VEHICLE TYPE CLASSIFICATION WITH DEEP LEARNING

A Thesis Submitted to
the Graduate School of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
Neriman YARAŞ

June 2020
İZMİR

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Mustafa Özuysal for sharing his information with me, directing me throughout my thesis and supporting me in this long period of time.

I am grateful to my friends for all the support and their motivation during busy and tiring times.

Finally, I would like to express my infinite gratitude to my parents, my husband Murat Yaraş and our daughter Arya Yaraş for their unconditional love, patience and endless support during this thesis and all my life. I dedicate this thesis work to them.

ABSTRACT

VEHICLE TYPE CLASSIFICATION WITH DEEP LEARNING

In this thesis, we studied the vehicle type classification problem from several perspectives. We apply a deep learning technique with different parameters such as image size and the number of images in data sets to the classification of an image as one of the nine vehicle types. After choosing the most appropriate one among trained models, we convert the problem into a hierarchical tree classification problem so that it could be analyzed in three different tree hierarchies. Experiments are performed using three computational methods for calculating possibilities for each of the nine classes that correspond to the leaves of the hierarchical trees. These studies result in a conclusion that 0.762812 average accuracy is obtained when traditional arithmetic mean computation applied on the hierarchical tree with level-2 using the Stanford Dataset by 224 image size on ResNet34 architecture.

ÖZET

DERİN ÖĞRENME İLE ARAÇ TİPİ SINIFLANDIRMA

Bu tez çalışmasında, taşıt tipi sınıflandırma problemi farklı açılardan incelenmiştir. Bir imgeyi dokuz araç türünden biri olarak sınıflandırmak için imge boyutu, veri kümelerindeki örnek sayısı gibi farklı parametreleri kullanan bir derin öğrenme tekniği uygulanmıştır. Eğitimli modeller arasında en uygun olanını seçtikten sonra, sorunu hiyerarşik bir ağaç sınıflandırma problemine dönüştürerek üç farklı ağaç hiyerarşisinde analiz ettik. Deneyler, hiyerarşik ağaçların yapraklarına karşılık gelen dokuz sınıfın her biri için olasılıkları hesaplamak için üç hesaplama yöntemi kullanılarak gerçekleştirilmiştir. Bu çalışmalar, ResNet34 mimarisinde 224 görüntü boyutuna göre Stanford veri seti kullanılarak seviye-2 ile hiyerarşik ağaçta geleneksel aritmetik ortalama hesaplama uygulandığında 0.762812 ortalama doğruluğunun elde edildiği sonucuna varmaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

Increased popularity and rapid development of novel algorithms for artificial intelligence lead to many new scientific experiments and approaches that include machine learning and deep learning as an important submodule. In this field, there are many problems and occasionally more than one specific approach to solve each problem. Deep learning achieves high performance by exploiting spatial coherence with convolutional layers and pretrained networks trained using large scale data sets with transfer learning. Increased size of the data sets forced technology to evolve around hardware in order to handle the availability of large-scale data sets. Thanks to the evolution of graphics processing units (GPUs) that specialize on processing large amounts of data in parallel, it became easier to work on the large-scale data sets and perform experiments in much shorter time frames. Increasing memory size along with the availability of powerful GPUs enabled researchers to obtain more accurate results for the complex real-world problems such as image classification, real-time pose estimation, and semantic segmentation. These improvements coupled with the availability of open source software libraries such as Keras (Chollet, 2015), Tensorflow (Abadi et al., 2016), and PyTorch (Paszke, 2017) for state-of-the-art algorithms makes it easier to apply deep learning to computer vision problems.

Classification problems involving images are prime candidates for learning based solutions using convolutional neural network (CNN) techniques developed to take advantage of spatial consistency pioneered by Yann LeCun et al. (1998) in 1998. The increase in computation power has allowed deeper neural networks to be constructed, so a solution could be sought for the problem of increasing data volume by using more powerful CNNs. Thanks to its potential to jointly learn both features and classification rules, CNNs give better classification results especially if the datasets are large scale (Kosmopoulos, Paliouras and Androutsopoulos, 2015, Simonyan and Zisserman, 2015).

In this thesis, we propose a deep learning solution for the vehicle type classification problem that builds upon the advances described above. Considering how much we use vehicles in our daily lives or how much we encounter them, we can understand how important it is to deduce information from the images that contain vehicles. Vehicle classification is important not only in the automotive field but also in the related fields such as traffic analysis and insurance (Jayawardena, 2013), it is demanded to create automated

solutions that produce robust and accurate classification output to put in use. Since it is potentially more difficult to learn features that separate all vehicle classes from each other, training a single classifier may not be the best possibility. With the aim of extracting rich features from the data, we propose a vehicle type classification approach utilizing multi-level classifier hierarchies that we built in order to distinguish subsets of vehicle types from each other at each layer.

We test our vehicle type classification approach by experimenting on two different data sets. BIT Dataset consists of images shot on the highway from a single front-view angle that contain vehicles of six different types. Stanford Dataset, on the other hand, contains images taken in different places from different angles and is organized in nine vehicle types. To classify vehicle types using these datasets, we first train single-level models by applying transfer learning (Pan and Yang, 2010) on four different pre-trained models. The aim is to show that with a relatively small number of images for each vehicle type, tangible results can be produced in the vehicle type classification problem.

During the single layer experiments, using deep neural networks on different architectures, their suitability for vehicle type classification problem was determined. Different architectures show different suitability for transfer learning, and some of them produce more successful results. In the single-level classifier trainings, both the last layers of neural networks were trained initially followed by all layers of the architectures and the results of the experiments were shared. Besides the deep neural network architecture, the effect of image size on the classification is another variable that was measured. Also, image size affects both the training speed and the amount of GPU memory that must be used. The results show that even with small images, different types of cars can be separated.

Evaluating the results of single-level training runs, we created a baseline model. Depending on the baseline model we build multi-level classifiers in order to distinguish nine vehicle types from each other; Hatchback, Mini, Minivan, Panel Van, Pick-up, Sedan, Sport, Station and SUV. We trained additional models corresponding to the nodes of the multi-level classifier trees of varying complexity. Then we computed the accuracy for each tree by employing three different computational methods that combines the classifier probabilities from the tree nodes into a single classification result over all nine classes.

Experiments showed that multi-level classifier training approach yields higher performance than single-level classifiers by enabling overall model to predict more accurate results. Taken experimental studies into consideration, the best results are observed when traditional arithmetic mean computation applied on the hierarchical tree with level-2 using the Stanford Dataset by 224 image size on ResNet34 architecture and 0.762812 average

accuracy is obtained by repeating experiments. Additionally, this result was supported by repeating experiments for multi-level classifier training by providing twice number of images for each vehicle types.

1.1 Motivation

According to the survey (Won, 2020), accurate classification of vehicles to different types is essential for effective transportation planning and traffic operation. Vehicle type classification plays a significant role for reducing traffic congestion of future transportation systems. Thanks to increasing usage of cameras, studies on vehicle type classification focused mainly on image-based approaches. Since traffic surveillance cameras provides large number of vehicle images, accurate estimation of vehicle types on those images becomes more important for efficient management of transportation. However, the focus of image-based methods is generally on the images taken from a single viewpoint. In order to cover this issue, we performed experiments on images which are taken from different viewpoints.

Accurate categorization of vehicles into different types is an important issue not only for sufficient transportation planning and management but also for effective damage detection of insurance solutions. In order to reduce the work accidents that may be caused by people during the assessment of damage, automation of the damage assessment process on vehicles is required. At this point, using deep learning techniques to classify vehicles into categories both reduce the wrongdoings and also accelerates the process of detection. By decreasing the discrepancies, deep learning enables to produce more accurate estimates. As already mentioned above, vehicles can be used in many different areas, it means that vehicle type classification subject is open to further development in order to contribute to these areas.

1.2. Thesis Goals and Contributions

This thesis aims to handle the vehicle type classification problem by building multi - level classifier hierarchy by differentiating classes among Hatchback, Mini, Minivan, Panel Van, Pick-up, Sedan, Sport, Station, SUV vehicle types. We employed more than one classification models to distinguish nine vehicle types from each other. We analyzed the machine learning and deep learning approaches for visual classification. To estimate the vehicle type class to which the vehicle, we evaluated several versions of convolutional

neural networks which varies in data sets, image sizes, computational methods and hierarchical tree structures. We applied transfer learning methodology to build single-level models and determined a baseline model for further analysis. Then, based on the baseline model, we built multi-level classifier trees and calculated the accuracy of each tree by computing the probabilities of each label via three computational methods.

1.3. Outline of Thesis

In the next section the related work is presented, while in Section 3 background search of this thesis is mentioned. Section 4 present experiments and discuss experiment results. In Section 5 proposed approach is introduces and related experiments are explained. Finally, Section 6 concludes and points to future work.

CHAPTER 2

RELATED WORKS

There has been a lot of literature on vehicle classification, and it is an inherently hard problem that has received significant attention over the years. In this section, we present previous work of vehicle type classification and hierarchical probability computing.

Thanks to success of neural networks, experimental studies are carried out in almost every area which results in practical applications became available for various fields such as finance, health care, education (Jordan and Mitchell, 2015). For example, in health care sector, Yu et al. (2017) proposed a very deep neural network solution for the automated recognition of skin cancer (Melanoma). In order to increase success rate of trained model, they proposed a two-stage scheme. Firstly they performed segmentation on dermoscopic images by separating the skin lesions from the images. Then they categorized images into two categories which are melanoma and non-melanoma. Instead of performing classification on whole dermoscopy images, aiming for the lesion regions they enabled classifier to obtain more representative and discriminative features. Secondly, they put a deep networks that has more than 50 layers into practice aiming for extracting discriminative features so that they trained model can make more accurate predictions. They utilized from residual learning technique (He et al., 2016) and built a fully convolutional residual network for effective and precise estimations for segmentation of skin injury and its classification. Lastly, connecting contextual information with each other they created a multi-scale system and aiming for overcoming of hardship melanoma recognition automatization they trained models with created with different set of schemes to determine for the most effective ones.

Due to its strong capacity to handle formidable problems and to improve system performance, deep learning can be applied to every type of daily life problems including vehicle detection and classification. According to the survey of intelligent traffic monitoring systems (Won, 2020), thanks to technological advances through learning, sensing, and wireless communication, numerous innovative vehicle classification have been developed. One of the vehicle classification method for Intelligent Traffic Control is proposed by Fazli, Mohammadi and Rahmani (2012) in 2012. They propose a deep learning algorithm in order to categorize mobile images into three vehicle types which are Heavy vehicles, Light vehicles and Motorcycle. The deep learning solution is explained in two-stages in general. The first step is about obtaining images from video of traffic scenes and applying some

image processing techniques such as background subtraction and edge detection. The second step is about extracting features from pre-processed images and creating a vector from these features such that a neural network can be trained on them in order to predict type of the vehicles. Another study is about automatic vehicle type classification using convolutional neural networks. Roecker et. al. (2018) proposed a convolutional neural network model for vehicle type classification that classifies vehicle types into six categories which are Bus, Microbus, Minivan, Sedan, SUV, Truck. They studied on images only with low resolution and taken from frontal perspective. These images are pre-processed by resizing and applying filters and the are given to the model as an input. In the convolutional stage of the model, using supervised learning with high-intensity data augmentation the model outputs a probability of each class. By getting more mature results from experiments and putting into practice, deep learning started to be served not only in traffic surveillance systems but also in other systems such as insurance systems. Patil et. al. (2017) studied on car damage classification problem aiming at car insurance industry. Since there is no public dataset for this problem, they started with preparing the dataset which is a difficult stage through the process of solving the problem. The dataset is created by collecting the image samples from the vehicle regions by taking damage types into consideration. In the first place, this study is carried on small sized dataset but it did not yield to good results using just neural network training, that's why fine-tuning process was applied to get better results. Furthermore, they investigated the effects of merging transfer learning to their solution. At last, experiments show that the most successful results was obtained by using transfer learning rather than applying just fine-tuning to the dataset. Visual classification is an important issue for insurance companies as well as vehicle owners. Insurance companies invest in deep learning methods. Because by providing solutions based on deep learning, delays in through visual inspections can be reduced and damage classification would be done in a reasonable amount of time. For example, a large amount of money would be wasted due to incorrect assessment of damage in the car insurance industry. Thanks to deep learning solutions, insurance companies can minimize money loss avoiding incorrect assessment of damage and reducing delays. Vehicle damage prediction is performed in another study in order to reduce insurance cost after traffic accidents (Jayawardena, 2013). In this study, solution is proposed based on 3D CAD models which are used as ground truth to differentiate mild damages from the undamaged vehicle model by analyzing images even under the conditions of environmental factor. In order to create undamaged model of a vehicle, multi-view geometry techniques are applied on two photographs of the vehicle taken from different view points. They presented 3D pose estimation algorithms and applied

the 3D model projection at the recovered 3D pose so as to differentiate vehicle components from an image in which there could be a slightly damaged vehicle. Because of the reflective body of vehicles, there could be inter object reflections such as ambient light or accident scene in an image of the damaged vehicle which may lead to misclassification of images. To deal with the inter object reflections, they also introduced a method which determines reliable point correspondences through images.

One of the common classification strategies for large scale datasets is building a hierarchy of classifiers. In 1989, Haskell and Noui-Mehidi (1989) defined the hierarchical classifiers as an agglomerative way of categorizing inputs into predefined output classes. With the aspect of "Divide and Conquer", he explained that firstly low-level classifications is trained by highly specific pieces of input data. Then these low-level classification of individuals come together to form larger hierarchical systems on a higher level which determines the output classes of the overall system. Following the trails of Haskell and Noui-Mehidi's thought, Ras, Dardzinska and Jiang (2010) proposes a cascade classification strategy investigated for musical instruments estimation. They introduce a methodology of building cascade classifiers for data sets that can be partitioned into non-singular subsets. They also worked on decreasing the incompleteness of the data before the knowledge extraction algorithms are applied. Then they built a multi-hierarchical decision system experimenting on two different classification hierarchy of instruments. The results showed that by implementing hierarchical decision system the identification of the pitch became easier in the process of recognizing a dominating musical instrument in a musical piece. Another study on the hierarchical classification subject proposes a generic and principled hierarchical architecture, Hierarchical Deep Convolutional Neural Network (HD-CNN). Yan et al. (2015) find subset of classes based on class confusion of a base network by decomposing the classification task into two steps. Firstly easy classes are separated from each other by using coarse category classifier. After that more formidable ones are directed through sub-level category classifiers for estimation to complete. In this proposed system, the classification is implemented by starting from general classifiers through ending detailed classifiers so that error rate could be lowered with the cost of complexity increase.

The problem covered by this thesis address to the many sectors and/or inter-sectors. It might be useful for traffic surveillance systems in terms of requirement for recognition of vehicle types. Or insurance companies might want to enhance their damage detection solutions by applying vehicle type classification in order to understand which types of vehicle mostly got damaged. Leading to more evidence-based decision-making across many walks of life, deep learning methods also help for creating for competitive solutions. They

enable us to not only reduce cost but also increase efficiency by enabling fast decision mechanisms for problems such as damage detection, face recognition, customer segmentation.

CHAPTER 3

RESEARCH BACKGROUND

In this section, ‘Machine Learning’, ‘Types of Machine Learning’, ‘Deep Learning’, ‘Convolutional Neural Network’ and ‘Fastai’ concepts are introduced.

3.1. Machine Learning

Machine learning is a branch of Artificial Intelligence (AI). It focuses on teaching an algorithm to progressively improve itself upon a given set of tasks. In a nutshell, machine learning designs a model of how a process works and implements it for practically building applications that present iterative enhancement. Dynamic mechanism of machine learning enables itself to change when required without human intervention. When a new instance is arrived, it processes historical information to determine which path to take, and takes action for adapting the new instance. Fundamentally, it functions as learning and self-development.

While all machine learning problems can be examined as artificial intelligence problem, on the other hand all artificial intelligence problems can not be assessed as machine learning problem. Because artificial intelligence problems cover machine learning problems. Machine learning is an applied form of artificial intelligence. It is for enabling machines to learn like humans by collecting, storing, analysing data and developing a decision making ability on its own. In order to extract predictable pattern or meaningful data whole data provided is needed to be analyzed. After that using statistical analysis a model to make predictions against new data is built which has the ability to make inferences according to the previous data provided to it. The most difficult part of this learning process is to ask right questions leaning on the collected data. In this case, it is necessary to have right data. For example, role information, such as an administrator role or a normal user role, can be important information about a user logged on to the management portal.

3.1.1. Types of Machine Learning

Machine learning algorithms are generally used to predict or classify data. There are three main categories: supervised learning, unsupervised learning and reinforcement learning.

3.1.1.1. Supervised Learning

Supervised learning requires training a model that would be useful for classifying new data by making predictions on predetermined classes. Therefore, training data which includes data points and labels are given to the learning algorithm. In other words, data being used for training and its labels of classes or categories are already known within the training system so that the learning algorithm is able to estimate based on the learned data.

In general, supervised learning is used to find a solution to classification and regression problems. For classification problems, data is needed to be divided into groups in accordance to the categories defined so that, the system is trained using these given categories and classifies new incoming data similarly to as it learns. For example, suppose that the task of recognizing a cat present within the given picture is given to the machine. On this account, the training dataset with containing a group of images with detail of whether there is a cat in image or not would be needed to be learned by the machine beforehand. Another example could be training a model by giving the size and the age of a group of people as input and expecting their weight as an output. And if enough data is given to the trained model, it would be able to make predictions of the weight of someone whose weight is unknown given his/her size and age.

The challenge of supervised learning is that the learning algorithm depends on the accuracy of the class predictions because it estimates generalized rules that supports the training dataset. Accuracy of trained model gets better by providing enough number of images. However, focusing too much on the training dataset would result in over-fitting which means that trained model exhibits poor performance on real-world scenarios. As an example, if a training dataset mostly includes breed of Van cats, the machine may find it harder to recognize the presence of cats of other breeds.

3.1.1.2. Unsupervised Learning

Unlike supervised learning, unsupervised learning does not require a labeled dataset, and no training dataset is provided to the machine anyway. Instead of a training dataset, unstructured data is given to the machine so that the machine work on exploring structures within the data provided. Unsupervised learning is often used to find patterns. It explores similarities in data in order to cluster related data close to each other. Therefore, this method is suitably used for clustering problems. To solve clustering problems, data should be divided into groups taking their similarities into account such that each group represent a

cluster. In order to minimize similarities between clusters, data that has different values from others should not be in a group and it should form another group. In terms of processing data, a new coming data is examined by the machine and assigned to a cluster which has similar data. Face recognition over an outsized dataset of photographs is an example of unsupervised learning. The machines cluster the photographs together finding similarities of faces. The task of machine is recognizing which images belongs to the same person while ascertaining how many different people exists in the dataset.

3.1.1.3. Reinforcement Learning

In reinforcement learning, the machine tries to investigate the most appropriate actions in certain scenarios. The machine is required to explore cause and effect relationship, whether these actions have short or long term consequences.

The idea behind reinforcement learning comes mainly from psychology of experiments on animal such as rats, and behaviors of these animals on certain situations like seeking for food. One example of reinforcement learning is that in order to make the rat to go and get the food, a stimulating sound is given to the mouse by an agent in the same environment. As a result the machine learns on its own behaviours that cause a gift or punishment according to its previous attempts that caused positive or negative effects. The most famous example for reinforcement learning is that an artificial intelligence product AlphaGo defeated the world champion Ke Jie in the game of go (Google AI Blog, 2016). Detailed information about types of machine learning can also be explained in (Başţanlar and Özuysal, 2014).

3.2. Deep Learning

Deep learning is machine learning technique which extract feature from data like images, texts or sounds. Extracted features by this technique are learned and can be used for next tasks. The main difference between how machine learning and deep learning works is demonstrated in Figure 3.1. While in machine learning, the way to make correct estimation should be taught to algorithm by providing more information, on the other hand, the algorithm in deep learning can learn this by processing data. In simple words, feature extraction is performed by human in machine learning while deep learning model figure it out by itself.

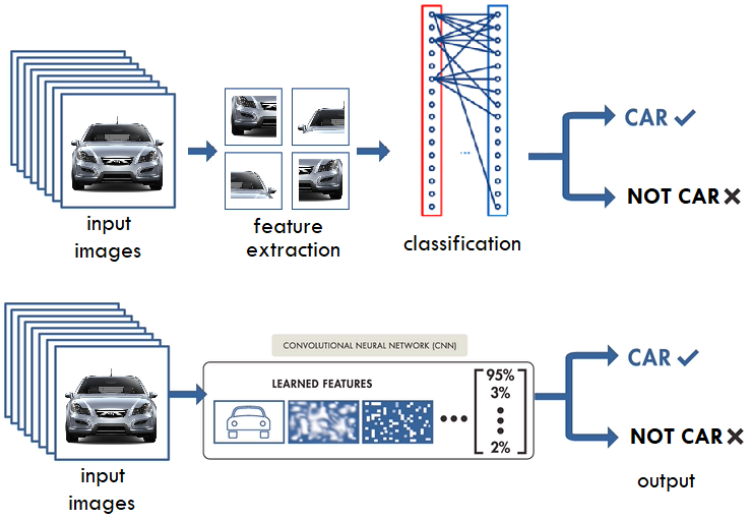

Figure 3.1. Difference between Machine Learning and Deep Learning (Mathworks, 1994)

Although deep learning is a new field of machine learning, it showed great development in a short time and became an important topic in artificial intelligence field. One of the reasons behind of this development is increasing dataset sizes. As technology evolves, hardware becomes more capable. Thanks to this, researchers can now work on huge datasets to get better results from their experiments. Another reason of success of deep learning is increasing memory size. After researchers performed experiment on very large datasets, another problem arose about how to process and store these huge amount of data. However, the obstacle has been overcome by the production of computers with superior features. Thanks to current CPUs and GPUs being faster and having larger memory, there happen to be enough resources to work on larger datasets. Last but not least, one of the reason behind success of deep learning is increasing accuracy, complexity and real world impact. After debut of deep learning, people did not use much is because larger datasets were needed to get better results. In order to extract every abstract features from hidden layers of network, large dataset should be given. Following the age of "Big Data", implementation of deep learning become much easier and effective. With the increasing digitization of daily life, computers have the power of taking more activities which result in increasing the number of computers connected together in network.

In simple neural network which has one hidden layer as in left side of the Figure 3.2, in order to solve reasonable problems, it is necessary for it to have exponentially increasing number of parameters. On the other hand, if multiple layers can be added as in right side of Figure 3.2, super linear scaling can be obtained. So as to solve much more complex problem, a few more hidden layers can be added. The ‘deep’ in deep learning simply refers to having

multiple ‘hidden’ layers; that is layers that aren’t the input, or the final output, but somewhere in between. In brief, deep learning is neural networks with multiple hidden layers.

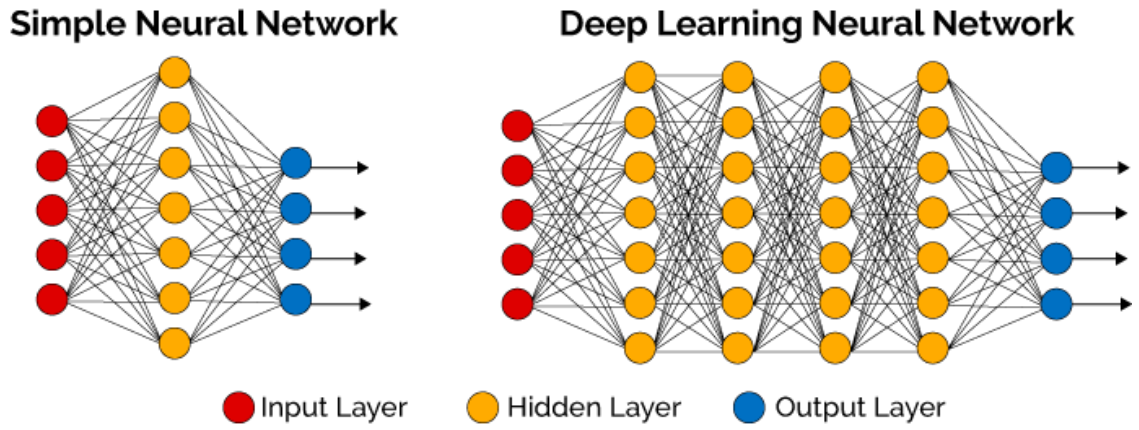

Figure 3.2. Difference between Simple Neural Network and Deep Learning Neural Network (Vazquez, 2017)

3.2.1 Convolutional Neural Network (ConvNet/CNN)

In neural networks, Convolutional Neural Network (CNN) is one of the main categories to work on problems such as image classification, image recognition. It is a Deep Learning algorithm that can take in an input image, process it and classify it under certain categories (Eg., sedan, hatchback categories for an vehicle image). In order to make good prediction, ConvNet aims at reducing the images into form that is easily processed without losing its important features. This is critical if an architecture is to be designed that it is required not only to be good at learning features but also is scalable to massive datasets. In deep learning, CNN algorithms train input data such as image by conveying them among convolutional layers to classify the input with probabilistic values between 0 and 1. Overview of the structure of CNN is explained below:

- Convolutional Layer is used to extract features by applying filters
- Non-Linearity Layer introduces non-linearity to the system
- Pooling(Downsampling) reduces the number of weights and control overfitting
- Flattening Layer prepares data for the fully connected layer
- Fully-Connected Layer is Standard Neural Network for classification

Using a very deep hierarchy of layers, ConvNet shows superior performance today. Thanks to the wide use of ConvNet, human-realizable tasks like voice recognition, object

detection, face recognition are now accomplished with deep learning algorithms programmed by researchers or IT companies (Jordan and Mitchell, 2015).

3.2.1.1 CNN Parameters

Neural network is an infinitely flexible function. In case of where appropriate inputs are given, even a simple neural network is able to symbolize a wide range of various functions. As long as enough parameters are added, any problem can be solved using neural network with a close accuracy. In this section some of the hyperparameters will be explained.

3.2.1.1.1. Stochastic Gradient Descent

In deep learning, there exists different optimization methods that enables us to find optimum value for nonlinear problems. Some of the widely used algorithms are Adam, Adamax, Stochastic Gradient Descent algorithms (Ruder, 2016). In this thesis, we use Stochastic Gradient Descent algorithm while training neural networks.

Gradient Descent is an algorithm that fits parameters of neural network to measure the quality a specific group of parameters depending on how well the final scores are in accordance to the ground truth labels in the training data. Suppose that we have given a set of parameters for training a network. Then gradient descent use these initial set of parameters and seeks out for a set of parameters that converges to local minima . It checks the current parameters and then estimates a slightly better set of parameters to predict how far it is from surface of the function by using current set of parameters.

As it is demonstrated in Figure 3.3, different initial set of parameters at starting point may result in different local minima. Thankfully, there is not only one value that good for neural networks, there might be many that are equally good. In other words, in deep learning even though solutions are not truly minimal they are generally accepted, as long as they are in accordance with significantly lower values of the cost function (Goodfellow, Bengio and Courville, 2016). Detailed information about stochastic gradient descent can be found in (Loshchilov and Hutter, 2017).

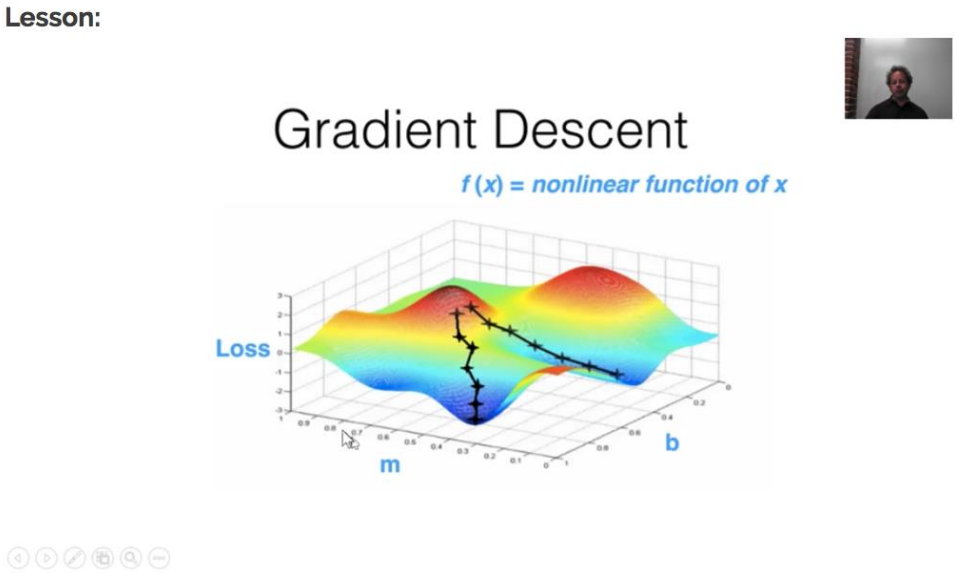

Figure 3.3. Initial set of parameters move towards to local minima (Howard and Thomas, 2017)

3.2.1.1.2. Learning Rate

The learning rate is one of the most significant hyper-parameters that adjust the weights of network while taken loss function gradients into account. It decides how much to change model in response to the error estimated when weights are updated. Suppose that there is a quadratic function and we would try to find a minimum of it using stochastic gradient descent technique that explained in 3.2.1.1.1. It starts by computing the value of the function and the slope at a point. Then, it goes down bit by bit of hill and make computation again. When it is seen that the value of is not decreasing anymore, it means that the value has reached its minimum. The difference between two consequent points which are pursuing slope through the downhill is called learning rate.

Learning rate decides how fast to reach local minima of a function. The biggest challenge is choosing a learning rate value to train a model. Because, by choosing a small value for learning rate, it would take so much time to get to the local minima, and choosing a large value might result in exceeding the minimum value trying to be reached. These are demonstrated in Figure 3.4.

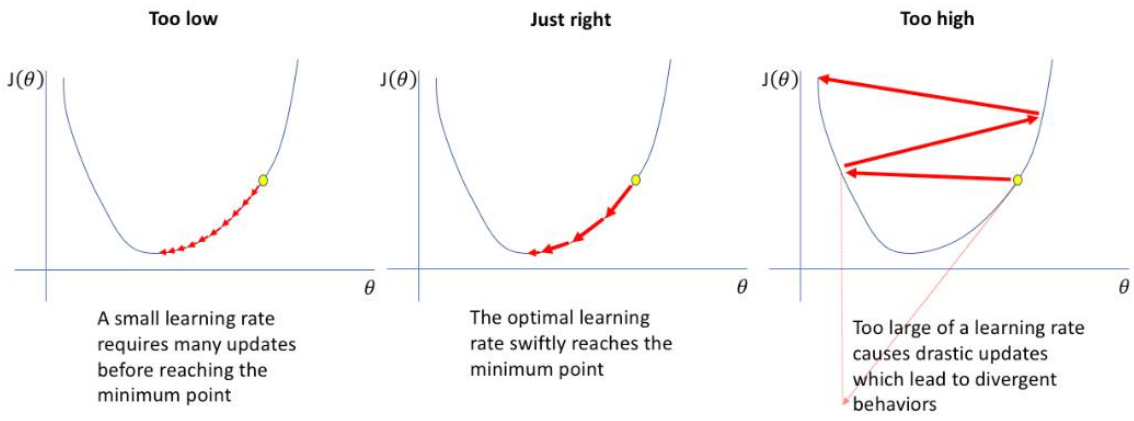

Figure 3.4. Visualization of what happens choosing three different learning rate (Jordan, 2018)

In order to eradicate the need to determine sensible values, one of the most robust techniques was proposed by (Smith, 2017). According L. N. Smith, if training is done using cyclical learning rates the accuracy of classification is enhanced and it does not reqire any tuning.

If we divide the logic behind of Smith's writing (2017) into steps, the first step is to start with a small learning rate and calculate the loss. Second is to increase the learning rate step by step and each time, calculate the loss. When the loss suddenly increase again, the increasing of learning rate should be stopped. Where the loss is still clearly improving, the highest learning could be chosen. Lastly, these steps are repeated in a cycle. Different implementation of cyclical learning rates can also be reviewed in Smith's writing (2017).

3.2.1.1.3. Number of Epochs

In deep learning, an epoch is a hyperparameter which refers to one cycle through full training data and it is determined before training the model. One epoch means passing through the neural network both forward and backward all at one. But one epoch is too big to feed to the network at once. Thats why, it is divided into several smaller batches such that more than one epoch is used to pass through the full dataset in multiple times. After each pass, it gives a chance to the network to evaluate success value and regulate model parameters taking previous data into account. Success value is generally lower in first epoch and as success rate will also increase as the number of epochs get bigger. This does not mean that the biggest value of number of epoch is best input for a network because after a certain step the success rate will no longer get better. In other words, increasing the value

of number of epochs would not provide better efficiency for each setting. On the other hand, decreasing the value of number of epoch hyperparameters would be useful in case of a model that require a long training period.

3.2.1.1.4. Batch size

A ConvNet doesn't process its inputs all at once, so as to increase performance, it processes the data in batches. The batch size is the number of training samples will be used in training in order to make an update to the model parameters. In an ideal way, all the training samples would be used to calculate the gradients for every single update, however that is not efficient. In simple terms batch size tries to simplify the process of updating the parameters.

One of the aspects of getting training right is adjusting the batch size. If chosen batch size is too small, then there will be a lot of variance within a batch, and the training loss would increase a lot. On the other hand, if it's too large, GPU would use so much memory to maintain it, or training would progress too slowly to catch a glimpse of if it's the optimization is diverging early on. Therefore, when choosing this value, neither too small nor too large should be selected.

3.2.1.1.6. Dataset Augmentation

As the more training is performed for the model, it will be more successful. When the number of data given for training is high enough it gets better fit for the model while preventing from overfitting. For the purpose of increasing size of the data, synthetic images can be produced so that prepared dataset will be augmented. There are several ways to create synthetic images such as rotating the image, flipping of the picture, random translations, changing brightness, cropping the picture. After applying these transformations to data, with respect to deep learning, there will be different versions of the data in a dataset. Besides, cross validation techniques were implemented by splitting data into five different groups in order to evaluate models.

Data augmentation technique is preferably used with other regularization method. It only increase the performance at a certain rate because, although data is synthetically generated, it does not replace completely different data. Detailed Information can be found in (Kukacka, Golgov and Cremers, 2017).

3.2.2 CNN Architectures

Neural networks may vary in terms of sizes and shapes, as a result they can be different depending on how they are configured. Thankfully, there is no need to reinvent the wheel for every situation. In order to create a classifier, one of the architectures that have proven to be successful can be reused. In this thesis instead of building a model from the scratch, transfer learning (Section 3.3) approach is applied. Therefore, to solve vehicle type classification problem, experiments performed on data sets reuses four different architecture which are listed below (Simonyan and Zisserman, 2015, He et al.,2016):

- resnet34
- resnet50
- resnext50
- vgg16

3.3. Transfer Learning

Transfer learning is a technique of machine learning in which an output of the algorithm can be reused as a starting point of another task even if it is designed for a specific work. In this method, knowledge gained while solving a problem is utilized for applying it to another but related problem. For example, knowledge gained while learning to drive a motorcycle can be used to some extent to when learning to drive a car. There is no need to acquire knowledge of everything from scratch when trying to learn something new.

In traditional learning, there is no knowledge that can be conveyed from a model to another and no knowledge is preserved. While traditional learning consist of specific tasks and training separate isolated models on different datasets, on the other hand, transfer learning comes from the idea of overcoming the isolated learning paradigm, and in order to train newer models it leverages knowledge from previously trained model. Thats why, pre-trained models make great contributions to solving new problems.

The most important advantage of transfer learning is saving time and resources needed thanks to leveraging a previously trained model. There is not much power is required for computing because weights are transferred from previous model. Transfer learning even get over with problems like having less data for newer tasks because there is no need for an extremely large training dataset when there is similarity between datasets for the original and new tasks. So, it reduces the need to re-collecting training data and time consumption drops to hours or even to minutes. Furthermore, Transfer Learning is easy to use in terms of

implementation on layers. Because instead of training all layers, some tasks can be defined by training only the last layers.

Using transfer learning approach, an application can be easily created to process a task that involves adjusting pretrained model. Firstly, it is started with choosing an existing network like GoogLeNet (Szegedy et al., 2015), and new data is fed in that architecture. After making some small adjustments to architecture, an application that can perform a new task can be produced. For example, this application can be used to categorize cats or dogs in an image set that hosts many unknown objects. Detailed information can be seen in (Pan and Yang, 2010). Transfer Learning is useful for creating applications that achieve many tasks including image recognition and classification, robotic process automation.

3.4. Ensemble Learning

Ensemble learning is a machine learning approach that merges several base models so that it produces a model that advances optimal prediction. In traditional ensembling, different models are combined and they predict on the same input. After that, averaging methods is used to firmly decide on the final prediction of the ensemble. It can be simple voting algorithm, calculating average, or even another model that predicts values and labels using model parameters. While some ensembling methods use different models to produce a optimal predictive model, on the other hand, models in the same architecture can also be combined and advance good results. One of the ensembling methods that makes use of this approach is Stochastic Weight Averaging.

3.4.1 Stochastic Weight Averaging (SWA)

Stochastic Weight Averaging technique is proposed in (Izmailov et al., 2018) and uses a novel ensembling in the weights space. In this approach, in order to make predictions it uses a model that takes advantage of an ensemble which is created by merging weights of the same network at different stages of training with weights itself.

Working algorithm of swa starts by making a copy of the model that is training such that it will be used for keeping track of the averaged weights. The weights of the copied model are updated after the each epoch of training using introduced averaging formula in (Izmailov et al., 2018). As a result, thanks working on one model at a time, SWA speeds up predictions.

3.5. Fastai

The fast.ai is a small library built on top of Python which is heavily used in research and it helps to build neural networks. It implements Supervised Learning approach such that a dataset containing labeled images of vehicle types can be given as an input for training data (Section 3.1.1).

As datasets are generally prepared, structures for two datasets in this thesis includes the valid, train and test folders containing images to train and for validation and to test close to real world scenarios. While valid and train folders have subfolders for each classes, test folder is directly contains test images. There is also "model" folder which stores the versions of models saved.

The training methodology of fastai is summarized into three step. First one is creating data variable to configure data for training. Creating data object in fastai it is required to specify two parameters. They are path of stored data, and selected architecture to apply its data transformations. These transformations may include resizing, normalizing or standardizing. Thanks to the transformations input data can be resized, rescales to values between 0 and 1. Second step is creating a learner object so as to train neural network on prepared dataset using selected pre-trained models. Fastai does not start training from zero but it uses a network architecture exist before in order to gain knowledge of pretrained model on a massive dataset like ImageNet so that it can transfer the knowledge to trained model. It means that fastai library utilize from Transfer Learning (Section 3.5). Third step is fitting these data and learner objects into together. Once data and learner object are determined actual training may start with specifying a learning rate and the number of epochs (eg. learn.fit(0.02, 3)). The fit method takes two parameters. First one is learning_rate which is controlled for updating the weights of the model all the while training (Section 3.2.1.1.2). Providing helper classes, the fastai library proposes a level of abstraction for application. One is used to determine learning_rate for fit method using gradient descent approach described in Section 3.2.1.1.1. Second parameter to fit the model into data is number of epoch (Section. 3.2.1.1.3).

In this thesis fast.ai library is used in order to build classifiers that categorizes an image it has never seen as one of the vehicle types like 'sedan', 'suv'. Trained network will be able to distinguish among vehicle types based on the labelled images provided. Detailed information about output of classifying by fastai and evaluating them can be found in later chapters.

3.6. Technical Environment

In order to perform experiments on data sets prepared, Fastai library founded on Pytorch is configured on a computer which has a GPU powered by 5.0 computing power. Codes written in Python have been visualized with jupyter notebook. The environment in which experiments are performed has characteristics described below:

Table 3.1. The differences between BIT Vehicle Dataset and Stanford Car Dataset

Name	Value
Processor	Intel(R) Core(TM) i7-4510U CPU @ 2.0 GHz 2.60 GHz
Memory	8.00 GB DDR3
System	64-bit Operation System, x64-based processor
GPU	NVIDIA GeForce 840M 4GB shared

CHAPTER 4

VEHICLE TYPE CLASSIFICATION

In this section, we introduce the problem of vehicle type classification and present our approach to deal with it. Then we give information about dataset and details of training and evaluation for vehicle type classification.

4.1. Approach

We approach the vehicle type classification problem from two different perspective. One uses a single - level classifier to categorize new instances and the other one uses a multi - level classifier.

Hierarchies are frequently used for the management of objects. Given a hierarchy of classes, these two approaches are also used to classify new instances labelling with a class name. Single - level classifier represents a traditional flat hierarchy which can be assumed as a tree with 1 unit - height (level-1). There is a node which acts as a classifier for an image given and gives probabilistic values for each class. In traditional flat hierarchies, these classes are independent of each other. In contrast to single - level classifier, there are some connections established in multi - level classifier in order to create hierarchical dependencies between nodes. Furthermore, the hierarchy is a cascade tree with height greater than one and the classification nodes are always the leaves of the hierarchy. It estimates the probability of each root-to-leaf path through cascade classification trees. It is also assumed that each image belongs to one class, which means that single - label classification is applied.

In this thesis, we handle vehicle type classification problem performing experiments using two different approaches which are single – level classifier training using transfer learning technique and multi – level classifier training using hierarchical cascade trees. According to experiment results provided by these two approaches, one can achieve better results with multi - level classification using three different computational methods compared to single - level classification.

In the next section, information on data sets that are used to train classifiers are introduced.

4.2 Dataset Preparation and Training

4.2.1. Information on Datasets

Two different data sets were prepared to train deep neural networks for vehicle type classification. They are BIT Vehicle Dataset (Dong et al., 2015) and a reorganized vehicle type dataset manually prepared from Stanford Car Dataset (Krause et al., 2013). There are six types of vehicle in BIT Vehicle image data set and nine types of vehicle in Stanford Car Dataset. Next, details of each data set are explained.

4.2.1.1. BIT Vehicle Dataset

As described in (Dong et al., 2015), BIT-Vehicle dataset is consist of 9,850 vehicle images collected from two different cameras located in two different place at different time intervals. Image size of photographs captured are 1600x1200 and 1920x1080. They are all captured from high viewpoint with two different angle that's why the top or bottom parts of some vehicles are not seen in the images. Some images are noisy because of motion, delay and illumination changes. Also there are no images of rear of a vehicle or side of a vehicle clearly. Total number of images used in this thesis are 9.652. Sample images of BIT Image Dataset for each classes are demonstrated in Figure 4.1.

Table 4.1. Number of images for folder arrangement per vehicle type in BIT Vehicle Dataset

Label	Training	Validation	Total
BUS	440	111	551
MICROBUS	670	168	838
MINIVAN	370	92	462
SEDAN	4502	1125	5627
SUV	1085	271	1356
TRUCK	654	164	818
TOTAL	7721	1931	9652

All vehicles in the dataset are divided into six categories: Bus, Microbus, Minivan, Sedan, SUV, and Truck. 80% of each class is allocated for training, the rest is placed in the validation folder. The number of vehicles per vehicle type are respectively given in Table 4.1.

Figure 4.1. Image examples from BIT Dataset for each classes

4.2.1.2. Stanford Car Dataset

This dataset is prepared from Stanford Car Dataset (Krause et al., 2013) and consists of 888 manually chosen vehicle images. It includes pictures with different lighting values at different points of view from various places. There are images from different viewpoint of vehicles such as back, front, sides, diagonal. Some images demonstrate interior view like studio while others demonstrate outdoor view like seaside, bridge. Because of location of the sun lighting changes occur in some outdoor images. Also a few surprising elements can

be found in images such as reflection in water, people or trees. Sample images of Stanford Dataset for each classes are demonstrated in Figure 4.2.

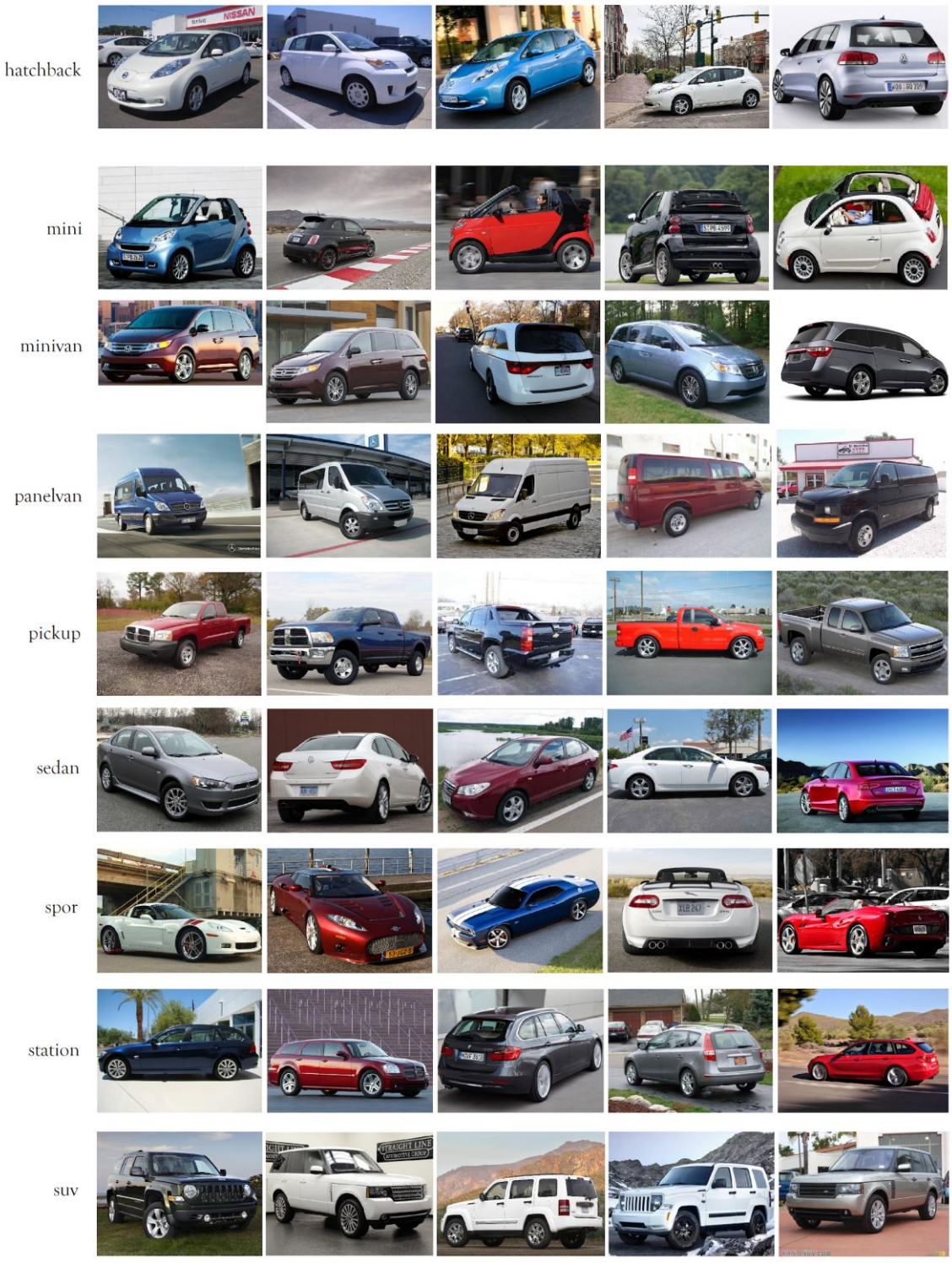

Figure 4.2. Image examples from Stanford Dataset for each classes

All vehicles in the dataset are divided into nine categories: Hatchback, Mini, Minivan, Panelvan, Pickup, Sedan, Sport, Station and SUV. These categories are

determined in accordance to the e - commerce websites such as sahibinden.com and arabam.com. 80% of each class is allocated for training, the rest is in the validation folder. The number of vehicle images per vehicle type class in Stanford Dataset are respectively given in Table 4.2.

Table 4.2. Number of images for folder arrangement per vehicle type in Stanford Car Dataset

Label	**Training**	**Validation**	**Total**
HATCHBACK	80	20	100
MINI	71	17	88
MINIVAN	80	20	100
PANELVAN	80	20	100
PICKUP	80	20	100
SEDAN	80	20	100
SPORT	80	20	100
STATION	80	20	100
SUV	80	20	100
TOTAL	611	177	888

The main differences of BIT and Stanford datasets are summed up in Table 4.3. The distinguishing features are specified as the number of classes, point of view as perspective, environmental elemets, and the number of images in total for each datasets.

Table 4.3. The differences between BIT Vehicle Dataset and Stanford Car Dataset

Name	**BIT**	**Stanford**
# classes	6	9
Perspective	Only front	Front, back, sides
Environment	Highway	Studio, seaside, bridge...
# images in total	9652	888

4.2.2. Training

In this study, firstly the single - level classifier approach is studied to train classifiers on introduced data sets in Section 4.2.1. Then, the use of a multi level classifier hierarchy is explained for predicting labels by applying different computing methods.

4.2.2.1. Single - Level Classifier Training with Transfer Learning

First, we studied the training of the single - level classifier on the BIT Dataset. We started to this training process by simply training our first model. So, the first step of training a model is creating data object. The data object contains the training and validation data. Training data is on what the model is trained upon while validation data is used to check how well our model performs on images it has not seen before. The data object carries information about not only labelled dataset but also image size, which decides the size of input image for training. In the first experiment we defined the class hierarchy of BIT Vehicle Dataset as a single - level hierarchy in Figure 4.3.

Next, we created a learner object for hosting model to train and we transferred the information data object carries to it. We make use of transfer learning methodology for training, that's why we introduced ResNet34 network to the single - level classifier learner object. In order to start training we determined learning rate for our training and number of epochs such that we will be able to manage how slowly or how quickly update the weights. For the first experiment of single - level classifier training, parameters are listed in Table 4.4.

Table 4.4. Parameters used for first experiment of single level classifier training

Name	**Value**
Dataset	BIT Vehicle Dataset
Image Size	32 x 32
Architecture	ResNet34
Learning Rate	1e-2
# Epoch	3

As a result, using given parameters with the single - level classifier represented in Figure 4.3, we train a classifier that predicts type of vehicle for labelling new instances as Bus, Microbus, Minivan, Sedan, Suv and Truck.

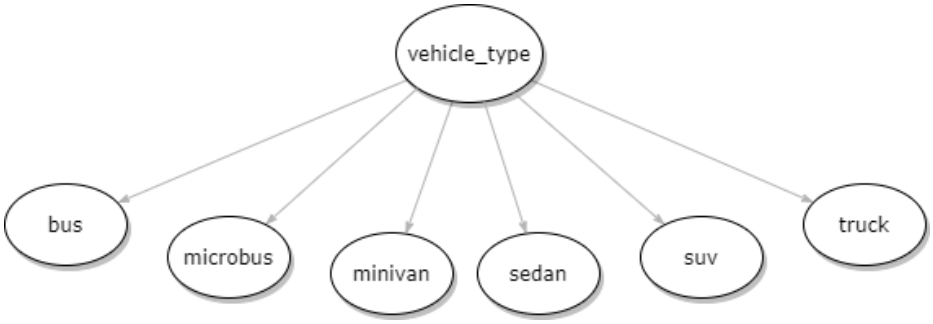

Figure 4.3. Tree representation of single-level classifier for BIT Vehicle Dataset

Secondly, we used augmentation methodology to increase the number of data from different perspectives, that's why restricted scaling and rotation transformations have been applied to add diversity to images during training. In this way, each time the data is used during the training, it is provided to perform training with various images. As examples can be seen in the Figure 4.4, six different synthetic pictures are created from one picture after transformations applied.

Figure 4.4. Examples of transformations applied to images from BIT Dataset (top) and Stanford Dataset (bottom)

Last but not least, we first opened and trained last layer of the network, then we unfreezed the remaining layers and released the lower convolutional layers so that the whole network can be trained as a one. Working on all layers, we assume that earlier layers need less fine-tuning for new datasets because they have more general purpose features. For this reason we used differential learning rates technique introduced in Fastai. In this technique,

learning rate is set as an array such that first learning rate is set for fully connected layers and as we go deeper in network layers, learning rate get smaller exponentially (Section 3.2.1.1).

In the rest of the thesis, the explanation of training above is referred as single training process because this process will be continued to be performed with different set of parameters. Three main categories were taken into consideration while forming set of parameters for conducting experiments. These categories are; two different data sets, architectures for training models and size of images. Details of the parameters can be found in Table 4.5.

Table 4.5. Categories of parameters used in experiments

Name	**Parameter_ 1**	**Parameter_ 2**	**Parameter _3**	**Parameter _4**	**Parameter _5**
Data sets	BIT Vehicle Dataset	Stanford Car Dataset			
Architecture	ResNet34	ResNet50	ResNext50	Vgg16	
Image Size	32 x 32	64 x 64	128 x 128	150 x 150	224 x 224

While working with BIT Vehicle Dataset and Stanford Car Dataset, deep neural networks trained in various architectures and ImageNet are used. The architectures selected to work on are Resnet34, Resnet50, Resnext50 and Vgg16. Fastai library (Howard and Rachel, 2017) is used to train classifiers utilizing these pre-trained models. The latest layers of these architectures contain a fully-connected neural network structure, while the other layers are convoluted blocks. Therefore, different image sizes for different application areas can be fed into these architectures and a fully-connected final layer can be trained for a different number of classes.

Aiming to interpret in a way that utilize from different viewpoints, in this study, 32, 64, 128, 150 and 224 dimensions have been studied for the selected architectures. In experiments, both the last layer was trained keeping other convolutional layers fixed and also all the layers were trained in given two datasets. The graphics card memory used for all stages is approximately 4 GB. Other technical information about experiment environment is given in Section 3.6.

As one can see from parameter categories detailed in Table 4.5, two different data sets can be given as parameter to training process. So experiments are also performed for

Stanford Dataset keeping other parameters fixed. Tree representation of single - level classifier training of Stanford Car Dataset can be found in Figure 4.5. In addition to these experiments, Ensemble Learning (Section 3.4) technique is used in order to improve accuracy of experiments performed on Stanford Dataset by averaging weights at the end of each epoch. So as to apply ensemble learning additional 14 experiments was conducted. In order to get best accuracy results of ensembling technique, the model trained with biggest image size on the same architecture is chosen as a second model for each architecture. For example, so as to implement ensemble learning on ResNet34 architecture, the model trained with the same architecture and has 224 image size is used as a second model for every other trained model with different image size.

4.2.2.2. Multi - Level Classifiers and Their Training

While the single - level classifier applies flat hierarchy that ignores the hierarchy, on the other hand multi - level classifier hierarchy traverse the hierarchy from the root to the predicted leaf. In this section, we study the cases where multi - level classifier hierarchy is constructed to estimate class probabilities.

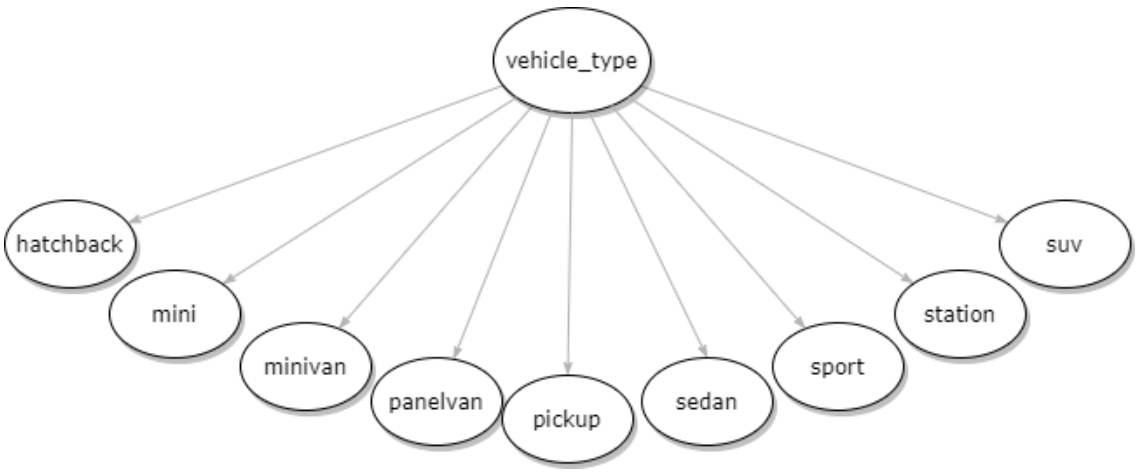

Figure 4.5: Tree representation of single - level classifier for Stanford Dataset

The aim is to classify new images through the hierarchy built which represents is-a relations. Two different multi - level classifier hierarchies are manually constructed for this purpose. The tree represents multi - level classifier hierarchy with tree height is 2 (level-2) can be found in Figure 4.6 and tree representation of multi - level classifier hierarchy with tree height is 3 (level-3) can be seen in Figure 4.7 respectively. In hierarchical trees, each node usually apply one classification to given test images. The problem can be divided into

smaller problems using hierarchical cascade classification such that fewer features are required to train each classifier compared to using overall dataset.

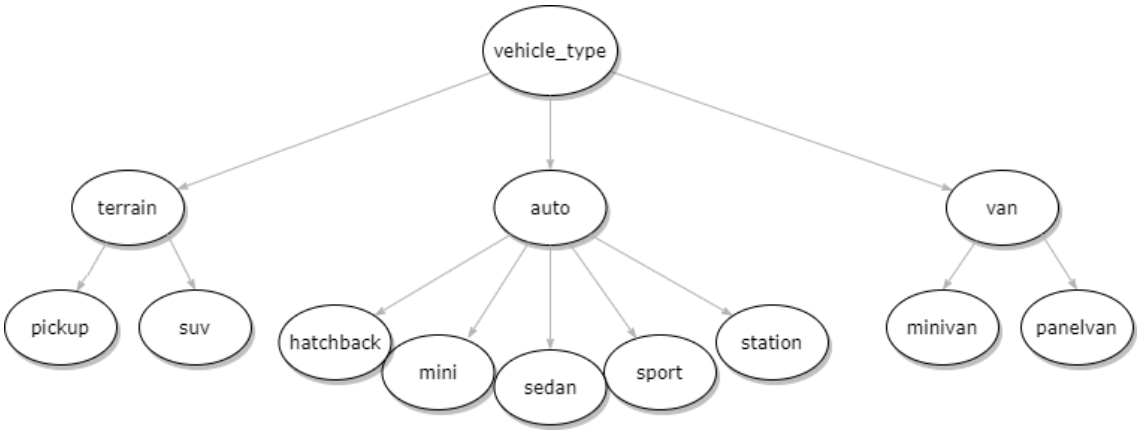

Figure 4.6: Level-2 tree hierarchy example

As already mentioned, for multi - level classifier hierarchy approach one classifier is trained for each of node of the hierarchy. For example, using the hierarchy of Figure 4.6, total of four classifiers are trained for each of the nodes vehicle_type, terrain, automobile and van. The classifier of any node is trained using the samples of that node's leaf descendants. That is, the vehicle_type node is trained using the train and validation image samples of the terrain, auto and van nodes.

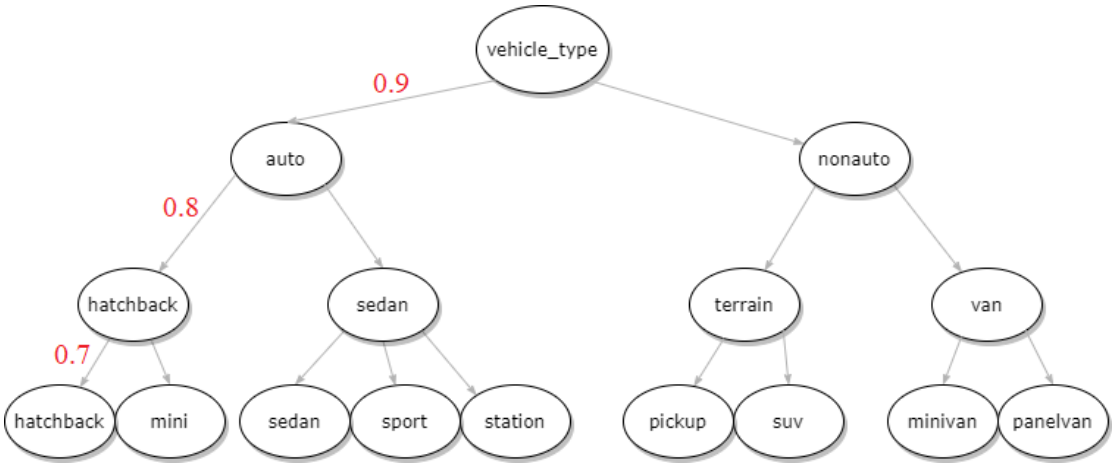

Figure 4.7: Level-3 tree hierarchy example

Similar situation also occurs in Figure 4.7. For the binary classifier of node auto_nonauto, all instances belonging to hatchback and sedan would be positive examples whereas all the instances belonging to Terrain and Van would be negative. Similarly, for the binary classifier of node auto, all hatchback and mini images belonging to hatchback class

would be positive instances while sedan, spor and station images belongs to sedan class would be negative. Comparing to flat hierarchy, these classifiers require fewer resources to be trained. Also they give more accurate results thanks to distinguishing between fewer classes compared to flat ones. For example, terrain node in Figure 4.6 has 2 leaves such that this classifier distinguishes between two classes, on the other hand the classifier of vehicle_type node in Figure 4.5 needs to distinguish between nine classes. For the flat hierarchy, if a node have 100 leaves then the classifier would need to separate one class from 99 others. So, hierarchical cascading may be a preferable option for large databases, as images only have to be separated between sibling classes.

The main disadvantage of multi - level classifier hierarchy is that any mistake due to wrong prediction is carried over from root to leaves. The severity of this mistake may be greater if it occurs closer to the root, in the same way smaller if it is farther away. Suppose that there is an instance belonging to sedan class and it is estimated to be non automobile by the binary classifier of auto_nonauto node in Figure 4.7. It means that this image classified wrongly at the beginning of hierarchical cascading tree even without taking into consideration the classifiers of sedan. Therefore, it lost a chance of getting high probability of its own class. Given this disadvantage, three different computational method are used to calculate the probability for each class at the end of hierarchical trees. Probabilistic cascading, traditional arithmetic mean and weighted average computation methods are used to compute estimated probability of a given instance. These three methods compute the probability of each root-to-leaf path for a testing image then the most probable path is determined.

If looking at the computation methods, **Probabilistic Cascading** (Kosmopoulos, Paliouras and Androutsopoulos, 2015), referred as **PC**, is one of the method studied to compute probabilities for testing images. It implements like standard cascading such that probabilities through root-to-leaf path are multiplied to find a class for a given image d (4.1). In order to find probabilities of each class for image d, a leaf C, a set S of all the ancestors of C, are needed to taken into computation. So its general formula is:

$$P(C|d) = \prod_{i=1}^{|S|} P(S_i|Ancestor(S_i), d) \tag{4.1}$$

As an example, determining the probable path of an image d belonging to Hatchback class for level-3 hierarchical tree in Figure 4.7 would be explained below for each computational methods. Assume that probabilities are determined from classifiers of each ancestor node of Hatchback class is given below:

P(Auto|Root, x) = 0.9 *Root refers to vehicle_type node

P(Hatchback|Auto, x) = 0.8

P(Hatchback, x) = 0.7

With calculated probabilities of classifiers of each ancestor nodes taking formula (4.1) into calculations, probability of an image d being Hatchback class is computed as 0.504.

Second computation method to calculate probabilities of test images is **Traditional Arithmetic Mean**, referred as **TAM**. In this method, probabilities calculated from classified from root to leaf nodes are added together and a total value is obtained. To compute the final probability, this total value is divided by the number of probabilities which are added to sum (4.2). It was assumed that initial probabilities given in above 0.9, 0.8 and 0.7. Using (4.2), the final probability of Hatchback class in level-3 hierarchical tree would be 0.8.

$$\bar{x} = \frac{\sum_{i=1}^{n} P_i}{n} \qquad \begin{array}{l} P_i : \text{probability of nodes} \\ n : \text{level of hierarchy} \\ \bar{x} : \text{arithmetic mean} \end{array} \qquad (4.2)$$

Last but not least, the third computation method to calculate probabilities of testing images is **Weighted Average Computation**, referred as **WAC**. In this method, probabilities produced from classifiers are multiplied by certain numbers that are weights of probabilities and added to sum. To compute the final probability, this total value is divided by the sum of weights (4.3). To compute the final probability, this total value is divided by the sum of weights (4.3). We assigned the weights according to the level that classifier resides. The lowest weight was assigned to classifier which is on lowest level, and weights were increased as the level gets bigger. In this case, when weights are set as 0.1 for vehicle_type classifier, 0.3 for auto classifier and 0.6 for hatchback classifier regards to initial probabilities given as 0.9, 0.8 and 0.7, the final probability would be 0.75.

$$\overline{x} = \frac{\sum_{i=1}^{n} P_i.w_i}{\sum_{i=1}^{n} w_i} \tag{4.3}$$

When computing final probabilities, for each image and an image's each class different probability is calculated. After calculation is done according to computing methods, results are saved in a file and presented in Chapter 5.

To conclude, in this chapter, we have introduced the datasets and classification techniques used in the thesis. In addition, we have explained the approaches we applied to vehicle classification problem along with the solutions we proposed. In the next chapter, results of the all experiments performed in this chapter are presented.

CHAPTER 5

EXPERIMENTAL RESULTS

In this section, we present experiment results of both single level classifier and multi level classifier approaches. We compare the results of experiments performed on BIT Dataset and Stanford Dataset when single level classifier applied. Also, multi level classifier and single level classifier approaches are compared by presenting their experiment results performed on Stanford Dataset.

5.1. Single Level Classifier Experiment Results

During the single level classifier training, the last layers were firstly trained. Then the lower convolution layers were also released and the whole neural network was trained. However, since the number of data was low, the lower layers were trained with a lower learning speed.

5.1.1. BIT Vehicle Dataset

In first experiments, models are trained on BIT Vehicle Type Dataset giving different architectures to the environment. Single level classifier training approaches is used to train on the dataset using ResNet34 architecture with different image sizes.The results for both last layer training and training all layers are recorded in Table 5.1 including loss of training, loss of validation, accuracy values and time spent on training.

Looking at the results of the last layer training for each image size, it is observed that accuracy gradually increase as image size get bigger. While training last layer with image size 32x32 gives 0.44786 accuracy value, on the other hand training with images which have 224x224 image size result in 0.65424 accuracy. It means that increasing image size from 32 to 224 can give better accuracy values by 46%. Besides, it can be observed that training loss and validation loss are inversely proportional to image size. To sum up, increasing image size decreases the loss of training and validation and enables models make more accurate predictions but it also requires more time to spent for training.

Table 5.1: Detailed result of training models on BIT Vehicle Dataset with different image sizes using ResNet34

Architecture	Image size	Trained layers	Training loss	Validation loss	Accuracy (%)	Time
ResNet34	32	Last Layer All Layers	0.63913 0.3936	1.88168 1.92105	0.44786 0.48298	11 min 05s 26 min 59s
	64	Last Layer All Layers	0.49206 0.23368	1.72825 1.61256	0.52788 0.57277	11 min 17s 27 min 44s
	128	Last Layer All Layers	0.28462 **0.02740**	1.62993 1.21851	0.61694 0.74366	12 min 17s 36 min 10s
	224	Last Layer All Layers	0.29082 0.08114	1.36875 **1.01714**	0.65424 **0.77009**	15 min 33s **1h 20min 36s**

When comparing last layer experiment results to all layers', time spent on training last layer with 32 image size is ~11 minutes and gives 0.44786 accuracy, and time spent on training all layers with the same image size takes ~27 minutes and gives 0.48298 accuracy. The time increase is by 146% while the accuracy increase is by 8%. But when we increase image size from 32 to 224 we observed that accuracy is increased from 0.48298 to 0.77009 by 60% on the other hand time spent is increased from 27 minutes to 81 minutes by 200%. This feature can be useful for a task requires high accuracy while time can be ignored.

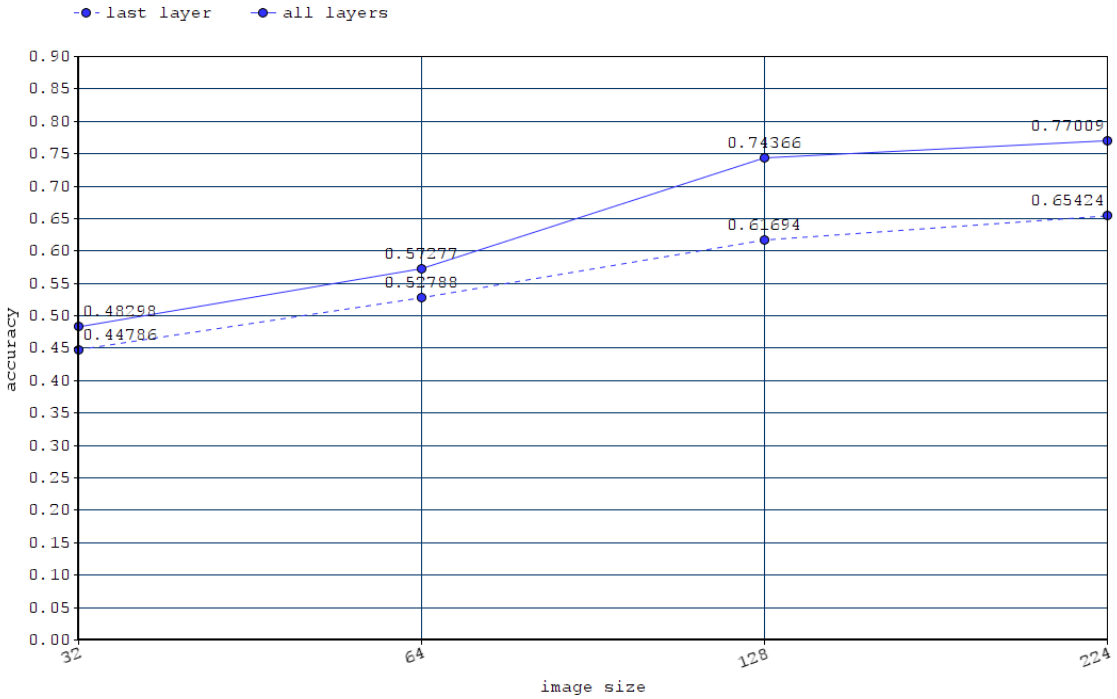

Figure 5.1: Graph of single level training accuracy values with respect to different image size of BIT Dataset images

Subsequent experiments were carried out by training different architectures. In addition to ResNet34 architecture, ResNet50, ResNext50 and Vgg16 architectures were used to perform experiments of vehicle type classification. Single level hierarchy training results of both last layer and all layers with 128 image size on BIT Vehicle Dataset using different architectures are represented in Table 5.2. Because of memory issues, with 224 image size experiments could not be performed on ResNet50 and other architectures. Again memory was insufficient for experimenting on ResNext50 with 150 image size. That's why the biggest image size which is 128 was able to be experimented in all of four architectures. Looking at accuracy values of last layer training results for each architecture, the best scenario occurs when using ResNet34 architecture. Compared to other architectures, it has high accuracy value and it spent least time for training. Furthermore, taking all layers experiments' results into consideration, ResNet34 has highest accuracy and best time usage again. The best values in terms of accuracy with respect to time usage are taken in ResNet34 architecture with the lowest loss of a training value and highest accuracy. Compared to training last layer, after opening all layers and training the network as a whole by ResNet34, it is observed that accuracy is increased by 21%, while training loss decrease by 90%. One can interpret from looking at Figure 5.2 that training loss gradually increases on architectures in order ResNet 34, ResNet50, ResNet50 and Vgg16. With the least time spent in training, the worst accuracy value was taken in ResNext50 architecture. Most time has been spent in Vgg16 architecture and has received relatively good accuracy.

Table 5.2: Training result of all layers with 128 image size of BIT Dataset images on different architecture models

Architecture	**Image size**	**Trained layers**	**Training loss**	**Validation loss**	**Accuracy (%)**	**Time**
ResNet34	128	Last Layer All Layers	**0.28462** **0.02740**	1.62993 1.21851	**0.61694** **0.74366**	12 min 17s 36 min 10s
ResNet50	128	Last Layer All Layers	0.33033 0.08073	1.50944 **1.09149**	0.61007 0.73316	12 min 20s 48 min
ResNext50	128	Last Layer All Layers	0.32642 0.18623	**1.41776** 1.49338	0.61405 0.64446	13 min 24s 30 min 05s
Vgg16	128	Last Layer All Layers	0.3853 0.96889	1.46764 1.22042	0.59232 0.71325	**18 min 36s** **1h 31min 20s**

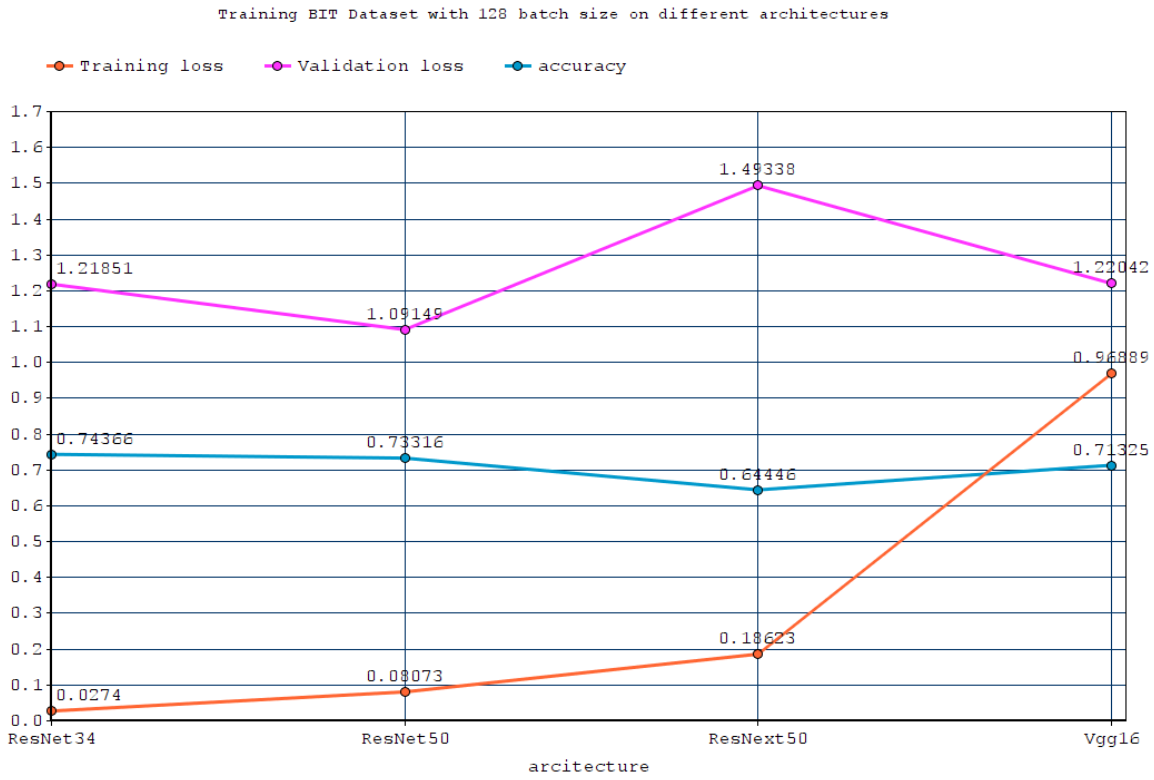

Figure 5.2: Graph of training all layers with 128 image size of BIT Dataset images on different architecture models

On BIT Vehicle Dataset, experiments are performed with ResNet34, ResNet50, ResNext50 and Vgg16. For each of these architectures, trainings are studied by different image sizes. 32x32, 64x64, 128x128 images size are used for each architecture. Additionally, 150x150 is used for ResNet50 and 224x224 is used for ResNet34. Giving these parameters as input to experiment environment, we got fourteen variation of single level classifier training. For each of fourteen variation, we trained fourteen models for 3 epoch with cyclical learning rate. Including training loss, validation loss and accuracy values the detailed results of all experiments performed on BIT Dataset are further detailed in Table A.1 in Appendix.

5.1.2. Stanford Dataset

In this section, single level hierarchy training approach is used to train models in order for classification of vehicles into nine vehicle type categories of the dataset. Models are trained on Stanford Car Dataset by differentiating parameters given to experiment environment such as image size and architectures. For first experiments ResNet34 architecture is used with different image sizes. Experiment results for both last layer training

and training all layers are recorded in Table 5.3 including loss of training, loss of validation, accuracy values and time spent on training.

Table 5.3: Detailed results of training models on BIT Vehicle Dataset with different image sizes using ResNet34

Architecture	Image Size	Trained layers	Training loss	Validation loss	Accuracy (%)	Time
ResNet34	32	Last Layer All Layers	2.20407 2.08773	2.08856 2.04804	0.28488 0.29349	18s 1 min 40
	64	Last Layer All Layers	2.067591 1.88601	1.90531 2.01203	0.30233 0.32651	26s 2 min 05s
	128	Last Layer All Layers	1.55797 1.27692	1.33347 1.33658	0.51744 0.54753	37s 3 min 19s
	224	Last Layer All Layers	**1.32541** **0.83031**	**1.09792** **1.05054**	**0.55814** **0.65116**	**1 min 25s** **8 min 20s**

If we look at accuracy column of Table 5.3 for 32x32 image size, it can be seen that the accuracy increases when all layers are opened as it is in BIT Vehicle Dataset experiments. But this increase is relatively small comparing to increase between trained layers in BIT Vehicle Dataset. On the other hand, comparing accuracy values of different image sizes, it become clear that accuracy increases as image size get bigger. While training last layer with image size 32x32 gives 0.28488 accuracy value, on the other hand training with images which have 224x224 image size result in 0.55814 accuracy. So, accuracy is increased by 96%. Looking at all layers training accuracy values, it can be interpreted that the percentage through accuracy values of bigger image sizes is even more increased. While training all layers with image size 32x32 gives 0.29349 accuracy value, on the other hand training with images which have 224x224 image size result in 0.65116 accuracy. The result is accuracy can be increased by 122% through increasing image size from 32 to 224.

According to line graph representation of accuracy values versus image size in Figure 5.3, it become clear that accuracy gradually increase as image size increases. This inference also can be supported with experiment results further detailed in Table A.2 in Appendix. As in BIT Vehicle Dataset, for Stanford Car Dataset experiments are also performed with ResNet34, ResNet50, ResNext50 and Vgg16 architectures. For each of these architectures, trainings are studied by different image sizes. 32x32, 64x64, 128x128 images size are used for each architecture. In addition to these sizes, 150x150 is used for ResNet50 and 224x224 is used for ResNet34. Giving these parameters as input to

experiment environment, fourteen variation of single level classifier training is conducted using Stanford Car Dataset. Including training loss, validation loss and accuracy values the detailed results of all experiments performed on Stanford Car Dataset are further detailed in Table A.2 in Appendix.

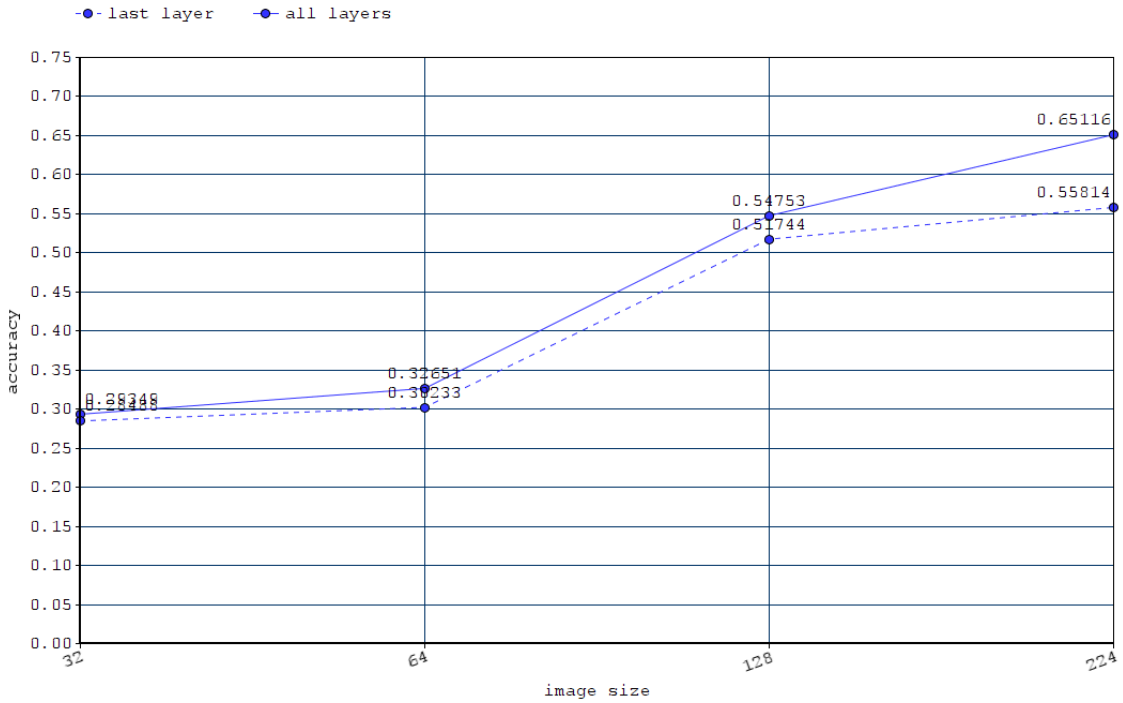

Figure 5.3: Graph of single level training accuracy values with respect to different image size of Stanford Car Dataset images

Comparing experiments results of single level classifier on BIT Vehicle Dataset with results on Stanford Car Dataset, we see that Stanford Car Dataset experiments are performed in less time than BIT Vehicle Dataset experiments. Main reason behind is that, while BIT Dataset has minimum 462 images per class of vehicle type, Stanford Dataset has 100 images per vehicle type label. Because the greater number of the images becomes, it takes more time to process those images. Another important difference between experiment results of two datasets is that the accuracy of the same architecture and same image size is smaller on Stanford Dataset than on BIT Dataset. For example while the accuracy value of last layer training experiment performed with 32x32 image size and ResNet34 architecture on BIT Dataset is 0.44786, on the other hand the accuracy value with the same parameters on the Stanford Dataset is 0.28488. The first reason comes to mind is that, number of the class of the Stanford Dataset is 9 while it is 6 for BIT Dataset. Another reason could be the lack of feature for a single label classifier can learn because Stanford Dataset has low number of images for each vehicle type class. For these reasons we focus on the accuracy of the models

trained on Stanford in the rest of the thesis. Firstly, we would like to improve accuracy by applying ensemble learning and perform fourteen experiments. Results of ensembling learning experiments are further detailed in Table A.3 in Appendix. Although we experienced that ensemble learning technique can give relatively better results on accuracy, we would like to examine the effects of multi - level classifier training for vehicle type classification problem. Next, we discuss the experiment results of multi - level classifier trainings.

5.2. Multi Level Classifier Experiment Results

In this section, we give information about experiments performed on Stanford Dataset using multi level hierarchy approaches. So as to implement multi level hierarchical classifier training, we visualize the vehicle type classification problem with trees and create dependency connections in order to build hierarchy as described in Section 4.2.2.2. Exploiting the dependencies between nodes, we performed experiments on three different hierarchy level proposed which estimates the probability of each root-to-leaf path through cascade classification trees. We provide experimental results which indicate that, one can achieve better results with cascade classifications compared to traditional flat classification. In order to proceed with multi level classifier experiments Stanford Dataset is re-organized and test folder was added as in the distribution of folders shown in Table 5.4.

In tree representation the flow starts from root and goes through the leaf nodes such that each node represents a classifier. Through the achieving at building multi level hierarchies, following the path from root to the leaves of each hierarchical cascade tree, ten classifiers are trained in total. Details of each classifier are given below with relation of nodes in Figures.

1. vehicle_type: Determines probabilities of an image for hatchback, mini, minivan, panelvan, pickup, sedan, sport, station, suv labels (see Figure 4.5)
2. vehicle_type: Determines probabilities of an image for terrain, automobile and van categories (see Figure 4.6)
3. auto_nonauto: Determines probabilities of an image for automobile and non-automobile categories (see Figure 4.7)
4. auto: Determines probabilities of an image for hatchback and sedan categories (see Figure 4.7)
5. auto: Determines probabilities of an image for hatchback, mini, sedan, sport and station labels (see Figure 4.6)

6. nonauto: Determines probabilities of an image for terrain and van categories (see Figure 4.7)
7. terrain: Determines probabilities of an image for pick-up and suv labels (see Figure 4.6 and Figure 4.7)
8. hatchback: Determines probabilities of an image for hatchback and mini labels (see Figure 4.7)
9. sedan: Determines probabilities of an image for sedan, sport and station labels (see Figure 4.7)
10. van: Determines probabilities of an image for minivan and panelvan labels (see Figure 4.7 and 4.7)

Table 5.4: Distribution of images per class for 100~ Stanford Car dataset

Class	Train	Valid	Test	TOTAL
Hatchback	70	15	15	100
Mini	70	9	9	88
Minivan	70	15	15	100
Panelvan	70	15	15	100
Pick-up	70	15	15	100
Sedan	70	15	15	100
Sport	70	15	15	100
Station	70	15	15	100
Suv	70	15	15	100
TOTAL	**630**	**129**	**129**	**888**

Next, experiments are performed by implementing three probabilistic computation method explained in Section 4.2.2.2. With given dataset information, multi level classifiers are trained on ResNet34 architecture with image size 224. Classification is only performed on the leaves of hierarchical cascade tree and each image belongs to only one class. The number of images in the test folder are 129 and the results are mainly evaluated on measuring accuracy. Experiments to compute different methods following different tree hierarchies are done in five times and recorded in detailed Table A.5 in Appendix. Calculated mean values are demonstrated in Table 5.5. The time required to train all

classifiers of level-2 and level-3 hierarchies including computing final estimation for test images took about 3 to 4 minutes for each cycle. For whole 5 five times experimenting it took 17 minutes and 50 seconds.

Table 5.5: Mean and standard deviation values for computation methods on hierarchical trees

Method	**Level-1**	**Level-2**	**Level-3**
PC	0.675723	0.686169	0.679479
std	0.055763	0.051349	0.063217
WAC	0.675723	0.644069	0.655853
std	0.055763	0.057612	0.054077
TAM	0.675723	**0.69018**	0.681934
std	0.055763	**0.044716**	0.057676

According to computed mean values for each computation method-hierarchical tree level tuples in Table 5.5, the best accuracy result is appear to be for Traditional Arithmetic Mean (referred as TAM in Section 4.2.2.2) computation method applied on the hierarchical tree with level-2. On the one hand, the Weighted Average Computation (referred as WAC in Section 4.2.2.2) method for all tree levels get the worst results, while on the other hand, the Probabilistic Cascading (referred as PC in Section 4.2.2.2) computation method applied on the hierarchical tree with level-2 is second, and the TAM computation method applied on the hierarchical tree with level-3 gets the third in the ranking. Looking at Table A.5 in Appendix, because of the accuracy values being close to each other, standard deviation is calculated for each computation method-hierarchical tree level tuples. Standard deviation gives information about distribution of measurements for a specific group, that is how far they are distanced from its average value. It assesses the amount of dispersion or unsteadiness around the mean. Dispersion is the difference between the actual value and the mean value. The larger the dispersion between these two values is, the higher the standard deviation gets. Therefore, a low standard deviation indicates that distribution is stable thanks to most values are at near of average, while a high standard deviation states that the values are distanced from aveage, which results in distribution becoming unstable. Standard deviation is computed using its general formula as in (5.1) and computed values are demonstrated as std in Table 5.5.

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \qquad \begin{aligned} n &= \text{The number of data points} \\ \bar{x} &= \text{The mean of the } x_i \\ x_i &= \text{Each of the values of the data} \end{aligned} \qquad (5.1)$$

The recorded standard deviation values in Table 5.5 also supports the first inference. The smallest standard deviation value is calculated for TAM computation method applied on level-2 hierarchical tree which means that accuracy values of each experiments of five is close to each other and dispersion is stable. For the purpose of supporting this conclusion, the same experiments were repeated by increasing the number of images in the data set. Number of images per class for new experiment are detailed below in Table 5.6.

Table 5.6: Distribution of images per class for 200~ Stanford Car dataset

Class Name	Train	Valid	Test	TOTAL
Hatchback	163	33	33	229
Mini	141	30	30	201
Minivan	160	34	34	228
Panelvan	138	29	29	196
Pick-up	140	30	30	200
Sedan	165	32	32	229
Sport	147	32	32	211
Station	151	31	32	214
Suv	157	33	33	223
TOTAL	**1362**	**284**	**285**	**1931**

With newly added images, total number of instances of the data set is doubled from 888 to 1931. Experiments performed for each computation method-hierarchical tree level tuple are performed five times using the new data set that is doubled in number of images, and keeping other values such as picture size and architecture fixed. So as to make comparisons, the mean of five experimental results was calculated for each tuple and recorded in the Table 5.7. The time required to train all classifiers of level-2 and level-3

hierarchies including computing final estimation for test images took about 7 to 8 minutes for each cycle. For whole 5 five times experimenting it took 41 minutes and 42 seconds.

Table 5.7: Mean values for computation methods applied on hierarchical trees for the number of images increased dataset

Method	**Level-1**	**Level-2**	**Level-3**
PC	0.695387	**0.755859**	0.721313
std	0.024096	**0.047230**	0.041244
WAC	0.695387	0.695875	0.712125
std	0.024096	0.047037	0.040596
TAM	0.695387	**0.762812**	**0.731125**
std	0.024096	**0.027773**	**0.042588**

As a result in these calculations, the best accuracy value is appear to be for TAM computation method applied on the hierarchical tree with level-2. The same order also occurs in last experiment results is that the PC computation method applied on the hierarchical tree with level-2 is second, and the TAM computation method applied on the hierarchical tree with level-3 gets the third in the ranking. In Figure 5.4, final accuracy values for each probability computation method are represented for both initial dataset and number of images increased.

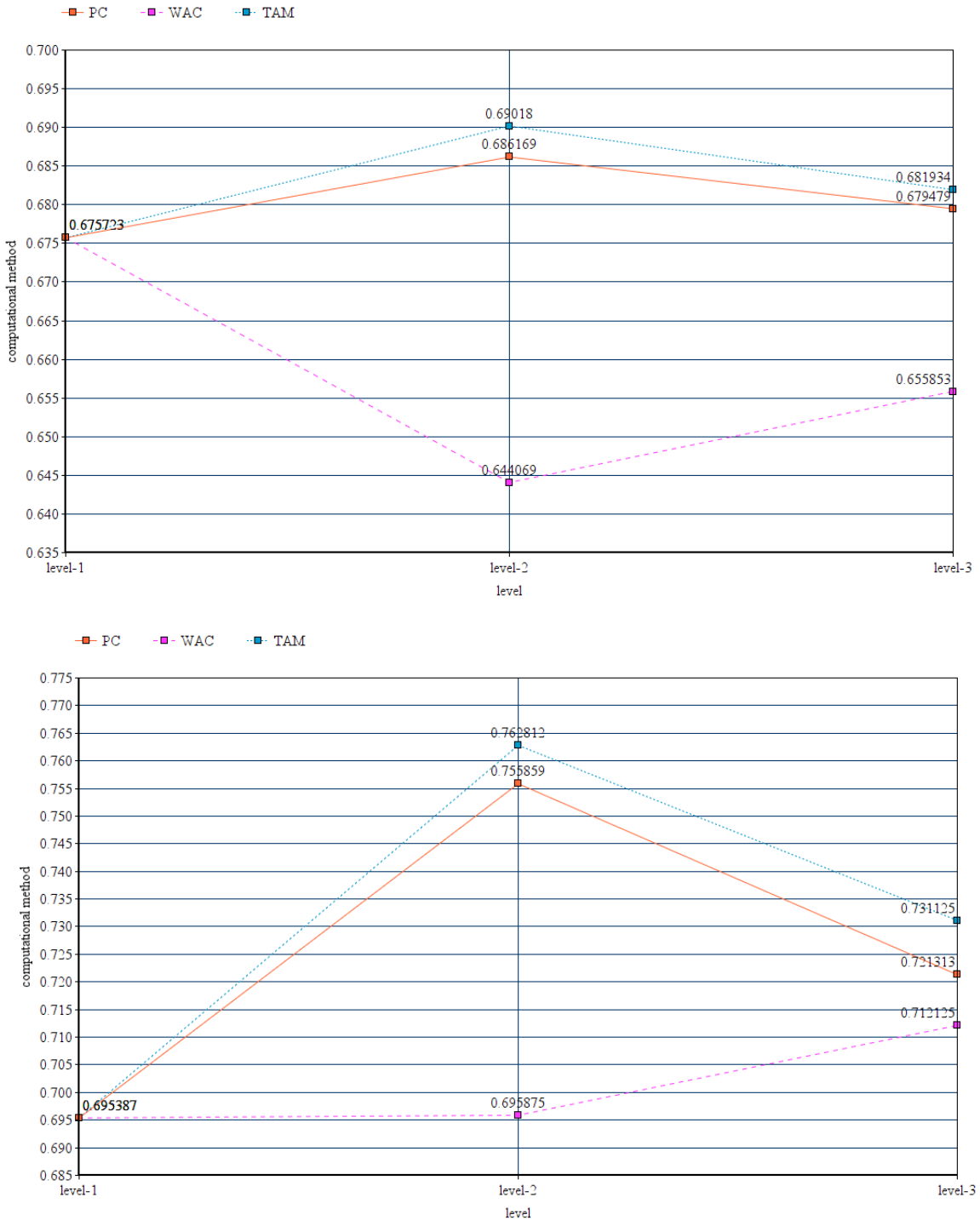

Figure 5.4: Mean values for computation methods applied on hierarchical trees for two datasets with 100~ per images (top) and with 200~ per images (bottom)

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

In this study, utilizing from transfer learning methodology, models are trained in order to construct multi - level classifier hierarchies, each of which determine types of vehicles. Two datasets were selected for classification and the results were shared. In the first, single - level classifier training experiments are performed for BIT Dataset and Stanford Dataset, while in the second multi - level classifier training experiments are carried out for three different classifier hierarchy. The results were evaluated using the same Stanford Dataset class arrangement for both single - level classifier training and multi - level classifier training experiments.

Experimental studies were started by preparing data sets in first. BIT Dataset is acquired by contacting Beijing Institute of Technology and Stanford Dataset is obtained from Stanford University's CarComp Dataset. BIT Dataset has six vehicle type classes which are Bus, Microbus, Minivan, Sedan, Truck, Suv, on the other hand, Stanford Dataset is manually arranged to have nine vehicle type classes which are Hatchback, Mini, Minivan, Panelvan, Pick-up, Sedan, Sport, Station, Suv. The number of images per class, the number of classes and their effects to results are represented in this thesis using two different datasets of which has different number of classes. The number of images per class in BIT Dataset and Stanford Dataset is different to the point that for same classes there is a difference of forty five times. For example, BIT Dataset has 5627 images for Sedan class while Stanford Dataset has 100 images. Due to the high number of images per class in BIT Dataset, trained model learn more about each class which allows the trained model to yield to better results. Furthermore thinking in terms of number of classes, it is expected for the dataset which have the lower number of classes to give more accurate predictions. Because as the number of classes increase it becomes more ambiguous for the classifier to make predictions. Naturally, the Stanford Dataset with the less number of classes gave relatively less accurate results but we enhanced the model by providing synthetic data and applying ensembling methodology. Furthermore, ensembling experiments prove that applying ensembling using two different models with different image sizes for the same architecture enables the merged

model to make more accurate predictions even better than the dataset that has more number of classes than itself.

In single - level classifier training, last layer or all layers training of pre-trained deep neural networks for vehicle type classification has been studied for two different datasets. For each dataset, experiments are carried out with different image size and different pre-trained deep neural networks are used. Image sizes differs from 32x32 to 224x224, including 64x64, 128x128, 150x150 image sizes. Pre-trained deep neural networks used in this thesis are ResNet34, ResNet50, ResNext50 and Vgg16. In experiments performed on both data sets, best results were obtained when image size is bigger for each network choice in terms of accuracy comparison. However, some situations may require involving low resolution images such as zooming when taking a photo. This the main reason why we include 32x32 images in experiments. It is to give insight about how model behave in certain conditions. When the same biggest image size in all networks are taken into consideration for each architecture, the results are very close to each other that it only differs in terms of decimals among architectures. In addition, although Stanford Dataset has more classes than BIT Dataset, training of Stanford Dataset requires much less time than training of BIT Dataset. That's why we researched on how to increase accuracy of training of Stanford Dataset by performing multi - level hierarchy training.

In multi - level training experiments, Stanford Dataset was used to perform experiments by building different training hierarchies on account of studying effects of cascade probability computation. The results of three different computational methods applied on three hierarchies were compared. In order to compare multi - level classifier training experiments results to, a base case scenario with flat hierarchy is created by separating test images from Stanford Dataset and training on ResNet34 with 224x224 image size. Next, the hierarchies with 2 unit height and 3 unit height are built by training 10 models for each of node of hierarchical trees with the same configuration to the base scenario. Therefore, probabilities for each class of each image in test folder are calculated by following the hierarchies. Three different computational techniques are compared in terms of number of correctly classified images and their percentage. As a result, it was observed how accurate the overall hierarchies make predictions by following three different training paths with different hierarchical structure and computing probabilities with three computational methods. After repeating experiments 5 times and calculating the mean of 5 results of applying each computational methods on multi - level training hierarchies, it was observed that level - 2 hierarchical tree makes better prediction than other hierarchies when Traditional Arithmetic Mean (TAM) computation is applied. Repeating the same

experiment by increasing the number of images from ~100 to ~200 per class in Stanford Dataset we repeated the same experiments. Again the more accurate prediction is performed by following level - 2 hierarchy applying TAM technique.

6.2. Future Work

In this thesis, it was observed how a data set was achieved by following three different training mechanisms/paths with different hierarchical structure. Studies have been done with a limited size image with Stanford Dataset. The main limitation here are the shortcomings in the existing graphics card infrastructure. It is foreseen that better results can be obtained if the data set is enlarged and the studies are repeated especially with larger images. It is considered to include more images into the dataset which are taken from different angles or vehicles that are more covered by surrounding objects. The results obtained at this stage are promising for successful classification for more difficult data.

One of the aspect that this study can be improved is applying pre-processing before training a model. The effects of removing background from the images or cropping the parts other than a vehicle can be studied. By separating the external factors from the pictures, the success rate can also be increased. Besides, techniques for creating synthetic data can also be applied for pre-processing such as virtually creating by predicting how an image can be of a larger image size.

Another future aspect could be observing the effects of different hierarchies. For example, firstly a model can be trained to recognize from which angle an image taken. Then, according to the prediction of firstly trained classifier, secondly trained classifiers are able to predict the vehicle type in an image among the dataset that has only one point of view of vehicle types. Furthermore, the effects of different cascade probabilistic computation methods can be studied.

The results in the study show that multi - level classifier training also increases the success when there is sufficient data. This increase in success is relatively long during multi - level training, but it is a fact that it should not be ignored that more accurate results can be produced in cases where there is no time problem.

REFERENCES

Abadi, Martin, Ashish Agarwal, Paul Barham, and et al. (2016) “Tensorflow: Large-scale machine learning on heterogeneous distributed systems”. arXiv:1603.04467.

Baştanlar, Yalın and Mustafa Özuysal. (2014) “Introduction to machine learning”, *miRNomics: MicroRNA Biology and Computational Analysis.* Humana Press, Totowa, NJ, pp. 105-128.

Chollet, François. (2015) Keras. Accessed Apr 4, 2020. https://keras.io/.

Dong, Zhen, Yuwei Wu, Mingtao Pei, and Yunde Jia. (2015) “Vehicle type classification using a semisupervised convolutional neural network”, *IEEE Transactions on Intelligent Transportation Systems*, Vol.16, Issue.4, pp. 2247–2256.

Fazli, Saeid, Shahram Mohammadi, and Morteza Rahmani. (2012) “Neural Network based Vehicle Classification for Intelligent Traffic Control”, *International Journal of Soft. Eng. & Applications* Vol.3, No.3, pp. 17-22. DOI: 10.5121/ijsea.2012.3302

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. (2016) “Deep learning”, *MIT press*, pp. 290-292. https://www.deeplearningbook.org/

Google AI Blog. (2016) AlphaGo: Mastering the ancient game of Go with Machine learning. https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html

Haskell, Richard, and Ali Noui-Mehidi. (1989) “Design of hierarchical classifiers”, *Comp. in the 90's: The First Great Lakes Comp. Science Con. Proc.*, Michigan, USA. pp. 118-124.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. (2016)“Deep residual learning for image recognition”, *IEEE CVPR*, Las Vegas, NV, USA. DOI:10.1109/CVPR.2016.90.

Howard, Jeremy, and Rachel Thomas. (2017) fast.ai: making neural nets uncool again, Accessed Feb 15, 2017. http://fast.ai.

Izmailov, Pavel, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew G. Wilson.(2018)"Averaging weights leads to wider optima and better generalization", *Proc. Of Inter. Conf. On Uncertainty in A.I. (UAI 2018),* arXiv: 1803.05407.

Jayawardena, Srimal. (2013) "Image based automatic vehicle damage detection", *Ph.D. dissertation,* Australian National University.

Jordan, Jeremy. (2018) Setting the learning rate of your neural network. 2018, accessed: Mar 1, 2020. https://www.jeremyjordan.me/nn-learning-rate/

Jordan, Michael, and Tom M. Mitchell. (2015) "Machine learning: Trends, perspectives, and prospects", *Science* Vol.349, Issue.6245, pp. 255-260.

Kosmopoulos, Aris, Georgios Paliouras, Ion Androutsopoulos. (2015) "Probabilistic cascading for large-scale hierarchical classification". arXiv: 1505.02251.

Krause, Jonathan, Michael Stark, Jia Deng, and Li Fei-Fei. (2013) "3D Object Representations for Fine-Grained Categorization", *4th IEEE Workshop on 3D Representation and Recognition, at ICCV 2013* (3dRR-13). Sydney, Australia.

Kukacka, Jan, Vladimir Golkov, and Daniel Cremers. (2017) "Regularization for deep learning: A taxonomy", *CoRR*. arXiv:1710.10686, 2017.

LeCun, Yann, Leon Bottou, Yoshua Bengio, and Patrick Haffner. (1998) "Gradient-based learning applied to document recognition", *Proceedings of the IEEE.* Vol.86, Issue.11, pp. 2278-2324. DOI:10.1109/5.726791

Loshchilov, Ilya, and Frank Hutter. (2017) "SGDR: stochastic gradient descent with restarts", *International Conference on Learning Representations.* arXiv:1608.03983.

Mathworks Inc. (1994) What Makes CNNs So Useful. Accessed: Feb, 2, 2020. www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Pan, Sinno Jialin, and Qiang Yang. (2010) "A survey on transfer learning", *IEEE Trans. Knowl. and Data Eng.*, Vol.22, Issue.10, pp. 1345–59. DOI: 10.1109/TKDE.2009.191.

Patil, Kalpesh, Mandar Kulkarni, Anand Sriraman, and Shirish Karande. (2017) "Deep learning based damage classification". *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 50-54.

Paszke, Adam, Sam Gross, Soumith Chintala, and Gregory Chanan. (2017) “PyTorch: tensors and dynamic neural networks in python with strong gpu acceleration”, Accessed Feb 15, 2020. https://pytorch.org/

Ras, Zbigniew W., Agnieszka Dardzinska and Wenxin Jiang. (2010) “Cascade Classifiers for Hierarchical Decision Systems”, in *Advances in Machine Learning I. Studies in Computational Intelligence, Vol 262. Springer, Berlin, Heidelberg*.

Roecker Max N., Yandre M.G. Costa, Joao L.R. Almeida, and Gustavo H.G Matsushite. (2018) “Automatic vehicle type classification with convolutional neural networks”, *25th Inter. Conf. on Systems Signals and Image Proc. (IWSSIP)*, pp. 1-5.

Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. arXiv: 1609.04747.

Simonyan, Karen, and Andrew Zisserman. (2015) “Very deep convolutional networks for large-scale image recognition”, *International Conference on Learning Representations*, pp. 1–14. arXiv:1409.1556.

Smith, Leslie N.(2017)“Cyclical learning rates for training neural networks” IEEE *Winter Conference on Applications of Computer Vision.*, Santa Rosa, CA, USA pp. 464–472.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, and Dumitru Erhan. (2015) “Going deeper with convolutions”, *Proc. IEEE Conf. Comp. Vis. Pattern Recog*., Boston, MA, USA, pp 1-9.

Vazquez, Favio. (2017) Deep Learning Made Easy with Deep Cognition. Accessed Feb 2, 2020. becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351

Won, Myounggyu. (2020) “Intelligent Traffic Monitoring Systems for Vehicle Classification: A Survey”, *IEEE Access,* Vol 8, pp. 73340 - 73358 arXiv:1910.04656.

Yan Zhicheng, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. (2015) “HD-CNN: Hierarchical Deep Convolutional Neural Network for Large Scale Visual Recognition”, *Proceedings of the International Conference on Computer Vision (ICCV)*, pp. 2740–2748. arXiv:1410.0736.

Yu, Lequan, Hao Chen, Qi Dou, Jin Qin, and Pheng-Ann Heng. (2017) “Automated melanoma recognition in dermoscopy images via very deep residual networks”, *IEEE Transactions on Medical Imaging,* Vol. 36, No. 4, pp. 994 - 1004.

APPENDICES

APPENDIX A

DETAILED EXPERIMENT RESULTS

1. BIT Vehicle Type Dataset Experiment Results Comparison

Table A.1: Experiment result of training last layer and all layers of each architecture using BIT Dataset images with six classes.

Architecture	Image size	Trained layers	Training loss	Validation loss	Accuracy (%)	Time
ResNet34	32	Last Layer All Layers	0.63913 0.3936	1.88168 1.92105	0.44786 0.48298	11 min 05s 26 min 59s
	64	Last Layer All Layers	0.49206 0.23368	1.72825 1.61256	0.52788 0.57277	11 min 17s 27 min 44s
	128	Last Layer All Layers	0.28462 **0.02740**	1.62993 1.21851	0.61694 0.74366	12 min 17s 36 min 10s
	224	Last Layer All Layers	0.29082 0.08114	1.36875 **1.01714**	0.65424 **0.77009**	15 min 33s **1h 20min 36s**
ResNet50	32	Last Layer All Layers	0.57006 0.33643	1.87802 1.82212	0.46742 0.49457	10 min 52s 26 min 19s
	64	Last Layer All Layers	0.47035 0.18029	1.8501 1.40715	0.52824 0.62925	11 min 19s 27 min 49s
	128	Last Layer All Layers	0.33033 0.08073	1.50944 **1.09149**	0.61007 0.73316	12 min 20s 48 min
	150	Last Layer All Layers	0.29839 **0.06378**	1.52645 1.1183	0.63903 **0.75634**	14 min 14s **1h 8 min 21s**
ResNext50	32	Last Layer All Layers	0.54844 0.34427	1.81678 1.84558	0.49131 0.49819	10 min 50s 25 min 53s
	64	Last Layer All Layers	0.45647 **0.15135**	1.71525 **1.40161**	0.5344 0.64337	12 min 02s 27 min 32s
	128	Last Layer All Layers	0.32642 0.18623	1.41776 1.49338	0.61405 **0.64446**	13 min 24s **30 min 05s**
Vgg16	32	Last Layer All Layers	0.59981 0.29486	1.87465 **0.66550**	0.48226 0.53005	11 min 12 s 26 min 26s
	64	Last Layer All Layers	0.51266 **0.17884**	1.71787 1.52944	0.53983 0.61115	12 min 35 min 43s
	128	Last Layer All Layers	0.3853 0.96889	1.46764 1.22042	0.59232 **0.71325**	18 min 36s **1h 31min 20s**

2. Stanford Vehicle Type Dataset Experiment Results Comparison

Table A.2: Experiment results of training last layer and all layers of each architecture using Stanford Dataset images with nine classes

Architecture	Image Size	Trained layers	Training loss	Validation loss	Accuracy (%)	Time
ResNet34	32	Last Layer All Layers	2.20407 2.08773	2.08856 2.04804	0.28488 0.29349	18s 1 min 40
	64	Last Layer All Layers	2.067591 1.88601	1.90531 2.01203	0.30233 0.32651	26s 2 min 05s
	128	Last Layer All Layers	1.55797 1.27692	1.33347 1.33658	0.51744 0.54753	37s 3 min 19s
	224	Last Layer All Layers	1.32541 **0.83031**	1.09792 **1.05054**	0.55814 **0.65116**	1 min 25s **8 min 20s**
ResNet50	32	Last Layer All Layers	2.14952 2.07112	1.98640 1.99301	0.27326 0.28535	24s 1 min 32s
	64	Last Layer All Layers	1.75417 1,63388	1.81545 1.76889	0.36628 0.38884	24s 1 min 55s
	128	Last Layer All Layers	1.24560 **0.33438**	1.20947 1.24685	0.61628 0.63372	48s 4 min 38s
	150	Last Layer All Layers	1.09324 0,6944	1.16414 **1.03003**	0.59883 **0.68023**	1 min 07s **6 min 54s**
ResNext50	32	Last Layer All Layers	2.00160 1,96781	1.94285 1.9163	0.23256 0.24186	20s 1 min 15s
	64	Last Layer All Layers	1.80058 1.60042	1.79406 1.76686	0.37209 0.39651	26s **2 min 20s**
	128	Last Layer All Layers	**1.2377** -	**1.0685** -	**0.60465** -	59s -
Vgg16	32	Last Layer All Layers	2.25661 2.02498	2.09126 2.05267	0.26163 0.27419	19s 1 min 24s
	64	Last Layer All Layers	2.06965 1.74158	1.75912 1.67859	0.37791 0.45349	42s 3 min 20s
	128	Last Layer All Layers	1.65253 **1.15283**	1.41063 **1.11932**	0.49419 **0.61047**	1 min 47s **9 min 12s**

3. Ensembling Results

In order to improve accuracy of experiments performed on Stanford Dataset, ensemble learning technique is applied. So as to apply ensemble learning additional 14 experiments was conducted. In order to get best accuracy results of ensembling technique, the model trained with biggest image size on the same architecture is chosen as a second model for each architecture. For example, implementing ensemble learning on ResNet34 architecture, the model trained with the same architecture and 224 image size is used as a second model for every other trained model with different image size. The result accuracy values for ResNet34 architecture can be seen in Table A.3.

Table A.3: Experiment results of ensembling on ResNet34 architecture

Architecture	Image Size	Accuracy (%)	Accuracy (by ensembling)	Second Model Image Size
ResNet34	32	0.28488	0.488372	224
	64	0.30233	0.552326	224
	128	0.51744	0.593721	224
	224	0.55814	0.602791	224

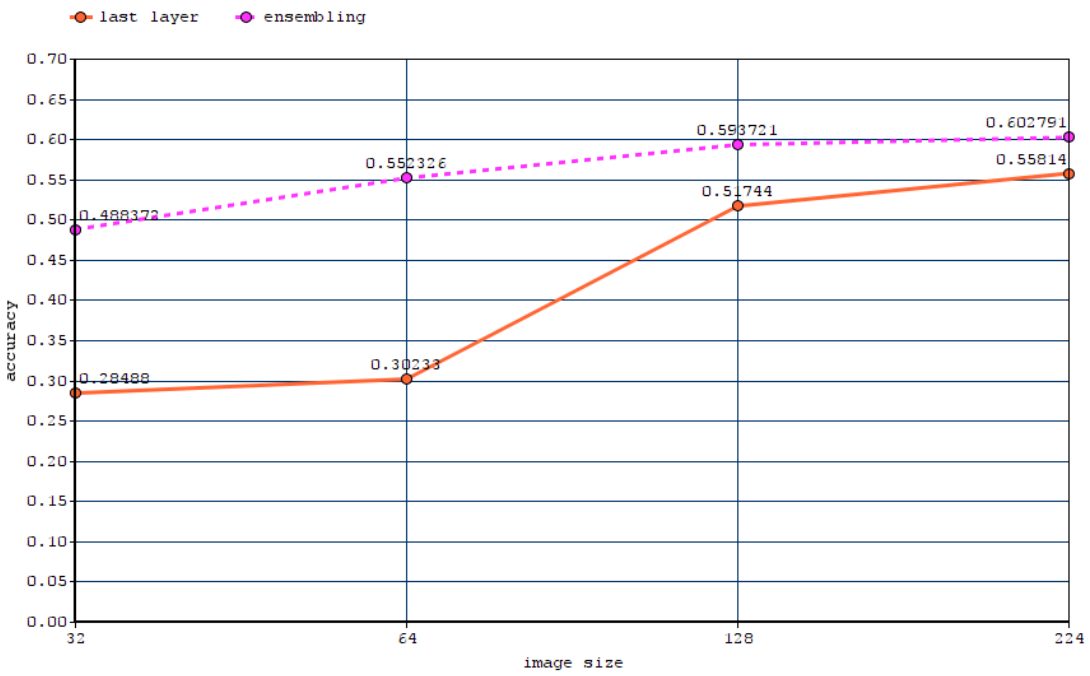

Figure A.1: Graph of comparing accuracy values of ensembling on ResNet34 architecture

When the experimental results of the last layer trainings on the ResNet34 architecture using different image sizes were compared, it was found that the accuracy values obtained when the ensembling technique was applied were better. As the image size gets smaller, the accuracy improvement rate increases. Experiment on size 224 shows that accuracy is improved from 0.55814 to 0.602791 with increase of 0.044651 by 8% while experiment on size 32 shows that accuracy is improved from 0.28488 to 0.488372 with the increase of 0.203492 by 71%. As for the experiment results on other selected architectures can be found in Table A.4. For ResNet50, ResNext50 and Vgg16 architectures, ensembling learning is applied with trained models by sizes 150, 128 and 128 in respectively.

Table A.4: Experiment results of applying ensembling learning for models trained on ResNet34, ResNet50, ResNext50 and Vgg16 architectures

Architecture	**Image Size**	**Accuracy (%)**	**Accuracy (by ensembling)**	**Second Model Image Size**
ResNet34	32	0.28488	0.488372	224
	64	0.30233	0.552326	
	128	0.51744	0.593721	
	224	0.55814	0.602791	
ResNet50	32	0.27326	0.50814	150
	64	0.36628	0.551395	
	128	0.61628	0.633721	
	150	0.59883	0.636977	
ResNext50	32	0.23256	0.494186	128
	64	0.37209	0.552326	
	128	0.60465	0.616279	
Vgg16	32	0.26163	0.446512	128
	64	0.37791	0.493023	
	128	0.49419	0.510465	

Comparing the accuracy values of each single level classifier of Ensemble learning experiments to BIT Dataset experiments, we can say that accuracy values obtained by

applying ensemble learning only better from Stanford Dataset experiment results but not better from the accuracy values obtained by experimenting on BIT Dataset.

4. Multi-Level Experiment Results with 100~ Images per Class

Table A.5: Mean values for computation methods applied on hierarchical trees for initial dataset

Method	Level-1	Level-2	Level-3
PC	0.604419	0.648438	0.671875
	0.680687	0.625	0.612812
	0.622308	0.664062	0.612812
	0.75	0.736472	0.773438
	0.7212	0.756875	0.726457
mean	0.675723	**0.686169**	0.679479
std	0.055763	**0.051349**	0.063217
WAC	0.604419	0.64979	0.689688
	0.680687	0.558925	0.568125
	0.622308	0.601362	0.617188
	0.75	0.709125	0.703125
	0.7212	0.701141	0.701141
mean	0.675723	0.644069	0.655853
std	0.055763	0.057612	0.054077
TAM	0.604419	0.664062	0.671875
	0.680687	0.63465	0.61467
	0.622308	0.65625	0.632812
	0.75	0.769062	0.773438
	0.7212	0.726875	0.716875
mean	0.675723	**0.69018**	**0.681934**
std	0.055763	**0.044716**	**0.057676**

5. Multi - Level Experiment Results with 200~ Images per Class

Table A.6: Mean values for computation methods applied on hierarchical trees for the number of images increased dataset

Method	**Level-1**	**Level-2**	**Level-3**
PC	0.701754	0.78125	0.734375
	0.69338	0.78523	0.726562
	0.723183	0.710938	0.710938
	0.707317	0.8125	0.78125
	0.651303	0.689375	0.653438
mean	0.695387	**0.755859**	0.721313
std	0.024096	**0.047230**	0.041244
WAC	0.701754	0.758438	0.752188
	0.69338	0.745312	0.757812
	0.723183	0.65375	0.691875
	0.707317	0.645312	0.647812
	0.651303	0.676562	0.710938
mean	0.695387	0.695875	0.712125
std	0.024096	0.047037	0.040596
TAM	0.701754	0.789062	0.734375
	0.69338	0.765625	0.757812
	0.723183	0.733125	0.710938
	0.707317	0.796875	0.789062
	0.651303	0.729375	0.663438
mean	0.695387	**0.762812**	**0.731125**
std	0.024096	**0.027773**	**0.042588**

APPENDIX B

DEEP LEARNING SCRIPT DEVELOPED IN THIS THESIS

```
# Put these at the top of every notebook, to get automatic reloading and inline plotting
%reload_ext autoreload
%autoreload 2
%matplotlib inline
# This file contains all the main external libs we'll use
from fastai.imports import *
from fastai.transforms import *
from fastai.conv_learner import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *

PATH_test = "test/"
sz=224# size that images will be resized to
arch=resnet34
result_ct_mult = []
result_ct_cond = []
result_ct_avg = []

def trainModel(path, v_test_with_labels):
    if v_test_with_labels == True:
        data = ImageClassifierData.from_paths(path, tfms=tfms_from_model(arch, sz),
test_name=PATH_test, test_with_labels=v_test_with_labels)
        learn = ConvLearner.pretrained(arch, data, precompute=True)
        learn.fit(0.01, 3)
    else:
        data = ImageClassifierData.from_paths(path, tfms=tfms_from_model(arch, sz),
test_name=PATH_test)
        learn = ConvLearner.pretrained(arch, data, precompute=True)
        learn.fit(0.01, 3)
    return data, learn

def trainAndSave(path, nameToSave, v_test_with_labels, isReset=False):
    if(isReset):
        shutil.rmtree(f'{path}tmp', ignore_errors=True)

    data, learner = trainModel(path, v_test_with_labels);
    learner.save(nameToSave)
    return data, learner

def loadModel(path, saved_model_name, v_test_with_labels):
    if v_test_with_labels == True:
```

```
        data = ImageClassifierData.from_paths(path, tfms=tfms_from_model(arch, sz),
test_name=PATH_test, test_with_labels=v_test_with_labels)
        learn = ConvLearner.pretrained(arch, data, precompute=True)
        learn.load(saved_model_name)
    else:
        data = ImageClassifierData.from_paths(path, tfms=tfms_from_model(arch, sz),
test_name=PATH_test)
        learn = ConvLearner.pretrained(arch, data, precompute=True)
        learn.load(saved_model_name)
    return data, learn

def createImgProbPairForGroundTruth(fpaths, preds):#an array of an img name relative to
its probability [img name]
    img_names = [os.path.basename(d) for d in fpaths]
    return np.asarray(img_names)

def getPreds(learn):#an array of an img name relative to its probability [img name]
    log_preds = learn.predict(is_test=True)#log_preds.shape
    preds = np.argmax(log_preds, axis=1)  # from log probabilities to 0, 1, 2,3,4,5,6,7,8
    probs = np.exp(log_preds[:])        # pr(hatchback)
    return preds, probs

def arrangeProbs(img_names_truth, img_names_new, preds_new, probs_new,
class_count):
    probs_result = np.zeros(len(img_names_truth)*class_count)
    probs_result = probs_result.reshape(len(img_names_truth), class_count)
    preds_result = np.zeros(len(preds_new))
    #img_names_result = []
    for i in range(len(img_names_truth)):
        img_idx = np.where(img_names_new == img_names_truth[i])
        preds_result[i] = preds_new[img_idx[0][0]];
        probs_result[i] = probs_new[img_idx[0][0]];
    return preds_result, probs_result

def calculateAccuracy(v_preds, v_labels):
    sum = 0
    for i in range(len(v_preds)):
        x = v_preds[i]
        y = v_labels[i]
        if x == y:
            sum = sum +1
    print(sum/len(v_preds))
    return sum/len(v_preds)

def getProbs(img_names, learn, data, classCount=2):#an array of an img name relative to
its probability [img name]
    preds, probs = getPreds(learn);
    img_names_created =
createImgProbPairForGroundTruth(np.asarray(data.test_ds.fnames),preds)
    res_preds, res_probs = arrangeProbs(img_names, img_names_created, preds, probs,
classCount)
    return res_probs
```

```
#hierachical classifiers computation methods
def computeWeightedAverage(v_prob_list):#1=WAC
    x = 0
    v_arr = v_prob_list
    if len(v_prob_list) == 2:
        return 0.3*v_arr[0]+0.7*v_arr[1]
    else:
        return 0.1*v_arr[0]+0.3*v_arr[1]+0.6*v_arr[2]

def computeMeanProbability(v_prob_list):#2=TAM
    x = 0
    calculated_prob = 0
    v_arr = v_prob_list
    if len(v_prob_list) == 2:
        return (v_arr[0]+v_arr[1])/2
    else:
        return (v_arr[0]+v_arr[1]+v_arr[2])/3

def computeProbabilisticCascading(v_prob_list):#3=PC
    x = 0
    calculated_prob = 0
    v_arr = v_prob_list
    if len(v_prob_list) == 2:
        return v_arr[0]*v_arr[1]
    else:
        return v_arr[0]*v_arr[1]*v_arr[2]

def computeProbs(v_prob_list, methodType):
    if methodType == 1:
        return computeWeightedAverage(v_prob_list)
    elif methodType == 2:
        return computeMeanProbability(v_prob_list)
    else:
        return computeProbabilisticCascading(v_prob_list)

def computeTreeHeightProbs(treeHeight, methodType, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class):
    if treeHeight == 2:
        return calculateProbsH2(methodType, probs_vehicle_type_3class,
probs_auto_5class, probs_van, probs_arazi)
    else:
        return calculateProbsH3(methodType, probs_auto_nonauto, probs_auto_2class,
probs_nonauto, probs_hatchback, probs_van, probs_arazi, probs_sedan)

def calculateProbsH2(methodType, probs_vehicle_type_3class, probs_auto_5class,
probs_van, probs_arazi):
    probs_hier = np.ones((128, 9))
    for i in range(len(probs_hier)):
```

```
        probs_hier[i][0] =
computeProbs([probs_vehicle_type_3class[i][1],probs_auto_5class[i][0]],methodType)#h
atchback
        probs_hier[i][1] =
computeProbs([probs_vehicle_type_3class[i][1],probs_auto_5class[i][1]],methodType)#m
ini
        probs_hier[i][2] =
computeProbs([probs_vehicle_type_3class[i][2],probs_van[i][0]],methodType)#minivan
        probs_hier[i][3] =
computeProbs([probs_vehicle_type_3class[i][2],probs_van[i][1]],methodType)#panelvan
        probs_hier[i][4] =
computeProbs([probs_vehicle_type_3class[i][0],probs_arazi[i][0]],methodType)#pickup
        probs_hier[i][5] =
computeProbs([probs_vehicle_type_3class[i][1],probs_auto_5class[i][2]],methodType)#se
dan
        probs_hier[i][6] =
computeProbs([probs_vehicle_type_3class[i][1],probs_auto_5class[i][3]],methodType)#sp
or
        probs_hier[i][7] =
computeProbs([probs_vehicle_type_3class[i][1],probs_auto_5class[i][4]],methodType)#st
ation
        probs_hier[i][8] =
computeProbs([probs_vehicle_type_3class[i][0],probs_arazi[i][1]],methodType)#suv
    return probs_hier

def calculateProbsH3(methodType, probs_auto_nonauto, probs_auto_2class,
probs_nonauto, probs_hatchback, probs_van, probs_arazi, probs_sedan):
    probs_hier = np.ones((128, 9))
    for i in range(len(probs_hier)):
        probs_hier[i][0] =
computeProbs([probs_auto_nonauto[i][0],probs_auto_2class[i][0],probs_hatchback[i][0]],
methodType)#hatchback
        probs_hier[i][1] =
computeProbs([probs_auto_nonauto[i][0],probs_auto_2class[i][0],probs_hatchback[i][1]],
methodType)#mini
        probs_hier[i][2] =
computeProbs([probs_auto_nonauto[i][1],probs_nonauto[i][1],probs_van[i][0]],methodTy
pe)#minivan
        probs_hier[i][3] =
computeProbs([probs_auto_nonauto[i][1],probs_nonauto[i][1],probs_van[i][1]],methodTy
pe)#panelvan
        probs_hier[i][4] =
computeProbs([probs_auto_nonauto[i][1],probs_nonauto[i][0],probs_arazi[i][0]],methodT
ype)#pickup
        probs_hier[i][5] =
computeProbs([probs_auto_nonauto[i][0],probs_auto_2class[i][1],probs_sedan[i][0]],meth
odType)#sedan
        probs_hier[i][6] =
computeProbs([probs_auto_nonauto[i][0],probs_auto_2class[i][1],probs_sedan[i][1]],meth
odType)#spor
```

```
        probs_hier[i][7] =
computeProbs([probs_auto_nonauto[i][0],probs_auto_2class[i][1],probs_sedan[i][2]],meth
odType)#station
        probs_hier[i][8] =
computeProbs([probs_auto_nonauto[i][1],probs_nonauto[i][0],probs_arazi[i][1]],methodT
ype)#suv
    return probs_hier

def computeTreeHeightPreds(probs_hier):
    tmp_arr_probs_hier = []
    tmp_arr_probs_class_idx_hier = []
    for i in range(len(probs_hier)):
        tmp_arr_probs_hier = probs_hier[i,:]
        max_prob_hier = np.max(tmp_arr_probs_hier)#max class prob for each image
        max_prob_idx_hier = np.where(tmp_arr_probs_hier == max_prob_hier)
        tmp_arr_probs_class_idx_hier.append(max_prob_idx_hier[0][0])
    return tmp_arr_probs_class_idx_hier

def doExperiment(PATH, i, isReset):
    #train and save classifiers
    data_9class, learner_9class = trainAndSave(PATH + "vehicle_type_9class/",
"vehicle_type_9class", True, isReset);#hatchback, mini, minivan, panelvan, pickup, sedan,
spor, station suv
    data_auto_nonauto, learner_auto_nonauto = trainAndSave(PATH + "auto_nonauto/",
"auto_nonauto", False, isReset);#auto, nonauto
    data_auto_2class, learner_auto_2class = trainAndSave(PATH + "auto_2class/",
"auto_2class", False, isReset);#hatchback, sedan
    data_nonauto, learner_nonauto = trainAndSave(PATH + "nonauto/", "nonauto", False,
isReset);#arazi, van
    data_hatchback, learner_hatchback = trainAndSave(PATH + "hatchback/", "hatchback",
False, isReset);#hatchback, mini
    data_sedan, learner_sedan = trainAndSave(PATH + "sedan/", "sedan", False,
isReset);#sedan, spor, station
    data_arazi, learner_arazi = trainAndSave(PATH + "arazi/", "arazi", False,
isReset);#pickup, suv
    data_van, learner_van = trainAndSave(PATH + "van/", "van", False, isReset);#minivan,
panelvan
    data_vehicle_type_3class, learner_vehicle_type_3class = trainAndSave(PATH +
"vehicle_type_3class/", "vehicle_type_3class", False, isReset);#arazi, automobile, van
    data_auto_5class, learner_auto_5class = trainAndSave(PATH + "auto_5class/",
"auto_5class", False, isReset);#hatchback, mini, sedan, spor, station

    #calculate accuracy for height=1 tree
    preds_9class, probs_9class = getPreds(learner_9class);
    #ground truth labels
    labels = [d[1] for d in data_9class.test_ds]
    img_names =
createImgProbPairForGroundTruth(np.asarray(data_9class.test_ds.fnames),preds_9class)

    #compute probs
    probs_auto_nonauto = getProbs(img_names, learner_auto_nonauto,
data_auto_nonauto);#probs_auto_nonauto
```

```
    probs_auto_2class = getProbs(img_names, learner_auto_2class,
data_auto_2class);#probs_auto_2class
    probs_nonauto = getProbs(img_names, learner_nonauto, data_nonauto);#probs_nonauto
    probs_hatchback = getProbs(img_names, learner_hatchback,
data_hatchback);#probs_hatchback
    probs_sedan = getProbs(img_names, learner_sedan, data_sedan,3);#probs_sedan
    probs_arazi = getProbs(img_names, learner_arazi, data_arazi);#probs_arazi
    probs_van = getProbs(img_names, learner_van, data_van);#probs_van
    probs_vehicle_type_3class = getProbs(img_names, learner_vehicle_type_3class,
data_vehicle_type_3class,3);#probs_vehicle_type_3class
    probs_auto_5class = getProbs(img_names, learner_auto_5class,
data_auto_5class,5);#probs_auto_5class

    #compute accuracy for each tree and computational method
    probs_h3_conditional = computeTreeHeightProbs(3,1, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);
    probs_h3_mean = computeTreeHeightProbs(3,2, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);
    probs_h3_multiplication = computeTreeHeightProbs(3,3, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);
    probs_h2_conditional = computeTreeHeightProbs(2,1, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);
    probs_h2_mean = computeTreeHeightProbs(2,2, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);
    probs_h2_multiplication = computeTreeHeightProbs(2,3, probs_auto_nonauto,
probs_auto_2class, probs_nonauto, probs_hatchback, probs_van, probs_arazi,
probs_sedan, probs_vehicle_type_3class, probs_auto_5class);

    preds_h3_conditional = computeTreeHeightPreds(probs_h3_conditional);
    preds_h3_mean = computeTreeHeightPreds(probs_h3_mean);
    preds_h3_multiplication = computeTreeHeightPreds(probs_h3_multiplication);
    preds_h2_conditional = computeTreeHeightPreds(probs_h2_conditional);
    preds_h2_mean = computeTreeHeightPreds(probs_h2_mean);
    preds_h2_multiplication = computeTreeHeightPreds(probs_h2_multiplication);

    #height=1
    h1a = calculateAccuracy(preds_9class,labels)
    h1b = calculateAccuracy(preds_9class,labels)
    h1c = calculateAccuracy(preds_9class,labels)
    #height=2
    h2a = calculateAccuracy(preds_h2_conditional,labels)
    h2b = calculateAccuracy(preds_h2_mean,labels)
    h2c = calculateAccuracy(preds_h2_multiplication,labels)
    #height=3
    h3a = calculateAccuracy(preds_h3_conditional,labels)
    h3b = calculateAccuracy(preds_h3_mean,labels)
    h3c = calculateAccuracy(preds_h3_multiplication,labels)
```

```
    #h1_tree = [h1a,h1b,h1c]
    #h2_tree = [h2a,h2b,h2c]
    #h3_tree = [h3a,h3b,h3c]

    ct_mult = [h1c, h2c, h3c]
    ct_cond = [h1a, h2a, h3a]
    ct_avg = [h1b, h2b, h3b]

    result_ct_mult.append(ct_mult)
    result_ct_cond.append(ct_cond)
    result_ct_avg.append(ct_avg)

    return

def calculateStandartDeviation(n_times, result_file, result_ct, ct_name):
    arr0 = []
    arr1 = []
    arr2 = []
    for i in range(n_times):
        arr_ct = result_ct[i]
        result_file.write(str(ct_name)+" \t " + str(round(arr_ct[0], 6))+" \t " +
str(round(arr_ct[1], 6))+" \t " + str(round(arr_ct[2], 6))+" \r\n")
        arr0.append(arr_ct[0])
        arr1.append(arr_ct[1])
        arr2.append(arr_ct[2])

    result_file.write("mean \t " + str(round(np.mean(arr0), 6))+" \t " +
str(round(np.mean(arr1), 6))+" \t " + str(round(np.mean(arr2), 6))+" \r\n")
    result_file.write("stnd \t " + str(round(np.std(arr0), 6))+" \t " + str(round(np.std(arr1),
6))+" \t " + str(round(np.std(arr2), 6))+" \r\n")
    result_file.write("\r\n")
    return

def printResults(result_file, n_times):
    result_file.write("x \t height=1 \t height=2 \t height=3 \r\n")

    calculateStandartDeviation(n_times, result_file, result_ct_mult, "mult")
    calculateStandartDeviation(n_times, result_file, result_ct_cond, "cond")
    calculateStandartDeviation(n_times, result_file, result_ct_avg, "mean")

    result_file.close()
    return

def computeNtimes(n_times, isReset):
    result_file= open("experiment_results_200_cv_22.txt","w+")
    result_file.write("START: \t " + str(np.datetime64('now'))+" \r\n")

    doExperiment("../../data_200/", 1, isReset)
    result_file.write("1. Exp: \t " + str(np.datetime64('now'))+" \r\n")
    doExperiment("../../data_200_f2/", 2, isReset)
    result_file.write("2. Exp: \t " + str(np.datetime64('now'))+" \r\n")
```

```
doExperiment("../../data_200_f3/", 3, isReset)
result_file.write("3. Exp: \t " + str(np.datetime64('now'))+" \r\n")
doExperiment("../../data_200_f4/", 4, isReset)
result_file.write("4. Exp: \t " + str(np.datetime64('now'))+" \r\n")
doExperiment("../../data_200_f5/", 5, isReset)
result_file.write("5. Exp: \t " + str(np.datetime64('now'))+" \r\n END.\r\n\r\n")
printResults(result_file, n_times)
return
```