conv5_2	14×14×512	3×3×512	14×14×412	3×3×512×512	1.7%	14×14×3×3×512×512	3.0%
conv5_3	14×14×512	3×3×512	14×14×412	3×3×512×512	1.7%	14×14×3×3×512×512	3.0%
pool/2	14×14×512	2×2/2	7×7×512	0	0	0	0
fc6: conv2d	7×7×512	7×7×4096	1×1×4096	7×7×4096×512	**74.3%**	7×7×4096×512	0.7%
fc7: conv2d	1×1×4096	1×1×4096	1×1×4096	4096×4096	12.1%	4096×4096	0.1%
fc8: conv2d	1×1×4096	1×1×1000	1×1×1000	4096×1000	3.0%	4096×1000	0.0%
			Total:	138M	100%	15,470M	100%

3.1.5 GoogLeNet or Inception v1

GoogLeNet (aka Inception v1)(Szegedy et al., 2015), the winner of ILSVRC 2014, has 22 layer having trainable weights (more than two times deeper than AlexNet v2(Krizhevsky, 2014)). The design of this model is very different as input flows in branches that later get joined using "DepthConcat" which as its name suggests places collect the output of different branches along the depth of output volume. That building block of this design is called "inception module" which is shown in figure 3.4, their CNN is composed of nine inception blocks (not to be confused with layers). Each block has four branches and it's two layers deep (except one branch, which has one layer)

- 1×1 followed by 3×3
- 1×1 followed by 5×5
- Max Pooling of size 3×3 (stride of 1) followed by 1×1
- 1×1

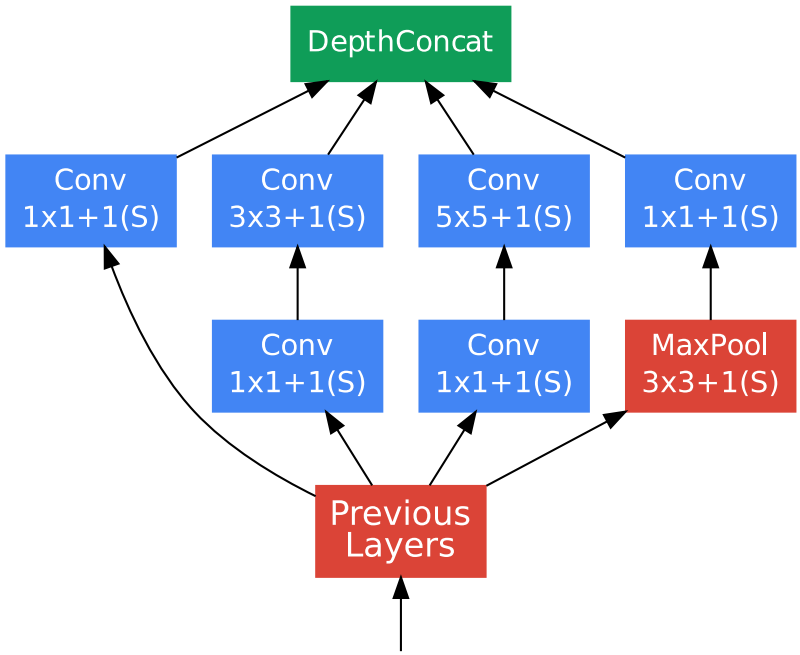


Figure 3.4: An Inception module block
Source: (Szegedy et al., 2015)

Table 3.5: Overhead of 3×3×128 convolution in Inception almost halved using separable operators

Name	Input	kernel	output	weights	mults
			Separable Version		
point-wise	28×28×192	1×1×96	28×28×96	1×1×96×192	28×28×96×192
depth-wise	28×28×96	3×3×128	28×28×128	3×3×128×96	28×28×3×3×128×96
			Total	129K	101M
			Non-separable Version		
original	28×28×192	3×3×128	28×28×128	3×3×128×192 =221K	28×28×3×3×192×128 =173M

As mentioned before those branches are merged using "depth concatenation", for example first inception block takes input of size 28×28×192 (coming from the output previous convolutions and pooling) which flows to the four mentioned branches the 1×1×64 branch outputs 28×28×64, the 3×3×128 branch outputs 28×28×128, the 5×5×32 branch outputs 28×28×32 and the max pool branch outputs 28×28×32 (because pooling stride is 1), resulting matching output size of 28×28 and total number of depth to be $64 + 128 + 32 + 32 = 256$ in depth.

Separable operators technique has been discussed previously and it's seen here, instead of making a branch of 3×3×128, they made it into 1×1×96 followed by 3×3×128 (here depth multiplier is 96). The table 3.5 compares the 3×3 part of first inception block compared to its non-reduced equivalent, using depth-multiplayer of 96 (reducing input depth from 192 to 96) and as a result of this reduction number of weights were reduced by about 41.7% and number of multiplications were reduced by about 41%.

Unlike separable operators we have seen before, the 1×1 point-wise stage is placed before the depth-wise stage in Inception. Table 3.6 shows the 5×5 part of first inception block compared to its non-reduced equivalent, using depth multiplier of 16 (reducing input depth from 192 to 16) and as a result of this reduction number of weights were reduced by 90% and number of multiplications were reduced by 90%.

General properties of this design:

- It can learn a features in a context of 3×3 or 5×5 (receptive fields) at the same layer level
- It can pass features from different context sizes from layer to layer (ex. combining a feature of 3×3 from one layer to a feature of 3×3 or 5×5 from previous layer) increasing and varying the size of receptive field.

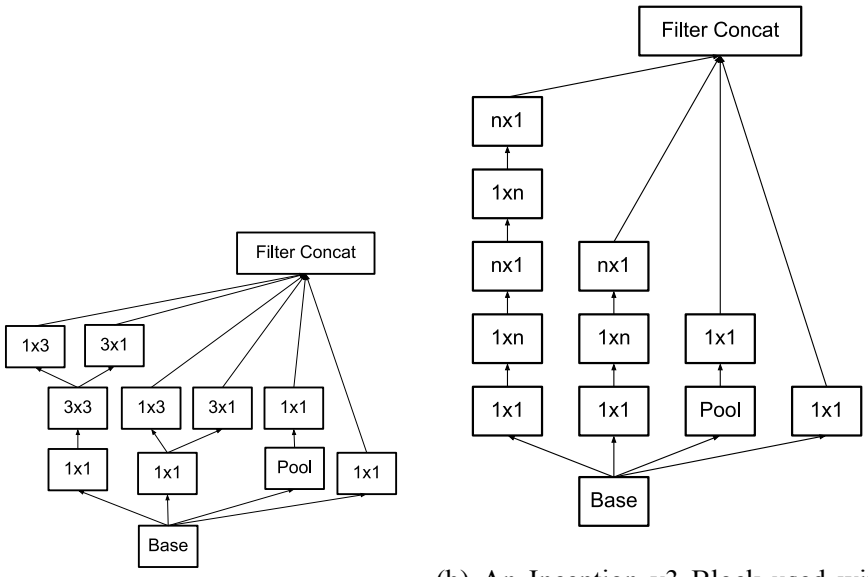
Table 3.6: Overhead of 5×5×32 convolution in Inception reduce by 90%

Name	Input	kernel	output	weights	mults
Separable Version					
point-wise	28×28×192	1×1×16	28×28×16	1×1×16×192	28×28×192×16
depth-wise	28×28×16	5×5×32	28×28×32	5×5×32×16	28×28×5×5×32×16
				Total: 16K	12M
Non-separable Version					
original	28×28×192	5×5×32	28×28×32	5×5×32×192 =154K	28×28×5×5×32×192 =120M

- the use of 1×1 branch is similar in effect to having a fully connected hidden layer (connecting neurons of input depth to neurons output depth) but much less expensive because it's used with OD much smaller than ID and since it's an intermediate layer it can be thought as an intermediate point-wise step in separable operator.
- No dense fully connected layers just before the classification layer (other CNN have a hidden layer with 4096 neurons with most of trainable weights) which is eliminated here and saved more than 70% of wasted weights (compare that to layer named fc6 in VGG-16 network as seen in table 3.4)
- Has a total of 6M weights, compared to 50M in AlexNet (as in table 3.2) and 140M in VGG-16(as in table 3.4) and 1.5G multiplications (compared to 15.4G multiplications in VGG-16) with better accuracy
- Their Paper(Szegedy et al., 2015) introduced the notion of "Auxiliary Classifiers" as counter measure for vanishing gradient. Beside the final so deep fully connected layers that is followed by "softmax" resulting the probabilities of each class, they have much earlier side fully connected layers and "softmax" towers. A fraction of the losss of those auxiliary classifiers is added to the final total loss, in that paper they used 0.3 of the auxiliary loss. This way, the ConvNet would be able to do the classification task as early as possible before going deeper. After training is done, those side towers are removed, because they are not needed in classification.

3.1.6 Inception v2 and v3

What is common in all inception versions(Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016)(Ioffe & Szegedy, 2015)(Chollet, 2017)(Krause et al., 2016) is the use of branched later depth-concatenated blocked called "inception module" and the use of 1×1 to reduce order. Table 3.7 compares all versions of inception.



(a) An Inception v3 Block

(b) An Inception v3 Block used with $n = 7$

Figure 3.5: Two variations of Inception v3 blocks
Source: (Szegedy, Vanhoucke, Ioffe, Shlens, & Wojna, 2016)

Table 3.7: Comparing all versions of Inception

Model Name	layers	weights	mults	Top-1 accuracy	Top-5 accuracy
Inception v1	22	6.6M	1,498M	69.8	89.6
Inception v2	42	11M	1,934M	73.9	91.8
Inception v3	53	27M	5,719M	78.0	93.9
Inception v4	81	46M	13,882M	80.2	95.2
Inception ResNet V2	130	59M	14,882M	80.4	95.3

In version two of inception, 5×5 was replaced with 3×3 (as two concatenated 3×3 would have the same receptive field of 5×5) Also separable convolution is done in this version too early as first layer which produce 64 depth using depth multiplier DM=8. Beside reduction along the depth, inception v2 introduced reduction along width and height which is called "Introduced Spatial Factorization" using "Asymmetric Convolutions" that is separating $k \times k$ kernels into $k \times 1$ followed by $1 \times k$. In Inception v1, all inception blocks were identical while later inception version have many variations of the inception block one of them having 3×3 followed by 1×1 kernels instead of 3×3 kernel. And another type with 1×1, 7×7, 1×1 then followed by 7×7. Two variants of this technique is seen in figure 3.5.

Counting layers in inception is tricky and misleading, because reducing an operator using "separable operator" methodology into two or more layers make a less generic filter with more layers.

3.1.7 Highway Networks

Deeper networks are more difficult to train due to vanishing gradients, a problem which is also faced by Recurrent Neural Networks (RNNs) that was approached by long short term memory (LSTM) blocks which uses gates that facilitate information passing from one block to another block as in figure 3.6.

Inspired by LSTM, Highway Networks(Rupesh Kumar Srivastava, Greff, & Schmidhuber, 2015)(Rupesh K Srivastava, Greff, & Schmidhuber, 2015) uses gated information flow in deeper CNNs. Hidden layer output H can be suppressed and replaced by passing input x, depending on the value of a special gate T which takes a value from zero to one (using sigmoid function) as in formula 3.1.

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T)) \tag{3.1}$$

By having $T = 0$, the first term will be zero and the second term will be the input x as is, that is information bypassing current layer and flow to next deeper layer.

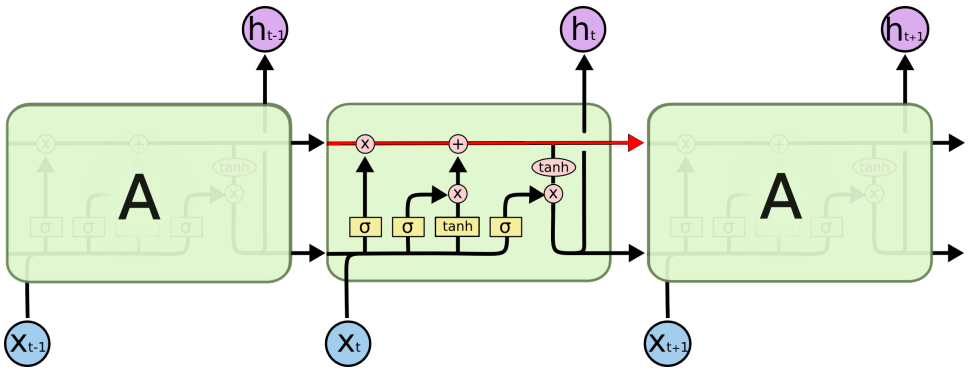


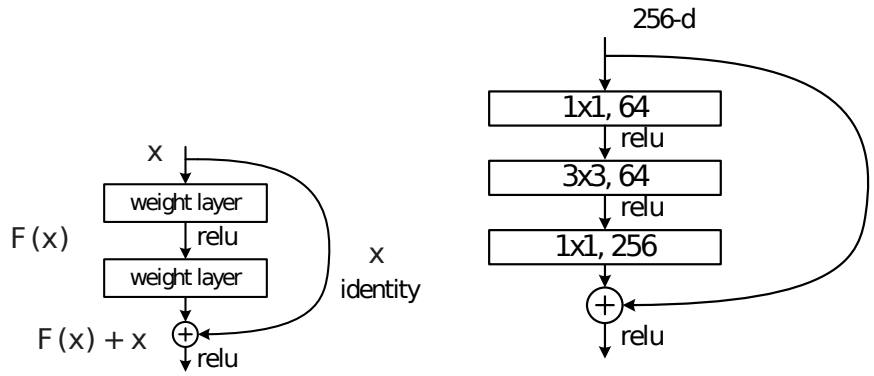
Figure 3.6: An LSTM-block showing gated flow of information in red color
Source: http://colah.github.io

3.1.8 ResNet

ResNet(He, Zhang, Ren, & Sun, 2016a)(He, Zhang, Ren, & Sun, 2016b) is the winner of ILSVRC 2015. ResNet extends the concept of highway in a more simpler way by just adding input every two layers, it's like having constant $T = 0.5$ in formula 3.1.

The main idea as in figure fig:resnet from first paper(He et al., 2016a) is to pass the original signal "as is" after every two layers with trainable weights using addition and "ReLU" (as seen in figure 3.7a) then just stack this building unit deeper and deeper.

This paper use a form of separable operator reduction called "bottleneck" (as in figure



(a) ResNet Building Block (b) ResNet Bottleneck

Figure 3.7: ResNet building block
Source: (He, Zhang, Ren, & Sun, 2016a)

3.7b) which places 1×1 kernel before and after 3×3 kernel.

The second paper(He et al., 2016b) changes the placement of batch normalization and ReLU as seen in figure 3.8

And because the original signal is not fading with depth this allowed them to go deeper up to one thousand layers. Different variation of this design have 50, 101, 152, 200, and even 1000 layers.

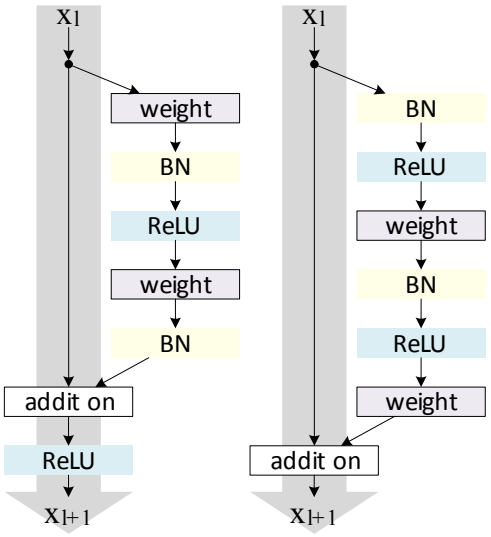
ResNet uses 7×7 convolutions with stride of 2, pool 3×3/2 then just repeat the building block of [1×1, 3×3, 1×1] many times, the last 1×1 output volume is 1×1×1000 and it's followed by average pooling from 2000 to 1000 then fully connected layer to classes.

And like inception, ResNet does not have a layer with most weights, the weight is well distributed as the heaviest layer in terms of number of parameters have about 8% of weights in the 50-layer version (that is the last fully connected layer), 4.7% in the 101 layer version, 3.9% in the 151 layers version.

And like inception number of layers is misleading because the use of separable operators.

3.1.9 Inception v4 And Inception-ResNet

Inception v4 And Inception-ResNet(Szegedy, Ioffe, Vanhoucke, & Alemi, 2017) are variations of previously discussed Inception that utilized residual connections in a way similar to ResNet(He et al., 2016a)



(a) original (b) proposed

Figure 3.8: Placement of Batch normalization in ResNet
Source: (He, Zhang, Ren, & Sun, 2016b)

3.1.10 DenseNet

Densely connected convolutional networks (DenseNet)(Huang, Liu, van der Maaten, & Weinberger, 2017), to counter vanishing gradients as network becomes deeper, a way for first input is allowed to pass through to deeper layers, Highway Networks which gates input and optionally pass it to deeper layers and ResNet which adds input every two layers/blocks, DenseNet just concatenate signals for all previous layers as in figure 3.9.

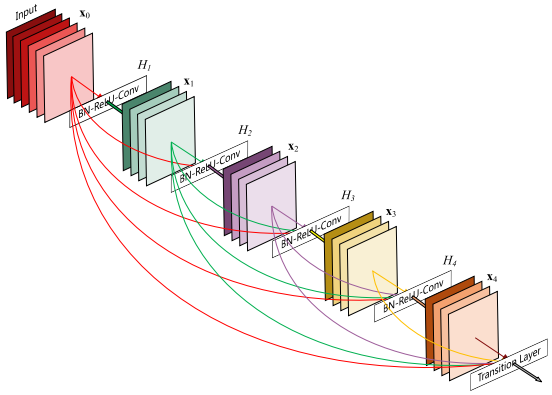


Figure 3.9: DenseNet have each layer gets input from all previous layers
Source: (Huang, Liu, van der Maaten, & Weinberger, 2017)

3.1.11 MobileNet and SqueezeNet

MobileNet(Howard et al., 2017) does not compete on accuracy but on speed. Similar to SqueezeNet(Iandola et al., 2016) in the use of the following strategies(Iandola et al., 2016)(He & Sun, 2015): Delay down-sampling toward the end of the network, that is pooling layers and convolutions with strides to get accuracy which other network put early in the network to reduce number of operations. Instead they use 1×1 convolutions instead of more larger 3×3 and reduce the depth of the 3×3 filters to reduce number of parameters and operations.

The idea of 1×1 did not first appear with MobileNet and SqueezeNet, as it's used heavily in Inception as seen before, but in MobileNet it's used as replacement of earlier stride to reduce the network computational cost, MobileNet v1.0 has competitive accuracy compared with inception v1 with half number of parameters. MobileNet v2.0(Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018) reaches better accuracy than Inception v1 with half the number of parameters.

There are no fully connected layer in SqueezeNet, which is called "Quantizing FC layer"(Wu, Leng, Wang, Hu, & Cheng, 2016).

3.1.12 NASNet

NASNet(Zoph, Vasudevan, Shlens, & Le, 2017) has multiple settings, some are small suitable for mobile applications that competes with MobileNet and Inception v1. While others has order of magnitude more operations and weights that compete with Inception v4 and ResNet in terms of accuracy. Here are two interesting settings on NASNet:

- NASNet-A (6 @ 4032), 331x331, 88.9M params, 23.8B mults
- NASNet-A (4 @ 1056), 224x224, 5.3M params, 564M mults

The main contribution of NASNet is not their specific network design, but rather is having that design done by a neural network instead of a human expert. Another aspect of it, that the training did not happen on ImageNet high resolution images of 1K classes but rather on a simpler task of classifying tiny images of ten classes (CIFAR-10), then the architecture is transferred to ImageNet 1k.

3.2 Pre-processing

The bare minimum pre-processing is to resize input image to make it fit the perceptive field of the network (typically 224×224 pixels). Some CNNs have special normalization done to input images, like subtracting image mean as in VGG(Simonyan & Zisserman, 2014), while others also divide by the standard deviation of the image. Such transformations should be done in both training phase and prediction phase.

Some pre-processing is done in training phase only to have better generalization, like: random cropping of a random bounding box in input image, random left/right flipping, and random color distorting (brightness, saturation, and contrast).

3.3 Stochastic Optimizers

The factors that affect the efficiency of ANN have been discussed in academia for ages for example (LECUN, 1998) (which was part of a book published 1998(Orr & Müller, 1998), that has more recent editions (Montavon, Orr, & Müller, 2012)).

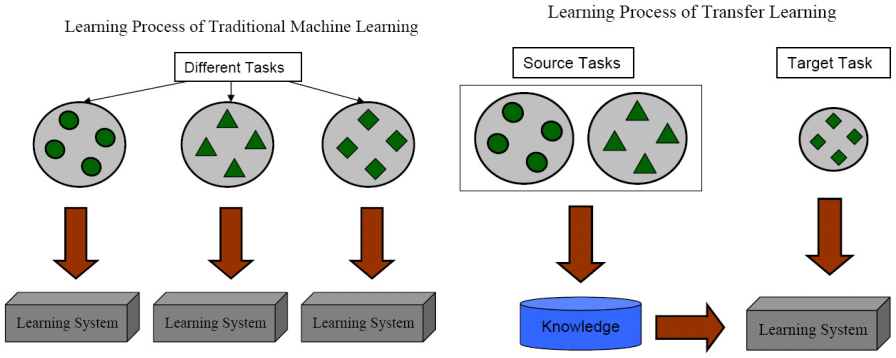
There are variations of plain stochastic gradient descent (SDG), the are adaptive and faster to converge like: SGD momentum(Sutskever, Martens, Dahl, & Hinton, 2013), Adagrad(Duchi, Hazan, & Singer, 2011), Adam(Kingma & Ba, 2014), Adadelta(Zeiler, 2012), and RMSprop(Dozat, 2016)(Sutskever et al., 2013)

PhD. student Sebastian Ruder has made a nice animation that compare how they work.

3.4 Knowledge Transfer and fine-tuning

Humans learn multiple tasks at once, and they learn how to generalize, and how to learn and the learning process is non-stopping continuous streams. If a computer program that work on a family of tasks each having its own training and its own experience and its own performance measures, then the computer program is said(Thrun & Pratt, 1998) to be capable of "learning to learn" if its performance depends on number of tasks (not just a single task). This field is called multi-task learning and in this type both tasks are trained simultaneously and one of them enhances the other.

"Self-taught Learning" as in (Raina, Battle, Lee, Packer, & Ng, 2007) use unsupervised machine learning to train the machine how to generalize and construct higher-level features despite the lack of labeled dataset in source task, and utilize that to in the labeled supervised target task.



(a) Traditional Machine Learning (b) Transfer Learning

Figure 3.10: Transfer-learning
Source: (Pan & Yang, 2010)

Knowledge Transfer is all about reusing some parts from an already solved task and borrowing knowledge from an already trained and accurate classifier as seen in figure 3.10. There are several settings for such process(Pan & Yang, 2010) depending on the relation between source domain D_s and target domain D_t, and the relation between source task τ_s and target task τ_t as they can be either the same or similar but different. Also Knowledge Transfer has to define(Pan & Yang, 2010)

- "What to transfer?"
- "How to transfer?"
- "When not to transfer?"

The transfer algorithm has to find what parts of knowledge can be transferred across domains and tasks, then how to actually do the transfer, depending on the availability of labeled training dataset in either of the domains (source and target). Domain adaptation(Saenko, Kulis, Fritz, & Darrell, 2010) considered same task on two different but similar domains, by finding a cross-domain transformation that maps points in source domain to target domain, as opposed to "Category Transfer" which maps labels as in figure 3.11.

There are several categorization for the "how"(Pan & Yang, 2010):

Instance-transfer To re-weight some labeled data in the source domain for use in the target domain.

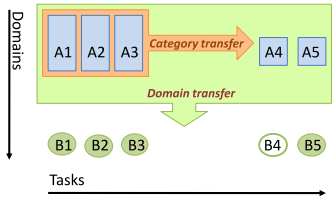


Figure 3.11: Difference between "Category Transfer" and "Domain Transfer". Source: (Saenko, Kulis, Fritz, & Darrell, 2010)

Feature-representation-transfer To find a "good" feature representation that reduces difference between the source and the target domains and the error of classification and regression models.

Parameter-transfer Discover shared parameters or priors between the source domain and target domain models, which can benefit for transfer learning

Relational-knowledge-transfer Build mapping of relational knowledge between the source domain and the target domains. Both domains are relational domains and i.i.d assumption is relaxed in each domain.

Architecture transfer train an Recurrent Neural Network (RNN) to find best architecture on a simple task, transfer architecture to difficult task as in NASNet(Zoph et al., 2017)

Other types of transfer learning have a prior (not simultaneous) knowledge (that is possibly static) in one task that is used to induce knowledge in another task both in same domain and feature space which is the type that this research would focus on where source domain and target domain are the same (which is the input image and its pixels are the feature). Source task is the ImageNet classification task from ILSVRC, and the prior knowledge is one of the pre-trained state-of-the-art models. Target task is to classify different dataset obtained from the e-Commerce platform or the fine-grained datasets of flowers, dogs, birds, food.

Fine-tuning(Zeng, Ouyang, Yang, Yan, & Wang, 2016)(Ouyang, Wang, Zhang, & Yang, 2016)(Sharif Razavian, Azizpour, Sullivan, & Carlsson, 2014) is a type of "Parameter-transfer". Taking an "off-the-shelf"(Sharif Razavian et al., 2014) pre-trained state of the art CNN model then re-fit it on different task by taking all weights except those in last fully connected layer or by adding an SVM after last convolution (called CNN-SVM). This works because several researches(Donahue et al., 2014)(Girshick, Donahue, Darrell, & Malik, 2014)(Oquab, Bottou, Laptev, & Sivic, 2014) have shown and even visualized(Zeiler & Fergus, 2014) that when moving toward the end of the neural network, the

deeper the more high-level features, moving from low primitives to complete objects or entities (eyes, faces, ..etc.)

Some other methods utilize knowledge from unlabeled datasets or noisy datasets and transfer knowledge to enhancing labeled but small dataset as in papers(Reed et al., 2014)(Sukhbaatar, Bruna, Paluri, Bourdev, & Fergus, 2014)(Sukhbaatar & Fergus, 2014)(Van Horn et al., 2015)

A different approach was done by Google Brain Team in their paper (Bello, Zoph, Vasudevan, & Le, 2017) as they proposed a way for "Architecture Search" but that was expensive for large database, that's why in their other paper(Zoph et al., 2017) used transferred knowledge from a very simple problem that is CIFAR-10 dataset to the state-of-the-art difficult problem ImageNet(Deng et al., 2009) dataset. They did not transfer weights, as they train from scratch after transfer, they transfer architecture and design not knowledge which resulted in NASNet discussed before.

Chapter 4. Implementation

4.1 Preparing datasets

4.1.1 Stock Datasets

There are many stock datasets in literature used to study image recognition techniques, like **"Oxford 102 Flowers Dataset"**(Nilsback & Zisserman, 2008), **"The Caltech-UCSD Birds-200-2011 Dataset"**(Wah, Branson, Welinder, Perona, & Belongie, 2011) and **"The Oxford-IIIT Pet Dataset"**(Parkhi, Vedaldi, Zisserman, & Jawahar, 2012) which have 37 species of cats and dogs. For short, those datasets are called flowers-102, birds-200, and pets-35 respectively.

For toy problems, a much simpler datasets with smaller number of classes are needed. That's why such datasets were crafted by manipulating previously mentioned datasets. For example, two classes dataset of "Cats and Dogs" was created by adapting pets-37 dataset, merging 12 cats breeds into a "Cats" class, and the rest are merged into "Dogs" class. A dataset with three classes "Cat, Dog or Bird" was created by merging all pictures from birds-200 dataset into one class (taking random 3,000 pictures out of around 11,000 pictures), then add them to the previous "Cats and Dogs" dataset.

Another toy dataset is the five flowers dataset(flowers-5) containing labeled images for Daisy, Dandelion, Roses, Sunflowers, and Tulips.

4.1.2 Vehicle Viewing Angles Dataset

For the sake of cleaning car images, a small hand-picked dataset was created to identify vehicle viewing angles into nine classes,

interior dashboard view , having either steering wheel or odometer visible.

interior seats view no steering wheel visible nor odometer visible.

front view both head lights are visible, no doors are visible.

back view both tail lights are visible, no doors are visible.

side view both front door and back door visible, both side wheels are visible, at most one head light visible and at most one tail light visible (if half of radiator grille is visible, then it's front-side view)

front-side view both front door and back door visible, both side wheels are visible, and both front head lights are visible (or more than half of the radiator grille)

back-side view both front door and back door visible, both side wheels are visible, and both back tail lights are visible

Wheels pictures of cars with only one wheel is visible and no-car wheels and alloy wheel.

under-hood without headlights or grill visible

Another dataset is created by subdividing each of "Side view", "Back view", and "Back-side view" into six classes: Sedan, Coupe, Station, Hatchback, SUV, and Pickup or truck.

Those two datasets are hand-picked from various websites, google image search, wikipedia, and OpenSooq.com, to have about 200 image from each class. Making sure to have different backgrounds, various brands, various models, designs of modern and legacy vehicles both stock "staged" images and natural images.

4.1.3 Noisy Vehicle Make-Model-Year Dataset

OpenSooq.com is the leading market place in the middle-east, having thousands of vehicles for sale added by their owners every day. Owners upload many pictures of their cars and tag them with make/model/year among other details (like mileage, transmission, ...etc.). Opensooq have tons of labeled images. But those user-generated images are noisy, as they have pictures of registration papers, insurance papers, inspection papers (local or CarFax), maintenance history, picture of business cards, dealer logos, ..etc. Beside non-car pictures, sellers upload pictures showing some selling points like moon roof, leather seats, door lock pin, large wheels size, alloy wheels, under the hood customization. Those picture (as in figure 4.1) have no association with any of the desired labels. Such pictures are important to the seller, but from the point of having pictures that conclude tags like car make and model those pictures are merely noise.

OpenSooq.com have generously provided us with hundreds of thousands of pictures of each class.

- 76 distinct car make
- more than 1600 classes of make-model
- more than 20 thousand class of make-model-year

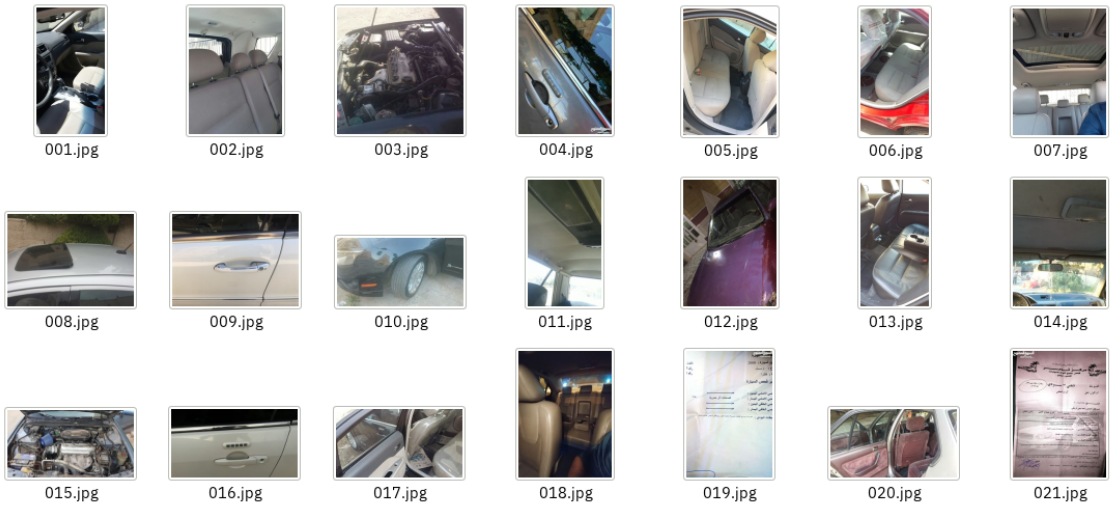


Figure 4.1: Examples of inconclusive pictures uploaded by sellers

Many of 20 thousand classes are distinct only on paper, many of them look similar and should be grouped together For example, Volkswagen Golf Mk3 looked the same from 1991 to 2002, and Beetle looked almost the same since 1956 as seen in figure 4.2. This means that this dataset need a lot of cleanup that is discussed later.

A cleaned dataset was produced based on this dataset, that is discussed in later sections.

4.1.4 Pre-processing Images

while most CNNs take input of size 224×224, most datasets have images of varying sizes, mostly too large images (full HD images or better) as they were scraped from social media or quality websites. It would be a waste of resources to store and photos of such large sizes.

To unify sizes and make it more suitable to such CNNs, All images have been re-sized keeping aspect ratio so that it would be at most 300×300 using the following command from ImageMagick Suite

```
$ convert input.jpg -resize 300x300'>' output.jpg
```

This is one time job that done once beside different run-time pre-processing done by different CNNs.

4.2 Tools

- SciKit Learn

(a) Original look of Volkswagen Beetle in 1956

(b) A Type 1 VW Beetle produced in 2003 Puebla, Mexico known as "Última Edición"

(c) The look of VW new beetle in 1998

Figure 4.2: Different models of VM beetle

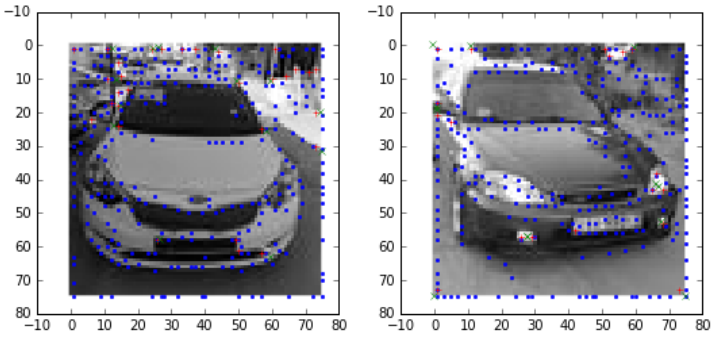
- TensorFlow
- TensorFlow slim models

4.3 Procedures

4.3.1 Failure of reduction via pre-processing before CNN

One of the initial ideas was the use of some pre-processing techniques to make input images into sparse "points of interest" or "key points". By focusing on smaller number of points, hoping to get a simpler problem that would require less resources. Hand crafted "hit-or-miss" convolution filters was used (hits are ones over their count, and misses are minus ones over their count). One of them those filters was corner filter (in figure 4.3) which should be applied in its four variations (rotated 90° each). Another tried was done using a small empty circle "hit-or-miss" (shown in figure 4.4). Those were applied to input images then taking local peaks (using "skimage.feature.peak_local_max()"). The results were comparable to Harris corner filter.

The idea was to make input images sparse to reduce the problem before feeding input images to ANN hoping to get a simpler problem that would require smaller resources to train. But ANN takes fixed-sized input or sequences in case on RNN. That's why the searcher could not make any use of the reduction and moved to reusing off-the-shelf pre-trained CNN.



$$\frac{1}{3} \cdot \begin{bmatrix} 0 & +1 & 0 \\ -1 & +1 & +1 \\ -1 & -1 & 0 \end{bmatrix}$$

Figure 4.3: Corner hit-or-miss transform convolutional matrix on left, results in blue compared to Harris corner in red

$$\begin{bmatrix} -\frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & -\frac{1}{4} \\ \frac{1}{12} & 0 & 0 & 0 & \frac{1}{12} \\ \frac{1}{12} & 0 & 0 & 0 & \frac{1}{12} \\ \frac{1}{12} & 0 & 0 & 0 & \frac{1}{12} \\ -\frac{1}{4} & \frac{1}{12} & \frac{1}{12} & \frac{1}{12} & -\frac{1}{4} \end{bmatrix}$$

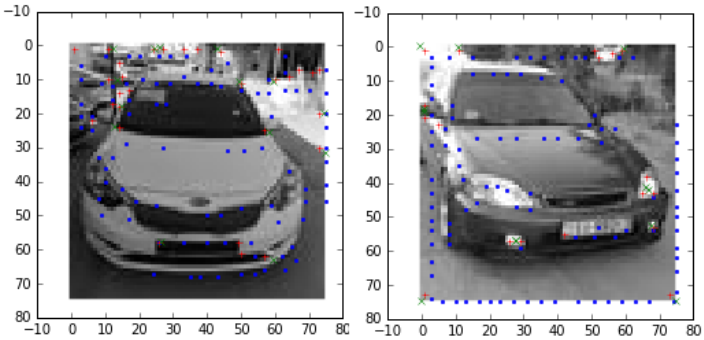


Figure 4.4: Small circle hit-or-miss transform convolutional matrix on left, results in blue

4.3.2 Category Adaptation

Category Adaptation works by finding mapping between any of state-of-the-art pre-trained off-the-shelf models (like ILSVRC-2012-CLS winners). For e-commerce like OpenSooq.com instead of training a model to identify cars, electronics, ...etc. one can just map the one thousand ImageNet classes into his/her own classes as in table 4.1, and flag non-mapped images for human moderation.

Human effort needed to carefully map the one thousand class to relevant categories is much less than preparing a clear dataset suitable for training.

Passing noisy labeled dataset and taking top most classes (using percentiles) is important in both validating human mappings and in finding unusual flaws in ImageNet, for example, images of car inspection papers usually get detected as “restaurant menu” or “website” and pictures showing car seats might get identified as “barber chair” or “stretcher”.

The benefit of “Category Adaptation” that it does not require training nor preparing dataset. On the other hand, it can’t get fine-grained, as you can merge categories but you can’t divide them, for example you can not identify different vehicle types (Sedan, Hatchback, SUV, ...etc.), makers, models. You can’t add new categories, for example smart phones might be detected as modems or other older devices that were part of ILSVRC-2012-CLS database(Russakovsky et al., 2015).

4.3.3 Understanding Weights

In case of usual ANN, the last layer (the fully connected layer) takes features and multiply them with weights matrix ($n \times m$ matrix where n is number of input features, and m is number of classes), add bias and then apply the activation function, then apply

Table 4.1: Example of "Category Adaptation" hand-picked mapping of ImageNet classes to some of OpenSooq e-Commerce Categories

OpenSooq Label	ImageNet Label
Autos	beach wagon minivan jeep limousine convertible pickup minibus sports car tow truck ambulance cab racer
Electronics	cellular telephone, cellphone, mobile phone desktop computer dishwasher, dish washer, dishwashing machine joystick laptop, laptop computer modem printer refrigerator, icebox remote control, remote television, television system vacuum, vacuum cleaner washer, automatic washer, washing machine

"softmax()" to convert them to probability like results. Some CNN have that usual ANN fully connected layer as last layer while others have an equivalent 1×1 convolution of depth m. In both cases we can have a weights matrix of size $n \times m$. As in equation 4.1 we can look to the weights matrix as if it's composed of m vectors $\vec{v}_i$ in feature space of dimension n. Each vector $\hat{v}_i$ represent the direction of that class, and the magnitude of the vector is used to control the probability when two classes are activated by same features.

$$W_{n\times m} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nm} \end{bmatrix} = \begin{bmatrix} \vec{v}_1 & \vec{v}_2 & \cdots & \vec{v}_m \end{bmatrix} \tag{4.1}$$

4.3.4 Using Category Adaptation to Cleanup Dataset

As mentioned before OpenSooq.com provided the researcher with a "Noisy Vehicle Make-Model-Year Dataset". "Category Adaptation" was used to clean this dataset by removing non-cars pictures from it.

A tool named "img-grep" that is inspired by UNIX philosophy and UNIX grep utility was created . This tool reads image file names from standard input, and output only matching file names (or non-matching if –exclude is passed). The matching criteria is either a list of label indices or a file containing such label indices.

Using mapping as in table 4.1, non-car images were removed from dataset, surprisingly more 20% of images were removed, which is understandable because on average a car post have 5 images one of them is a picture of some papers (registration/inspection).

Program 4.3.1 A simple bash script that uses "img-grep" to create a cleaned up version of dataset

```
#! /bin/bash
find src -type f -name '*.jpg' | \
  img-grep --model_dir inception_v1 \
    --idx 437,657,610,628,512,718,655,818,865,408,469,752 |
  while read filename
  do
    cp "${filename}" "${filename/src/dst}"
  done
```

4.3.5 Category Adaptation using weights manipulations

One can produce cars vs. electronics using category adaptation by mapping classes of the results as in table 4.1. One might consider doing this by simply averaging vectors of source classes to be merged in one target class

$$\vec{v_t} = (\vec{v_1} + \vec{v_2} + \ldots + \vec{v_k})/k \tag{4.2}$$

or even better using unit vectors adjusted by its probability in target task p_i and the result vector magnitude is set to be the average magnitude in source task if we are constructing a target model from scratch or the average magnitude in acceptor model in case of knowledge transfer to an existing target model.

$$\vec{v'}_t = \frac{\sum_i p_i \hat{v}_i}{\sum_i p_i} \tag{4.3}$$

$$\vec{v_t} = \bar{V}\hat{v'}_t \tag{4.4}$$

4.3.6 Application on "Cats, Dogs and Birds" dataset

As discussed before4.1 the weights matrix is composed of vectors in feature space, one can use a formula to 4.3 create weights. A small number of pictures were passed from "Cats, Dogs, and Birds" dataset, exactly 100 pictures (out of thousands) from each of the three classes to Inception v1 pre-trained on ImageNet to find probability of donor class in target class and got results as in table 4.2.

4.3.7 Fine Tuning off-the-shelf models

Fine-Tuning an already trained model work by reconstructing a new ANN, transferring weights from the pre-trained network (called donor model or source model) to the newly constructed one (called acceptor or target model) up to some point as seen in figure 4.5.

Weights transferred from the donor model carries already learned knowledge on how to extract features and how to pre-process them(Oquab et al., 2014). On the other hand, weights left to be trained are the decision making parts (decision making layer), which is not transferred but left to be trained on target task which might be different than source task.

The same thing with ConvNets as seen in figure 4.6, as they typically have some fully

Table 4.2: Results of passing some cats, dogs and birds pictures to ImageNet Inception v1

Target Class	Probability p_i	Donor Model Class
Cat	49%	286:Egyptian cat
	19%	285:Siamese cat
	10%	284:Persian cat
	10%	282:tabby cat
	2%	288:lynx, catamount
Dog	14%	181:pit bull terrier
	10%	243:boxer
	8%	213:English setter
	5%	211:German short-haired pointer
	5%	200:Scotch terrier
	4%	262:keeshond
	4%	259:Samoyed
	4%	256:Leonberg
	3%	258:Great Pyrenees
	3%	248:Saint Bernard
	3%	238:miniature pinscher
	3%	203:soft-coated wheaten terrier
	3%	188:Yorkshire terrier
	3%	163:beagle
	2%	274:Canis dingo
	2%	264:Pembroke
	2%	257:Newfoundland dog
	2%	220:cocker spaniel
	2%	158:papillon
	2%	154:Maltese dog
Bird	10%	11:brambling
	9%	17:bulbul
	9%	13:linnet
	6%	147:albatross, mollymawk
	6%	12:goldfinch, Carduelis carduelis
	5%	20:chickadee
	5%	19:magpie
	5%	18:jay
	4%	96:jacamar
	4%	144:oystercatcher
	4%	14:snowbird
	3%	95:hummingbird
	3%	22:kite
	3%	16:Turdus migratorius
	2%	93:bee eater
	2%	92:coucal
	2%	21:water ouzel
	2%	15:indigo bird
	2%	134:bittern
	2%	129:black stork

connected layers (just like usual ANN) toward the end, or have an equivalent 1x1 convolution that act as a fully connected ANN layer, that leads to the classes after being passed to softmax function.

Fine-tuning an existing model gives state-of-art accuracy(Ouyang et al., 2016) with fraction of training time (compared to fully train the new model) as millions of weights are transferred from the donor model without training.

If training one layer was not enough to get the desired accuracy, one might add more fully connected layers toward the end of the network or include more layers in the trainable parts (blue colored layers in figure 4.5).

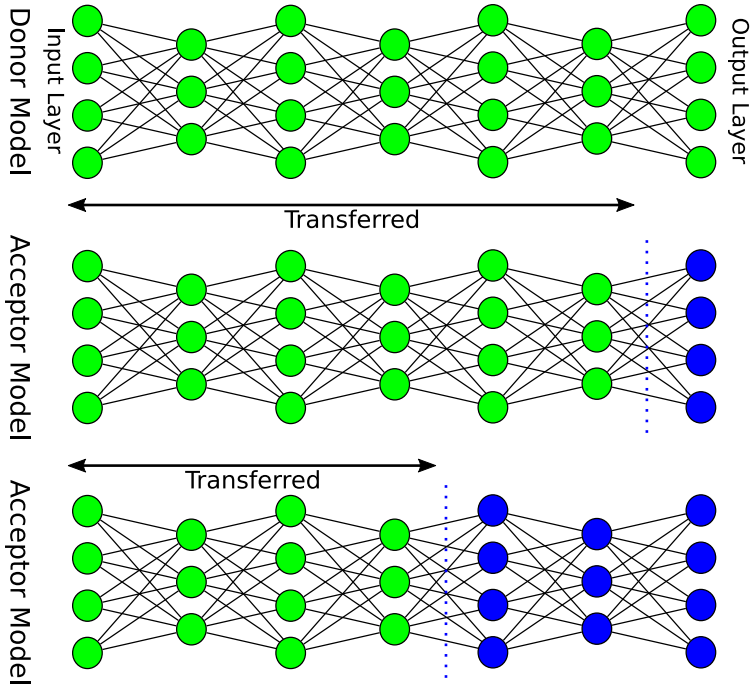


Figure 4.5: How weights are transfered from the donor model to the acceptor model in ANN

4.3.8 Estimating performance of fine-tuning off-the-shelf models

Fine-tuning some models were found to be way more slower than others, to identify the factors that affects the speed of training and fine-tuning, two networks that have very different properties were studied: VGG-16(Simonyan & Zisserman, 2014) which has 15 billion multiplication-accumulation operations and around 140 million weights 75% of which are in a single bottleneck layer (that is FC6), and GoogLeNet or Inception v1(Szegedy et al., 2015) which have small fraction of that, that is less than 1.5 billion multiplication-accumulation operations and less than ten million weights and the weights are almost evenly distributed on layers.

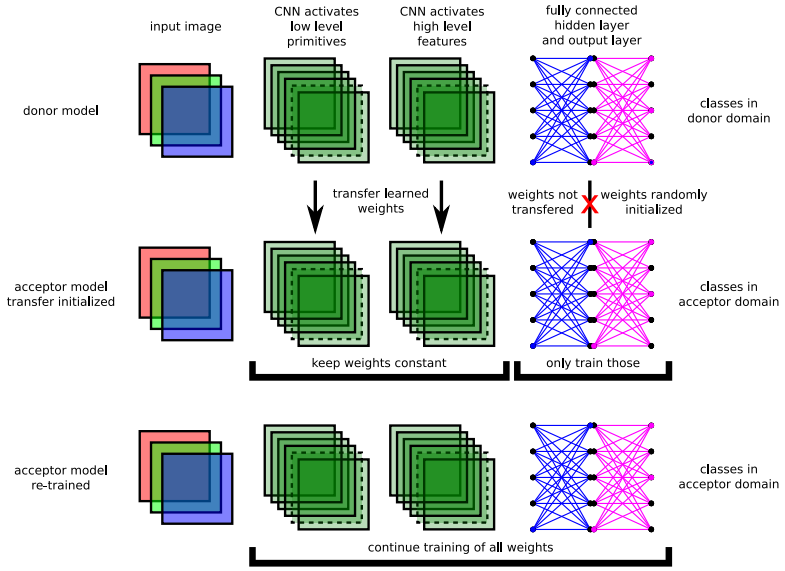


Figure 4.6: How weights are transfered from the donor model to the acceptor model in CNN

The number of weights left for training varies from model to another, so do number of multiplication-accumulation operations, for example, assuming you have n classes, in VGG-16(Simonyan & Zisserman, 2014), the fully connected layers are FC6, FC7, and FC8, one can just retrain FC8 (which has $4096 \times n$ weights) or the whole fully connected ANN starting from FC6 (which is 120 million more weights than FC8 only). or the whole CNN network having 138 million weights, for more details refer to table 3.4.

All experiments in this section were done with 8 images per batch per step, and measure the time needed to reach 500 steps. Table 4.3 shows the time and accuracy of different fine-tuning settings of VGG16(Simonyan & Zisserman, 2014) on toy dataset of five flowers

VGG16(Simonyan & Zisserman, 2014) pre-trained on ImageNet's ILSVRC 2012(Russakovsky et al., 2015) which was previously analyzed in table 3.4. The number of multiplication/addition operations is about 15,470M operation to be exact it's $15466168320 + 4096 \times n$ where n is number of classes. The number of weights to be trained is last layer (named "FC8") is $4096 \times n$, while number of weights in previous layers does not depend on number of classes. As you can see in that table most weights (74.3%) are in "FC6" layer.

Formula 4.5 estimates training time by assuming it's proportional to number of multiplication/addition operations M in forward pass and number of trainable weights W and number of multiplication/addition operations in training part T which is the same as the number of trainable weights times output width and height (which is usually 1×1. Need-

less to say that those constants depends on the speed of the machine, how it's configured, how it handles parallelism, and the model design itself (like non-weights operations like pooling, pre-processing, ..etc.).

$$t = C_0 + C_1 \cdot M + C_2 \cdot W + C_3 \cdot T \tag{4.5}$$

To find C_0 (which is the time to load the model and store it without training), just run zero steps, and that was 8 seconds. To find C_1, C_2, and C_3, by substituting last 3 rows from table 4.3 we get the following linear equation.

$$\begin{bmatrix} 15,466,188,800 & 16,797,696 & 16,797,696 \\ 15,466,188,800 & 119,558,144 & 119,558,144 \\ 15,466,188,800 & 121,917,440 & 581,980,160 \end{bmatrix} \times x = \begin{bmatrix} 772.9 - 8 \\ 2,012.242 - 8 \\ 2,016.7 - 8 \end{bmatrix} \tag{4.6}$$

we get

$$C_1 = 3.64 \times 10^{-8}$$

$$C_2 = 1.21 \times 10^{-5}$$

$$C_3 = -5.21 \times 10^{-8}$$

Having negative value for C_3 indicate that assumption should be changed by removing the term $C_3 \cdot T$ removing the effect of number of trainable weights as in formula 4.7

$$t = C_0 + C_1 \cdot M + C_2 \cdot W \tag{4.7}$$

By substituting first and last rows, and solve the linear equation 4.8

$$\begin{bmatrix} 15,466,188,800 & 20,480 \\ 15,466,188,800 & 121,917,440 \end{bmatrix} \times x = \begin{bmatrix} 582.8 - 8 \\ 2016.7 - 8 \end{bmatrix} \tag{4.8}$$

We get

$$C_1 = 3.71 \times 10^{-8}$$

$$C_2 = 1.18 \times 10^{-5} = 316.8 \times C_1$$

Writing C_2 and C_3 in terms of C_1, we got 4.9

$$t = 8 + C_1 \cdot M + 316.8 C_1 \cdot W \tag{4.9}$$

Using 4.9 substituting second and third rows of table 4.3

$$t_{fc7} = 8 + C_1 \times 15,466,188,800 + C_2 \times 16,797,696 = 780.1$$

$$t_{fc6} = 8 + C_1 \times 15,466,188,800 + C_2 \times 119,558,144 = 1,988.9$$

which is very close from actual value of 772.9 and 2,012.2 with error 0.9% and 1.1% respectively.

Repeating same experiment with different dataset, this time with Birds 200(Wah et al., 2011), results are shown in table 4.4. Using formula 4.5 gave very close values

$$C_1 = 3.62 \times 10^{-8}$$

$$C_2 = 1.23 \times 10^{-5}$$

$$C_3 = -5.44 \times 10^{-8}$$

And when omitting T term as in formula 4.7, we got the following result

$$C_1 = 3.69 \times 10^{-8}$$

$$C_2 = 1.20 \times 10^{-5} = 324.4 C_1$$

$$t = 8 + C_1 \cdot M + 324.4 C_1 \cdot W$$

which is very close from our previous result.

Although the cost of trainable weights is 353 times that of multiplications, in case of fine-tuning last layer as in FC8 layer, number of multiplication-accumulation operations M is 755 thousand times greater than number of trainable weights W.

This huge gap between number multiplication-accumulation operations and number

weights in case of fine-tuning neglect any effect of number of classes on time The cost is merely controlled by number of multiplication-accumulation operations in forward pass is almost constant 15,466M as seen in table 4.5. Same conclusion applies to Inception v1 as seen in table 4.6 where the number of classes does not have any effect of time with fixed trainable layers.

Several experiments were applied on fine-tuning different layers of Inception V1 and results were recorded on table 4.6. Forming an equation similar to 4.7 by substituting the time of fine-tuning a single layer of Inception V1 on flowers 5 dataset and that of fine-tuning both last layer and last inception block "Mixed_5c"

$$\begin{bmatrix} 1,497,357,312 & 5,120 \\ 1,497,357,312 & 1,349,632 \end{bmatrix} \times x = \begin{bmatrix} 94.0 - 8 \\ 102.8 - 8 \end{bmatrix} \tag{4.10}$$

Solving it would result in:

$$C1 = 5.74 \times 10^{-8}$$

$$t = 8 + C_1 \cdot M + 114.0 C_1 \cdot W$$

and by applying it on fine-tuning last layer and last two inception blocks "Mixed_5c" and "Mixed_5b"

$$t = 8 + C_1 \cdot M + 114.0 C_1 \cdot W$$

$$t = 8 + 1,497,357,312c1 + 2,326,528c2 = 109.2$$

which matches the actual observed value of 109.5 seconds with error of 0.3 seconds.

When more classes are included, converging becomes slower as it's harder to reach good accuracy this is seen in tables 4.5 and 4.6.

The time needed to fine-tune last layer of Inception V1 is $3.7\times$ faster than full training (as in table 4.6) and with VGG-16 it's $5.6\times$ faster (as in table 4.3).

Not only it's faster but also better accuracy, because:

- it can run more training steps in same amount of time.
- all convolution filters/pre-processors and high-level feature-extractors are already

Table 4.3: Comparing fine-tuning VGG-16 performance up to different layers on same Flowers-5 dataset

Model	Layer	Classes	Total Mults	Trainable Weights	Training Mults	Time	Accuracy
VGG-16	FC8	5	15,466M	20K	20K	582.8s	80.0%
VGG-16	FC7	5	15,466M	16,798K	16,798K	772.9s	76.1%
VGG-16	FC6	5	15,466M	119,558K	119,558K	2,012.2s	45.9%
VGG-16	Conv5-3	5	15,466M	121,917K	581,980K	2,016.7s	57.6%
VGG-16	All	5	15,466M	134,269K	15,466,189K	3,249.8s	42.7%

Table 4.4: Comparing fine-tuning VGG-16 performance up to different layers on Birds-200 datasets(Wah, Branson, Welinder, Perona, & Belongie, 2011)

Model	Layer	Classes	Total Mults	Trainable Weights	Training Mults	Time	Accuracy
VGG-16	FC8	200	15,467M	819K	819K	588.6s	22.8%
VGG-16	FC7	200	15,467M	17,596K	17,596K	784.1s	10.0%
VGG-16	FC6	200	15,467M	120,357K	120,357K	2,043.9s	00.5%
VGG-16	Conv5-3	200	15,467M	122,716K	582,779K	2,047.8s	00.3%

trained.

This is only valid up to a limit, because the second point does not hold, the target task might require different pre-processing or feature extractor than source task to achieve the desired accuracy, in that case fine-tuning can be used for initialization.

4.3.9 Effectiveness of smaller batch sizes

Trying to achieve high accuracy fine-tuning "The Caltech-UCSD Birds-200-2011 Dataset"(Wah et al., 2011). The test was done 4 times using 200, 100, 50, and 10 batch size having fixed learning rate of 0.01 and the result was as in figure 4.7

Metrics measured on the batch while training (like cross-entropy-loss) will be over

Table 4.5: Comparing fine-tuning same layer of VGG-16 network on different datasets

Model	Layer	Dataset	Total Mults	Trainable Weights	Training Mults	Time	Accuracy
VGG-16	FC8	Cats Dogs 2	15,466,176,512	8,192	8,192	582.4s	99.3%
VGG-16	FC8	Flowers 5	15,466,188,800	20,480	20,480	582.8s	80.0%
VGG-16	FC8	Pets 37	15,466,319,872	151,552	151,552	582.6s	82.9%
VGG-16	FC8	Flowers 102	15,466,586,112	417,792	417,792	584.0s	57.8%
VGG-16	FC8	Birds 200	15,466,987,520	819,200	819,200	588.6s	22.8%

Table 4.6: Comparing fine-tuning Inception V1 performance up to different layers on different datasets

Model	Layer	Dataset	Total Mults	Trainable Weights	Time	Accuracy
Inception V1	Logits	Cats Dogs 2	1,497,354,240	2,048	94.0s	99.3%
Inception V1	Logits	Flowers 5	1,497,357,312	5,120	94.1s	71.8%
Inception V1	Logits	Pets 37	1,497,390,080	37,888	94.3s	75.0%
Inception V1	Logits	Flowers 102	1,497,456,640	104,448	94.1s	18.8%
Inception V1	Logits	Birds 200	1,497,556,992	204,800	94.6s	2.2%
Inception V1	Mixed_5c	Flowers 5	1,497,357,312	1,349,632	102.8s	75.0%
Inception V1	Mixed_5c	Birds 200	1,497,556,992	1,549,312	103.5s	8.0%
Inception V1	Mixed_5b	Flowers 5	1,497,357,312	2,326,528	109.5s	79.2%
Inception V1	Mixed_5b	Birds 200	1,497,556,992	2,526,208	110.8s	7.0%
Inception V1	All	Flowers 5	1,497,357,312	8,718,784	355.4s	61.4%
Inception V1	All	Birds 200	1,497,556,992	9,118,144	355.6s	6.3%

estimated as they will give results based on the small non-representative batch (for example, the ten images in the batch). That's why 10% of dataset were used for validation (more than one thousand image, independent of training batch size) to evaluate the model periodically.

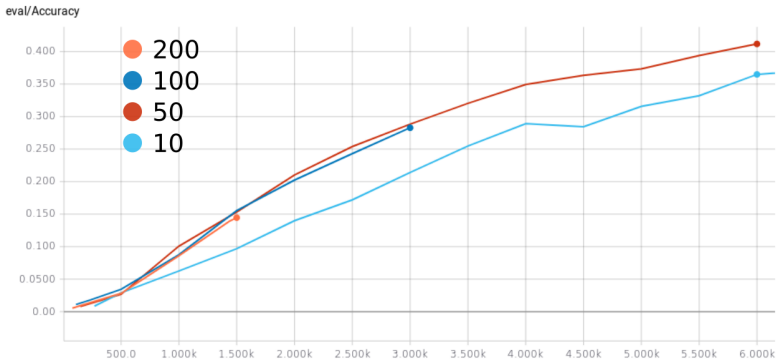


Figure 4.7: Top-1 Accuracy fine-tuning Birds 200 dataset with different batch sizes, x-axis is in steps

By looking at accuracy over steps one might think that a batch size of 10 is the worst, after 1500 step it was only 10% accuracy, while others were around 15%. But this is not a good measure as a batch size of 10 is 5 times faster than batch size of 50, and 20 times faster than batch size of 200. By making x-axis measure in time (hours) instead of steps, the result was as in figure 4.8

This is good accomplishment as more than 50% accuracy was reached in less one and half hour, specially that this dataset contains too many classes (200) and they look very similar even to expert human.

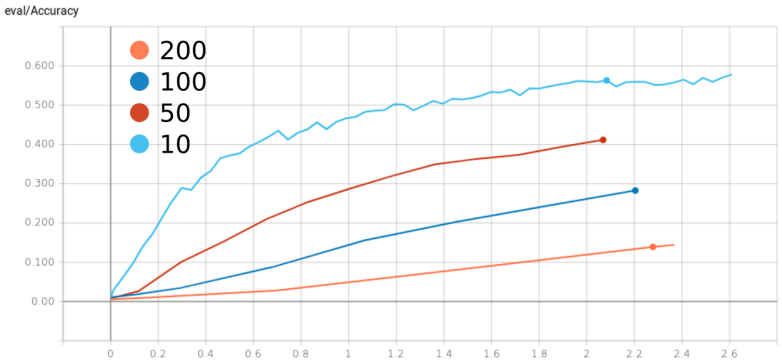


Figure 4.8: Top-1 Accuracy fine-tuning Birds 200 dataset with different batch sizes, x-axis is in hours

4.3.10 Adaptive batch size

Previous section showed how small batch size can be effective (as seen in figure 4.8), it can make speed of training multiple times faster. But it will get stuck after some iterations, that's why one need to use an adaptive approach, that is keep using "fast-forward" mode of small batch sizes as long as it's giving better accuracy compared to previous run, if not switch to slower mode with larger batch size.

Program 4.3.2 adaptive batch size algorithm in pseudo code

```
1  LET old_accuracy = 0
2  LET batch_size, steps_count = batch_size_normal, steps_count_normal
3  train_steps(steps_count, batch_size)
4  accuracy = evaluate()
5  IF (accuracy>old_accuracy) THEN
6    batch_size, steps_count = batch_size_ff, steps_count_ff
7  ELSE
8    batch_size, steps_count = batch_size_normal, steps_count_normal
9  END IF
10 old_accuracy = accuracy
11 IF not done THEN goto 3
```

More generic algorithm, that is to define some fast forward criteria "FF_CRITERIA", when satisfied, training settings are set to faster mode in terms of batch size, steps count, learning rate, ..etc. and when not, normal slower settings are used.

4.3.11 Fine Tuning "Cat, Dog, or Bird" with adaptive batch size

Fine tuning just the last layer of Inception V1 pre-trained on ImageNet 1K classes, is very effective. The three-classes "Cat, Dog, or Bird" database as seen in figure 4.9 took only two hours (on CPU-only setup) to reach 98.9% top-1 accuracy, see table 4.7 for all

Program 4.3.3 more general adaptive batch size algorithm in pseudo code

```
1  init_settings()
2  train_steps()
3  accuracy = evaluate()
4  IF (FF_CRITERIA) THEN
5    apply_settings(ff_settings)
6  ELSE
7    apply_settings(normal_settings)
8  END IF
10 IF not done THEN goto 2
```

Table 4.7: Performance metrics of fine-tuning single layer of Inception v1 on "Cat, Dog or Bird" Task

Metric	Top 1	Top 2	Precision	Recall	F1-Score
Cat, Dog or Bird	98.90%	100%	98.86%	98.87%	98.87%

metrics, and see confusion matrix in table 4.8.

This is very impressive as this layer only have 3072 trainable weights.

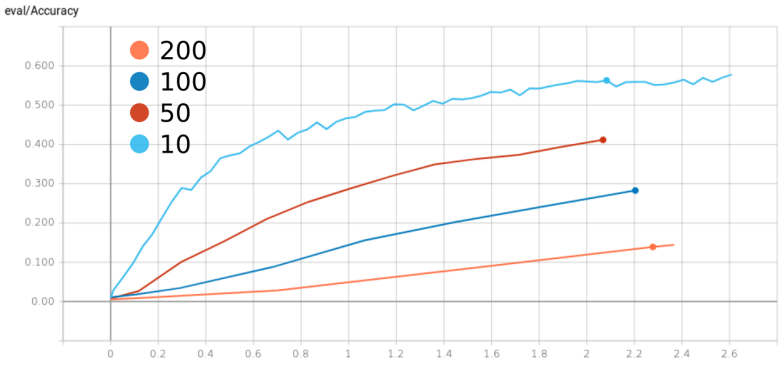


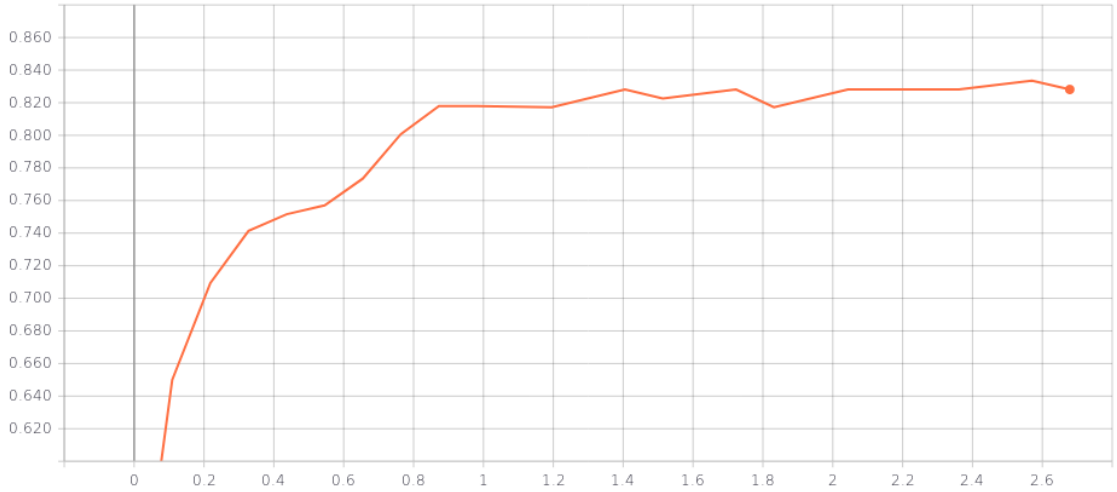
Figure 4.9: Accuracy over time (in hours) of fine-tuning last layer of Inception V1 on "Cat, Dog or Bird" dataset

4.3.12 Fine Tuning Car Sides Dataset

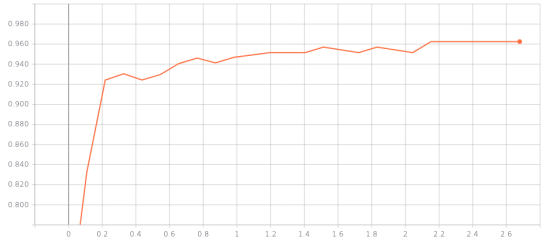
Same procedure was applied to the nine-classes dataset of car sides. And as seen in figure 4.10, it reached 82% top-1 accuracy in less than one hour. This is impressive because this dataset by design has some confusion as it's difficult to draw a strict line between back view, partly-back partly-side, and side view, That's why top-3 accuracy of 99% was reached as seen in table 4.9.

Table 4.8: Confusion matric for fune-tuning “Cat, Dog or Bird” Task

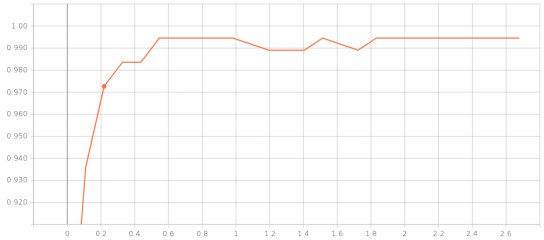
-	Cat	Bird	Dog	Total
Cat	428	0	8	436
Bird	0	455	0	455
Dog	6	0	383	389
Total	434	455	391	1280



(a) Reaching 82% top-1 accuracy after in less than one hour of training



(b) Reaching 96% top-2 accuracy in about two hours of training



(c) Reaching 99% top-3 accuracy in about one hour of training

Figure 4.10: Top-1, Top-2 and Top-3 Accuracy over time (in hours) Fine-Tuning last layer of pre-trained Inception v1 on 9-classes car sides dataset

Table 4.9: Performance metrics of fine-tuning single layer of Inception v1 on “Car sides” Task

Metric	Top 1	Top 2	Top 3	top 5	Precision	Recall	F1-Score
Car Sides 9	83.36%	96.25%	99.45%	100.00%	85.14%	79.71%	82.34%

4.3.13 Failure of stalling accuracy of fine-tuning of single layer

ImageNet 1K-classes has so many generic high-level features and it can be easily fine-tuned into identifying a cat from dog. But if the task is very specific to the level that the already learned generic features are not sufficient to achieve accuracy. Birds-200-2011 Dataset(Wah et al., 2011), has 200 breads of birds, many of them look very similar even to human eyes as you can see in figure 4.11. And as seen in figure 4.12, fine-tuning of a last layer got stuck at around 55% for more than two and half hours.

(a) An image labeled with “Baltimore Oriole” (b) An image labeled with “Hooded Oriole”

Figure 4.11: Two bird breads that look similar even to some humans

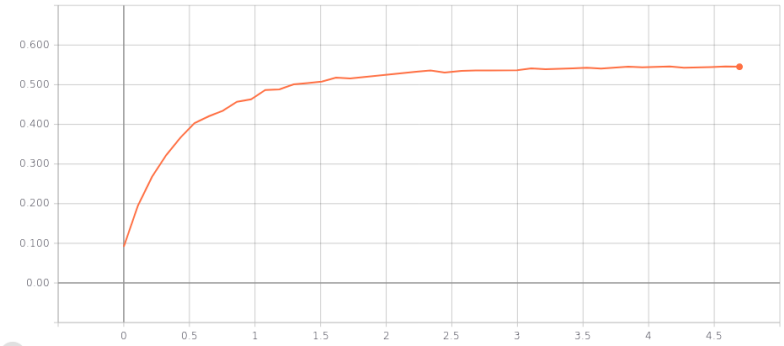


Figure 4.12: Accuracy over time (in hours) of fine-tuning last layer of Inception V1 on “Birds-200” dataset

The solution to this problem is to either add extra layers that are not part of the original design of the ConvNet, or include more layers in the training process after fine-tuning a single layer as in figure 4.5.

Table 4.10: Performance metrics of fine-tuning single layer of Inception v1 on "Birds-200" Task

Birds 200	Top 1	Top 2	Top 3	top 5	Precision	Recall	F1-Score
Logits	54.30%	70.23%	77.19%	84.69%	55.56%	56.29%	55.92%
Mixed-5c	60.39%	74.14%	81.17%	88.13%	61.81%	62.34%	62.08%

Since Inception v1 has branches, the first layer just before last layer (Logits) is an entire inception module block named "Mixed_5c" consisting of four branches similar to that in figure 3.4 previously discussed in section 3.1.5.

Figure 4.13 shows that including an inception module block beside last fully connected layer was able to push accuracy from 55% to 60% within one and half hours for more metrics refer to table 4.10.

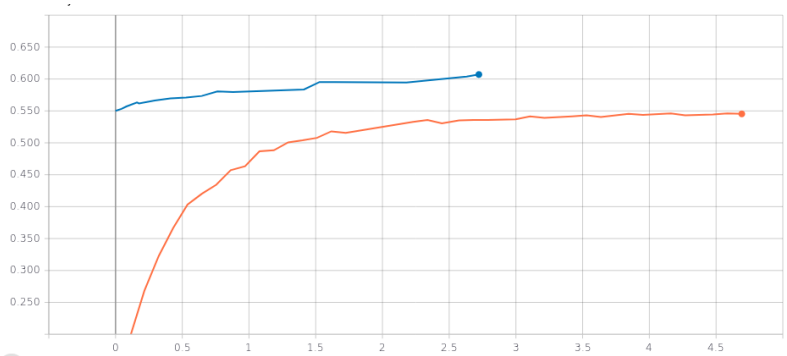


Figure 4.13: Accuracy over time (in hours) of fine-tuning a block "Mixed_5c" (blue) along with last layer (orange) of Inception V1 on "Birds-200" dataset

4.3.14 The problem of "none" class

When training a model that distinguish a car from a none car, then the needed dataset should have images for cars and all kind of non-cars. It's very easy to have a dataset of car front view, car back view, car side view but adding a fourth class "none" is very difficult, because it would require having photos all kinds of other things (animals, humans, plants, ...etc.).

Trying to add opposite of the mean vector of the classes did not work at all. Taking weights from ImageNet and averaging none-car vectors resulted on almost zero vector. Passed weight vectors to "sign()" function showed that each feature has almost equal number of positive and negative items, which is the reason why averaging did not work, and components canceled each other.

That's why a different way is needed to calculate weights vector of "none" class.

4.3.15 Jointly Activated Classes and Cosine Similarity

When a pivot class is taken (like "wagon"), its weights vector $\hat{v}_i$ can be compared to other classes $\hat{v}_j$ using cosine similarity, and when using unit vectors, the similarity equals to the dot product as in 4.11

$$\cos\theta = \frac{\vec{v_i} \cdot \vec{v_j}}{|\vec{v_i}| \cdot |\vec{v_j}|} = \hat{v}_i \cdot \hat{v}_j \tag{4.11}$$

The result would be in the interval [-1, +1] from where -1 means the opposite and 1 means exact match. Table 4.11 shows top similar and dissimilar classes to some chosen pivots.

Definition 4.3.1. Jointly-Activated Classes: Two classes are said to be "Jointly-Activated Classes" if the dot product of their weights unit vectors is positive (they have positive positive cosine similarity).

In a diverse and rich CNN like ImageNet models, the number of similar classes is found to be very close to dissimilar, that is having 450 positive similarity and 550 negative similarity (as seen in first column of table 4.11). Which make it possible to use zero as threshold between similar and dissimilar.

For other models one might set a different threshold other than zero. One might use a percentile-based threshold, the 50th-percentile would take half of the classes as similar, setting the threshold to the 75th-percentile would take only 25% of the classes as similar, and the rest 75% as dissimilar.

Definition 4.3.2. Tolerated Jointly Activated Classes: For a given tolerance τ, two classes are said to be "Tolerated Jointly-Activated Classes" if the dot product of their weights unit vectors is greater than of equal τ.

$$\hat{v}_i \cdot \hat{v}_j \geq \tau \tag{4.12}$$

One can't partition classes into "Equivalence Classes" directly using jointly activated classes because they do not form a partition on the set of classes. But there are many ways of partitioning the classes, one of them is by taking the class having the largest magnitude of weight vector, then find all jointly activated classes then only take a fixed percentile of the similarity, remove most similar classes then repeat.

Table 4.11: Cosine Similarity between ImageNet Classes measured on MobileNet weights. Under pivot classes two numbers are shown, number of similar classes and dissimilar classes

Pivot Class	Top Similar Classes	%	Top Dissimilar Classes	%
437:Wagon +443 -557	657:minivan	24.7%	611:T-shirt	11.3%
	512:convertible	22.4%	778:scabbard	10.6%
	610:jeep	20.4%	595:harp	10.1%
	628:limousine	19.2%	328:starfish	10.0%
	582:radiator grille	17.4%	924:plate	09.7%
852:television +441 -559	783:CRT screen	27.4%	54:ring-neck snake	09.0%
	599:home theater	24.6%	936:mashed potato	08.8%
	549:entertainment center	24.1%	318:leafhopper	08.4%
	665:monitor	23.8%	484:castle	07.9%
	528:desktop computer	15.1%	162:basset hound	07.7%
285:Siamese cat +443 -557	286:Egyptian cat	20.6%	298:sloth bear	11.0%
	288:lynx	19.7%	818:sports car	10.9%
	255:pug	19.7%	308:weevil	10.0%
	226:malinois	19.0%	545:Dutch oven	09.9%
	287:cougar	16.9%	123:American lobster	09.8%
89:macaw +432 -568	91:lorikeet	34.8%	535:dish washer	11.5%
	88:African gray parrot	26.3%	175:Norwegian elkhound	10.7%
	97:toucan	24.5%	71:Phalangium opilio	09.7%
	90:sulphur-crested cockatoo	24.5%	180:Staffordshire bull-terrier	09.2%
	93:bee eater	23.1%	55:hognose snake	09.0%

4.3.16 Category Adaptation using Fuzzy logic on Cosine similarity

Given pivot class (or classes or a weight unit vector), instead of creating "none" class by averaging vectors of all other classes and suffering from weights canceling each other, one can average only dissimilar classes (either those negative cosine similarity or with cosign similarity below some percentile-based threshold) or even better give weights to all other classes by their dissimilarity ($1 - similarity$).

Depending on the model and how feature space is distributed, a negative feature might never appear due to ReLU activation from previous layers, that's why a non-negative multiplier on weights unit vectors is needed so that the resulting direction of unit vector is valid in feature space. To find the unit vector of "none" class, one can exclude similar unit vectors similar classes instead of multiply them by negative number. Event better one might use probabilistic similarity4.3.3 and fuzzy logic.

Definition 4.3.3. Probabilistic Similarity: For a given cosine similarity S_{ij} between classes i and j, the probabilistic similarity p_{ij}^d is defined to be

$$p_{ij}^s = \frac{1 + S_{ij}}{2} = \frac{1 + \hat{v}_i \cdot \hat{v}_j}{2} \tag{4.13}$$

Definition 4.3.4. Probabilistic Dissimilarity: The probabilistic dissimilarity p_{ij}^d is defined to be

$$p_{ij}^d = 1 - p_{ij}^s \tag{4.14}$$

The probabilistic version of similarity4.13 and dissimilarity**??** are in the range [0, 1]. And since it's in that range it can also be treated as probability and apply fuzzy logic-like operations like "AND"/"OR" using addition for "OR" (as in formula 4.3) and multiplication for "AND".

For example, if one wants "not this class" and "not that class" one can use multiplication as in 4.15

Definition 4.3.5. Joint Probabilistic Dissimilarity: Given multiple classes $c_1, c_2, c_3 \dots c_n$, the joint probabilistic dissimilarity of them against a given class c_j is the multiplication of all normalized dissimilarity

$$\prod_i p_{ij}^d \tag{4.15}$$

4.3.17 Injecting “None” class using weights manipulations

As discussed before in equation 4.1 the weights matrix is composed of vectors in feature space, one can use a formula similar to 4.3 create “none” class, for example, if we have Bird, Cat or Dog model, we can inject a fourth class of “none of them”.

”Non-car“ class were injected into the CNN that identifies car sides in section 4.3.12 so the result is a model that can identify cars from non-cars, and if it’s a car it would indicate from which side the picture was taken. As seen in figure 4.14, a cat picture which was previously identified as front side view is now identified as “none”

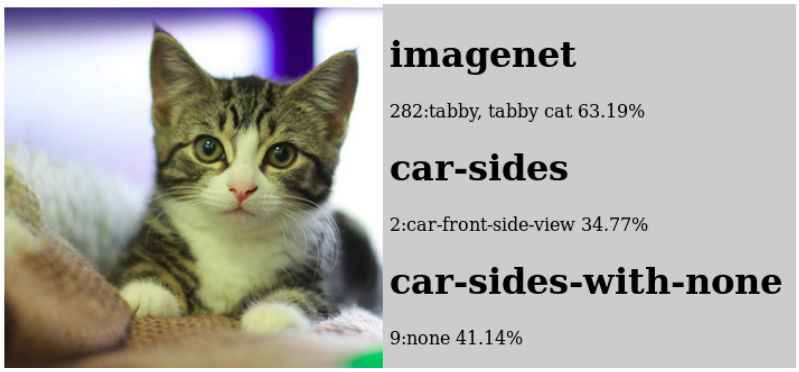


Figure 4.14: Cat mistakenly identified as car front side view before model is manipulated to get it identified as “none”

4.3.18 Fine-Tuned CNN to Cleanup Dataset

Figure 4.15 and 4.16 shows that the design identity of a car is mainly seen in the front and back view. It’s less likely for a human expert to identify car’s make and model from a picture of the interior (seats and dashboard). There are many cars that looks similar from a perfect side view specially if they both are sedan or SUV.

Inception v1 ImageNet were fine-tuned on “Vehicle Viewing Angles Dataset”, then injected non-cars using weights manipulation as in a previous section, The resulted model was used to clean up the noisy dataset, by removing interior pictures, side pictures, and non-cars.

4.3.19 Fine-tuning small number Make/Models/Years

The goal of this research is to identify car’s make/model/year from its picture. Initially, toy problem of recognizing very small number of popular sedan make/model/years was examined:

(a) Toyota Camry Front View

(b) Toyota Camry side view with partial front view

Figure 4.15: How the partly-front partly-side view of this car carries the design identity of front view

(a) Toyota Camry Back View

(b) Toyota Camry side view with partial back view

Figure 4.16: How the partly-back partly-side view of this car carries the design identity of back view

1. Honda,Civic,1996-2000
2. Honda,Civic,2001-2005
3. Honda,Civic,2006-2008
4. Hyundai,Avante,HD-2007-2010
5. Hyundai,Avante,MD-2011+
6. Hyundai,Avante,Old-1990-1999
7. Hyundai,Avante,XD-2000-2006
8. Toyota,Camry,2007-2011
9. Toyota,Camry,2012-2017

Inception V1 model pre-trained on ImageNet 1K task was fine-tuned on those nine classes task by just training last layer.

4.3.20 Fine-tuning 110-classes of Make/Models/Years

110 make/models/years were included from top sold cars, that collectively form more than 70% of cars sold in the region. Some models are almost identical that get rebadged differently by different dealers in different countries, for example, “Hyundai Elantra” is rebadged as “Hyundai Avante”, and “Chevrolet Tahoe” which is rebadged “GMC Yukon”. For this reason, only one of them is picked in the dataset. And if two cars looks very similar, we only pick the most popular one.

Accuracy of 80% top-1 single picture was achieved after two weeks of training. This is very impressive result because, users upload 5 pictures on average, using the previous side model, front or back views can be given higher priority which will result on more accurate results.

4.3.21 Faster way to including one more car model: “Introduced Confusion”

Given a model that can identify 110 make model year, that needs to be extended to include one more car model Making it 111-classes task without repeating weeks of training. To add “Mercedes-Benz G-Class” to the 110-models task. Passing a sample of training data to the old 110-classes model which does not have G-Class, and identify most frequent class that is confused with it, in that case it was Toyota Land Cruiser 2012. Weights were

Table 4.12: Performance metrics of fine-tuning Inception v1 on “Car-Models” Task

task	Top 1	Top 2	Top 3	Top 5	Precision	Recall	F1-Score
Car Models 110	79.53%	87.89%	89.92%	91.88%	78.82%	70.47%	74.41%
Car Models 116	79.22%	87.19%	89.92%	92.34%	77.87%	72.13%	74.89%
Car Models 134	76.80%	87.19%	90.00%	92.42%	77.10%	74.27%	75.66%
Car Models 205	79.53%	89.14%	91.95%	94.53%	79.70%	78.42%	79.06%
Car Models 229	81.17%	88.20%	90.94%	93.67%	80.89%	79.70%	80.29%

crafted in last layer of target task, by taking source task's weights of size 1024×110, extending it to size 1024×111 by duplicating weights corresponding to “Toyota Land Cruiser 2012” into those corresponding to the newly added class “Mercedes-Benz G-Class” And craft bias values in a similar way. This is merely a good initialization. This proposed technique is called “Introduced Confusion”.

The initial Top-1 accuracy dropped from 80% in source task to 75% in target task, but after 6 hours of fine-tuning it recovered and reached 80%. But top-1 accuracy is not the metric to look for (as a stop condition), but it should be confusion matrix as it should recover, if measured before training all Benz G-class would be wrongly predicted as Toyota Land Cruiser, after enough training the top confusion should return to what it was before “Introduced Confusion”.

This procedure can be used multiple times, This technique was used to extend 110-classes task into 116-classes task without any notable sacrifice in accuracy metrics as seen in table 4.12. And the confusion matrix after training have recovered reporting the same top confusing classes as what was before.

4.3.22 Oversampling The Crafted Confusion

The previous method extended the trained model saving weeks of training time to include one more class. By crafting weights creating confusion between the newly added class (Benz G-Class) and one of the existing classes (Toyota LC). Since stochastic training is used with adaptive mini-batches (mostly of 8 images) and assuming equally likely distribution of classes, then the probability of having a batch that has both “Toyota Land Cruiser 2012” and “Mercedes-Benz G-Class” is less than 0.5% which means after 1000 steps, it would have seen less than 5 batches that can train it on how to resolve this confusion.

To overcome this limitation, we need to over-sample images that belong to the newly injected class and the one that it was used to initialize it creating confusion, in our case

to over-sample “Toyota Land Cruiser 2012” and “Mercedes-Benz G-Class” and to under-sample the rest classes.

This should be done for limited number of steps, just to recover the created confusion, then continue using normal fair sampling. The criteria can be based on number of steps or based on accuracy or confusion matrix.

Chapter 5. Discussion and Recommendations

5.1 Results

Given Inception V1 model trained to solve ImageNet one thousand classes task which have accuracy of 69.8% in that task. Transferring knowledge from that task to solve the task of identifying 229 car models, high accuracy was achieved as seen in table 5.1.

Top 1 accuracy of 81.17% with very close Precision and Recall values indicates that it does not suffer from either false positive problem nor false negative problem. Compared to the accuracy of original source model (69.8%) and the complexity of the target task this is an impressive result.

Another important aspect of this work, that it's expandable, starting with only 110 car models, then expanded it to 116, 134, 205 and finally to 229 car models. Each expansion process took only hours.

Table 5.1: Final model performance for "Car-Models-229" Task

Metric	Value
Top 1	81.17%
Top 2	88.20%
Top 3	90.94%
Top 5	93.67%
Precision	80.89%
Recall	79.70%
F1-Score	80.29%

5.2 Model Choice

There are so many pre-trained models to choose from. Model of choice in this research is Inception v1(Szegedy et al., 2015) due to its relatively small number of multiplication-accumulation operations that falls in the same level of MobileNet, but it's both faster to train, converge and resulted in higher accuracy. The reasons for not using more sophisticated later versions of Inception like v3, v4 and ResNet version of Inception that those models are more complicated and require more time to converge, Using Inception v1, top-1 accuracy of 82% was achieved within only one hour on car sides dataset as we have seen before in section4.3.12. Using more advanced model was a waste of resources for this kind of problem, for example, Inception v4 has seven times more weights, and nine

times more multiplication-accumulation operation than Inception V1. VGG has inferior performance in both speed and accuracy.

DenseNet(Huang et al., 2017) is not suitable for knowledge transfer because most weights are in last layers, transferring weights to inner layers, then training last layer would mean training most weights.

NASNet was evaluated but that resulted in worst speed than "Inception v1" despite the promised speed due to smaller number of parameters and weights, and the use of same pre-processing. This might be attributed to a problem in the implementation due to being very new model.

When MobileNet was evaluated it was at version 1.0, it was faster inference than Inception v1, but it had inferior accuracy and training speed was on bar with Inception v1. MobileNet 2.0(Sandler et al., 2018) was not evaluated as it came after writing most of this research.

5.3 Proposed Procedure

The researcher demonstrated that cosine similarity works well on signal going into last layer and weights vectors forming weights matrix, and thus can be used to craft weights matrix for different tasks like

- add "none" class.
- used for initialization.
- identify most similar classes in the 1K-class ImageNet to be used in category adaptation.

A method to extend an existing model to add more classes was demonstrated, and how that was very effective. "Introduced Confusion", by duplicating weights vector of a similar class for the new class, then fine-tune it, over-sampling those known injected confusion, under-sampling other classes, for enough steps to remove confusion. The researcher have successfully extended A network that recognize different 110 car models was successfully extended into recognizing 134 car models (adding more 24 classes), without repeating weeks of training. One can add more and more classes, as this method scales well, the latest model have 229 classes.

Smaller mini-batch sizes were demonstrated to be very effective, using only ten images per step would work even if you have more than a hundred of classes, where as having

at least an image from each class in each step would no nothing but make training ten times slower. Using ridiculously small sized mini-batches would accelerate reaching a stalled accuracy, in that case one had better use larger batches to overcome this, which the researcher calls "Adaptive batch size". The reason why this works, is that the time needed to process a number of images is proportional to number of images having ten times less images means ten times faster, but the effect of smaller batch size on accuracy (if any) is not linear.

AdaDelta(Zeiler, 2012) optimizer was used in this research.

5.4 Recommendations

- Models with more multiplication-accumulation operations and more trainable weights takes more time to train. For faster models, pick models with less multiplication-accumulation operations.
- For fine-tuning (for example, a single layer), number of multiplication-accumulation operations in the whole network is an important factor (due to forward propagation) where as multiplication-accumulation operations in the trainable part of the network is negligible.
- Number of trainable weights to be fine-tuned is also important factor.
- Shallow networks are faster to converge, but they got stuck in accuracy too soon.
- Deeper networks are more difficult and slower to train, but they can achieve higher accuracy after too long training time.
- Training time is linearly proportional to batch size (number of images in each step), but accuracy is not. This means one can fast-forward training by using smaller batch size.
- Adaptive batch size and gradual layer inclusion exploit the above notes to achieve both higher accuracy and faster training time.
- For generic simple tasks, fine-tuning can reach 99% accuracy in hours by fine-tuning a single layer on commodity Central Processing Units (CPUs) without special hardware nor expensive GPUs.
- More difficult tasks would require training more layers.
- Weights in last layer can be treated as vectors in feature space, and cosine similarity can be applied to them.

- Weights can be crafted from feature-space vectors, this is specially useful for “none” class, as it’s not feasible to have training data for all kinds of not-something (for example, not a car).
- Use trained CNN on a simpler task, cleaning the dataset of a more difficult task, in the case of this research train a CNN to identify car interior, and non-cars to remove them from dataset.
- A trained model can be expended to add a new class, using method of “introduced confusion”.
- Over-sampling/under-sampling can be used to solve confusion.

References

Battiti, R. & Brunato, M. (2017). *The LION way. Machine Learning plus Intelligent Optimization*. University of Trento, Italy. Retrieved from http://intelligent-optimization.org/LIONbook/

Bello, I., Zoph, B., Vasudevan, V., & Le, Q. V. (2017). Neural optimizer search with reinforcement learning. *Proceedings of the 34th international conference on machine learning*, International Convention Centre, Sydney, Australia: PMLR, *70*, 459–468. arXiv: 1709.07417. Retrieved from http://proceedings.mlr.press/v70/bello17a.html

Chollet, F. (2017). Xception: deep learning with depthwise separable convolutions. *IEEE conference on computer vision and pattern recognition (CVPR)*, 1800–1807. IEEE. doi:10.1109/CVPR.2017.195

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273–297. doi:10.1023/A:1022627411411

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: a large-scale hierarchical image database. *IEEE conference on computer vision and pattern recognition (CVPR)*, 248–255. IEEE. doi:10.1109/CVPR.2009.5206848

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2014). Decaf: a deep convolutional activation feature for generic visual recognition. *International conference on machine learning*, 647–655. Retrieved from http://dl.acm.org/citation.cfm?id=3044805.3044879

Dozat, T. (2016). Incorporating nesterov momentum into adam. *International conference on learning representations: Workshop track*.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159. Retrieved from http://dl.acm.org/citation.cfm?id=1953048.2021068

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of human genetics*, *7*(2), 179–188.

Garris, M. D. [Michael D] & Wilkinson, R. A. (1992). NIST special database 3: handwritten segmented characters. *NIST, Gaithersburg, Md.*

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 580–587. IEEE. doi:10.1109/CVPR.2014.81

Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., … He, K. (2017). Accurate, large minibatch sgd: training ImageNet in 1 hour. *arXiv e-prints*. arXiv: 1706.02677. Retrieved from http://arxiv.org/abs/1706.02677

Hahnloser, R. H. & Seung, H. S. (2001). Permitted and forbidden sets in symmetric threshold-linear networks. *Advances in neural information processing systems*, 217–223.

He, K. & Sun, J. (2015). Convolutional neural networks at constrained time cost. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 5353–5360. IEEE. doi:10.1109/CVPR.2015.7299173

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 770–778. IEEE. doi:10.1109/CVPR.2016.90

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks. *European conference on computer vision (ECCV)*, 630–645. Springer.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … Adam, H. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications. *arXiv preprint*. arXiv: 1704.04861. Retrieved from http://arxiv.org/abs/1704.04861

Hsu, C.-W. & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, *13*(2), 415–425. doi:10.1109/72.991427

Huang, G., Liu, Z., van der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. *IEEE conference on computer vision and pattern recognition (CVPR)*, 2261–2269. IEEE. doi:10.1109/CVPR.2017.243

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint*. arXiv: 1602.07360. Retrieved from http://arxiv.org/abs/1602.07360

Ioffe, S. & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. *International conference on machine learning*, 448–456. Retrieved from http://dl.acm.org/citation.cfm?id=3045118.3045167

Jarrett, K., Kavukcuoglu, K., LeCun, Y., et al. (2009). What is the best multi-stage architecture for object recognition? *IEEE 12th international conference on computer vision*, 2146–2153. IEEE. doi:10.1109/ICCV.2009.5459469

Kingma, D. & Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint*. arXiv: 1412.6980. Retrieved from http://arxiv.org/abs/1412.6980

Knerr, S., Personnaz, L., & Dreyfus, G. (1990). Single-layer learning revisited: a stepwise procedure for building and training a neural network. *Neurocomputing: algorithms, architectures and applications*, *68*(41-50), 71.

Krause, J., Sapp, B., Howard, A., Zhou, H., Toshev, A., Duerig, T., … Fei-Fei, L. (2016). The unreasonable effectiveness of noisy data for fine-grained recognition. *European conference on computer vision (ECCV)*, 301–320. Springer. doi:10.1007/978-3-319-46487-9

Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *arXiv preprint*. arXiv: 1404.5997. Retrieved from http://arxiv.org/abs/1404.5997

Krizhevsky, A. & Hinton, G. (2009). Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, *1*, 7.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Proceedings of the 25th international conference on neural information processing systems*, Curran Associates Inc., *1*, 1097–1105. Retrieved from http://dl.acm.org/citation.cfm?id=2999134.2999257

LECUN, Y. (1998). Efficient backprop. *Neural Networks: Trick of the Trade*.

LeCun, Y. et al. (1989). Generalization and network design strategies. *Connectionism in perspective*, 143–155.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. doi:10.1109/5.726791

LeCun, Y., Cortes, C., & Burges, C. J. (2010). Mnist handwritten digit database. *AT&T Labs*, *2*. Retrieved from http://yann.lecun.com/exdb/mnist

Mamalet, F. & Garcia, C. (2012). Simplifying ConvNets for fast learning. *International Conference on Artificial Neural Networks and Machine Learning (ICANN)*, 58–65.

Mangasarian, O. L., Street, W. N., & Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, *43*(4), 570–577.

Montavon, G., Orr, G., & Müller, K.-R. (Eds.). (2012). *Neural networks: tricks of the trade* (2nd ed.). Lecture notes in computer science. Springer.

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (icml-10)*, 807–814.

Nasrabadi, N. M. (2007). Pattern recognition and machine learning. *Journal of electronic imaging*, *16*(4), 049901.

Negnevitsky, M. (2005). *Artificial intelligence: a guide to intelligent systems* (2nd ed.). Addison-Wesle.

Ng, A. Y. (2004). Feature selection, L1 vs. L2 regularization, and rotational invariance. *Proceedings of the twenty-first international conference on machine learning*, 78. ACM. doi:10.1145/1015330.1015435

Nilsback, M.-E. & Zisserman, A. (2008). Automated flower classification over a large number of classes. *Sixth indian conference on computer vision, graphics & image processing (CVGIP)*, 722–729. IEEE. doi:10.1109/ICVGIP.2008.47

Oquab, M., Bottou, L., Laptev, I., & Sivic, J. (2014). Learning and transferring mid-level image representations using convolutional neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 1717–1724. IEEE. doi:10.1109/CVPR.2014.222

Orr, G. B. & Müller, K.-R. (Eds.). (1998). *Neural networks: tricks of the trade*. Lecture notes in computer science. Springer.

Ouyang, W., Wang, X., Zhang, C., & Yang, X. (2016). Factors in finetuning deep model for object detection with long-tail distribution. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 864–873. IEEE. doi:10.1109/CVPR.2016.100

Pan, S. J. & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359. doi:10.1109/TKDE.2009.191

Parkhi, O. M., Vedaldi, A., Zisserman, A., & Jawahar, C. V. (2012). Cats and dogs. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3498–3505). IEEE. doi:10.1109/CVPR.2012.6248092

Raina, R., Battle, A., Lee, H., Packer, B., & Ng, A. Y. (2007). Self-taught learning: transfer learning from unlabeled data. *Proceedings of the 24th international conference on machine learning*, ACM, 759–766. doi:10.1145/1273496.1273592

Reed, S., Lee, H., Anguelov, D., Szegedy, C., Erhan, D., & Rabinovich, A. (2014). Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint*. arXiv: 1412.6596. Retrieved from http://arxiv.org/abs/1412.6596

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., …, Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. doi:10.1007/s11263-015-0816-y

Russell, S. J. & Norvig, P. (2003). *Artificial intelligence: a modern approach* (2nd ed.). Prentice Hall.

Saenko, K., Kulis, B., Fritz, M., & Darrell, T. (2010). Adapting visual category models to new domains. *European Conference on Computer Vision (ECCV)*, 213–226.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Inverted residuals and linear bottlenecks: mobile networks for classification, detection and segmentation. *arXiv preprint*. arXiv: 1801.04381. Retrieved from http://arxiv.org/abs/1801.04381

Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN features off-the-shelf: an astounding baseline for recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 806–813. IEEE. doi:10.1109/CVPRW.2014.131

Simonyan, K. & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint*. arXiv: 1409.1556. Retrieved from http://arxiv.org/abs/1409.1556

Srivastava, R. K. [Rupesh K], Greff, K., & Schmidhuber, J. (2015). Training very deep networks. *Advances in neural information processing systems*, MIT Press, 2377–2385. Retrieved from http://dl.acm.org/citation.cfm?id=2969442.2969505

Srivastava, R. K. [Rupesh Kumar], Greff, K., & Schmidhuber, J. (2015). Highway networks. *arXiv preprint*. arXiv: 1505.00387. Retrieved from http://arxiv.org/abs/1505.00387

Street, W. N., Wolberg, W. H., & Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. *Biomedical image processing and biomedical visualization*, *1905*, 861–871. International Society for Optics and Photonics.

Sukhbaatar, S., Bruna, J., Paluri, M., Bourdev, L., & Fergus, R. (2014). Training convolutional networks with noisy labels. *arXiv preprint*. arXiv: 1406.2080. Retrieved from http://arxiv.org/abs/1406.2080

Sukhbaatar, S. & Fergus, R. (2014). Learning from noisy labels with deep neural networks. *arXiv preprint*, *2*(3), 4. arXiv: 1406.2080. Retrieved from https://arxiv.org/abs/1406.2080v1

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *International conference on machine learning*, PMLR, 1139–1147.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, Inception-ResNet and the impact of residual connections on learning. *The association for the advancement of artificial intelligence (aaai)*, *4*, 4278–4284.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 1–9. IEEE. doi:10.1109/CVPR.2015.7298594

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2818–2826. IEEE. doi:10.1109/CVPR.2016.308

Thrun, S. & Pratt, L. (1998). Learning to learn: introduction and overview. In *Learning to learn* (pp. 3–17). Springer.

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the turing test* (pp. 23–65). Springer.

Van Horn, G., Branson, S., Farrell, R., Haber, S., Barry, J., Ipeirotis, P., … Belongie, S. (2015). Building a bird recognition app and large scale dataset with citizen scientists: the fine print in fine-grained dataset collection. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 595–604. IEEE. doi:10.1109/CVPR.2015.7298658

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer science & business media.

Wah, C., Branson, S., Welinder, P., Perona, P., & Belongie, S. (2011). *The caltech-ucsd birds-200-2011 dataset* (tech. rep. No. CNS-TR-2011-001). California Institute of Technology.

Wilson, C. & Garris, M. (1990). NIST special database 1. *National Institute of Standards and Technology (NIST)*.

Wolberg, W. H., Street, W. N., & Mangasarian, O. (1994). Machine learning techniques to diagnose breast cancer from image-processed nuclear features of fine needle aspirates. *Cancer letters*, *77*(2-3), 163–171.

Wu, J., Leng, C., Wang, Y., Hu, Q., & Cheng, J. (2016). Quantized convolutional neural networks for mobile devices. *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 4820–4828. IEEE. doi:10.1109/CVPR.2016.521

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint*. arXiv: 1212.5701. Retrieved from http://arxiv.org/abs/1212.5701

Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. *European conference on computer vision (ECCV)*, 818–833. Springer.

Zeng, X., Ouyang, W., Yang, B., Yan, J., & Wang, X. (2016). Gated bi-directional CNN for object detection. *European conference on computer vision (ECCV)*, 354–369. Springer.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. *arXiv preprint*. arXiv: 1707.07012. Retrieved from http://arxiv.org/abs/1707.07012

استعمال الشبكات العصبونية الملتفة ونقل المعرفة في التعرف على الصور

إعداد: مؤيد السعدي

إشراف: أ.د. عرفات عوجان

الملخص

يعتبر التعرف على الصور من موضوعات الرؤية الحاسوبية التي تهدف إلى تحديد واحد أو أكثر من الكائنات أو أصناف الكائنات أو المزايا أو النشاطات الموجودة في الصورة المدخلة أو إطارات الفيديو حتى وإن كان محجوبا جزئيا عن المشهد. وتعتبر تقنية الشبكات العصبونية العميقة الملتفة ذروة التقدم فيها هذا المجال لكنها تحتاج للكثير من وقت التدريب وقدرة حسابية عالية.

المقاربة التي تسلكها هذه الرسالة تفترض وجود قيود تتمثل في موارد محدودة من حيث زمن التدريب والقدرة الحسابية العادية المتاحة في الأسواق. إلى جانب ذلك أضيف قيد آخر وهو أن يكون النموذج المطروح قابل للتوسعة. تقنية "نقل المعرفة" استخدمت لإعادة استعمال نماذج مدربة مسبقا متاحة للعموم. إلى جانب تقنيات أخرى لجعلها مجدية وتسريع العملية وجعلها قابلة للتوسعة.

هناك العديد من التطبيقات للتعرف على الصور في نطاقات متعددة مثل التعرف على الوجوه والتعرف على المحارف والتفحص الآلي في خطوط الإنتاج وضبط الجودة والتشخيص الطبي والمهام المتعلقة بالمركبات ذاتية القيادة مثل التعرف على المشاة. إلا أن التطبيق المطروح في هذه الرسالة هو التعرف على الشركة الصانعة للسيارة ونوعها (الموديل) وسنة الصنع من خلال صوتها التي يرفعها البائع على منصة للتجارة الإلكترونية. وتأتي أهمية هذا التطبيق مع تزايد شعبية الهواتف الذكية المزودة بكاميرات حيث أن أخذ صورة أسهل من نقر وصف السيارة حرفا حرفا.

جرى نقل الأوزان من نموذج ImageNet إلى نموذج أول جرى إعادة توليفه ليقوم بتنظيف بيانات التدريب ثم تم عمل عملية نقل أخرى إلى نموذج ثان يستعمل للتعرف على أنواع السيارات المهمة تبعا لحصتها من السوق وبعدها يتم توسعة النموذج كي يشمل نماذج أكثر فأكثر من خلال إجراء تم تقديمه في هذا البحث.

تم إنجاز دقة تزيد عن ٨١٪ في التعرف على ٢٢٩ نوع مختلف من السيارات من خلال إعادة استعمال نموذج Inception V1 المدرب مسبقا لحل مسألة ImageNet ذات الألف فئة بدقة ٦٩,٨٪. ولضمان أن الطريقة المطروحة عامة ولا تتعلق بنطاق معين تم تقييمها من خلال مهام أكاديمية معروفة.