Training Strategies for Deep Architectures in Computer Vision

Wenliang Dai [1]

MSc Data Science and Machine Learning

Supervisor: Iasonas Kokkinos

Submission date: September 2018

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Data Science and Machine Learning degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text. *Either:* The report may be freely copied and distributed provided the source is explicitly acknowledged *Or:* The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

In recent years, the development of Deep Learning and Convolutional Neural Networks has made a large improvement in most tasks of Computer Vision. This thesis examines and explores a training strategy called "ImageNet pre-training". To achieve our goal, two extra training strategies are also applied: multi-task learning and regularization. We offer two hypotheses in this thesis: (1) is ImageNet pre-training necessary and non-replaceable for semantic segmentation? (2) what is the effect of ImageNet pre-training on deep architectures? Is it initialization, or regularization, or both?

Our main contribution is that we provide a detailed analysis and understanding for training strategies, especially the ImageNet pre-training, through well designed methods and experiments. In addition, we proved that non pre-trained models can achieve a better performance on the training data than pre-trained models. As a result, with proper model designs and regularization, pre-training might not be necessary. We make conclusions that can be useful when designing an architecture for Computer Vision in the future.

For accompanying code, see this repository: github link.

Acknowledge

I would like to thank my supervisor, Dr. Iasonas Kokkinos, for his very helpful proposals, advice, and support throughout this project. Without him, this project would never have gone very far. Furthermore, I would like to also thank my parents for their encouragement and assistance to my studies and life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we firstly make an introduction and state the background of the problems that have been researched in this thesis. Next, we discuss the motivation behind this. Finally, we describe the aim and objectives of this thesis.

1.1 Background

Computer Vision is a large and active research field that has existed for more than 50 years. In this field, researchers try to automate tasks that the human visual system can do. Recently, with the development of Deep Learning (DL) and Convolutional Neural Networks (CNNs), a huge improvement has been made in most tasks of Computer Vision, such as image classification, object detection, semantic segmentation, instance segmentation, scene parsing, etc.

A common concern of Computer Vision tasks is about extracting information from images. Only with sufficient and hierarchical information, we can make reasonable decisions. Pre-training models on the ImageNet [39] dataset is a common methodology used by many models to achieve state-of-the-art results on Computer Vision tasks, e.g. [29, 35, 14]. On the contrary, in recent years, there are also models trained from scratch that also achieve state-of-the-art performance, e.g. [42, 21]. In this project, we want to know what kinds of effects does the ImageNet pre-training have, whether pre-training on the ImageNet is necessary or not, and whether other training strategies can have more competitive advantages than it.

1.2 Motivation

In 2012, the AlexNet [25] paper brings Neural Networks back to the front of Computer Vision, and Deep Learning becomes one of the most popular sub-field of Machine Learning. In the following years, a lot of research has been conducted in Deep Learning and there are many excellent deep architectures that achieve start-of-the-art performance in many tasks, such as VGG-Net [43], ResNet [16], R-CNN [8], etc. In the popular architectures, some training strategies are commonly used to improve the performance of the models. But sometimes it is not clear that why a strategy work, when to use a specific strategy, and whether multiple strategies have impacts within themselves.

One popular strategy is pre-training models on the ImageNet [39] classification dataset, which contains more than 14 million labelled images. Many Deep Neural Networks (DNNs) are pre-trained on it because they require a large amount of training data to converge, and most task-specific datasets are not large enough. As a result, most popular DNNs are pre-trained on the ImageNet dataset and then fine-tuned on the task-specific dataset to get a good performance. Recently, there are also architectures trained from scratch achieve state-of-the-art performance, e.g. [42, 21]. This raises a question: is pre-training really necessary? In addition, ImageNet pre-training has some already known disadvantages. Firstly, the pre-trained network contains a large number of parameters, and usually not all of them are necessary. Secondly, the pre-trained network is not flexible, it significantly limits the space of designing architectures for other Computer Vision tasks.

In our very fist experiment, we noticed that a non pre-trained model (FCN with ResNet-18) can achieve a much better performance on the training data, which means both pre-training initialization and random initialization can optimize the loss function quite well, and the effect of ImageNet pre-training is regularization. Motivated by that, we designed the other experiments. Specifically, two strategies are used and manipulated: multi-task learning and regularization.

1.3 Aim and Objectives

As described in Section 1.1 and 1.2, deep learning is a very active research field and there are many outstanding architectures and training strategies. In this thesis, we discuss the effects of a widely-used training strategy for Computer Vision called ImageNet pre-training. As introduced in Section 1.2, we want to explore whether its regularization effect can be achieved by other methods. Specifically, two methods are explored: multi-task learning

and weight decay regularization.

Aim. Analyze and verify the effects of training strategies (pre-training on ImageNet, multi-task learning, and regularization) through theoretical analysis and empirical experiments. Compare and contrast them, which can help people to make appropriate decisions when designing new deep learning architectures in the future.

Objectives:

- Review related deep learning papers, summarize approaches used for pre-training, multi-task learning, and regularization.
- Implement appropriate deep learning models for our experiments, they should be able to turn on/off these three training strategies easily.
- Verify and explore the effects of ImageNet pre-training based on our discovery (as stated in Section 1.4). Carry out experiments to see whether its regularization effect can be achieved by weight decay and multi-task learning. Specifically, we use semantic segmentation task and human part segmentation task for the multi-task learning.
- Train our models on the High Performance Computing (HPC) platform, which belongs to the department of computer science at UCL.
- Evaluate, visualize, and analyze the results. Make an elaborate summary and a guidance for future deep learning model training.

1.4 Summary of Main Results

This section gives an overview of our experimental results, please see Chapter 6 for a discussion with more details.

In our first series of experiments, we explore model's performance with/without ImageNet pre-training. As shown in Table 1.1, we discovered that the training performance of non pre-trained model is much better than the pre-trained one. This empirically proves that with our model (FCN with ResNet-18) and optimization settings, we do not need ImageNet pre-training to optimize the loss function. The pre-training only provides a regularization effect in our case. To our best knowledge, this phenomenon has not been found in other works before. We also tried VGG-16 and AlexNet with the same experimental

settings to confirm our hypothesis, VGG-16 shows a similar phenomenon to ResNet-18, but AlexNet does not have this. Although VGG-16 also achieves a better training performance, it is not as good as ResNet-18. If we use pre-trained and non pre-trained "Train mIoU" to compute a ratio, VGG-16 is $\frac{77.2\%}{73.5\%} = 1.05$, and ResNet-18 is $\frac{70.7\%}{63.6\%} = 1.11$. As a result, ResNet-18 has a better optimization scheme.

Based on this discovery, we tried to regularize the model instead of pre-training. We first applied weight decay, as shown in Table 1.2, we increase the value of weight decay step by step, and the validation performance of the non pre-trained model increases with it, reaching the top at weight decay $5e^{-3}$. But it is not as good as the results of the pre-trained model.

We also applied multi-task learning to regularize our model. Multi-task learning does improve the generalization ability for non pre-trained model, but still not as good as the pre-trained model. The result of this is shown in Section 6.2.2 in details.

	Train loss	Val loss	Train mIoU	Val mIoU
VGG-16	0.183	0.623	77.2%	37.9%
VGG-16 pre-trained	0.216	0.382	73.5%	50.4%
AlexNet	0.339	0.669	64.3%	31.1%
AlexNet pre-trained	0.307	0.521	66.9%	38.4%
ResNet-18	0.257	0.471	70.7%	41.1%
ResNet-18 pre-trained	0.344	0.392	63.6%	47.1%

Table 1.1: The results of FCN with three different backbones on the semantic segmentation task. The main purpose is to compare the performance of pre-trained and non pre-trained models, and the differences with different backbone architectures.

weight decay	0	$1e^{-4}$	$5e^{-4}$	$1e^{-3}$
Train loss	**.270**/.323	**.236**/.329	**.257**/.344	**.330**/.381
Train mIoU	**69.7%**/66.0%	**72.7%**/65.2%	**70.7%**/63.6%	**66.1%**/60.6%
Val loss	.565/**.402**	.514/**.400**	.471/**.392**	.484/**.399**
Val mIoU	38.0%/**47.2%**	39.8%/**47.4%**	41.1%/**47.1%**	39.7%/**45.7%**

Table 1.2: The performance of pre-trained and non pre-trained models with the changing of weight decay. In each cell of the table, the result is shown in the format of "non pre-trained/pre-trained". The better result is written in bold. For the losses, the zero before decimal point is omitted for clearance.

Chapter 2

Literature Review

In this chapter, we aim to give an overview of the papers and approaches that are related to this project in a basic-to-complex order. The main focus of this literature review is on Convolutional Neural Networks, pre-training, and multi-task learning. They are pre-requisites to verify our hypothesis and can help us to give a clear explanation of our experiments design and result analysis in the following sections.

This review is divided into three sub-sections. Firstly, we introduce the neural network and important concepts of it. These basic concepts are very important for understanding latter algorithms. As stated in [1], deep learning itself is a re-branding of the neural network. Secondly, we discuss some useful techniques for improving the performance of deep learning models. These techniques are also used in our models. Finally, we will make a review of papers that are related to pre-training and multi-task learning. Our hypothesis is based on this knowledge, and the experiments learn from them with appropriate modifications.

2.1 Deep Learning

2.1.1 History

Deep Learning is an evolution and also a re-branding of the Artificial Neural Network (ANN), which has a very long history. In 1943, Warren McCulloch and Walter Pitts created a simple computational model named "threshold logic unit", trying to mimic how real neurons work [32]. In 1958, Rosenblatt created the Multi-Layer Perceptron for pattern recognition [37]. It drew a lot of attention at first, but after a few years, it was proven

not being able to learn simple functions such as exclusive-or (XOR) as it is linear. In 1986, Rumelhart, Hinton and Williams proposed an algorithm to train a multi-layer neural network efficiently, which is called back-propagation (BP) [38]. This algorithm plays an important role in the development of neural network and deep learning, because it is a general purpose method to train a network end-to-end. BP is an application of the chain rule for derivatives, the derivative of an input module's objective can be calculated by taking the gradient backwards with respect to the output of that module. This process can be repeated for all modules in an ANN. Therefore, an ANN can be trained by doing forward and backward propagation unlimited times until convergence.

ANN experienced its first winter in 1990s. There are three main reasons for that. Firstly, at that time, training an ANN with more than two hidden layers is not achievable because of the vanishing gradients problem. As a result, it was impossible to have a deep/complex ANN model. Secondly, the computing power in 1990s are not as powerful as today's. Training process was extremely slow and not applicable to large scale problems. Thirdly, Support Vector Machine (SVM) became very popular and powerful in 1990s, the publication of non-linear SVM classifier and soft margin enabled SVM to outperform neural networks.

In 2006, Geoffrey Hinton announced a new initialization method for training a deep neural network named "unsupervised pre-training". This pre-training method helps to train a much deeper model than before (2 layers). More about this algorithm will be discussed in section 2.3.1. The real breakthrough of deep learning came in 2012, a deep Convolutional Neural Network called AlexNet won the ILSVRC2012 and reduced state-of-the-art error rate from 26.2% to 15.3% [25]. There are two main reasons for its success, the use of GPUs and more advanced training techniques (this will be introduced in section 2.2). After AlexNet, deep learning is truly boosted and attracts a huge amount of attention from the world. A lot of more advanced deep learning architectures and training methods were invented, more and more labs and companies are getting into this exciting research area. Figure 2.1 shows the trend of deep learning in recent years. Nowadays, deep learning is applied to many areas in research and in our life, such as Computer Vision, Speech Recognition, Drug Detection, Autonomous Cars, etc. It will keep evolving in the future.

2.1.2 Convolutional Neural Networks

According to [26], convolutional neural networks (ConvNets) are a class of deep models designed to process data that come in the form of multiple arrays, such as RGB images,

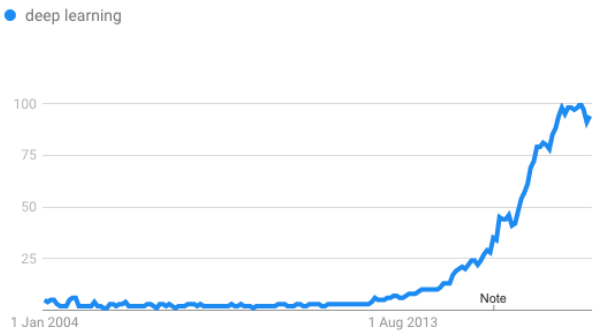


Figure 2.1: Worldwide trending of deep learning according to Google Trends.

sequences of texts/signals, audio spectrograms, etc. Especially. ConvNets are very good at processing visual data and widely used in Computer Vision tasks. In this subsection, we discuss ConvNets in its key concepts: local connection, parameter sharing, pooling layer, and model depth.

Local Connection

For image data or some similar types of data, In a convolutional layer, features are usually groups of locally correlated values, they provide the most useful information of the input data. ConvNets make use of this property to improve the computational efficiency. As shown in figure 2.2 as an example, each filter in a convolutional layer is only connected to a local area of the data at a time when convolving. Filters with different sizes can be applied to achieve local connection with different scales, 3×3, 5×5, and 7×7 are typical sizes.

Parameter Sharing

Parameter sharing is also an important architecture design. It makes use of a data property named translation in-variance, which means if a feature is detected at some location in an image, it can also be detected in any other location. Therefore, it is feasible and reasonable to use the same set of parameters to convolve the data. Although parameters are shared on the x and y axes of an image, we do not share parameters along the depth (z) axis, as different kinds of features can exist on different channels. These two schemes can largely reduce the number of parameters. For example, for a RGB input image with size $64 \times 64 \times 3$ and a $3 \times 3 \times 3$ filter, there are $3 \times 3 \times 3 = 27$ parameters with parameter sharing, and $3 \times 3 \times 3 \times (64-2) \times (64-2) = 103788$ parameters without parameter sharing. Obviously,

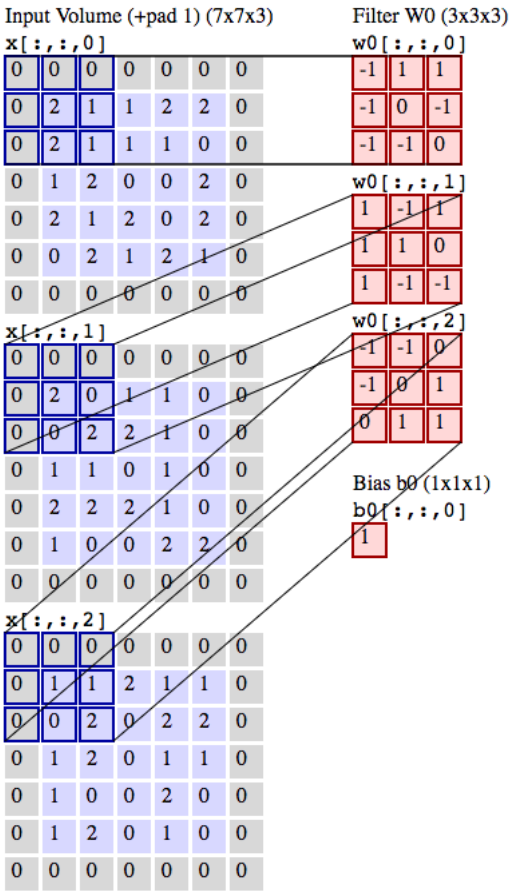


Figure 2.2: A $3 \times 3 \times 3$ filter is convolving the $5 \times 5 \times 3$ input data with *padding* $= 1$. The filter is locally connected to the data at each step. Figure source: [27].

the latter one is a waste of parameters and computing power. Furthermore, a huge number of parameters can cause the over-fitting problem.

To better understand local connection and parameter sharing, there is a vivid animation here: https://github.com/vdumoulin/conv_arithmetic, which is created by [5].

Pooling

Pooling layers are very common in ConvNets. They are used to downsample the data to reduce the number of parameters and remove redundant representations. Two types of pooling layers are usually used: max-pooling and average-pooling. In most cases, max-pooling can achieve a better result, and average-pooling is usually used in the last layer of a ConvNet for image classification task. An example of max-pooling is shown in figure 2.3. Although pooling works well in ConvNets, Geoffrey Hinton said the pooling operation

is a mistake because valuable information is lost. Researchers are finding replacements of pooling operation, for example the Capsule Networks [40], but there is no mature solution yet that is found to be better than pooling. Pooling is still necessary in many tasks.

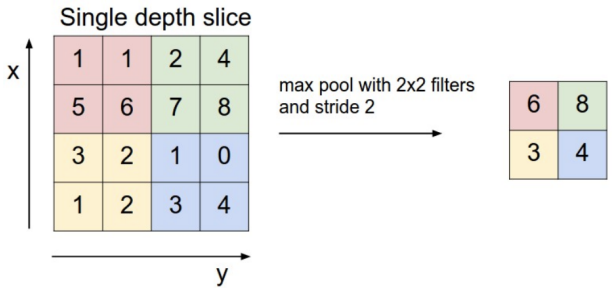


Figure 2.3: Example of the max-pooling operation. Figure source: [27].

Model Depth

Model depth is an important design factor of CNN architectures. As shown in the leading results [25, 45, 43, 44, 16] of the ImageNet [39] challenge in the past few years, all of them are exploiting very deep models. Figure 2.4 visually this trend and the benefits of the growing depth. In addition, there are many other Computer Vision tasks benefit from deep models, e.g. [29, 8]. In despite of all other factors, the depth of a model should have a positive effect on its performance. But it is difficult to train a very deep model. In other words, when the model is growing deeper, it is getting harder and harder to fully utilize the model's real capacity. For example, in [16], He et al. empirically proved that if we naively stack modules to make a model deeper, the model will experience a degradation issue, which is shown in figure 2.5, both training and validation loss will increase. Therefore, training a deep model requires crafted model structure and wise strategies.

Deep supervision is one of the most effective method to solve the degradation issue and train a very deep model. The deep residual neural network (ResNet) [16] uses shortcut connections to achieve deep supervision and enhance gradient flow. This is the first time that we can train a network with more than one hundred layers successfully. In addition, the DiracNet achieves a similar effect by weight pasteurization. Furthermore, the DenseNet [19] strengthens deep supervision by connecting all convolutional layers in a block. Their purposes are same, to truly make use of a deeper model's capacity.

How to train a very deep model is still an active research topic, it needs to be addressed theoretically and empirically.

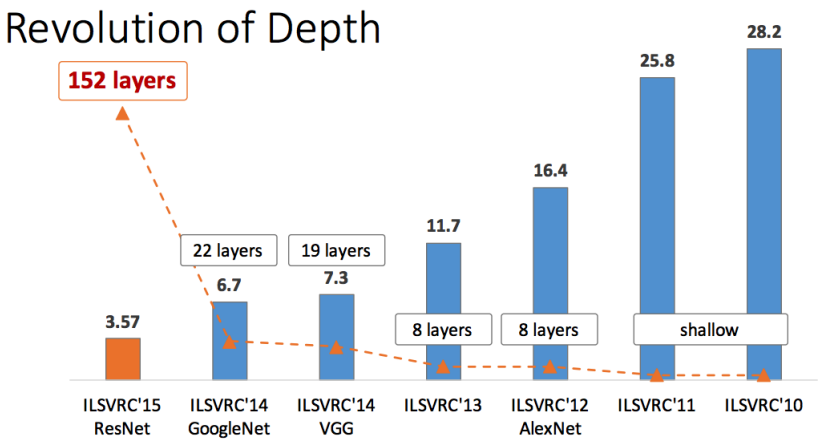


Figure 2.4: The depth of the best models of ImageNet challenge from 2010 to 2015. The model's depth is getting deeper and deeper, and the growth of depth can actually bring improvements in terms of performance. Figure source: https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf.

2.1.3 Cross-entropy Loss

Cross-entropy is also called log loss, it is widely used as an objective function for categorical predictions. It is used to measure the difference or "distance" between the ground truth labels and model predictions. For example, if we have a prediction task with $N = 10$ classes, the model's prediction P will be a vector of 10 float numbers represent the probability of each class, and the ground truth label Y is a one-hot vector with the correct class to be 1, other classes to be 0. In this case, the cross-entropy loss L can be calculated as:

$$L = -\sum_{n=1}^{N} Y_n log(P_n) \tag{2.1}$$

In Computer Vision, many tasks are related to categorical prediction, e.g. image classification, dense prediction like semantic segmentation, etc.

2.2 Weight Initialization

Weight initialization plays a crucial role in deep architecture training. The purpose of finding a good initialization is to make the flow of gradients more fluent and converge more easily. Before 2010, weights for neural networks were initialized using Gaussian distribution with zero mean and small standard deviation. As shown in [27], this is not

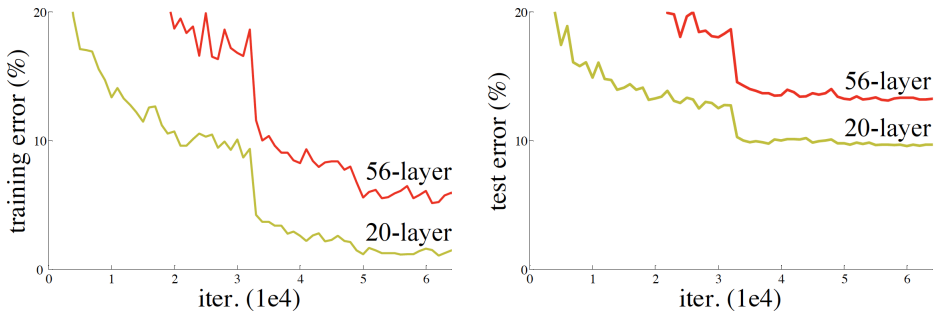


Figure 2.5: The figure shows the degradation problem of deep learning models. As the depth grows from 20 to 56, both training and test errors increase (on CIFAR-10 dataset, similar problem exists on other datasets). Therefore, this problem is not caused by over-fitting. Figure source: [16].

good enough for deep models because almost all neurons (except those in the first layer) are saturated easily and the gradients will be zero. As a result, no learning will happen in these neurons. Currently, there are two more advanced and mainly used initialization methods in deep learning: Xavier initialization [9] and He initialization [15]. Equation 2.2 shows how Xavier draws initial weights, where U represents uniform distribution, N_{in} and N_{out} represent the number of neurons in the input layer and output layer. As stated in the paper, He initialization uses Gaussian distribution, which is shown in equation 2.3. The difference of them is that Xavier is better when using activation functions like sigmoid and tanh because it assumes linear activation, He initialization is better when using the ReLU [33] activation.

$$W \sim U(-\frac{\sqrt{6}}{\sqrt{N_{in}+N_{out}}}, \frac{\sqrt{6}}{\sqrt{N_{in}+N_{out}}}) \tag{2.2}$$

$$W \sim \mathcal{N}(0, \sqrt{\frac{2}{N_{in}}}) \tag{2.3}$$

Pre-training can also be thought as a way to initialize the parameters of a neural network. More of this is discussed in section 2.3.

2.3 Model Pre-training

In deep learning, the pre-training technique is mainly used for two purposes at different stages. In the early days of deep learning, pre-training is used for helping to train a deep network successfully. In recent years, as the "training very deep neural networks" problem has been mostly solved by more advanced architecture designs [16, 19] and training tricks

(e.g. [33, 20]), pre-training is then largely used in transfer learning. In this section, these two parts of pre-training is reviewed, and exploring the effects of pre-training is one of the key purpose of this project.

2.3.1 Pre-training for Training Deep Neural Networks

As briefly discussed in section 2.1.1, before 2006, it was difficult to train a neural network with more than two hidden layers. It is caused by the gradient vanishing problem. At each hidden layer of a neural network, an activation function is applied for adding non-linear transformation. In the past, the activation is usually the sigmoid or tanh function, which are both "squashing" functions. As visualized in figure 2.6, no matter what value the input is, it will be mapped to range $(0, 1)$ by sigmoid or $(-1, 1)$ by tanh. If this type of activation is applied many times, even there is a large change of the parameter values in the first layer of the neural network, there is almost no change in a deeper layer's output. Therefore, the gradients vanish in deeper layers and there will be no learning happen in the neurons of these layers.

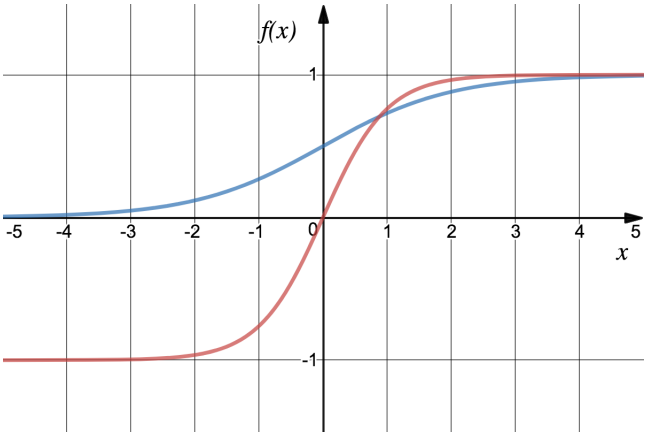


Figure 2.6: The sigmoid (blue) and tanh (red) activation functions. They both squash the input to a value in a small range.

In 2006, Geoffrey Hinton et al. created a new weight initialization method called "unsupervised pre-training" for training deep belief nets [17]. It is a greedy learning algorithm that initializes the network layer by layer. For example, to pre-train a network with 3 hidden layers (figure 2.7), we need to train an autoencoder for each layer. In addition, when pre-training one layer, all its previous layers' weights must be fixed, and this is why this algorithm is greedy. The pre-training process of the first two hidden layers is illustrated in

figure 2.8, and all hidden weights are initialized in this method. For the output layer, it is pre-trained to predict the one-hot ground truth vector $[y_1, y_2, y_3, y_4]$ rather than recovering its last previous layer. After this pre-training process, the network can be fine-tuned integrally by supervised learning on the training dataset.

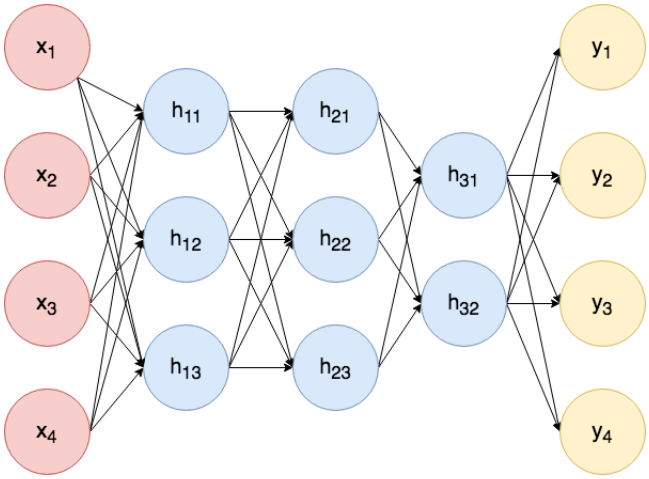
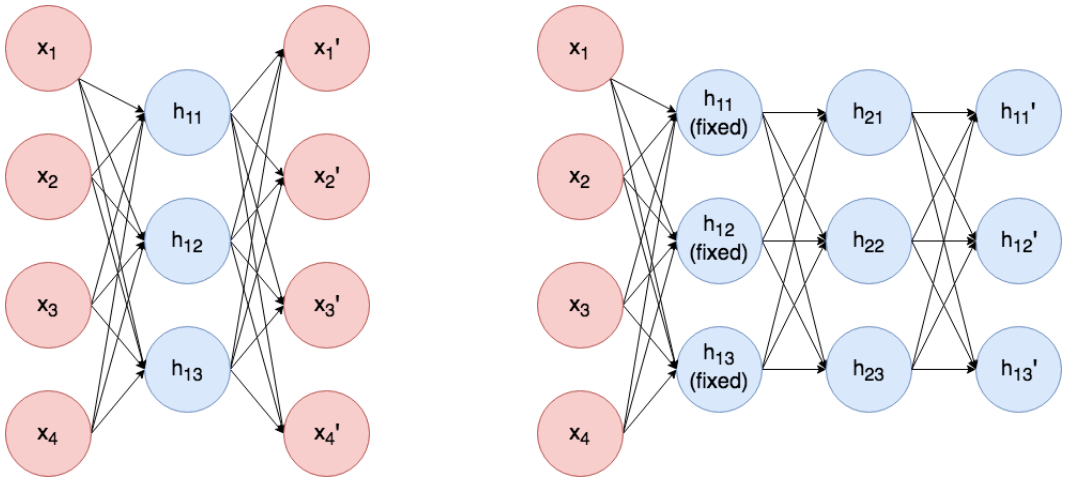


Figure 2.7: A neural network with 3 hidden layers. The output is a 1×4 one-hot vector.

The are two main reasons for the positive effects of "unsupervised pre-training". Firstly, it has a regularizing effect to the subsequent supervised fine-tuning by introducing a prior to it [6]. Compared to random initialization, "unsupervised pre-training" makes the neural network function already very non-linear, and the cost function is also more complicated. Therefore, the parameter space is restricted in a small area and easier to find a good local optimum. This increase the stability of the learning afterwards. Furthermore, as the network is pre-trained greedily layer by layer, each layer learns features of the input data X on a different scale. The knowledge of X is leveraged to all hidden layers, which is much better ordered way than random initialization. Secondly, neural networks use stochastic gradient descent (SGD) or its variants to optimize the parameters. The parameter space consists of many basins, plateaus, and plains. During the optimization procedure, SGD draws a trajectory in the parameter space until convergence. Figure 2.9 shows an example of trajectory of SGD with momentum. The optimization trajectory is easier to be changed from one basin to another in the early stage than later stage of training. Unsupervised pre-training is this kind of perturbation that happens very early to trap parameters in regions that are less difficult to find good local optimum. As a result, with proper hyper-parameter settings, it is very unlikely that the supervised fine-tuning later can move the trajectory to another bad basin. "Unsupervised pre-training" is the first technique that helps to train



(a) pre-training of the first hidden layer (b) pre-training of the second hidden layer

Figure 2.8: This is an example of unsupervised pre-training. In figure (a), the first hidden layer is pre-trained by encoding the input x and trying to recover it by decoding. This is a one-layer autoencoder, which is a generative model. It is assumed to have a good understanding of the underlying structures in the images. Similarly, in figure (b), the second hidden layer is pre-trained to recover the first hidden layer, with the parameters of the first layer fixed.

a network with more than two hidden layers. There are more advanced techniques came out, such as ReLU activation [33], dropout [18], batch formalization [20], etc. As a result, "unsupervised pre-training" is not used for weight initialization anymore.

The famous VGG-Net [43] also uses pre-training to overcome the problem of training very deep networks with 16 and 19 layers. It divides the training procedure into two parts. Firstly, a shallow network with 8 convolutional layers and 3 fully-connected layers is trained. Then, its first four convolutional layers and all fully-connected layers are applied for the initialization of deeper networks.

2.3.2 Transfer Learning with Pre-trained Networks

Transfer learning is an active research area in machine learning. In recent years, it increasingly draws a lot of attention from academia and industry. In this subsection, we firstly make a definition and introduction of transfer learning. Then, we review the main transfer learning methods in Computer Vision and their corresponding applications.

Currently, machine learning has achieved or even surpassed the human level performance in many specific supervised learning tasks. Many of the models require a huge

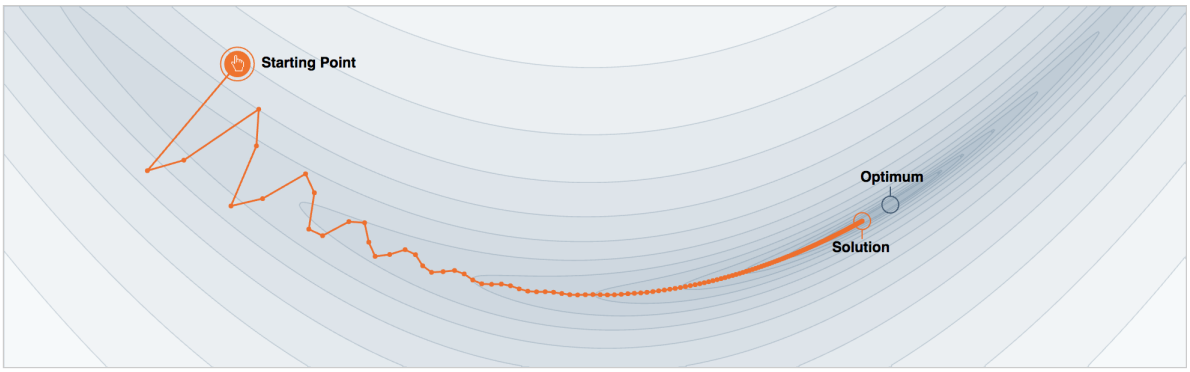


Figure 2.9: An example of trajectory of SGD with momentum. The trajectory zigzags in the basin and finally hits a local optimum that is close to the true optimum. Figure source: [10].

amount of labelled data to achieve their results. However, this is not possible for tasks with only little amount of training data. In addition, collecting and labelling data is very expensive and time consuming. Transfer learning can help to mitigate this problem. It aims to use the knowledge and information learned in one task to improve the performance of a similar task. Sinno Jialin Pan and Qiang Yang [34] give a very clear mathematical definition of the transfer learning: Given a source domain D_S and its corresponding task T_S, and a target domain D_T and its corresponding task T_T, the aim of transfer learning is to use the knowledge in D_S and T_S to enhance the performance of the target predictive function $f_T(\cdot)$ in D_T, where $D_S \neq D_T$ and $T_S \neq T_T$.

In Computer Vision, transfer learning using pre-trained networks/features is widely deployed and has an unparalleled positive effect compared to other machine learning disciplines, e.g. natural language processing (NLP), reinforcement learning (RL), etc. One of the main reason of its success is the ImageNet dataset. ImageNet holds an annual challenge called "ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" [39], which provides 1.2 million labelled training data with 1000 categories for object localization and classification. It has boosted many excellent deep learning models for Computer Vision, such as AlexNet, VGG-Net, GoogLeNet, ResNet, DenseNet, etc. For transfer learning, there are basically two steps to do. Firstly, one of these classification models is usually pre-trained on the ImageNet classification dataset. Secondly, this pre-trained network (except the final fully-connected layers) is adapted as the backbone of another task's model. Then, additional task-specific layers are added to the backbone to make up a new end-to-end deep learning model. Finally, this new model is fine-tuned on its target task's training

dataset. In practice, during the fine-tuning, the learning rate of the weights in pre-trained backbone is usually much smaller than the new layers to prevent losing the information it has learned.

There are many outstanding models using ImageNet pre-trained networks that have achieved previous state-of-the-art results in the past few years. For example, in 2014, Jonathan Long et al. created the fully convolutional networks (FCNs) [29] for semantic segmentation task. This is the first time that semantic segmentation task is solved by an end-to-end model. They experimented with different backbones: AlexNet, VGG-Net (16-layer version), and GoogLeNet. Without changing anything else, the FCN with VGG-Net has a 23.75% improvement compared to the FCN with AlexNet version.

Why does pre-trained networks work so well on other tasks? The main reason is that we can capture the information of image features with different scales using a deep CNN. With 1.2 million training images, the network not only learns how to classify 1000 categories, but also captures general information about image structures. As shown in FCN, the backbone can be replaced by one another, and different pre-trained backbone can boost the performance of FCN by different degrees. This indirectly proves that pre-trained CNN networks do learn general information of visual data. Figure 2.10 shows an example of high-level features learned by the first convolutional layer of AlexNet. These features can be edges and corners in images.

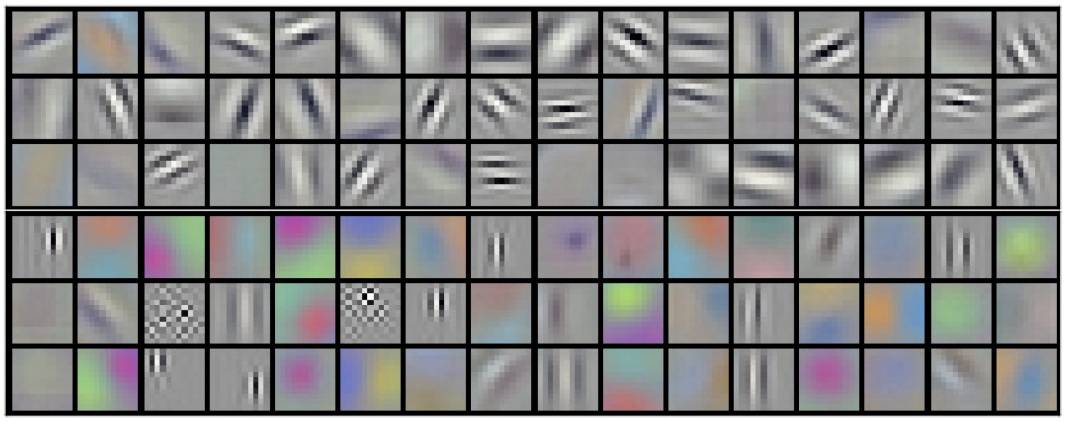


Figure 2.10: 96 convolutional features learned by the first convolutional layer of AlexNet. In CNNs, lower convolutional layers captures low-level features, e.g. edges/corners in this figure, whereas higher convolutional layers capture more complex features, which are details in the images. Figure source: [25].

2.4 Multi-task Learning (MTL)

2.4.1 What is multi-task learning?

Multi-task learning (MTL) is a learning method that multiple tasks are learned jointly while sharing a single representation [2]. In other words, MTL trains several different tasks on a single model, and this model can be used to perform all the tasks it is trained with. These tasks can be trained in parallel, or in a specific order, or using a well-designed strategy. There are two main MTL approaches, either training auxiliary tasks intermediately to improve the performance of target task(s), or directly learning multiple target tasks together.

2.4.2 Benefits of MTL

1. With MTL, we can build a single system that can process different tasks we need, instead of having many separate systems. Currently, there are many works achieve state-of-the-art results in Computer Vision tasks, e.g. object detection [8, 35, 14], image classification [25, 43, 44, 16, 19], semantic segmentation [29, 3], and many others. However, each of them provides a task-specific CNN model, and potentially they are very different architectures. There are three disadvantages of using separate models compared using one unified model. Firstly, multiple models have to be trained and evaluated for multiple tasks. In addition, if we want to perform more than one tasks on a image, we have to feed it to each model separately. This is not very time efficient. Secondly, different architectures may accept input data of different sizes and dimensions. With separate models, we need to do data pre-processing many times. Thirdly, one important property of modern general purpose AI system is evolving with time. The evolution of multiple models is much less convenient than a unified system. In my point of view, having a unified single system is a must for developing general AI and many AI applications, such as robots and virtual AI assistants. It is simpler and faster.

2. As proven in [22, 30, 28, 12], learning one task can benefit from jointly learning other related tasks. To explain the intuition behind this phenomenon, imagine there is a student with no prior knowledge, if we teach him two related disciplines jointly (e.g. probability theory and statistics), he would have a better understanding of both disciplines than only learning one of them. As shown in [28, 12], auxiliary tasks

can provide additional supervision which are partially complementary. In addition, if different tasks use different datasets, we can get more training data and therefore more related information. Furthermore, MTL also has a regularization effect to the model, because multiple tasks introduce more noise and the neurons have to take care of more than one tasks. As a result, a MTL model can have a better generalization ability and is less possible to over-fit.

2.4.3 Difficulties of MTL

One of the difficulties of MTL is catastrophic forgetting, i.e. when a deep neural network is trained for task A, it is difficult to train it again for a new task B and not forgetting A. Ronald Kemker et al. [23] analyze five methods to mitigate this problem: regularization, ensembling, rehearsal, dual-memory, and sparse-coding. As analyzing the difficulties of MTL is not quite relevant to this project, we do not get into details.

Chapter 3

Datasets

In this chapter, we introduce the datasets used in our project. As mentioned in Section 1.3, two Computer Vision tasks are experimented in this project: semantic segmentation and human part segmentation. We use four datasets in total. In section 3.1, we give descriptions of each dataset and we talk about the reasons for using them. In section 3.2, we define three abbreviations for writing the following chapters clearly. A summary of these four datasets are shown in Table 3.1.

3.1 Descriptions of Datasets

3.1.1 PASCAL VOC 2012 Dataset

The PASCAL Visual Object Classes challenge started from 2005 and finished in 2012. It holds five challenges in the Computer Vision field: object detection, object classification, semantic segmentation, person layout, and action classification. It is a widely used benchmark dataset, and one of its official paper [7] has been cited by 5553 times according to Google Scholar (at the time we write this thesis). In this project, we use the training images with semantic segmentation annotations in PASCAL VOC 2012.

3.1.2 Semantic Boundaries Dataset

The Semantic Boundaries Dataset (SBD) [13] contains training data for semantic segmentation and semantic boundaries tasks. Although it uses the same set of images as the PASCAL VOC 2011 dataset, it has much more images with semantic segmentation

Dataset	Task	#Training Images	#Val Images	Total
PASCAL VOC 2012	SEMSEG	1464	1449	2913
SBD	SEMSEG	8498	2857	11355
LIP	PARTSEG	28280	5000	33280
PASCAL-Person-Part	SEMSEG, PARTSEG	1716	1817	3533

Table 3.1: A summary of four datasets used in this project. Here, SEGSEM denotes semantic segmantation, PARTSEG denotes part segmentation, and symbol # is "the number of".

annotations. We use it together with the PASCAL VOC 2012 dataset for the semantic segmentation task, for achieving much more stable learning curves and easier model convergence.

3.1.3 Look Into Person Dataset

Ke Gong et al. created the Look Into Person (LIP) [11], which is a large scale dataset for understanding person semantically. It has advantages of good scalability, diversity and difficulty. We use the multi-person portion of it for the human part segmentation task. It provides enough training data for our first series of experiments.

3.1.4 PASCAL-Person-Part Dataset

The PASCAL-Person-Part dataset is a subset of PASCAL-Part dataset [4], which is a collection of additional annotations for PASCAL VOC 2010 dataset. It provides segmentation masks for each body part of the object in images, which exceeds the original PASCAL VOC object detection task. It is used for our second series of experiments for two advantages. Firstly, small training data is not a problem in the second series. As we do not have much computing power (only 1 GPU is available), this small dataset can shorten our experiment time. Secondly, if it is combined with the SBD dataset, each image has both ground truth for semantic segmentation and human part segmentation, which is very convenient for our multi-task learning experiments. Here is a link for this dataset: http://liangchiehchen.com/data/pascal_person_part.zip.

3.2 Dataset Abbreviations

In this project, we use different combinations of the four datasets described in section 3.1 for different tasks. For the purpose of writing the following chapters clearly, we define three abbreviations in Table 3.2:

Abbreviation	Train	Val	Test
D_1	VOC_2012_Train + SBD_Train + SBD_Val (removed overlap in VOC_2012_Val)	VOC_2012_Val	VOC_2012_Test
D_2	LIP_Train (multi-person)	LIP_Val (multi-person)	LIP_Test (multi-person)
D_3	Intersection of PASCAL-Person-Part_Train and SBD_Train	Intersection of PASCAL-Person-Part_Val and SBD_Val	VOC_2012_Test

Table 3.2: D_1, D_2, D_3 denote three combinations of datasets used in this project. Especially, "_Train", "_Val", "_Test" denote the training, validation, testing sets in the dataset respectively.

Chapter 4

Model Design

As discussed in Section 1.1 $\sim$ 1.3, we achieve the goal of this project by making a few hypotheses and verify them through elaborate experiments. The purpose is not achieving a state-of-the-art result or making improvements to some existing models. Therefore, in this project, two existing models are used: Stacked U-Nets [41] and Fully Convolutional Neural Networks [29]. We make appropriate modifications to them for conducting our experiments. In this chapter, we firstly talk about the models that have been used in this project. Next, we discuss how they are modified and adapted for our experiments.

4.1 Stacked U-Nets

The Stacked U-Nets (SUNets) is inspired by the U-Net [36], which is a CNN model created for biomedical image segmentation. The U-Net architecture is shown in figure 4.1. Although U-Nets achieve excellent results in biomedical image segmentation task, it does not perform well when processing images in life. Compared to the original U-Nets, the SUNets are much deeper architectures by stacking many U-Net blocks together.

The SUNets expend the power of plain U-Net and have three main advantages. Firstly, it can pass information of input data and extract features while preserving resolution. This is very important for segmentation tasks, which require the result images to be both accurate and precise. Secondly, it can effectively encode semantic information. This is attributed to its architecture design, all features from high-level to low-level can get many layers of non-linear computation by passing through its stacked encoder-decoder blocks. As a contrast, traditional bottom-up architectures can not make good use of the high-level features. Thirdly, SUNets have much fewer parameters, which is a save of computational

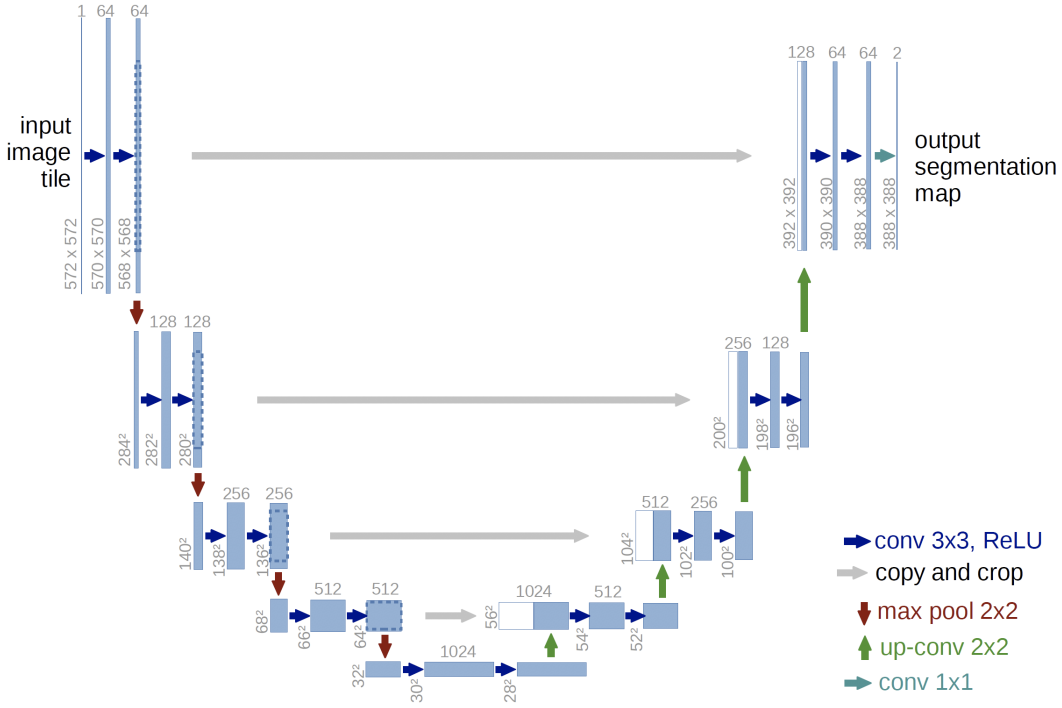


Figure 4.1: U-Net architecture. The arrows represent different operations. Each blue box is a stack of feature maps, and the number of them is shown on the top of the box. Figure source: [36].

time and resources. As discussed in Section 2.3.2, many modern deep architectures for the semantic segmentation task are built upon pre-trained networks. This is not flexible, and the pre-trained network itself has already contained a large number of parameters. SUNets do not have this issue as they are designed and trained from scratch. As stated in its original paper, the smallest SUNet model surpass the ResNet-101 by 4.5% mIoU on the PASCAL VOC 2012 semantic segmentation task, while containing about 7 times fewer parameters.

There are three architecture settings of SUNets with increasing complexity in the paper, with 6.9M, 24.6M, and 37.7M parameters respectively. In this project, we choose to use the smallest one because of short project duration and computing power limitation. More details will be discussed in Section 4.3.1.

4.2 Fully Convolutional Neural Networks

The Fully Convolutional Neural Networks (FCNs) were created by Jonathan Long et al. [29] in 2015. It is the first end-to-end deep architecture for semantic segmentation task and achieves state-of-the-art results at that year. As also mentioned in Section 2.3.2, FCNs use pre-trained networks as their backbone. In addition, if the pre-trained networks contain fully connected layers at the end, these layers will be replaced with convolutional layers of equal number of parameters. This is necessary for upsampling and generating dense predictions in later steps. Upsampling is done by using transposed convolutional layer, which is learnable by back propagation. This in-network upsampling is very efficient for dense predicting. Bilinear interpolation is used as well for adjusting the output size to be the same as input size. Furthermore, to improve the resolution, as shown in figure 4.2, they fuse outputs at multiple levels. Finally, to generate dense predictions, the output of the model has the same size as the input, and the depth is equal to the number of categories. For example, if we use the PASCAL VOC 2012 dataset with 21 categories, feeding an input tensor with size $batch \times 3 \times width \times height$, the output would be of size $batch \times 21 \times width \times height$, i.e. there is a classification prediction for the 21 categories at every pixel.

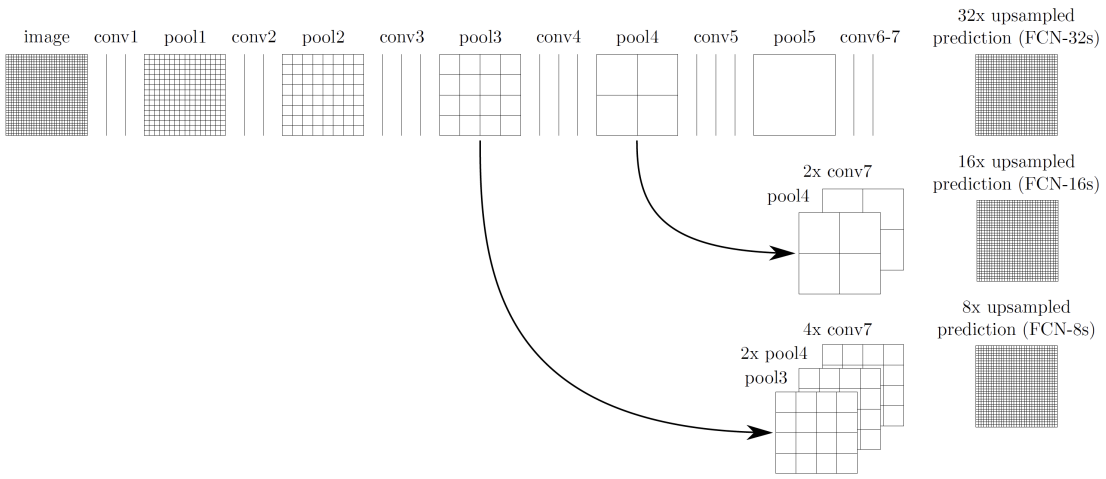


Figure 4.2: Pooling layers (or downsampling) decrease resolution of data in a CNN model. By combining the final coarser output with finer outputs from previous layers (processed by less downsampling operations), we can have a dense prediction map will higher resolution. Figure source: [29].

The FCNs are used in our first series of experiments. The main reason is that the model complexity of a FCN can be easily changed by alternating the backbone of it, i.e.

the pre-trained network. This is very important for our experiments because we want to verify the effects of "pre-training" and "multi-task learning" on the model's capacity, with the increase of model complexity.

4.3 Our Models

In this section, we discuss the three models used in this project. Two single-task models are discussed in Section 4.3.1, and one multi-task model is introduced in Section 4.3.2.

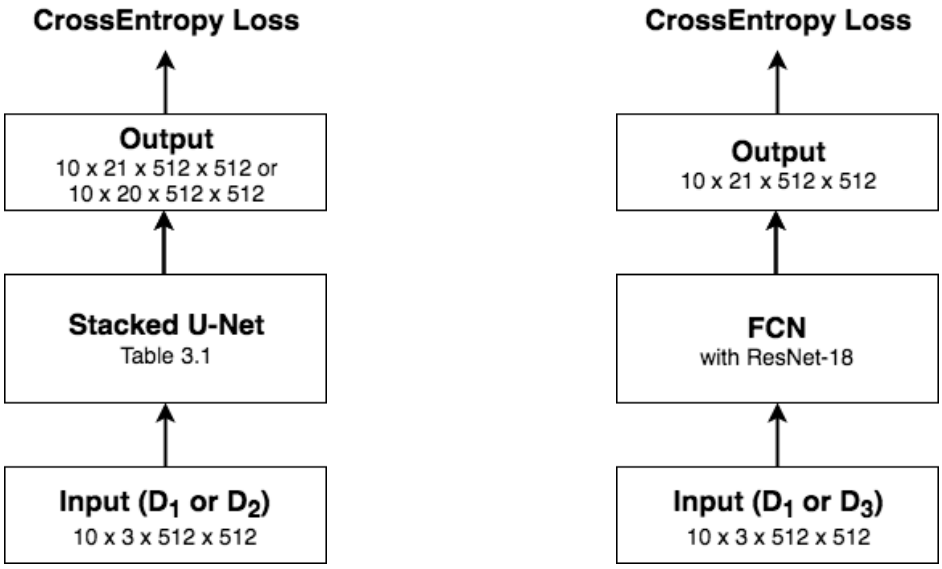
4.3.1 Single-task Model

The first sing-task model is shown in figure 4.3b. The backbone is the ResNet-18, which is a ResNet with 18 layers. The reason we did not use a deeper version of ResNet is that the architecture design and optimization method of ResNets with different depth are the same. Deeper ResNet can achieve a better performance, but this is not the key point in our experiments. In addition, a shallow ResNet takes much less training time. It is used for the first series of experiments.

The second single-task model is shown in figure 4.3a. The backbone is the Stacked U-Net, and the details of this architecture are listed in table 4.1. It is used for the second series of experiments.

4.3.2 Multi-task Model

The multi-task learning model is visualized in figure 4.4. The two tasks share the same backbone (Stacked U-Net) but using different upsampling blocks. In addition, because the input dataset is a combination of two datasets D_1 and D_2, we use asynchronous SGD updates for training this deep model. More experiment details are discussed in Section 6.1.



(a) Stacked U-Net model for single-task (b) FCN model for single-task

Figure 4.3: Two single-task models in this project. They have similar designs but different backbones. The details of the Stacked U-Net we use is shown in table 4.1.

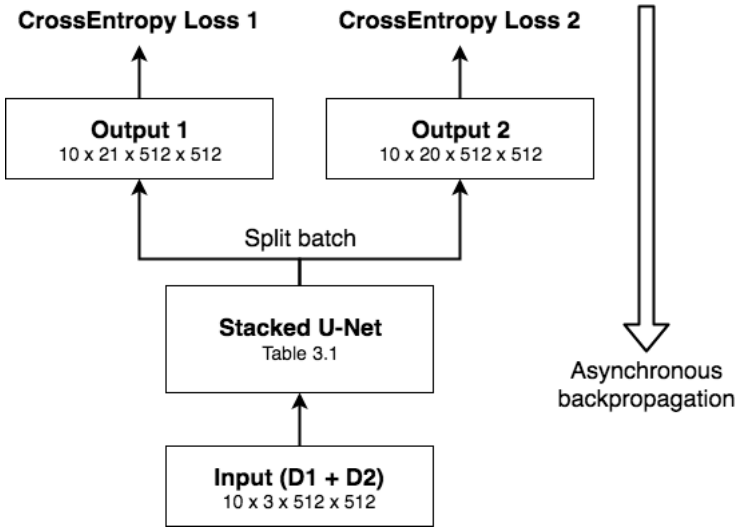


Figure 4.4: The multi-task learning model for the second series of experiments. The two tasks share the same backbone but using different upsampling blocks.

Layers	Output Size	Block Details
Convolution	256×256	7×7 conv, 64, stride 2
Residual Block	128×128	$\begin{bmatrix} 3 \times 3 \text{ conv, } 128\text{, stride } 2 \\ 3 \times 3 \text{ conv, } 128\text{, stride } 1 \end{bmatrix} \times 1$
U-Net Block (1)	128×128	$\begin{bmatrix} 1 \times 1 \text{ conv, } 64 \\ \text{U-Net}, N = 64 \\ 1 \times 1 \text{ conv, } 256 \end{bmatrix} \times 2$
Transition Layer	56×56	2×2 average pool, stride 2
U-Net Block (2)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv, } 64 \\ \text{U-Net}, N = 64 \\ 1 \times 1 \text{ conv, } 512 \end{bmatrix} \times 4$
Transition Layer	28×28	2×2 average pool, stride 2
U-Net Block (3)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv, } 64 \\ \text{U-Net}, N = 64 \\ 1 \times 1 \text{ conv, } 768 \end{bmatrix} \times 4$
Transition Layer	14×14	2×2 average pool, stride 2
U-Net Block (4)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv, } 64 \\ \text{U-Net}, N = 64 \\ 1 \times 1 \text{ conv, } 1024 \end{bmatrix} \times 4$
De-gridding Block	14×14	$\begin{bmatrix} 3 \times 3 \text{ conv, } 512\text{, dilation } 2 \\ \text{batchnorm } 512\text{, relu} \\ 3 \times 3 \text{ conv, } 512\text{, dilation } 1 \\ \text{batchnorm } 512\text{, relu} \\ 1 \times 1 \text{ conv, \#Categories} \end{bmatrix} \times 1$
Upsampling Block	512×512	bilinear interpolation
Dense Prediction		2D-CrossEntropy Loss (negative log loss + softmax)

Table 4.1: The Stacked U-Net architecture used in this project for semantic segmentation or human part segmentation. In this table, the N represents the feature map depth of each convolutional layer in U-Net. This architecture has 112 layers and 6.8M parameters.

Chapter 5

Method

In this chapter, we discuss the methods we use to verify the effects of training strategies mentioned in Section 1.3. Firstly, we introduce the metrics used for evaluating the performance of models in Section 5.1. Secondly, we discuss our methods for achieving our objectives. To define our methods clearly and formally, we use mathematical symbols to express, compare, and contrast.

5.1 Evaluation Metrics

In this project, as the tasks are both dense prediction, we evaluate using the same two measurements. The first one is objective function loss. For dense prediction tasks, cross-entropy loss is used, which is introduced in Section 2.1.3. It is an effective metric to measure the convergence and performance during a model's training process. The second one is called mean Intersection over Union, or mIoU. To define it mathematically, let n_{ca} be the number of categories, n_{ij} be the number of pixels of class i predicted to be class j, and t_i be the total number of pixels of class i, the mIoU is defined in Equation 5.1. An visualization of IoU is also shown in Figure 5.1. As dense prediction task is classifying all pixels in images, mIoU can measure the number of correctly classified pixels and misclassified pixels in one value.

$$mIoU = \frac{1}{n_{ca}} \sum_i \frac{n_{ii}}{t_i + \sum_j n_{ji} - n_{ii}} \tag{5.1}$$

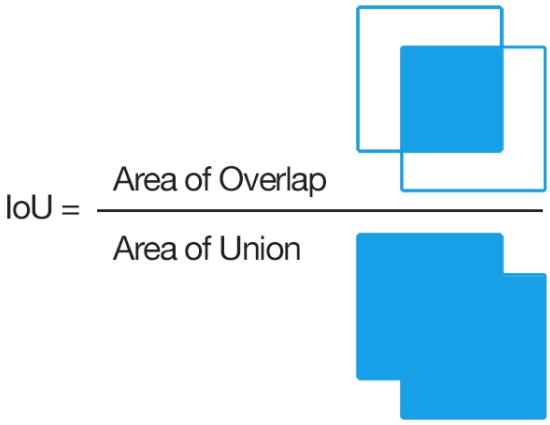


Figure 5.1: An illustration of metric intersection of union. Figure source: link.

5.2 Verification Methods

First series of experiments

The first point we want to research and verify is whether the effect of ImageNet pre-training is purely a better initialization, or a better initialization with some regularization. If the pre-trained model can achieve a better performance on both training and validation data than a trained from scratch model, then the effect of pre-training is a better initialization, and this initialization can not be easily found by optimization. If pre-trained model just achieves a better validation performance, but a worse performance on training data, then its effect is a better initialization plus regularization, and this initialization might be found by a good optimization. Therefore, to verify this, we can compare the performance of a model with and without pre-training by increasing the level of regularization step by step. The intuition is that if a trained from scratch model can achieve a lower training loss, we can add more regularization to have a better generalization performance on unseen data.

We choose to use weight decay as our regularization method. In addition, drop out can also be a good choice. Weight decay (L2 penalty) is a common regularization method, which adds an extra term in the objective function to control the size of the values of weights.

For this purpose, we carry out $E_1 \sim E_2$, the first series of experiments defined as below:

> **E**$_1$ Training the single-task model with ResNet-18 as the backbone for **semantic segmentation** task, **without** ImageNet pre-training. Using weight decay = $0, 1e^{-4}, 5e^{-4}, 1e^{-3}$ and for each value of weight decay, record the best training loss as

$L_1 \sim L_4$, the best validation loss as $L_5 \sim L_8$, the best training mIoU as $M_1 \sim M_4$, and the best validation mIoU as $M_5 \sim M_8$.

E_2 Training the single-task model with ResNet-18 as the backbone for **semantic segmentation** task, **with** ImageNet pre-training. Using weight decay $= 0, 1e^{-4}, 5e^{-4}, 1e^{-3}$ and for each value of weight decay, record the best training loss as $L_9 \sim L_{12}$, the best validation loss as $L_{13} \sim L_{16}$, the best training mIoU as $M_9 \sim M_{12}$, and the best validation mIoU as $M_{13} \sim M_{16}$.

For $E_1 \sim E_2$, we use the single-task model in Figure 4.3b. As these experiments are started in a late stage of this project, only 50% of D_1 is used for a faster speed. Results are shown and analyzed in Section 6.2.

Second series of experiments

The second point we want to research and verify in our project is that whether the multi-task learning can improve a model's generalization performance. To do that, we carry out $E_3 \sim E_8$, the second series of experiments, and control variables in them, i.e. change one factor at a time, to compare and contrast the effects.

E_3 Training the single-task model for **semantic segmentation** task, **without** ImageNet pre-training. Record validation loss as L_{17} and validation mIoU as M_{17}.

E_4 Training the single-task model for **semantic segmentation** task, **with** ImageNet pre-training. Record validation loss as L_{18} and validation mIoU as M_{18}.

E_5 Training the single-task model for **human part segmentation** task, **without** ImageNet pre-training. Record validation loss as L_{19} and validation mIoU as M_{19}.

E_6 Training the single-task model for **human part segmentation** task, **with** ImageNet pre-training. Record validation loss as L_{20} and validation mIoU as M_{20}.

E_7 Training the multi-task model jointly for **semantic segmentation and human part segmentation**, **without** ImageNet pre-training. For semantic segmentation, record validation loss as L_{21} and validation mIoU as M_{21}. For human part segmentation, record validation loss as L_{22} and validation mIoU as M_{22}.

E$_8$ Training the multi-task model jointly for **semantic segmentation and human part segmentation**, **with** ImageNet pre-training. For semantic segmentation, record validation loss as L_{23} and validation mIoU as M_{23}. For human part segmentation, record validation loss as L_{24} and validation mIoU as M_{24}.

For $E_3 \sim E_6$, we use the single-task model with Stacked U-Net, which is illustrated in Figure 4.3a. For $E_7 \sim E_8$, the multi-task model in Figure 4.4 is applied. After getting results $L_{17} \sim L_{24}$ and $M_{17} \sim M_{24}$, we can draw 8 learning curves to compare the performance of with/without ImageNet pre-training and with/without multi-task jointly learning. The results and their analyses are discussed in Section 6.2.

Chapter 6

Experiments

In this chapter, we firstly discuss the experiments using the methods stated in chapter 5. Specifically, we talk about data pre-processing, hyper-parameter settings, model optimization. Then, we show experiment results in Section 6.2, and give analyses of them. Finally, we talk about implementation details. You can view the code of this project on GitHub: github link..

6.1 Experimental Methods

Data Pre-processing. During the training process, before each minibatch of input data fed to the model, we do five steps of pre-processing for the purposes of data normalization (steps $2, 5$) and augmentation (steps $1, 3, 4$). Firstly, the size of images is re-scaled to $0.5 \sim 2.0$ times to their original size. The scaling level L is uniformly sampled from Equation 6.1. Secondly, all images in the batch are cropped to size 512×512. A unified image size is necessary for training with a batch of more than one images. Thirdly, this batch of images are flipped with a probability of 0.5. Fourthly, images are rotated with an angle A sampled from Equation 6.2. Finally, we normalize the RGB values of images by mean $= [0.485, 0.456, 0.406]$, std $= [0.229, 0.224, 0.225]$. The mean and standard deviation are calculated using the ImageNet dataset.

$$L = Uniform(0.5, 2) \tag{6.1}$$

$$A = Uniform(-10, 10) \tag{6.2}$$

Hyper-parameter settings. We have two settings of hyper-parameters, one for the first series of experiments, one for the second. In each setting, the number of epochs and the learning rate for raw (not pre-trained) models and pre-trained models are different. For pre-trained models, the learning rate of the pre-trained backbone is $\frac{1}{10}$ to the raw part to make sure the pre-trained information is not lost. The raw models and the raw part of the pre-trained model use the same learning rate, to ensure we only change one thing at a time. They are shown in Table 6.2 and 6.1.

Params / Type	Epochs	Batch Size	Learning Rate	Momentum	Weight Decay
Raw	300	10	$5e^{-3}$	$9e^{-1}$	$0, 1e^{-4}, 5e^{-4}, 1e^{-3}$
Pre-trained	100	10	$5e^{-4}$ for pre-trained part, $5e^{-3}$ for raw part	$9e^{-1}$	$0, 1e^{-4}, 5e^{-4}, 1e^{-3}$

Table 6.1: Hyper-parameters for the first series of experiments.

Params / Type	Epochs	Batch Size	Learning Rate	Momentum	Weight Decay
Raw	300	10	$2e^{-3}$	$9e^{-1}$	$5e^{-4}$
Pre-trained	90	10	$2e^{-4}$ for pre-trained part, $2e^{-3}$ for raw part	$9e^{-1}$	$5e^{-4}$

Table 6.2: Hyper-parameters for the second series of experiments.

Parameter initialization. We use He initialization [15] with normal distribution to initialize parameters of our models, except the pre-trained backbones. This method has been discussed in Section 2.2. It can help deep models with ReLU [33] activation to propagate gradients better. In addition, to make our experiments precise and robust, we set the random seed manually and fix it to zero for all experiments, to ensure a changing of our results is not caused by different random numbers.

Optimization. We use Stochastic Gradient Descent (SGD) with momentum for optimizing model parameters. Furthermore, we use the asynchronous SGD (ASGD) for multi-task

learning with more than one training datasets, e.g. experiments $E_5 \sim E_6$ stated in Section 5.2. Because in each minibatch of images, although the minibatch size is still fixed to 10, the number of images belong to a task is random. With ASGD, we can update each part of the model (common part and task-specific parts) using the same batch size, so our hyper-parameters can still be effective. You can find the pseudocode of asynchronous SGD in Table 3 of [24], created by Iasonas Kokkinos. In addition, we learn from [41, 31] to decrease the learning rate LR in every iteration by Equation 6.3.

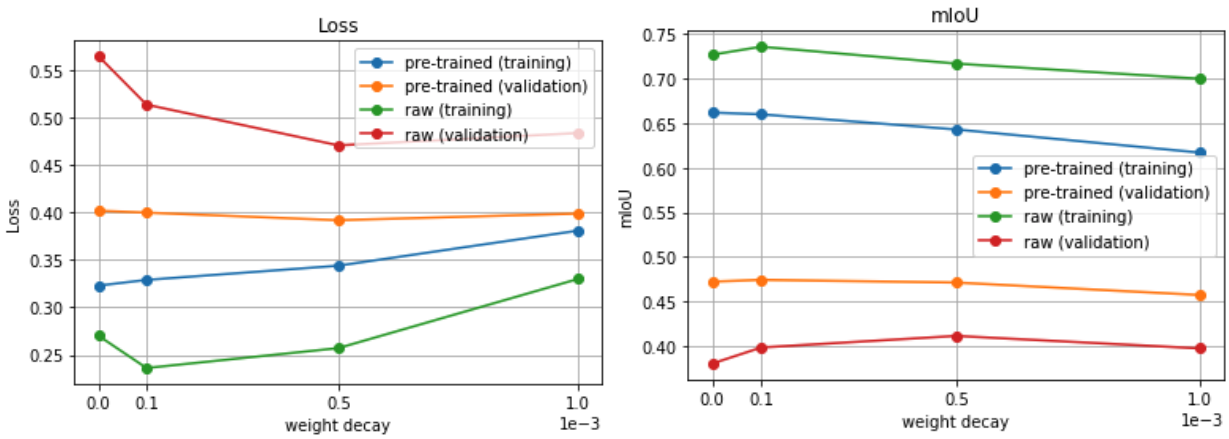
$$LR = LR \times 0.5 \times (1 + cos(\pi \frac{\#\text{iterations}}{\#\text{total\_iterations}})) \tag{6.3}$$

6.2 Results

6.2.1 Result Analysis of the First Series of Experiments

As explained in Section 1.2 and 5.2, by carrying out the first series of experiments, we want to explore the effect of pre-training, and whether its regularization effect can be achieved by other methods. The result is shown in Figure 6.1. As shown by the blue and green curves in the graphs, the model without pre-training can reach a lower training loss and a higher mIoU value. This means the pre-training not only providing a better initialization, but also a regularization effect. More importantly, this better initialization can be reached by our optimization, it only gives a quicker convergence. This also means there is a lot of space in the raw model that we can add regularization to improve its validation performance. Regularization is usually used to improve a model's generalization ability, i.e. the performance on unseen data, and it hurts the performance on training data. If we can improve the not pre-trained model's performance to the same level of pre-trained model by using regularization, then it means our optimization is good and ImageNet pre-training is not needed. Common approaches for regularization in deep learning are weight decay and dropout. We use weight decay in our experiments.

To increase the level of regularization step by step, we use four values of weight decay: $0, 1e^{-4}, 5e^{-4}, 1e^{-3}$. As shown in Figure 6.1, the model's performance is improved by using a suitable regularization, and weight decay of $5e^{-4}$ has the best effect in our experiments. Larger weight decay will hurt the optimization and it will become harder to converge. Although regularization has a positive effect, it does not solve the problem, for the raw model, its performance on training data is still better than the pre-trained model, and its performance on validation data is still worse. As a result, only with weight decay is not



(a) Training and validation losses.

(b) Training and validation mIoUs.

Figure 6.1: In (a), the graph shows the training and validation losses of the best epoch with different values of weight decay. The best epoch means the epoch with the highest mIoU value on the validation data. Figure (b) shows how mIoU changes in a similar manner.

enough to regularize it.

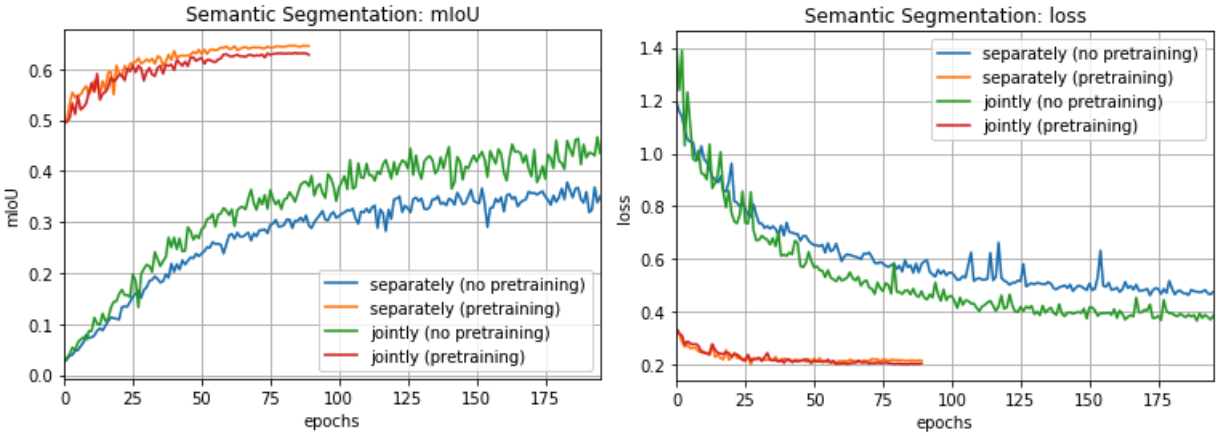
Here is a list of potential methods to further improve our optimization of the raw model:

- Use more regularization. In this case, we can fix weight decay to $5e^{-4}$, and using a suitable value of dropout probability.
- Use a larger batch size. A large batch size can make the SGD updates more stable. It can also improve the batch normalization performance. In our experiments, due to GPU memory limitation, the max batch size we can use is ten.

6.2.2 Result Analysis of the Second Series of Experiments

As stated in Section 5.2, we draw sixteen curves for the results of experiments $E_3 \sim E_8$ in Figure 6.2 and 6.3. Specifically, Figure 6.2 shows results corresponding to semantic segmentation task, and Figure 6.3 shows results corresponding to human part segmentation task. Results of both tasks show a similar pattern for the effects of "pre-training" and "multi-task learning".

As shown in the graphs, pre-training gives a much better initialization of parameters, the validation mIoU starts from 50%, compared to the raw models' starts from only 5%. For multi-task learning, it can improve raw models' performance on validation data by a large gap (as shown by the green and blue lines). But this kind of improvement does



(a) Semantic segmentation task, mIoU curves (b) Semantic segmentation task, loss curves

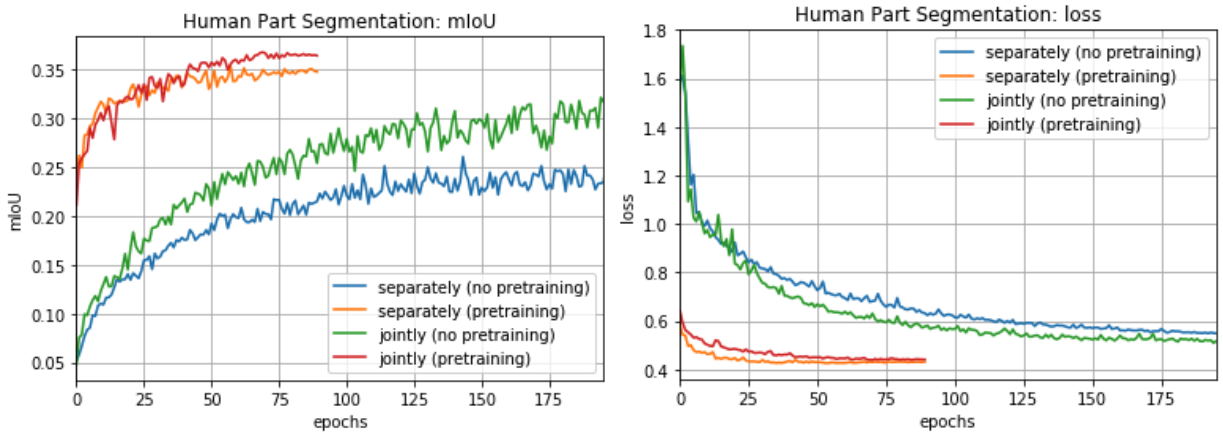
Figure 6.2: Learning curves of semantic segmentation task. Figure (a) and (b) show mIoU and loss respectively. In the figures, "separately" means no multi-task learning, "jointly" means using multi-task learning.

not beat the pre-trained models. The pre-trained networks still have a huge advantage on performance compared to multi-task learned raw models. In addition, multi-task learning has no help on pre-trained models. This might because the pre-trained models have already know a lot of general information about images, and multi-task learning does not provide additional information for one specific task, it may even be a distraction (as shown by the red and orange lines).

6.3 Implementation

6.3.1 Deep Learning Library

We choose to use the PyTorch deep learning library as our framework for programming. It has five advantages. Firstly, PyTorch is beginner friendly and easy to use with the help of many techniques, e.g. dynamic computational graphs. Secondly, the speed of PyTorch is very outstanding compared to other deep learning libraries. There is a benchmark you can see here. Thirdly, it is GPU friendly. It is extremely easy to write code running on GPU in PyTorch. Fourthly, PyTorch has a very detailed documentation and an active community. Finally, it provides a Computer Vision model zoo. This saves us a lot of time when using popular architectures.



(a) Human part segmentation task, mIoU curves (b) Human part segmentation task, loss curves

Figure 6.3: Learning curves of human part segmentation task. Figure (a) and (b) show mIoU and loss respectively. In the figures, "separately" means no multi-task learning, "jointly" means using multi-task learning.

6.3.2 Computational Resources

In recent years, started from AlexNet [25], GPUs are more and more widely utilized in machine learning field to accelerate the training process of deep learning models. In this project, we use GPUs on the High Performance Computing (HPC) platform, which is freely provided by the department of computer science. You can check it out through this link. Although it hugely increases the training speed, it has some limitations. Firstly, there is only one GPU available for one running job. As a result, we can not use data parallelism to further accelerate the training. In addition, this also limits the max minibatch size of SGD, as there is only about 10G GPU memory available. Secondly, because there are many users of HPC, sometimes our jobs have to wait in queue for a few hours before running.

Chapter 7

Discussion

In this chapter, we firstly make a conclusion for our experiments, and our contribution is shown in the conclusion. Then, we talk about some further works to do in the future.

7.1 Conclusions of Our Contribution

In this project, we explored and verified the effects of training strategies for deep architecture in Computer Vision. Specifically, we explored the necessity and effects of "pre-training on the ImageNet dataset" strategy through using other strategies: multi-task learning and regularization. Two series of experiments were carried out for this purpose, the methods and details are discussed in Section 5 and 6.

Conclusion 1. Pre-training on the ImageNet not only provides a better initialization of parameters, but also has a good regularization effect on the model. As empirically proved in our experiments, its initialization does not have an advantage of optimizing the loss function, we can achieve a even lower training loss without pre-training, while its generalization effect can not be easily achieved by manually adding regularization using weight decay. In our experiments, we found that a model can achieve a better performance on training data without pre-training. But regularization using weight decay can not improve its performance on unseen data to the same level of pre-trained model. The most possible reason is that pre-training on ImageNet provides much more training images, although they are not directly related to the target task, they provide general information and can regularize the model. Manually regularizing the model with weight decay does not provide additional

information.

Conclusion 2. Multi-task learning can significantly improve the performance on the validation data of non pre-trained models. But it does not help on pre-trained models and may even decreasing the performance by about 1% absolute value. To understand this intuitively, imagine there are three related academic subjects A, B, and C that all belong to one field. For a student with no knowledge in this field, studying A and B jointly can help the student to achieve a better understanding on both A and B. Because the student is unbiased and easy to take in new knowledge, and more subjects can provide more information of common basic knowledge. For a professor who is an expert in subject C, studying A and B jointly could not help because he/she has already studied enough basic knowledge in this field, and does not have the energy to study two subjects well at the same time. To explain this in a more formal way, pre-training provides an initialization that is very close to the local optimum, and fine-tuning on another task does not have the strength to pull it to another path. In addition, pre-trained models learned enough general information about images on ImageNet (1.2 million images), usually other tasks' dataset is much smaller and barely has an influence. Without pre-training, models are more sensitive and multiple similar datasets can provide more useful information.

7.2 Future Works

In this project, although we get exciting results, there is still space to do further experiments because we do not have enough time and computing resources. We want to list them here and complete them in the future. These supplementary experiments are potentially helpful to solidify or even modify our conclusions.

- Use more related tasks other than semantic segmentation and human part segmentation for multi-task learning to solidify our conclusion.
- Use dropout [18] together with weight decay to regularize our model in our second series of experiments. The may provide a better generalization effect.
- Use larger minibatch size for SGD. In our experiments, we used minibatch size 10 due to GPU memory limitation. If we can use multiple GPUs or a better GPU, we can

use a larger batch size to achieve more stable updates and better batch normalization performance.

Bibliography

[1] Andrew L. Beam. Deep learning 101 - part 1: History and background, Feburary 2017.

[2] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[3] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[4] Xianjie Chen, Roozbeh Mottaghi, Xiaobai Liu, Sanja Fidler, Raquel Urtasun, and Alan Yuille. Detect what you can: Detecting and representing objects using holistic models and body parts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[5] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *ArXiv e-prints*, mar 2016.

[6] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[10] Gabriel Goh. Why momentum really works. *Distill*, 2017.

[11] Ke Gong, Xiaodan Liang, Xiaohui Shen, and Liang Lin. Look into person: Self-supervised structure-sensitive learning and A new benchmark for human parsing. *CoRR*, abs/1703.05446, 2017.

[12] Riza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. *CoRR*, abs/1802.00434, 2018.

[13] Bharath Hariharan, Pablo Arbelaez, Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Semantic contours from inverse detectors. In *International Conference on Computer Vision (ICCV)*, 2011.

[14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[17] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[18] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[19] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[21] Simon Jégou, Michal Drozdzal, David Vázquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. *CoRR*, abs/1611.09326, 2016.

[22] Zhuoliang Kang, Kristen Grauman, and Fei Sha. Learning with whom to share in multi-task feature learning. In *ICML*, pages 521–528, 2011.

[23] Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *arXiv preprint arXiv:1708.02072*, 2017.

[24] Iasonas Kokkinos. Ubernet: Training a 'universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. *CoRR*, abs/1609.02132, 2016.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436 EP –, 05 2015.

[27] Fei-Fei Li, Justin Johnson, and Serena Yeung. Lecture 6: Training neural networks, part i. In *CS231n: Convolutional Neural Networks for Visual Recognition*. 2018.

[28] Yu Liu and Guanlong Zhao. Pad-net: A perception-aided single image dehazing network. *CoRR*, abs/1805.03146, 2018.

[29] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[30] Mingsheng Long and Jianmin Wang. Learning multiple tasks with deep relationship networks. *CoRR*, abs/1506.02117, 2015.

[31] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

[32] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[33] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[34] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[35] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[36] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[37] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[38] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[39] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[40] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017.

[41] Sohil Shah, Pallabi Ghosh, Larry S Davis, and Tom Goldstein. Stacked u-nets: a no-frills approach to natural image segmentation. *arXiv preprint arXiv:1804.10343*, 2018.

[42] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. DSOD: learning deeply supervised object detectors from scratch. *CoRR*, abs/1708.01241, 2017.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[45] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.