3D Densely Connected Convolutional Network for the Recognition of Human Shopping Actions

by

Dongfeng Gu

Thesis submitted in partial fulfillment of the requirements for the
Master of Computer Science degree

uOttawa

School of Electrical Engineering and Computer Science
Faculty of Engineering, University of Ottawa
Ottawa, Ontario

Abstract

In recent years, deep convolutional neural networks (CNNs) have shown remarkable results in the image domain. However, most of the neural networks in action recognition do not have very deep layer compared with the CNN in the image domain. This thesis presents a 3D Densely Connected Convolutional Network (3D-DenseNet) for action recognition that can have more than 100 layers without exhibiting performance degradation or overfitting. Our network expands Densely Connected Convolutional Networks (DenseNet) [32] to 3D-DenseNet by adding the temporal dimension to all internal convolution and pooling layers. The internal layers of our model are connected with each other in a feed-forward fashion. In each layer, the feature-maps of all preceding layers are concatenated along the last dimension and are used as inputs to all subsequent layers. We propose two different versions of 3D-DenseNets: general 3D-DenseNet and lite 3D-DenseNet. While general 3D-DenseNet has the same architecture as DenseNet, lite 3D-DenseNet adds a 3D pooling layer right after the first 3D convolution layer of general 3D-DenseNet to reduce the number of training parameters at the beginning so that we can reach a deeper network.

We test on two action datasets: the MERL shopping dataset [69] and the KTH dataset [63]. Our experiment results demonstrate that our method performs better than the state-of-the-art action recognition method on the MERL shopping dataset and achieves a competitive result on the KTH dataset.

Acknowledgements

I would like to thank my supervisor, Professor Robert Laganière, and my co-supervisor Professor Emil M. Petriu for their guidance. Their trust in my abilities and the academic latitudes they provided were extremely invaluable during my M.A.Sc study. They gave me far more than just the required supervisory feedback and instead provided me with excellent guidance throughout my research journey and the production of this work.

Likewise, I would like to thank all the members of the VIVA Lab for their thoughtful comments on my thesis. My sincere thanks go to Dr. Yong Wang for his invaluable assistance and guidance throughout my research. I also thank Muye Jiang, Yang Liu, Xile Li and Xuelu Wang for their insightful suggestions and their help with my research.

Finally, I express my profound gratitude to my parents and my wife, Lexin Li, for their unfailing support and continuous encouragement throughout my years of study and throughout the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Batch Normalization (BN)

Batch Normalization [33] is a method to reduce internal covariate shift in neural networks. In principle, the method adds an additional step between the layers, in which the output of the layer before is normalized. It has the potential to speed up the training procedure and increase the accuracy.

Convolutional Neural Network (CNN)

Convolutional Neural Network is a deep, feed-forward artificial neural network that can be used for computer vision. It usually consists of one or more convolutional layers, pooling layers, followed by several fully connected layers.

Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients is a feature descriptor used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in localized portions of an image. It takes advantage of edge orientation histograms, scale-invariant feature transform descriptors, and shape contexts, and is computed on a dense grid of uniformly spaced cells.

Long-Short Term Memory (LSTM)

Long-Short Term Memory is a special kind of RNN, which has the ability to learn long-term dependencies. It has the same chain-like structure as RNN, but different hidden unit structure. It is explicitly designed to avoid the long-term dependency problem.

Rectified Linear Unit (ReLU)

Rectified Linear Unit is an activation function defined as $f(x) = max(0, x)$. It has the benefits of reducing the likelihood of vanishing gradient and allowing a network to obtain sparse representations.

Recurrent Neural Network (RNN)

Recurrent Neural Network is a neural network that performs the same action for ev-

ery unit of a sequence, with the output being depended on the previous computations. It has a chain-like structure that can store dynamic temporal information, which is really useful for video or audio processing.

Scale-invariant Feature Transform (SIFT)

Scale-invariant Feature Transform is a feature descriptor to describe local features in images. It has the benefits of invariant to translations, rotations and scaling transformations of images and robust to moderate perspective transformations and illumination variations.

Chapter 1

Introduction

The computer vision community has worked hard on human action recognition for years. Recognizing different actions from video data is critical for many important applications, such as fight detection from surveillance video, human-robot interaction, video content analysis for online video-sharing websites, and security surveillance at home. The primary goal of action recognition is to identify human actions in a video sequence. It is one of the essential steps in analyzing human behavior from video data.

There are countless surveillance cameras in the world and these cameras play an important role in fighting crime. The police can use a surveillance video to track crime suspects after the occurrence of a crime. However, the ability to identify illegal or suspicious behaviors as they happen is a game changer, and this is where action recognition is good at. It can be used not only after a crime has been perpetuated but also during the crime itself or, sometimes, even before the crime happens. For example, if a person

deliberately abandons a bag in the airport and leaves the airport immediately, an action recognition system can send an alarm signal to security guards.

Action recognition is also an essential tool for video-sharing websites, such as YouTube and Twitch. It can understand the content of a video and make a decision on whether this video can be made public. This tool can help filter some dangerous video content, such as bomb making instructions, choking games, and use of hard drugs.

Security surveillance at home with the use of action recognition can be helpful for people who leave their children or elderly people alone at home. The system can notify the homeowner when someone falls down or when the children exhibit dangerous behaviors. Surveillance at convenience store equips with action recognition can prevent theft from stealing goods, and we can also use action recognition to analyze behaviors of a customer in front of the showcase of a clothing shop so that the owner knows which display is better. For human-robot interaction, robots that integrate action recognition can understand human behaviors with just a camera, and they execute instructions based on human movements.

1.1 Problem Statement

Nowadays, people share millions of videos every day on the internet and the surveillance system also produces a considerable amount of videos. Security becomes a crucial problem with this amount of messy data but opportunities are also hidden between these data if we can utilize it efficiently. Facing this explosion of visual information and opportunities, it is now essential to develop methods that can analyze video data and understand their content. Among the numerous video analysis tasks that have been the object of intense research are the ones related to human actions such as action recognition [42, 67, 66, 87], action detection [30], and abnormal event detection [6]. Remarkable progress has been achieved in these individual fields through the discovery of new algorithms or the application of different solutions. However, there is still a growing need for better video understanding methods both in terms of accuracy and efficiency.

This thesis addresses the problem of action recognition and considers the problem of human shopping actions in videos. This technique can apply on customer services analysis, such as service time estimation, or fully automated convenience store, such as Amazon go. We aim to build an end-to-end (video to classification result) deep neural network that classifies human shopping actions from a video sequence (with only RGB

information). However, our method has the potential to be expanded to any kind of action with corresponding dataset.

1.2 Motivation of the Problem

The earliest works on action recognition used 3D models [29, 61] to describe human actions. However, generating a 3D model from videos is difficult and expensive. As a result, people instead use global representations [5, 8, 86] or local representations [42, 13, 84] for action recognition. These methods are called representation-based methods, as described in section 2.1.

In recent years, with the rapid development of GPU computing, deep networks-based methods have been able to achieve remarkable results in action recognition. Tran et al. [78] create a generic video descriptor called C3D that achieves 82.3% accuracy on the UCF101 dataset [70]. Singh et al. [69] propose a multi-stream bi-directional recurrent neural network [64] that achieves 80.31% accuracy on the MERL [69] dataset. However, most of the deep networks in action recognition do not have a very deep structure compared to the networks used in image analysis [41] such as Highway Network [73], Residual Networks (ResNets) [26] and Densely Connected Convolutional Network (DenseNet) [32] that can use more than 100 layers. Therefore, we try to create a deep neural network for action recognition in surveillance videos that can have more than 100 layers.

1.3 Contributions

In this thesis, we propose a 3D Densely Connected Convolutional Network (3D-DenseNet) based on DenseNet [32] with 3D kernel [78] for recognition of actions taken by people interacting with items in a shopping context. We thus explore an architecture that can accommodate more than 100 layers as we believe that a deeper network increases its capability of learning more complex patterns of actions. We test our model on the MERL shopping dataset [69] showing overhead views of people manipulating items on a shelf. We also provide performance results for the KTH action recognition dataset [63].

To summarize, our contributions in this thesis are:

- We propose a new deep 3D convolutional network structure for action recognition that can have more than 100 layers.

- Our model performs better than the current state-of-the-art method on the MERL [69] dataset.
- We empirically find that 3D-DenseNet performs better as the depth (number of layers) and growth rate increase.
- We fully implement the 3D-DenseNet with Tensorflow and the code is available at Github: https://github.com/frankgu/3d-DenseNet

1.4 Thesis Outline

The thesis is organized as follows:

- Chapter 2 introduces the background and related work on action recognition. We group the methods for action recognition into two categories, representation-based methods and deep networks-based methods. Global representation and local representation are introduced in the representation section. The deep networks section covers spatio-temporal architecture, multiple-stream architecture, and deep generative architecture. At the end of this chapter, we present the widely used human action datasets.
- Chapter 3 presents our method for action recognition. We introduce the four basic layers of the 3D CNN, 3D convolution, 3D pooling, rectified linear unit, and fully connected layers. Then we explain dense connectivity, composite function, and some configuration parameters of our model. Finally, we present some implementation details of our model in section 3.3.
- Chapter 4 shows the pre-processing procedure for the dataset and the training setting of our network. We also present the TensorFlow implementation details of our model and the experiment result at the end of this chapter.
- Chapter 5 summarizes the thesis and presents its limitations and possible future research directions.

Chapter 2

Related Work

In this chapter, we introduce the state-of-the-art representation-based methods and deep networks-based methods for action recognition in realistic videos. In the representation-based methods section, we divide representation into 2D or 3D modeling, global representation, and local representation. In the deep networks-based methods section, we categorize the model architecture into spatio-temporal architecture, multiple-stream architecture and deep generative architecture. The densely connected convolutional network (DenseNet) will be introduced in a separate section because our 3D-DenseNet takes inspiration from it. Finally, two different datasets are introduced in detail, as well as a list of popular action datasets.

2.1 Representation-based Methods

In this section, we review some state-of-the-art human action recognition methods based on the representation of human action, i.e., those use handcrafted features, such as scale-invariant feature transform (SIFT), which has the benefits of invariant to translations, rotations and scaling transformations of images and robust to moderate perspective transformations and illumination variations. Generally, we can group these representation-based methods into the following three categories:

- **2D or 3D modeling methods** use a 2D pattern or a 3D model to represent human actions.
- **Global representation methods** use the whole human body structure, shape and movement as the representation of human actions.
- **Local representation methods** use a collection of patterns or distinct structures that are extracted from the human body to represent human actions.

2.1.1 2D or 3D modeling

2D or 3D modeling use either 2D motion patterns [24, 57] or 3D models [29, 7, 47, 61] to represent human actions. In the 2D motion pattern method, Guo et al. [24] record human motion as a sequence of stick figure parameters and then use the backpropagation neural network to classify these stick figures. Johansson [37] uses a few bright spots of the main joints from the human body to describe human motion (Figure 2.1).

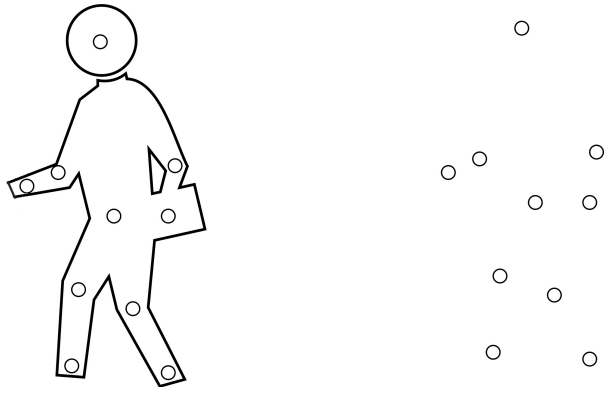


Figure 2.1: Illustration of a walking action on the left hand side and the corresponding spot of the walking action on the right hand side.

Much of early research on action recognition used the 3D modeling method. The walker hierarchical model [29] represents a person by using a series of hierarchical levels. Rohr [61] uses connected cylinders and their evolution to identify pedestrians on a video.

In general, 2D or 3D modeling methods show good results on simple and clean datasets. However, the performance of these methods heavily relies on some challenging techniques, such as image/video segmentation, human tracking, and 3D reconstruction. Most of these techniques are prone to error as a result of occlusion, background clutter, scale change, and multiple subjects in one image. Therefore, the majority of recent solutions avoid using 2D or 3D modeling methods and try to find a good representation of actions at the global or local level.

2.1.2 Global representation

Global representation methods use the whole human body structure, appearance, and movements for the representation of an action. Global representation methods are good at preserving the structural information of actions and are better than 2D or 3D modeling methods in terms of efficiency and robustness because they only need to construct global appearances and the actions of the human body instead of modeling every part of the body. To encode the global appearance of the human body, we commonly use silhouettes, optical flow, or a template to represent the human body.

The silhouettes of the human body can be obtained by background subtraction. Bobick and Davis [5] are among the earliest researchers who have used silhouettes for human action recognition. They proposed motion-energy image (MEI), which uses a binary image to indicate where motion occurs, and motion-history image (MHI), which adds the time variable to the weight of the silhouette. The MEI stacks several silhouette images into a single image and uses this image to represent the region of motion. On the other hand, MHI weights a motion region by applying a function to it over time to record how motion happens. This representation works well when the background is stable. In addition, Briassouli et al. [8] use a similar approach and they experiment their method on various human motions contained in both indoor and outdoor videos. Meng et al. [50] propose a new hierarchical motion history histogram that combines MHI to produce a low-dimension feature vector.

Wang et al. [86] use the $\mathfrak{R}$ transform to represent the low-level features for human silhouettes. This $\mathfrak{R}$ transform has the advantage of computational inexpensiveness, and it can distinguish similar activities when there is a disjoint silhouette, a silhouette with

holes, or frame loss data. Souvenir and Babbs [71] generalize the $\mathfrak{R}$ transform curve to the surface by adding temporal information to the $\mathfrak{R}$ transform.

Yamato et al. [90] experiment their algorithm on real time-sequential silhouette images to distinguish different tennis actions, such as backhand volley and forehand stroke. They use vector quantization to transform a set of time-sequential images to a symbol sequence, and then use the hidden markov model to learn different actions.

Wang et al. [85] use average motion energy (AME) and mean motion shape (MMS) as a representation of human silhouettes. The AME converts a set of time-sequential images to one binary image of one human motion. The MMS uses a border-following algorithm to obtain a single connectivity binary silhouette. Three simple classification methods and various distance metrics are used for classification.

Huang and Xu [31] use two orthogonal cameras with a similar height and distance to gain a viewpoint-insensitive representation of human action. A silhouette has been created by these two cameras, and an enveloping shape has been calculated by the silhouette. With the help of the enveloping shape, we will have enough discriminating features for action recognition.

Silhouettes have the advantage of efficiency and insensitivity to variable changes. However, generating silhouette representation highly relies on an accurate background subtraction, which is sensitive to the viewpoint, occlusion, and background, among other factors. When we have a good control of these factors, silhouettes usually perform well.

Unlike silhouette representation, optical flow representation does not depend on background subtraction. Polana and Nelson [56] generate an optical flow representation for the region of interest of a person and the stack motion magnitude feature in a square grid of non-overlapping bins. Classification is performed with the nearest centroid algorithm, which is efficient and easy to implement.

Efros el al. [17] compute optical flow in a very small person-centred image window and test their algorithm on soccer videos. They divide the horizontal and vertical components of the flow field into positively and negatively directed components, creating four different channels. Then, a blurry process is used to avoid noisy displacement. The classification is done with the k-nearest-neighbour algorithm, and four channels are matched separately.

Ali and Shah [1] derive a set of kinematic features from the optical flow. The set of features includes divergence, vorticity, symmetric and antisymmetric flow fields, second and third principal invariants of the flow gradient and rate of the strain tensor, and the third principal invariant of the rate of the rotation tensor. Principal component analysis

is performed on the spatio-temporal volumes to compute the kinematic modes. The coordinates of the kinematic mode-based feature space are used for classification with the k-nearest-neighbour algorithm.

Unlike silhouettes and optical flow, which are computed within a short period of time, template-based methods usually need a long sequence of video frames. Some template-based methods can be built by stacking silhouette images, such as in [4]. Other methods use spatio-temporal volume filters [60] and 3D Gaussian third-derivative filters [62].

2.1.3 Local representation

Local representation methods use a collection of local descriptors or patches as representations. They do not rely on accurate localization and background subtraction, which means that local representations can do better when we have illumination, occlusion, deformation, and multiple motions. Local representations can be divided into two different components, the feature detector and the feature descriptor.

Feature detectors

Feature detectors usually select the most significant spatio-temporal areas in an image, such as the corner of an image. These areas are called spatio-temporal interest points, and the neighborhoods of these interest points vary significantly both in the spatial and temporal domains.

Laptev and Lindeberg [42] are the first to build a space-time interest point detector by extending the Harris corner detector [25] to the 3D-Harris detector. In 3D-Harris, they detect the interest points in the spatio-temporal domain and calculate a spatio-temporal second-moment matrix for interest points with different spatial and temporal scales. Then, they smooth the function and space-time gradients by applying a Gaussian function on it. Another example [88] uses second-order derivatives instead of gradients. Schuldt et al. [63] use salient sparse spatio-temporal features based on the Harris corner detector.

When we direct 3D counterparts to commonly used 2D interest point detectors, the detection of 2D interest point detectors does not work well for detection. Dollár et al.[13] addressed this problem by creating a cuboid detector that applies the 1D Gabor-filter in the temporal dimension to convolve with a spatial Gaussian function and select the local maxima of the response convolution as interest points.

Because feature detectors are computationally expensive when we have a large amount of data and sparse interest points have the possibility of missing important aspects of the

scene, researchers tend to use densely sampled points that capture almost all relevant information and trajectories that track spatial interest points over a sequence of videos.

Scovanner et al. [65] introduce the 3D SIFT descriptor based on the 2D SIFT descriptor [46] to capture the spatio-temporal information of the video data, and use a bag of words method to represent the video. They use random sampling of a video at different locations, times, and scales to extract feature points.

Wang et al. [84] compare three different spatio-temporal feature point detectors, namely, 3D-Harris [42], Cuboid [13], and Hessian3D with dense sampling and use a bag-of-features SVM method for action recognition. The result shows that dense sampling at regular space-time grids is better than state-of-the-art interest point detectors.

Uemura et al. [79] extract a large number of features for every frame by multiple interest point detectors.They use the KLT tracker [46] and SIFT to create a robust feature extractor and they also create a method for estimating the dominant planes in the scene. Messing et al. [51] use the KLT tracker to track 3D-Harris interest points and obtain feature trajectories. They apply uniform quantization in log-polar coordinates to represent feature trajectories with a different length. In addition, a generative mixture of Markov chain models has been used for action recognition.

Feature descriptors

A feature descriptor is calculated in a local neighborhood centered at a spatio-temporal interest point to obtain the shape and motion information of a video sequence. The feature descriptor should be distinctive and insensitive to local image deformations to facilitate the matching procedure.

Laptev and Lindeberg [43] propose and compare several descriptors over local spatio-temporal neighborhoods, such as the histograms of spatio-temporal gradients and those of optical flow. They compute spatio-temporal gradients and the histograms of optical flow for each cell over a $M*M*M$ grid layout to generate the local motion and appearance at the same time. In addition, they concatenate the optical flow or gradient information of each pixel to reduce the dimension of features. The result shows that descriptors based on the spatio-temporal gradients and histograms of optical flow outperform those based on traditional global histograms.

Klaser et al. [39] propose the histograms of the oriented 3D spatio-temporal gradients descriptor. They initially divide the 3D patch into a grid of cells and then divide each cell into sub-blocks. A mean 3D gradient is computed and quantized using a polyhedron for each sub-block. Finally, they concatenate the gradient histogram of all cells and create

the 3D histogram of the oriented gradients for a given 3D patch.

Wang et al. [83] use motion boundary histogram (MBH) descriptor to describe dense trajectories. MBH was first introduced by Dalal et al. [11]. Because the derivatives of MBH are computed separately for the horizontal and vertical components of the optical flow, this will cause motion compensation. Wang et al. quantize the orientation information into histograms and process the magnitude for weighting to reduce motion compensation.

2.2 Deep Networks-based Methods

There are two major type of deep neural networks: CNN, which [44] plays an important role in learning image content [41, 9, 76, 32, 26] and recurrent neural network (RNN), [19] which is commonly used for sequential data, such as video [81, 15] and audio [19, 52]. The training procedure of these networks generates a complicated decision function from a large amount of data by composing multiple levels of nonlinear operations and learning the hyper-parameter based on the gradient descent approach. Generally speaking, a large amount of annotated data, a powerful GPU, and a gradient descent learning algorithm lead to the success of deep learning.

In this section, we group deep networks into the following three categories based on the network architecture they use:

- **Spatio-temporal architecture**
- **Multiple-stream architecture**
- **Deep generative architecture**

2.2.1 Spatio-temporal networks

In general, a convolutional architecture involves three main steps: convolution, pooling (also called sub-sampling), and normalization. These steps extract the most relevant image structure while reducing the search space of the network. Zeiler and Fergus [93] visualize the kernel learned by CNN and find that beginning layers learn low-level features, whereas top layers learn high-level representations. This shows that convolutional architecture can be used for feature extraction.

Unlike images, videos have a temporal dimension. One common way of utilizing the deep networks for action recognition is by directly adding the temporal information to a

convolutional architecture. To achieve this, Ji et al. [36] propose the 3D CNN, which uses 3D kernels (with the temporal dimension added) to extract both spatial and temporal information. Figure 2.2 shows the convolution and pooling operation of the 3D CNN.

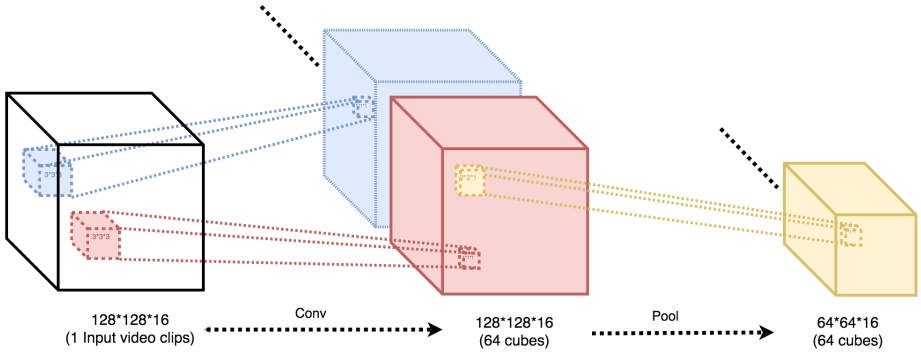


Figure 2.2: 3D convolution operation on input video clips (height * width * depth = $128 * 128 * 16$) with 64 kernels of size $3 * 3 * 3$ (small blue and red cubes), 3D pooling operation on big red and blue cubes with kernel size $2 * 2 * 1$ (height * width * depth) to create a $64 * 64 * 16$ feature cube.

In general, the 3D CNN outperforms the 2D frame-based method, but it has a very rigid temporal structure. We need to predefine the video clip sequence length, which is the number of frames, used as the input of the network. This may be unreasonable for action recognition because different actions may have different speeds and durations, so defining the video clip sequence length in advance is difficult.

Many researchers have contributed to the area of integrating temporal information into CNNs. Ng et al. [92] find that maximum pooling is better than average and other pooling methods in the temporal domain. Karpathy et al. [38] propose a new convolutional architecture called slow fusion model (Figure 2.3), which accepts video clips and processes them through a similar set of layers (with shared parameters) to produce outputs for fully connected layers. These fully connected layers will then produce the video descriptor.

Tran et al. [78] take the idea from visual geometry group (VGG) [9], Decaf [14], and the 3D CNN [36] to create a generic video descriptor called C3D (Figure 2.4). They train their network on the Sports-1M [38] dataset and extract the video features from a fully connected layer, as shown in Figure 2.4. Empirically, the authors find that a $3 * 3 * 3$ kernel size outperforms other sizes. For classification, they use linear SVM to classify the video features generated by a fully connected layer.

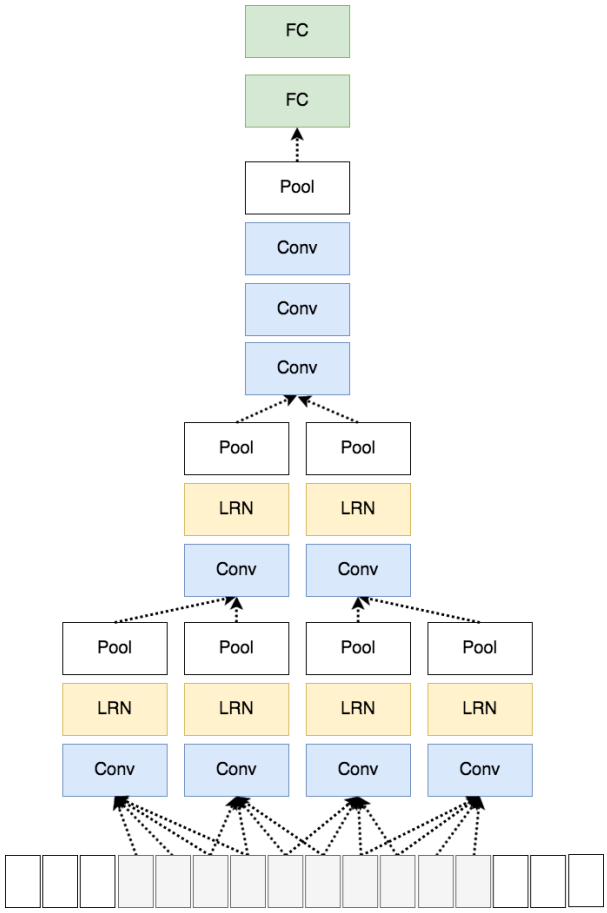


Figure 2.3: The blue box represents the convolution layer, the yellow box represents the local response normalization layer, the white box represents the pooling layer, and the green box represents the fully connected layer.

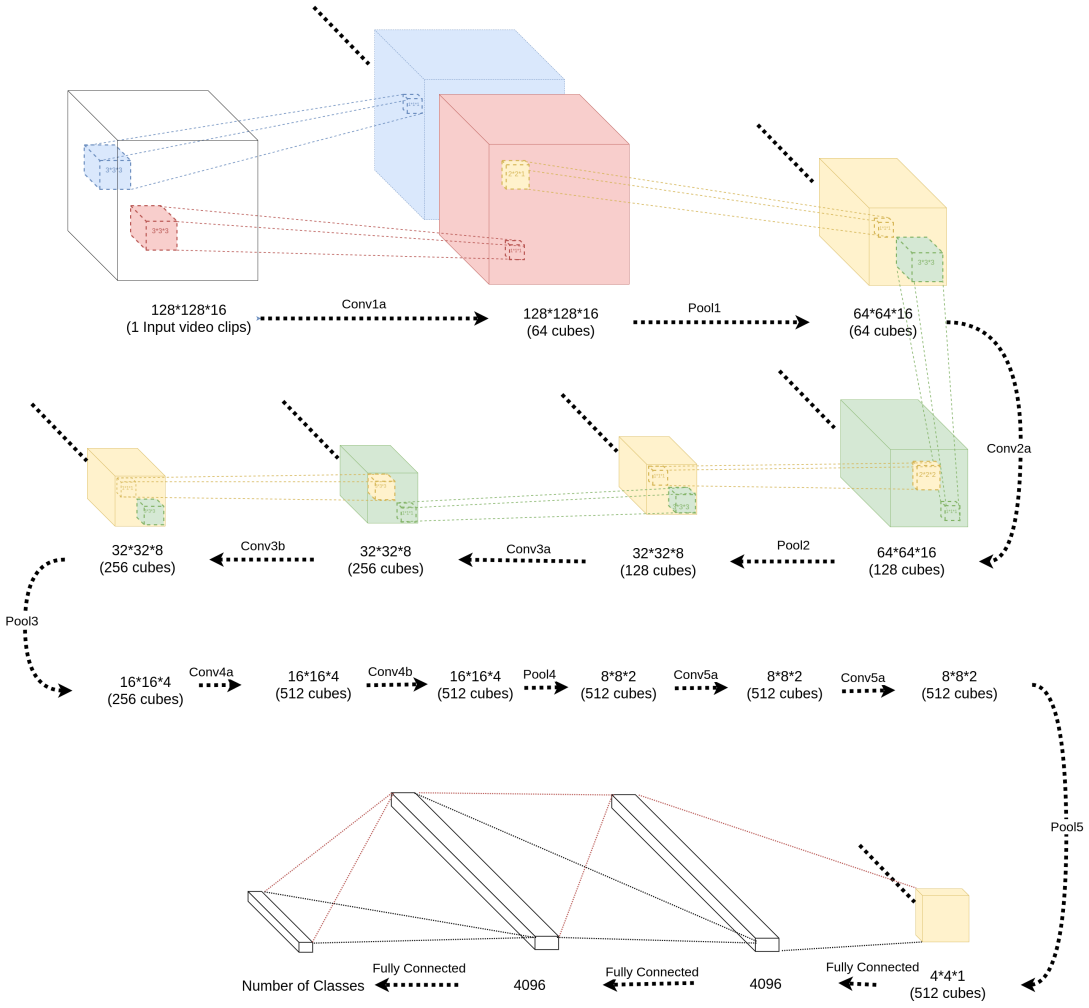


Figure 2.4: C3D has eight convolution layers, five maximum pooling layers, and two fully connected layers, followed by a softmax output layer. All the convolution kernel size is $3 * 3 * 3$ and all the pooling kernel size is $2 * 2 * 2$, except for the first pooling layer that has a size of $2 * 2 * 1$.

Exploring the 3D CNNs with long temporal durations at the input layer, Varol et al. [80] find that the performance increases with an increment in temporal depth. Sun et al. [74] use the combination of 2D and 1D filters to replace a 3D filter and thus reduce the number of training parameters of the convolutional network.

Recurrent structures can also be used for action recognition. Baccouche et al. [2] and Donahue et al. [15] use a cascade of convolutional networks and long-short term memory (LSTM) [28] (Figure 2.6), which is a special kind of RNN [59] (Figure 2.5). The chain-like structure of RNN reveals that it is closely related to sequences and lists. Therefore, RNN can be used for tasks, such as video processing, speech recognition, and video captioning, to name a few. However, the activation functions of RNN have gradients in the range $(-1, 1)$; if we have an n-layer network, we might encounter the gradient vanishing problem [3] because the gradient decreases exponentially with n ($0.99^n \approx 0$). LSTM solves this problem fundamentally by controlling the states and outputs of the RNN through control gates.

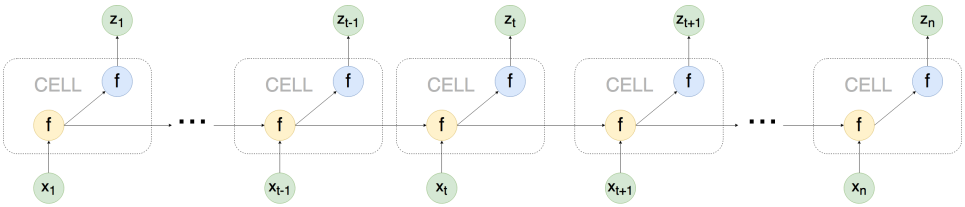


Figure 2.5: x represents the sequence input, z represents the output, and f represents an element-wise non-linearity, such as a sigmoid function or hyperbolic tangent function.

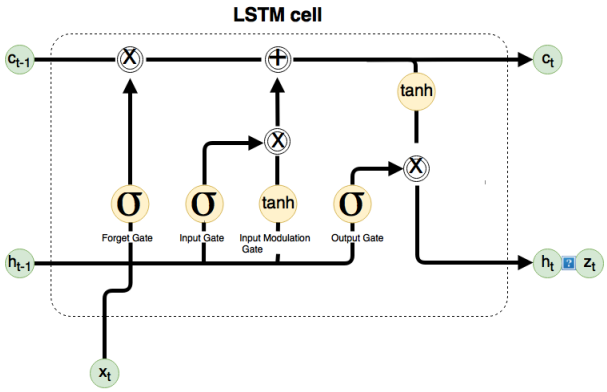


Figure 2.6: The LSTM cell has four different control gates that add or remove the information from the top cell state C_t.

Baccouche et al. [2] combine the 3D CNN and LSTM for action recognition. They use

a 3D CNN as a feature extractor to generate representations of video clips and then feed the representations to an LSTM network for classification. In addition, both networks are trained separately. Unlike Baccouche et al. [2], Donahue et al. [15] create an end-to-end network called long-term recurrent convolutional network that can be used for action recognition, video captioning, and image captioning.

2.2.2 Multiple-stream networks

Generally speaking, a multiple-stream network can take fully use of the rich multimodal information in videos. Such as raw RGB images, optical flow, and audio clues. Simonyan and Zisserman [68] are two of the earliest researchers who have proposed multiple-stream deep CNNs for action recognition (Figure 2.7). They use two different CNNs to handle both spatial and temporal information. The spatial stream network takes the raw RGB images, while the temporal stream network takes a set of optical flow images as input. In terms of implementation, they use multi-GPU to accelerate the training time, pre-train their spatial stream network on ImageNet ILSVRC-2012 to increase the accuracy, and use multi-task learning for the temporal stream network to increase the training set.

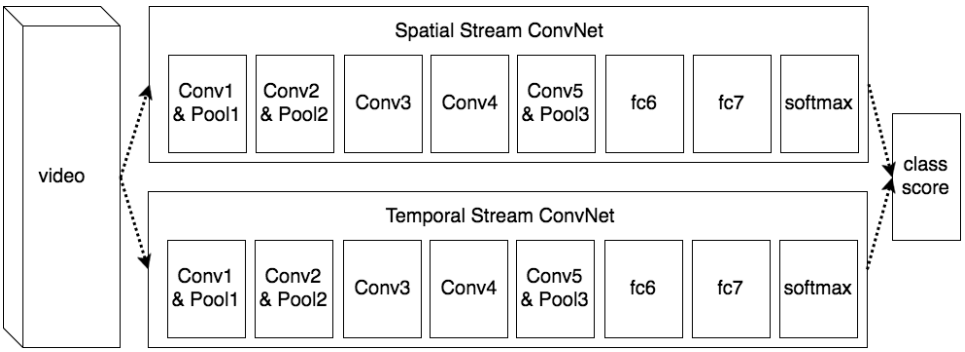


Figure 2.7: The two-stream network by Simonyan and Zisserman [68]. It takes the RGB and a set of optical flow frames as inputs for the network.

By extending the two-stream network [68], Feichtenhofer et al. [18] find that adding a fusion operation at an intermediate layer of the two-stream network can improve performance while decreasing the training parameters significantly (Figure 2.8). They perform fusion right after the last convolution layer, which will increase accuracy and remove the requirement of costly fully connected layers in both streams. They can reach the same accuracy as in the original network [68] with only half of the training parameters.

Wu et al. [89] use image frame, short-term motion, which is stacked optical flow over

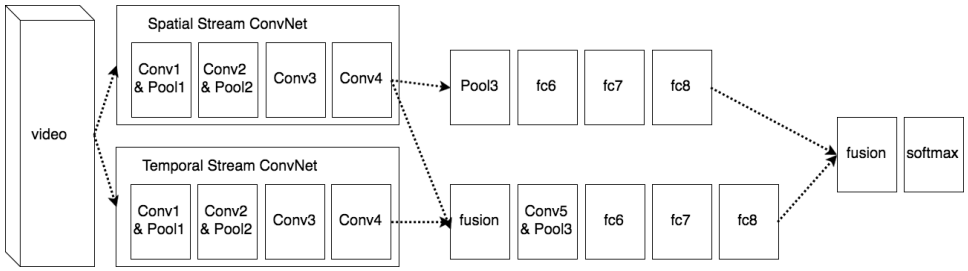


Figure 2.8: The two-stream fusion network by Feichtenhofer et al. [18]

a short temporal window, and audio clues to train three different CNNs separately. After that, they use the first fully connected layer from both image frame CNN and short-term motion CNN as input to two different LSTMs. By combining the outputs from three CNNs and two LSTMs, the author generates the final predictions.

Singh et al. [69] present a four streams bi-directional RNN for action recognition. The first stream of the network is the origin RGB frame, second is the bounding box around the person generated by a state-based tracker [69], third is a motion stream that uses pixel trajectories of a frame from the first stream, and the last one is the pixel trajectories of the second stream. All these streams have been trained by separate CNN followed by a bi-directional LSTM [22].

2.2.3 Deep generative networks

Nowadays, people post millions of videos on the web, but most of these do not have any labels or tag information. If we can create a deep model that requires little or no supervision, then data will no longer be a bottleneck for training the model. A generative model is a type of model that can use unsupervised data for video sequence analysis. Sutskever et al. [76] and Srivastava et al. [72] take a video sequence $x_1, x_2, ..., x_t$ as the input for their generative model and predict the future (e.g., the next instance x_{t+1}) of the video sequence. We can evaluate the model by checking whether the content and dynamics (e.g., motion primitives) of the video have been captured. In general, the goal for a deep generative network [82, 21, 28] is learning temporal information from unsupervised data.

Yan et al. [91] take the linear dynamic system modeling idea from Doretto et al. [16] and propose a deep dynencoder to capture video dynamics. They initially generate the hidden states h_t from raw pixel input x_t, and then they predict the next hidden states h_{t+1} from the current h_t. Finally, they map the predicted hidden state h_{t+1} to

produce the output frame x_{t+1}. They train the layer (autoencoder and dynpredictor) in the network separately in the pretraining phase and then fine-tuning the network to reduce training complexity.

Srivastava el al. [72] use the LSTM [28] cell to create the LSTM autoencoder model as shown in Figure 2.9. This model consists of the encoder LSTM and the decoder LSTM. The input of the model is a sequence of vector data $(v1, v2, v3)$, which is a set of images or features. The encoder LSTM reads the input sequence in order and learns the corresponding feature representation. After that, the decoder LSTM takes the learned representation to reconstruct the input sequence $(v3, v2)$ and output a prediction $(c3, c2, c1)$.

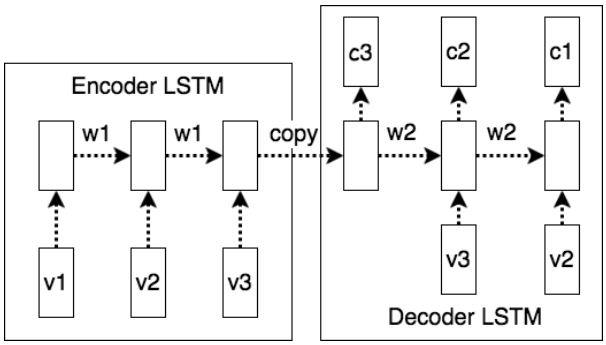


Figure 2.9: The LSTM autoencoder model from Srivastava el al. [72]

Goodfellow et al. [21] focus on the training difficulties in deep generative networks and propose an adversarial network. Adversarial refers to the competition between a generative model and a discriminative one. The generative model learns to capture the data distribution and create a sample that is similar to the input data; at the same time, the discriminative model estimates the probability whether a sample comes from the generative model or the origin data themselves. While training, the program tries to make the generative model create samples that are more similar to the origin data. Mathieu et al. [49] train a multi-scale convolutional network with the adversarial principle and discuss the advantages of pooling in a generative model.

2.3 Densely Connected Convolutional Network

Densely connected convolutional network (DenseNet) [32] is a CNN for image classification that surpasses 100-layer barrier. Usually, when CNNs get deeper and deeper, information about the input or gradient that passes through many layers can vanish by

the time it reaches the end (or beginning) of the network. However, DenseNet uses a simple connectivity pattern, which is dense connectivity, to tackle this problem. It ensures maximum information flow between layers in the network by connecting all layers with each other directly using the same size of feature-map. Each layer requires information from all preceding layers as input, then it passes its own feature-maps to all subsequent layers. Example of dense connectivity has been shown in Figure 2.10. Our 3D-DenseNet takes inspiration from DenseNet and creates a 3D version of DenseNet by adding a temporal dimension to all the convolution and pooling layers in DenseNet.

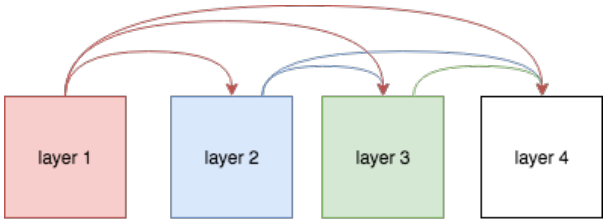


Figure 2.10: Layer 1 sends its feature-maps to all its subsequent layers $(2, 3, 4)$ and Layer 4 receives feature-maps from all its preceding layers $(1, 2, 3)$.

The overall architecture of DenseNet can be found in Figure 2.11. The composite function consists of three consecutive operations: BN [33] (section 3.2.2), followed by a ReLU [20] (section 3.1.3), and a convolution layer. Layers between each dense block are transition layers, which do the convolution and pooling operations. For 3D-DenseNet, we replace all the internal convolution and pooling layers of DenseNet with the 3D convolution and 3D pooling layers and we also change the input from images to videos.

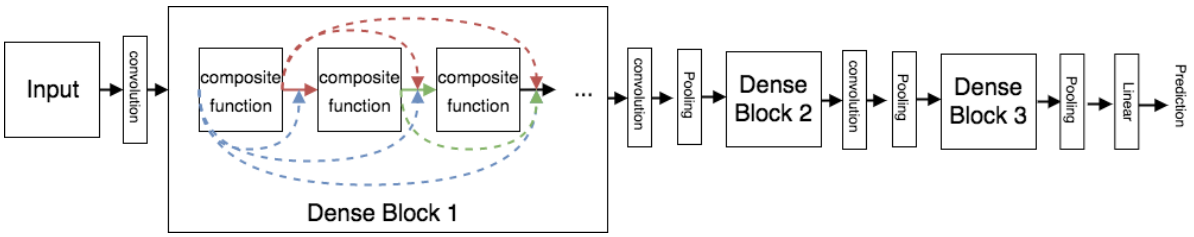


Figure 2.11: DenseNet with three dense blocks. The layers between two adjacent blocks are transition layers.

DenseNet obtains significant improvements over the state-of-the-art on most of the object recognition benchmark tasks among CIFAR-10 [40], CIFAR-100 [40], SVHN [53] and ImageNet [12]. It has better parameter efficiency (ratio of accuracy to training parameter) compare with residual neural network (ResNet) [26], ability to control over-

fitting, and gradually improvement with the increment of layers. During the experiment, our 3D-DenseNet shows the same behaviors as DenseNet and performs better than the current state-of-the-art on the MERL shopping dataset [69].

2.4 Datasets

In recent years, with the increasing popularity of human action recognition in the computer vision community, more and more video datasets have been created. One of the main advantages of using datasets is the ability they afford researchers to compare different action recognition models within the same input data. In this thesis, we choose two different video datasets (the KTH [63] and MERL shopping datasets [69]) because we want to perform recognition of human actions in the static background.

KTH [63] consists of the following six different classes: walking, jogging, running, boxing, hand waving and hand clapping. These actions are performed several times by 25 subjects in four different scenarios: outdoors, indoors, outdoors with scale variation, and outdoors with different clothes. Sample frames are shown in Figure 2.12. The dataset has 2,391 sequences, with slight camera movement. All clips in this dataset have 25 frame per second (fps) rates and $160 * 120$ resolutions.

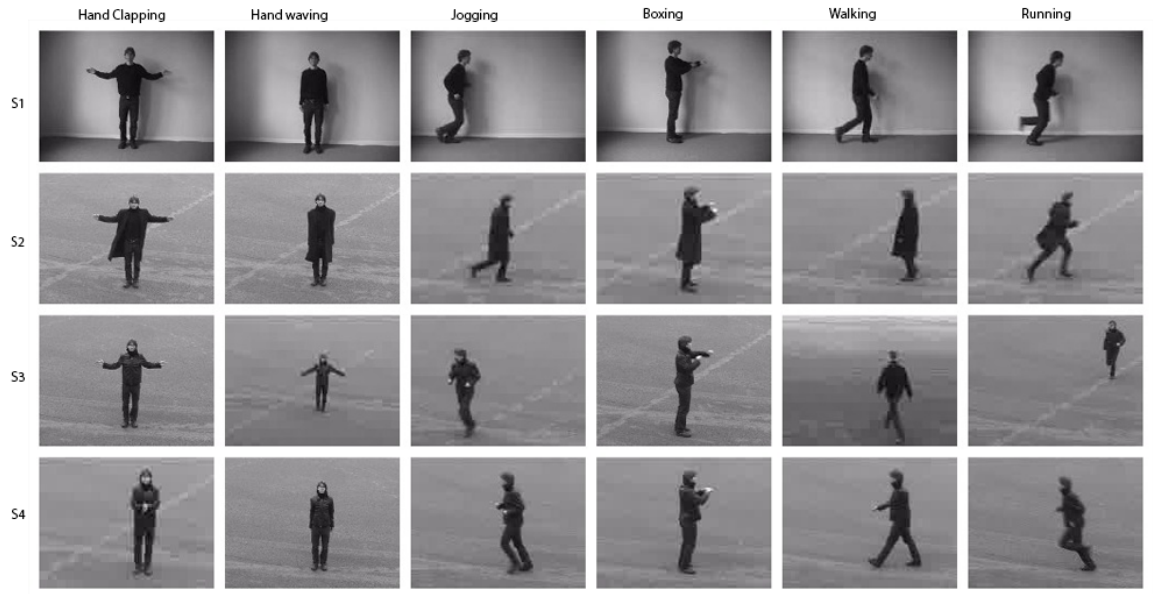


Figure 2.12: Example frame of six different actions in the KTH dataset [63]: $S1$,$S2$,$S3$, and $S4$ represent the different scenarios.

The **MERL shopping dataset** [69] consists of the following five different classes: reach to shelf, retract from shelf, hand in shelf, inspect product, and inspect shelf. These actions basically cover all possible behaviors for shopping in a grocery store. The dataset consists of 96 video clips with a stable background, and the setting is a lab space. There are 41 subjects in total and each video clip is labeled with the start and end times for different actions. The sample frames are shown in Figure 2.13. Currently, the best performance is reported in [69] with a mean accuracy of 80.31%.

Figure 2.13: The MERL shopping dataset [69] represents five different human shopping actions from a top view camera. From left to right, the actions are: retract from shelf, inspect shelf, inspect product, hand in shelf, and reach to shelf.

Table 2.1 is a list of commonly used datasets in action recognition. Large variations can be found in these datasets, such as in terms of scale changes, dynamic background, illumination, and people size. These variations give us an excellent simulation of the realistic environment but they also equate to challenges when we use them for training on our deep model. UCF Sports, UCF11, UCF50 and UCF101 in the list are ordered by time, and each dataset includes its precursor. The KTH dataset is the oldest dataset, and the MERL shopping dataset is the latest one.

In this thesis, we propose a deep network-based method to recognize human actions in videos. This method differs from other proposed methods because we can have more than 100 layers in our network without any signs of performance degradation or overfitting. Furthermore, our network performs better than the state-of-the-art action recognition method on the MERL shopping dataset and achieves a competitive result on the KTH dataset.

Dataset			
Name	Classes	Background	Resource
MERL[69]	5	Static	Actor Staged
KTH[63]	6	Static	Actor Staged
UCF Sports[60]	9	Dynamic	TV, Movies
UCF11[45]	11	Dynamic	YouTube
HOHA[48]	12	Dynamic	Movies
Olympic[54]	16	Dynamic	YouTube
UCF50[58]	50	Dynamic	YouTube
HMDB51[34]	51	Dynamic	Movies,YouTube,Web
UCF101[70]	101	Dynamic	YouTube

Table 2.1: List of major action recognition datasets

Chapter 3

3D Densely Connected Convolutional Network

In this chapter, we propose 3D-DenseNet, a deep network that adds temporal information to all the internal convolution and pooling layers in DenseNet [32] and makes it work on an action recognition task. We introduce the four basic layers of our 3D-DenseNet, namely, the 3D convolution layer, 3D pooling layer, rectified linear units (ReLU) layer, and the fully connected layer of a 3D CNN. Then, we explain dense connectivity, composite function, 3D pooling operation, growth rate k, 3D bottleneck layers, and the compression factor of our model in details. Finally, we describe the parameter setting for our network in Section 3.3.

3.1 Basic Layers

In this section, we explain the following four basic layers of the 3D CNN in detail: the 3D convolution layer, 3D pooling layer, ReLU layer, and the fully connected layer. The 3D convolution layer and the 3D pooling layer are well suited for spatio-temporal feature learning because they have the ability to model the temporal information of video data, whereas the two other layers perform the same functionality as they do in a general CNN.

3.1.1 3D convolution

Compared with 2D convolution layers, *3D convolution* operations are performed spatio-temporally; 2D convolution operations can only handle spatial information.

The details of the 2D convolution operation can be found in Figure 3.1. First, we slide the orange matrix (kernel) over the input image (green image). Normally, the kernel is moved by 1 pixel such that every pixel of the image is hit by the kernel center. However, skipping over the pixel is also possible; this is the stride of the operator. Then, we compute the element-wise multiplication between the two matrices and add the multiplication outputs to obtain the final integer result. We may notice that when we apply the $3 * 3$ kernel on the $5 * 5$ input image, the output feature size decreases to $3 * 3$. However, we will want to preserve as much information as possible during the entire process, so we can use a zero padding technique on the input image to pad the input image with zeros around the border as shown in Figure 3.2.

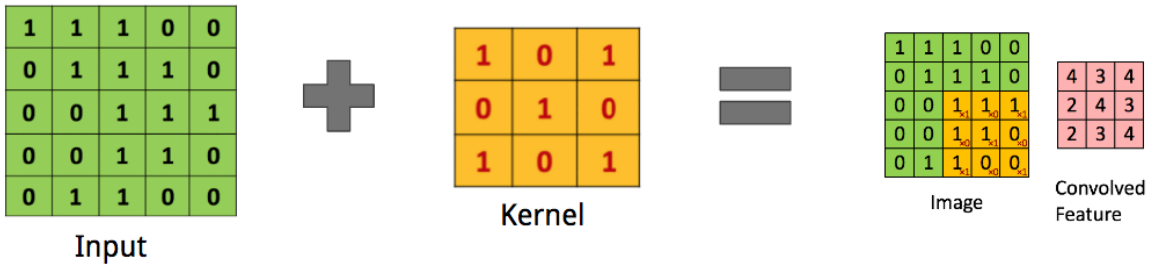


Figure 3.1: **2D convolution operation**. The green image is the input image for 2D convolution, the orange image is the kernel of size $3 * 3$ and the result is the pink image.

The extension from 2D convolution to 3D convolution is easy to imagine. All we need to do is add the temporal dimension into the input volume, kernel, and the output volume. The input is a video clip with three dimensions: height, width, and depth, which

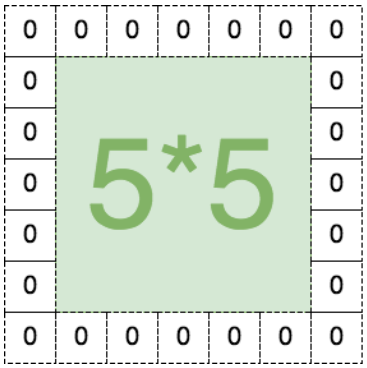


Figure 3.2: The input image is $5 * 5$: we add one border of zeros around the input image, which increases the image size to $6 * 6$. Then, when we apply the $3 * 3$ kernel over the input image. We can get a $5 * 5$ output image instead of a $3 * 3$ one.

is the number of images in a video clip. The kernel can be, for example a $3 * 3 * 3$ cube as shown in Figure 3.3 b). Finally, the output has the same dimension with the input video when we apply zero padding.

3.1.2 3D pooling

Inserting a 3D pooling layer between 3D convolution layers is quite common in the 3D CNN. It can reduce the spatio-temporal size of the video representation, as well as the parameters of the network. This operation can save computation power and control the overfitting problem. Pooling operations has many types, such as maximum pooling, average pooling, and L2-norm pooling. The basic steps for 3D pooling operation are as follows:

1. Take an input video clip of size $D_i * H_i * W_i * C_i$, where D_i is the number of images in this video clip, H_i is the height of the image, W_i is the width of the image and C_i is the channel of the image.

2. Define a kernel of size $H_k * W_k * C_k$, where H_k is the height of the kernel, W_k is the width, and C_k is the depth.

3. Define the stride S.

4. Produce the output of size $D_o * H_o * W_o * C_o$, where

   - $D_o = D_i$

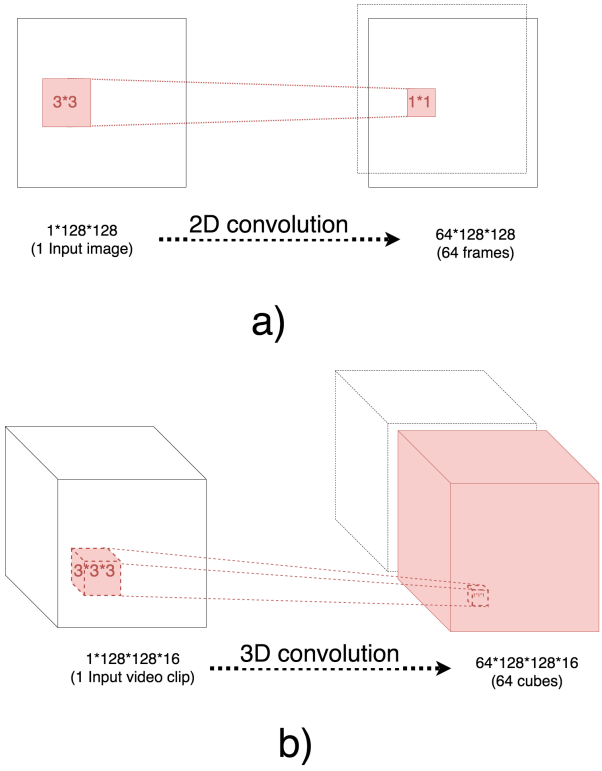


Figure 3.3: **2D and 3D convolution operations**. a) With 2D convolution applied on a $128 * 128$ (height * width) image with 64 different kernels (the size of the kernel is $3 * 3$), the result is a list of 64 images. b) With 3D convolution applied on a $128 * 128 * 16$ (height * width * channel) video clip with 64 different kernels (the size of the kernel is $3 * 3 * 3$), the result is a list of 64 video clips.

- $H_o = (H_i - H_k)/S + 1$
- $W_o = (W_i - W_k)/S + 1$
- $C_o = (C_i - C_k)/S + 1$

Before we proceed to the 3D pooling operation, we need to understand the details of the 2D pooling operation. We take Figure 3.4 as an example. The filter size is $2 * 2$. We stride the filter with a step of 2 along the height and width of the input. Every maximum operation will take the maximum number over four numbers in the box of size $2 * 2$ and every average operation will take the average of these four numbers.

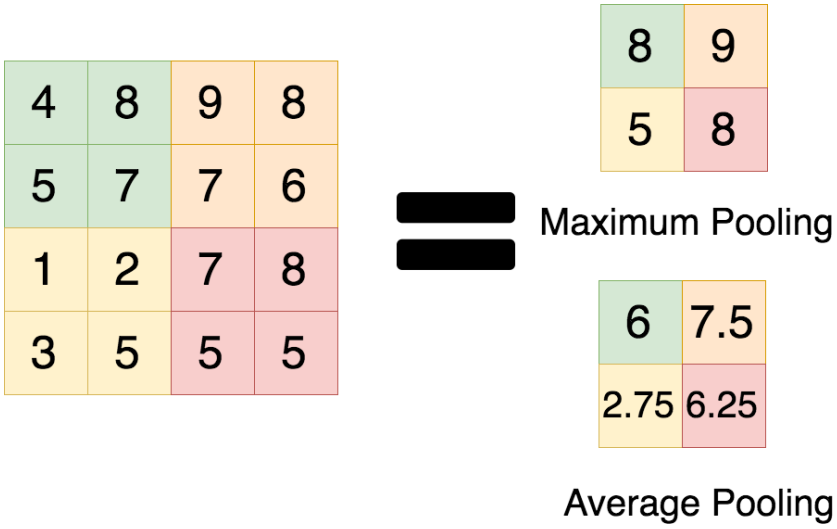


Figure 3.4: 2D maximum pooling and average pooling example with stride 2. Every four numbers of one color on the left produce one number of the same color on the right

The 3D pooling operation involves just adding the temporal dimension to 2D pooling. For example, we stride the filter of size $2 * 2 * 2$ with a step of 2 along the height, width, and channel in Figure 3.5). The input video of size $128 * 128 * 16$ will generate an output video of size $64 * 64 * 8$.

3.1.3 Rectified linear units (ReLU)

ReLU [20] is an activation function of the CNN. This layer has the ability to prevent the input from saturating [41] and increasing the nonlinear properties of the decision function. The ReLU is defined as

$$f(x) = max(0, x), \tag{3.1}$$

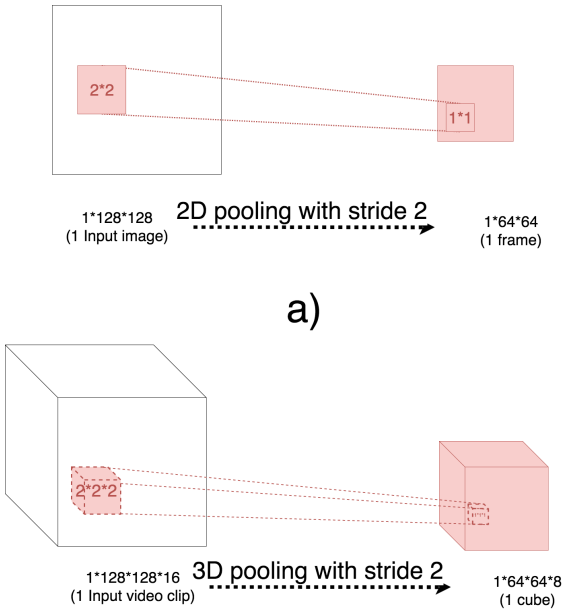
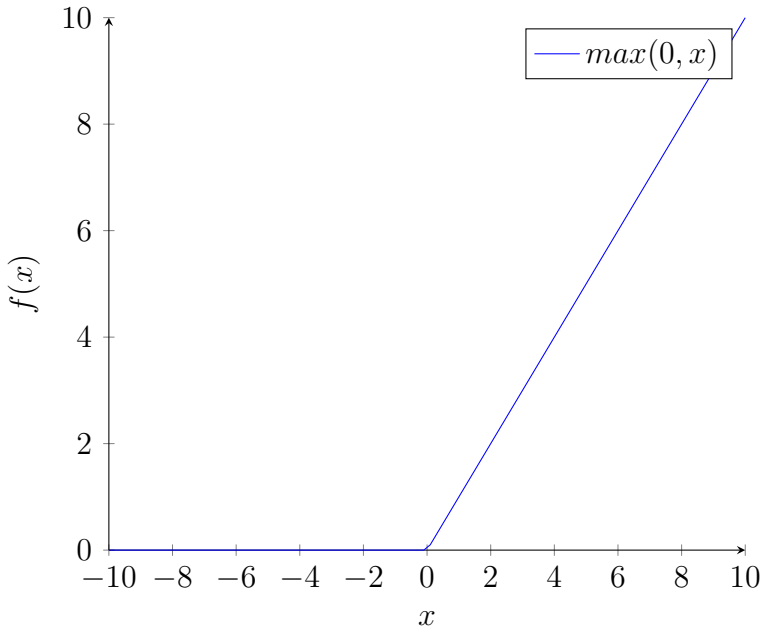


Figure 3.5: **2D and 3D pooling operations**. a) With 2D pooling applied on a 128∗128 (height * width) image with a stride of 2 (the size of the filter is $2 * 2$), the result is an image of size $64 * 64$. b) With 3D pooling applied on a $128 * 128 * 16$ (height * width * channels) video clip with a stride of 2 (the size of the filter is $2 * 2 * 2$), the result is a cube of size $64 * 64 * 8$.



where x is the output of the convolution layer. ReLU has the advantage of reducing the likelihood of a vanishing gradient without affecting the receptive fields of the convolution layer.

Many other functions can also be used for activation, similar to ReLU. The hyperbolic tangent is defined as

$$f(x) = tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}, \tag{3.2}$$

where x is the output of the convolution layer and the sigmoid function is defined as

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{3.3}$$

Overall, ReLU performs better than other activation functions do in terms of speed and accuracy. A network with ReLU will train several times faster than those using other activation functions [41].

3.1.4 Fully connected layer

After several layers of 3D convolution, 3D pooling, and ReLU, a fully connected layer is generally added to learn the non-linear functions of high-level features from previous convolution layers. The output from the 3D convolution is initially flattened and connected to every neuron in a fully connected layer. The neurons in the fully connected layer have full connection to all neurons in the previous layer, as shown in Figure 3.6.

The mathematics behind a fully connected layer can be shown as follows:

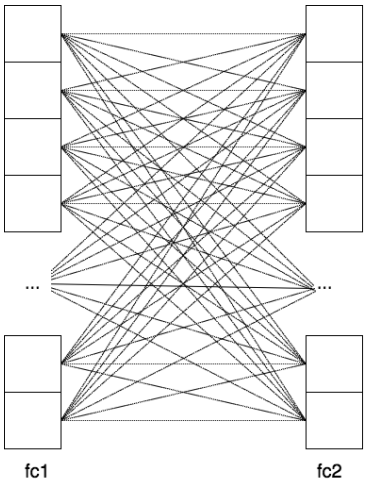


Figure 3.6: Each square represents a neuron in a CNN. $fc1$ is the first fully connected layer and $fc2$ is the second fully connected layer. Each neuron in $fc1$ is connected to the every neuron in $fc2$.

S.1 All neurons in the first fully connected layer are $x_1, x_2, ..., x_n$.

S.2 All neurons in the second fully connected layer are $y_1, y_2, ..., y_z$.

S.3 Then $y_z = x_1 * w_{1,z} + x_2 * w_{2,z} + ... + x_n * w_{n,z} + b_z$.

S.4 b_z is the bias offset of the matrix multiplication.

The last layer of the 3D CNN is the classification layer; it contains the number of classes when we perform the classification operation. Take Figure 3.7 as an example; we have four different classes, a,b,c, and d, in total. Each class is fully connected to the last fully connected layer. The probability of each class can be computed by steps **S.1** to **S.4** followed by the softmax function:

$$f(x)_i = \frac{e^{x_i}}{\sum_{n=1}^{n} e^{x_n}} \tag{3.4}$$

where x is the output from the last fully connected layer and i ranges from 1 to n; n is the number of classes.

3.2 3D Densely Connected Convolutional Network

3D-DenseNet is a network that adds the temporal dimension to DenseNet [32] for action recognition. The 3D-DenseNet consists of L layers, with each layer implementing a

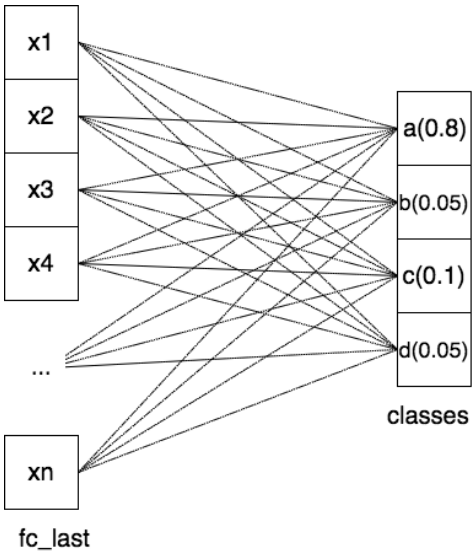


Figure 3.7: fc_{last} is the last fully connected layer and *classes* represent the last layer of the CNN. *a*,*b*,*c*, and *d* are the different classes of the dataset, and the number on the right is the predicted probability of each class.

non-linear transformation F_l, where l is the index of the layer. F_l can be a composite function that includes operations, such as batch normalization (BN) [33], ReLU, and 3D convolution. In the following subsections, we will introduce the concept of dense connectivity, operations of the composite function, 3D pooling, growth rate, 3D bottleneck layers, and compression in detail.

3.2.1 Dense connectivity

Dense connectivity is the direct connections from any layer to all subsequent layers. As shown in Figure 3.8, layer l sends the feature-maps y_l to all subsequent layers, $l+1, l+2, ..., L$, where L is the number of the layer. Reciprocally layer l will receive feature-maps from all preceding layers as input:

$$y_l = F_l([y_0, y_1, ..., y_{l-1}]) \tag{3.5}$$

where $[y_0, y_1, ..., y_{l-1}]$ is the concatenation of composite function outputs from layer $0, ..., l-1$. Dense connectivity is the core idea behind DenseNet [32]. Our method inherits this idea and creates 3D-DenseNet for action recognition. The equation remains the same as Eq. (3.5), except that y_l now has a temporal dimension to record video representations.

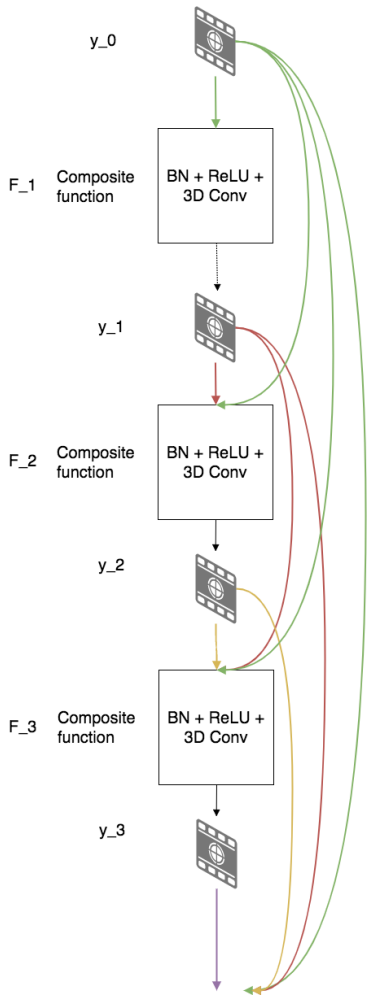


Figure 3.8: The composite function consist of BN, ReLU, and 3D convolution. Each layer has a direct connection to all of its subsequent layers.

3.2.2 Composite function

The composite function F_l is defined by the following three consecutive operations: BN [33], ReLU and a 3D convolution that has, in our case, a kernel size of $3 * 3 * 3$.

BN is accomplished as follows (Algorithm 1):

Algorithm 1 Batch Normalizing Transform

Input: Mini-batch $B = \{x_{1...n}\}$, where n is the number of the input

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$; γ and β are parameters to be learned

mini-batch means: $\mu_B \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$

mini-batch variance: $\sigma_B^2 \leftarrow \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$

normalize: $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2}}$

scale and shift: $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$

BN has been added before ReLU layer in each composite function to reduce internal covariate shift. It has the potential to speed up the training procedure and increase the accuracy. The mini batch in Algorithm 1 means several video clips (each video clip is a sequence of images) that are independent with each other.

ReLU in section 3.1.3 can be defined as follows:

$$f(x) = max(0, x) \tag{3.6}$$

Function $f(x)$ uses element-wise maximization between 0 and x, where x is the output from previous BN layer.

3D convolution is an extension of 2D convolution with a temporal dimension. Formally, 3D convolution [36] can be defined as follows:

$$v_{ij}^{xyz} = b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1}\sum_{r=0}^{R_i-1} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)} \tag{3.7}$$

where R_i is the size of the 3D kernel along the temporal dimension, and P_i and Q_i are the sizes of the 3D kernel along the spatial dimension, which is, in our case, $P_i = Q_i = R_i = 3$. Values (x, y, z) are the position in the data cube, i is the index of the layer, and j is the index of the feature map. w_{ijm}^{pqr} is the $(p, q, r)^{th}$ value of the kernel connected to the m^{th} feature map in the previous layer.

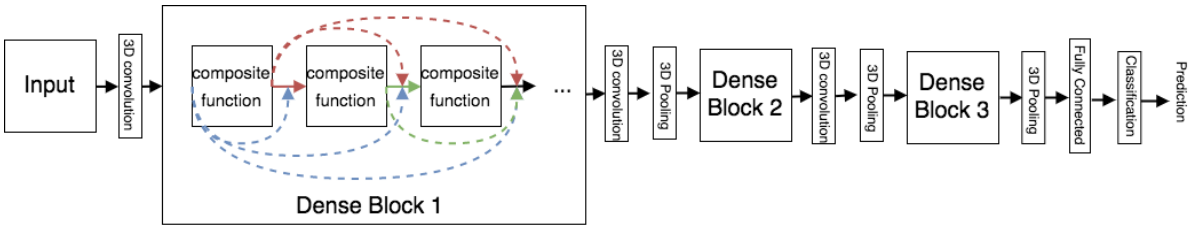


Figure 3.9: The overall representation of 3D-DenseNet with three dense blocks. Each dense block contains several composite functions that are connected in a feed-forward fashion. The feature-maps of all preceding composite layer are concatenated along the last dimension and are used as inputs for current composite function. The output feature-maps are also used as inputs for all subsequent layers. The 3D convolution and 3D pooling are transition layers that resize the feature-maps. The last linear layer is a fully connected layer that creates a connection between feature-maps and the different type of actions. Detail implementation can be found in Table 3.1.

3.2.3 3D pooling

3D pooling is an essential operation to reduce the size of the feature-maps. To use 3D pooling in our architecture we take the *multiple densely connected dense blocks* idea from [32]; (see Figure 3.9.) In our experiment, the layers between two adjacent blocks are transition layers that perform the 3D convolution and 3D pooling operations. The transition layer between dense block 1 and dense block 2 consists of a BN layer and an $1 * 1 * 1$ 3D convolution layer, followed by a $1 * 2 * 2$ (depth*height*width) 3D average pooling layer. The 3D average pooling layer between dense block 2 and dense block 3 has a size of $2 * 2 * 2$, and the 3D pooling layer after dense block 3 has a size of $\frac{D}{2} * 2 * 2$, where D is the number of images in the video clip.

3.2.4 Growth rate

Assume each composite function F_l of layer l in Eq. (3.5) produces number of k feature-maps, then the l^{th} layer has number of $k * (l-1) + k_0$ input feature-maps, where k_0 is the number of channels in the input image of video clip. k has been limited to a small integer, e.g., $k = 12$, to prevent the network from growing too rapidly. The hyper-parameter k is the *growth rate* in our network. In the experiment chapter, we have shown that even a small growth rate can reach the results of the state-of-the-art method on the datasets that we tested on because each layer has access to all preceding feature-maps within its

block. If we treat the overall feature-maps as the global state of the network, each layer appends k feature-maps of its own to this global state, which means that growth rate k controls how much new information each layer contributes to the global state. The global state can be accessed by every layer within the network, so duplicating the global state to every layer in the network is not needed.

3.2.5 3D bottleneck layers

Even if each layer only produces k (growth rate) output feature-maps, we still have a large number of inputs because we have a very deep structure in our network. By extending the $1 * 1$ convolution layer (bottleneck layer) [77, 26, 32] to the $1 * 1 * 1$ 3D convolution layer, we propose to introduce *3D bottleneck layers* to reduce the number of input feature-maps. This 3D bottleneck layer should be added before each $3 * 3 * 3$ 3D convolution layer to reduce the input to $4k$ feature-maps in all experiments, where k is the growth rate. An example is as follows, if k is 12 and the input volume is $100 * 100 * 100 * 512$, where $100 * 100 * 100$ is height, width, and depth of the video, we will convolve this input volume with $4k = 48$ kernels that have size $1 * 1 * 1 * 512$ and produce a volume of size $100 * 100 * 100 * 48$. This operation reduces the input volume from $100 * 100 * 100 * 512$ to $100 * 100 * 100 * 48$. This design can highly reduce the training parameters and improve computational efficiency according to our experiment result in section 4.5 (Parameter Efficiency part). We refer to our network with such a 3D bottleneck layer as 3D-DenseNet-B.

3.2.6 Compression

To further improve computational efficiency and reduce the training parameters, we can also reduce the number of feature-maps at transition layers by using the same principle as that in the 3D bottleneck layer. If a dense block produces m feature-maps, we add a $1 * 1 * 1$ 3D convolution layer before the 3D pooling layer shown in Figure 3.9. This $1 * 1 * 1$ 3D convolution layer will generate $\llcorner m\theta \lrcorner$ output feature-maps, where $0 < \theta \leq 1$ is the compression factor. When $\theta < 1$, we name the 3D-DenseNet as 3D-DenseNet-C. In our experiment, we set the compression factor θ to 0.5. The network with both a 3D bottleneck layer and a compression factor less than 1 is called 3D-DenseNet-BC.

3.3 Implementation Details

In our experiments, two different settings (general and lite) are used for our 3D-DenseNet. Each setting has two different versions, 3D-DenseNet and 3D-DenseNet-BC. For each setting, 3D-Densenet has three dense blocks with the same numbers of layers. As we can see in Table 3.1, we use a 3D convolution layer with kernel size $3 * 3 * 3$ on the input images before we enter the first dense block. The output channel for this convolution layer is twice the growth rate for 3D-DenseNet. For each convolution layer inside the dense block, each side of the input is zero padded (Figure 3.2) by one pixel to keep a fixed-size feature-map. The transition layer between two dense blocks consists of a $1 * 1 * 1$ convolution layer and a $d * 2 * 2$ average pooling layer, where d is 1 on the first transition layer and 2 on the second transition layer. Inside the classification layer, a global average pooling is used, follow by a softmax classifier. Table 3.2 presents the general 3D-DenseNet-BC version of our model. Each dense block has one more $1 * 1 * 1$ *conv* operation, compared with that of the general 3D-DenseNet in order to mitigate overfitting and reduce the training parameters.

Layers	Output Size	3D-DenseNet-20(k=12)	3D-DenseNet-40(k=12)
3D Convolution	$16 * 100 * 100$	$3 * 3 * 3$ conv	
Dense Block 1	$16 * 100 * 100$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Transition Layer 1	$16 * 100 * 100$	$1 * 1 * 1$ conv	
	$16 * 50 * 50$	$1 * 2 * 2$ average pool	
Dense Block 2	$16 * 50 * 50$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Transition Layer 2	$16 * 50 * 50$	$1 * 1 * 1$ conv	
	$8 * 25 * 25$	$2 * 2 * 2$ average pool	
Dense Block 3	$8 * 25 * 25$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Classification Layer	$1 * 1 * 1$	$8 * 25 * 25$ average pool	
		fully connected	
		classification and softmax	

Table 3.1: General 3D-DenseNet architecture. Each "conv" layer shown in the table represents a sequence of BN-ReLU-3DConvolution operations

Table 3.3 and 3.4 show the lite setting of 3D-DenseNet. We change the input size from $16 * 100 * 100$ to $16 * 128 * 128$, compared with that of the general 3D-DenseNet, and we also add a 3D pooling layer right after the first 3D convolution layer. The kernel

Layers	Output Size	3D-DenseNet-BC-20(k=12)	3D-DenseNet-BC-40(k=12)
3D Convolution	$16 * 100 * 100$	$3 * 3 * 3$ conv	
Dense Block 1	$16 * 100 * 100$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Transition Layer 1	$16 * 100 * 100$	$1 * 1 * 1$ conv	
	$16 * 50 * 50$	$1 * 2 * 2$ average pool	
Dense Block 2	$16 * 50 * 50$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Transition Layer 2	$16 * 50 * 50$	$1 * 1 * 1$ conv	
	$8 * 25 * 25$	$2 * 2 * 2$ average pool	
Dense Block 3	$8 * 25 * 25$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Classification Layer	$1 * 1 * 1$	$8 * 25 * 25$ average pool	
		fully connected	
		classification and softmax	

Table 3.2: General 3D-DenseNet-BC architecture. Each "conv" layer shown in the table represents a sequence of BN-ReLU-3DConvolution operations

size of the first 3D convolution has been changed to $7 * 7 * 7$ with stride $1 * 2 * 2$. Notice that the depth dimension of the stride for the first 3D convolution and 3D pooling layer is always 1 because we want to keep as many temporal information as possible at the beginning for better accuracy. This 3D-DenseNet is called lite 3D-DenseNet because it requires less GPU memory than the general 3D-DenseNet does, with the same depth and growth rate setting.

Layers	Output Size	3D-DenseNet-20(k=12)	3D-DenseNet-40(k=12)
3D Convolution	$16 * 64 * 64$	$7 * 7 * 7$ conv with stride $1 * 2 * 2$	
3D Pooling	$16 * 32 * 32$	$3 * 3 * 3$ average pool with stride $1 * 2 * 2$	
Dense Block 1	$16 * 32 * 32$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Transition Layer 1	$16 * 32 * 32$	$1 * 1 * 1$ conv	
	$8 * 16 * 16$	$2 * 2 * 2$ average pool	
Dense Block 2	$8 * 16 * 16$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Transition Layer 2	$8 * 16 * 16$	$1 * 1 * 1$ conv	
	$4 * 8 * 8$	$2 * 2 * 2$ average pool	
Dense Block 3	$4 * 8 * 8$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 5$	$\begin{bmatrix} 3*3*3 & conv \end{bmatrix} * 12$
Classification Layer	$1 * 1 * 1$	$4 * 8 * 8$ average pool	
		fully connected	
		classification and softmax	

Table 3.3: Lite 3D-DenseNet architecture. Each "conv" layer shown in the table represents a sequence of BN-ReLU-3DConvolution operations

Layers	Output Size	3D-DenseNet-BC-20(k=12)	3D-DenseNet-BC-40(k=12)
3D Convolution	$16 * 64 * 64$	$7 * 7 * 7$ conv with stride $1 * 2 * 2$	
Pooling	$16 * 32 * 32$	$3 * 3 * 3$ average pool with stride $1 * 2 * 2$	
Dense Block 1	$16 * 32 * 32$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Transition Layer 1	$16 * 32 * 32$	$1 * 1 * 1$ conv	
	$8 * 16 * 16$	$2 * 2 * 2$ average pool	
Dense Block 2	$8 * 16 * 16$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Transition Layer 2	$8 * 16 * 16$	$1 * 1 * 1$ conv	
	$4 * 8 * 8$	$2 * 2 * 2$ average pool	
Dense Block 3	$4 * 8 * 8$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 2$	$\begin{bmatrix} 1*1*1 & conv \\ 3*3*3 & conv \end{bmatrix} * 6$
Classification Layer	$1 * 1 * 1$	$4 * 8 * 8$ average pool	
		fully connected	
		classification and softmax	

Table 3.4: Lite 3D-DenseNet-BC architecture. Each "conv" layer shown in the table represents a sequence of BN-ReLU-3DConvolution operations

Chapter 4

Experiments

We present here the experiment results of our model when applying different parameter settings. The implementation setting can be found in Table 4.1 and the classification results on the MERL shopping dataset [69] and the KTH dataset [63] are shown in Table 4.2. All computations are done on an Intel i7-6800K 3.4GHZ PC with 32 GB RAM and a NVIDIA TITAN X video card with 12 GB memory. The average training time for each setting is one and a half days. The program is implemented in Python with TensorFlow. (Code implementation can be found on Github: https://github.com/frankgu/3d-DenseNet and code readme can be found in Appendix B)

4.1 Dataset Pre-processing

For each video in the dataset, we transform the video data into a sequence of image data with a 25 fps rate, as shown in Figure 4.2. The folder structure of the dataset is as follows:

{Dataset Name} -> {Action Name} -> {Video Name} -> {number.png}

One folder structure from the MERL dataset after we transform the video to sequence of images can be

merl -> Hand_In_Shelf -> 1_1_1

The data split of the MERL dataset follows the same rule as that in [69]. Subjects with an id of 1 to 20 belong to the training set, and id of 21 to 26 to the validation set, and id of 27 to 41 to the testing set. In addition, we use the HOG [10] people detector on **KTH** to extract the bounding box of a person as the input of our network. Image examples can be found in Figure 4.1.

Figure 4.1: Bounding box of people for the KTH dataset [63].

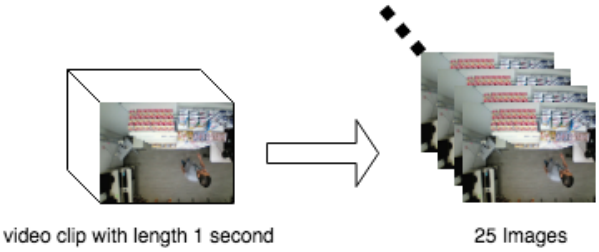


Figure 4.2: With a 25 fps rate, the top view video clip with a length of 1 second will be transformed into a 25-image sequence.

4.2 Accuracy

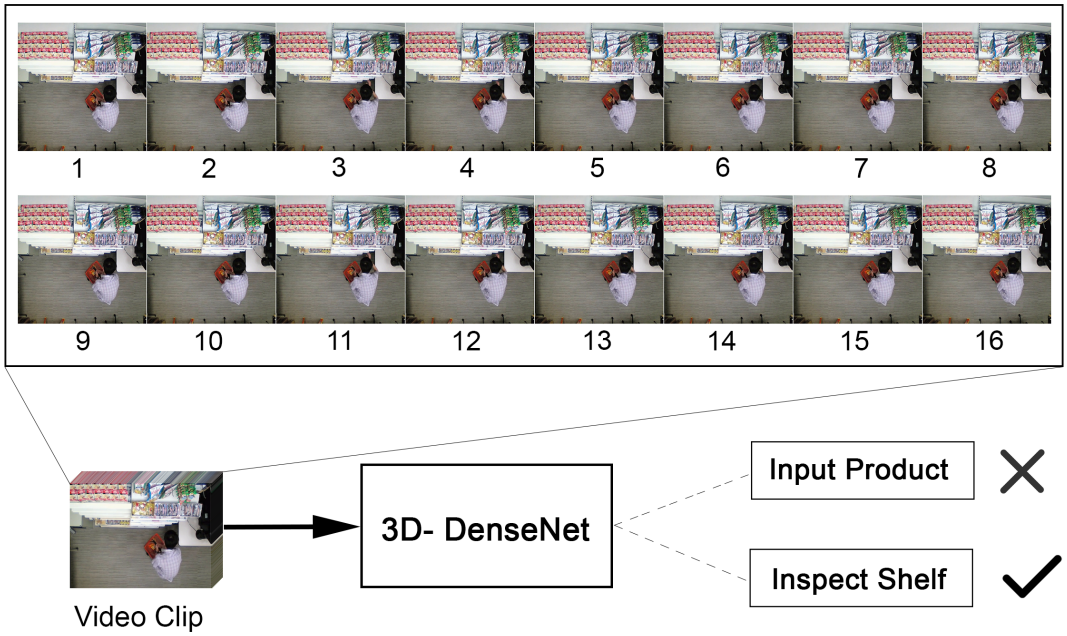


Figure 4.3: Workflow of 3D-DenseNet.

Figure 4.3 gives an example of the 3D-DenseNet workflow. A video clip in the figure consists of 16 sequential images. We can use this video clip as the input for 3D-DenseNet and 3D-DenseNet will generate a prediction, which is the name of the action, for this video clip. If the prediction matches the ground truth, then we increase the correct count by one. Overall, the accuracy of 3D-DenseNet is equal to the correct count divided by the total number of detection.

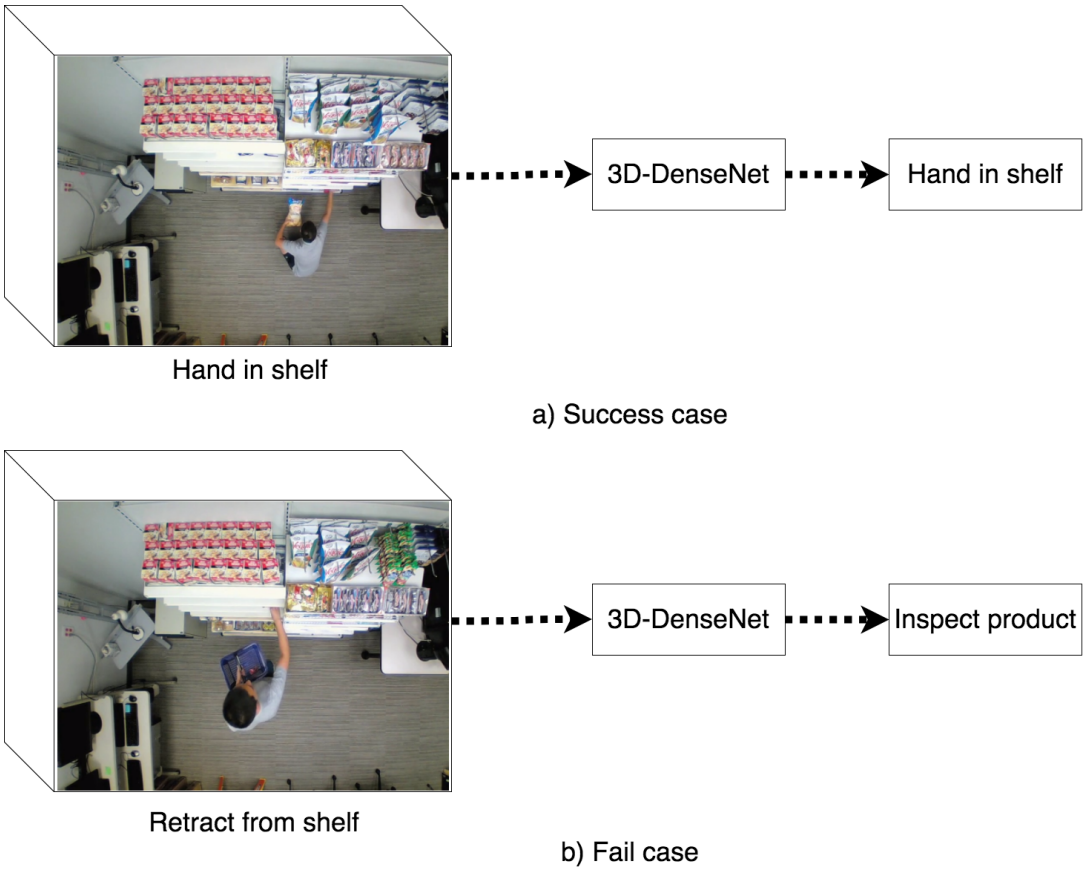


Figure 4.4: Detection examples of 3D-DenseNet. a) is a successful detection of hand in shelf action. b) is a failure detection of retracting from shelf action.

Figure 4.4 gives a success and a fail detection example of 3D-DenseNet. Part a) shows a successful detection of hand in shelf action, the input is a video clip with hand in shelf action and the model prediction is hand in shelf. Part b) shows an unsuccessful detection of retracting from shelf action, the input is a video clip with retract from shelf action and the model prediction is inspecting product, which does not match the input action.

4.3 Training

The training procedure has been done using stochastic gradient descent. Because of limitations in GPU memory, the crop size of the video, which is the (width, height) of the image, has been set to (64, 64), (100, 100), or a larger value accordingly, and the batch size has been set to 4, 6, 8, or 10 according to the growth rate and the depth of the network; the higher are the growth rate and depth, the lower are the crop size and batch size. The number of training epoch is 70 for both the MERL and KTH datasets.

The initial learning rate is 0.1, and it will decrease to 0.01 at 30 epochs and 0.001 at 55 epochs. All parameter settings for our network can be found in Table 4.1.

With the idea from [23] followed, a weight decay of 10^{-4} has been applied for all the weights, and a Nesterov momentum [75] of 0.9 without dampening has been used in our model. The initialization operation takes the idea from [27]. The test accuracy will be evaluated once for each training epoch.

	Growth Rate	Depth	Batch Size	Crop Size	Sequence Length
(1)General 3D-DenseNet	12	20	10	(100, 100)	16
(2)General 3D-DenseNet-BC	12	20	10	(100, 100)	16
(3)General 3D-DenseNet-BC	12	20	10	(64, 128)	16
(4)General 3D-DenseNet-BC	12	40	10	(64, 64)	16
(5)General 3D-DenseNet-BC	24	20	10	(100, 100)	16
(6)General 3D-DenseNet-BC	24	30	4	(64, 128)	32
(7)General 3D-DenseNet-BC	24	30	8	(64, 128)	16
(8)General 3D-DenseNet-BC	24	40	8	(64, 64)	16
(9)Lite 3D-DenseNet-BC	12	20	10	(128, 128)	16
(10)Lite 3D-DenseNet-BC	24	40	10	(128, 128)	16
(11)Lite 3D-DenseNet-BC	24	40	10	(256, 128)	32
(12)Lite 3D-DenseNet-BC	24	40	10	(256, 256)	16
(13)Lite 3D-DenseNet-BC	24	40	10	(288, 188)	16
(14)Lite 3D-DenseNet-BC	24	40	6	(400, 256)	16
(15)Lite 3D-DenseNet-BC	24	100	10	(128, 128)	16

Table 4.1: Growth rate, Depth, Batch size, Crop size and Sequence length values for 3D-DenseNet. The number at the beginning is the reference number used in Table 4.2.

4.4 TensorFlow Implementation Details

In this section, we will explain the implementation details of our general 3D-DenseNet-BC network (Table 3.2 with depth 20) by expanding Figure 4.11. Each node will be expanded, and the operations inside the node will be introduced.

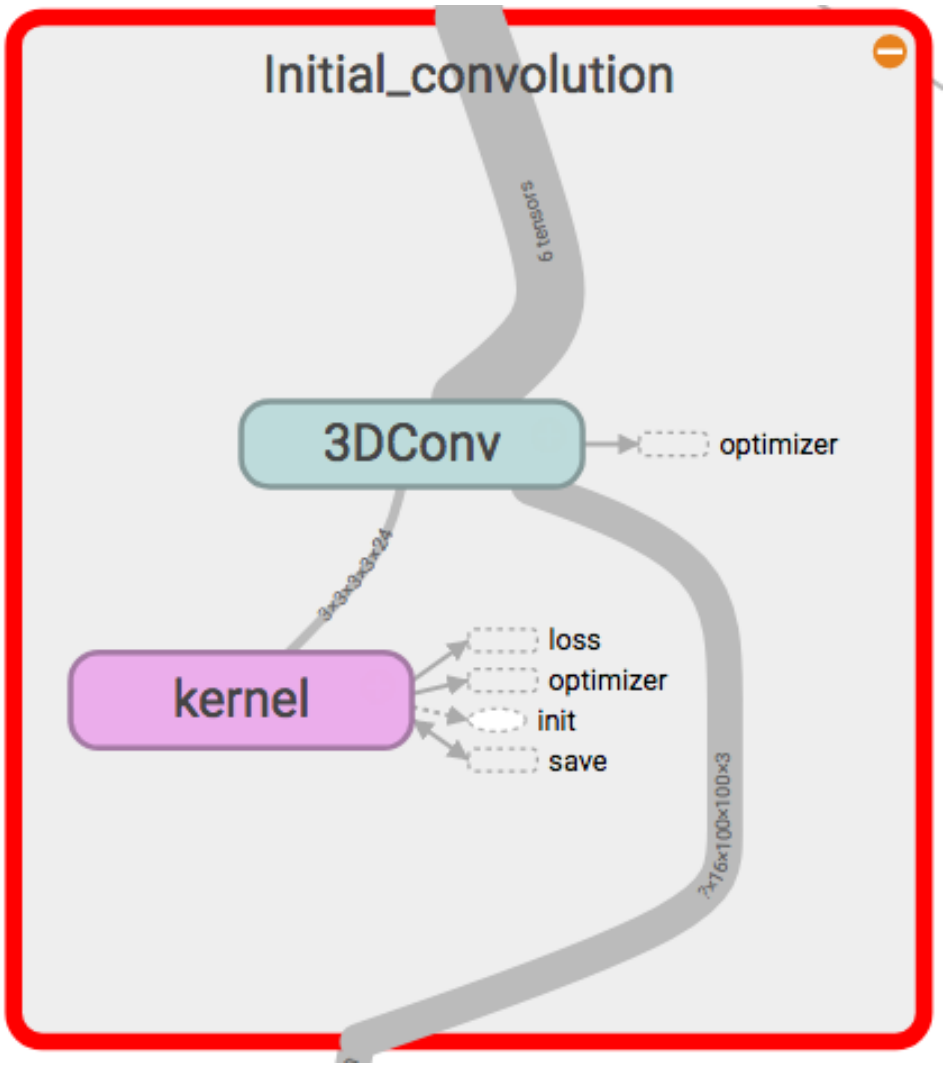


Figure 4.5: The first 3D convolution layer of the general 3D-DenseNet-BC network.

The **initial convolution** (Figure 4.5) node takes the video clip as input and applies the 3D convolution operation, as shown in section 3.1.1. The kernel node represents the weights of the 3D convolution.

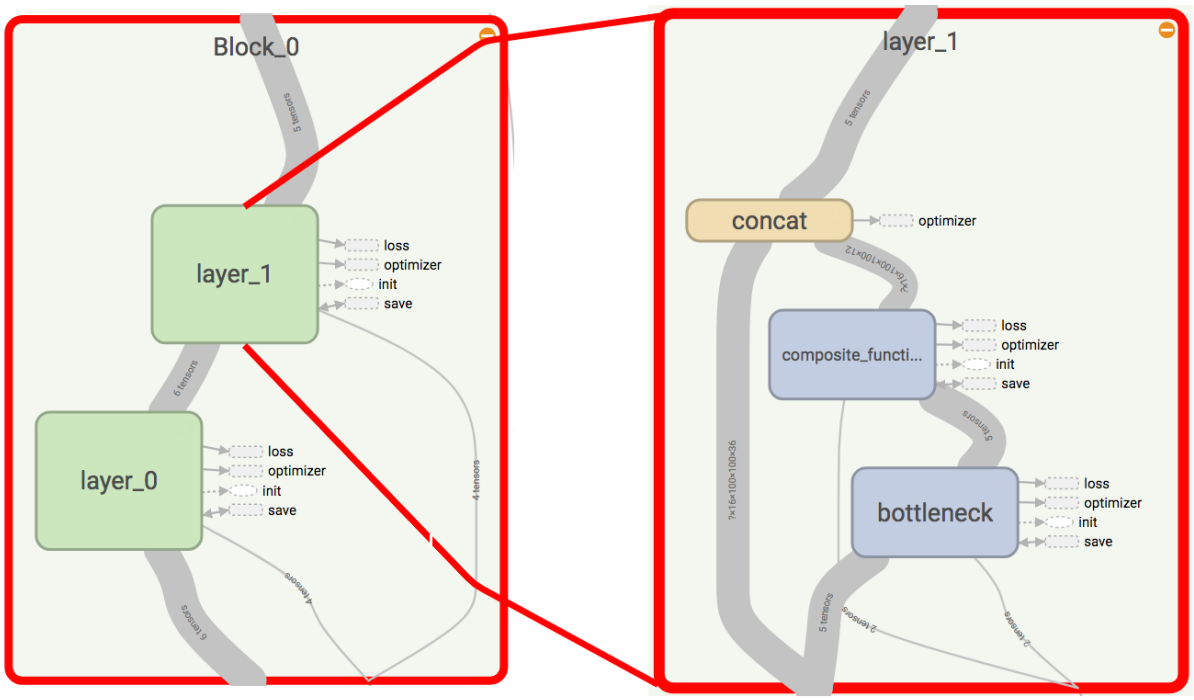


Figure 4.6: The block layer of the general 3D-DenseNet-BC network.

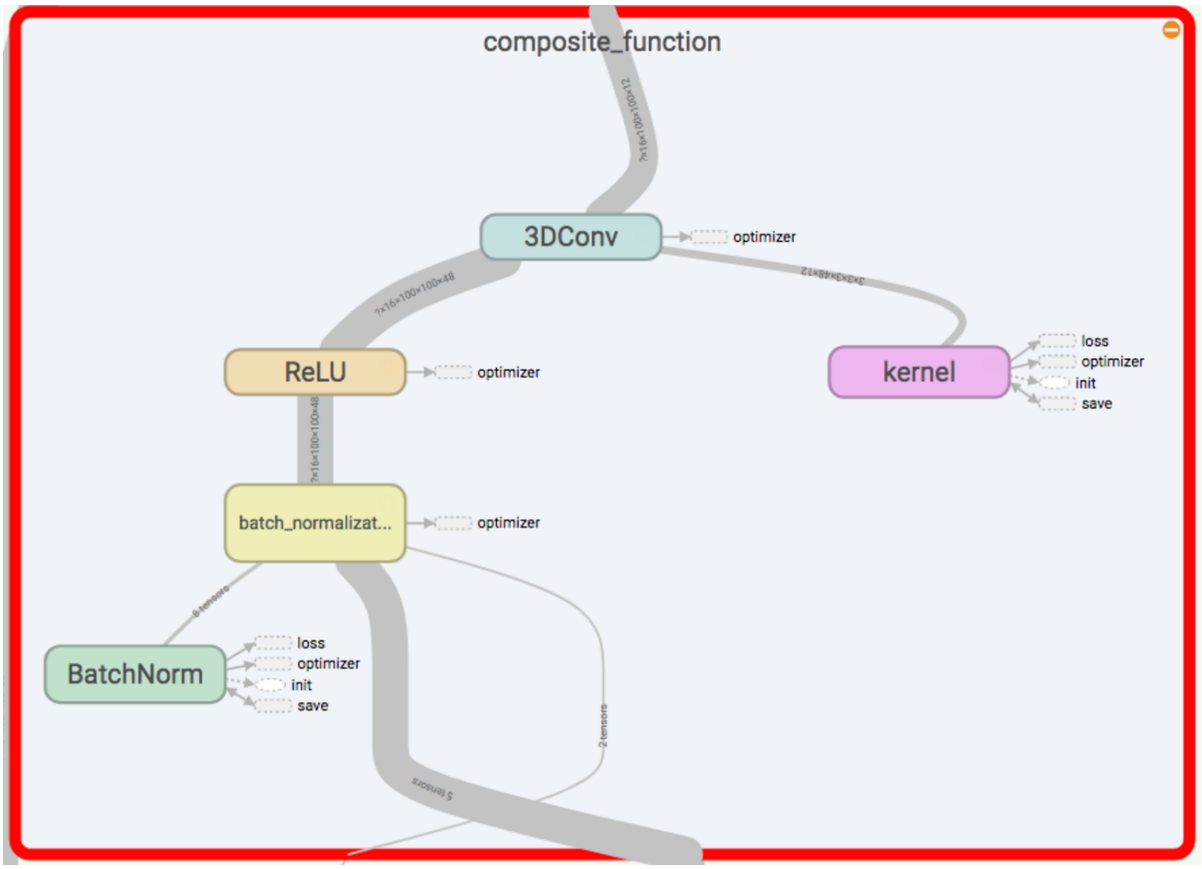


Figure 4.7: The composite function inside block layer in Figure 4.6.

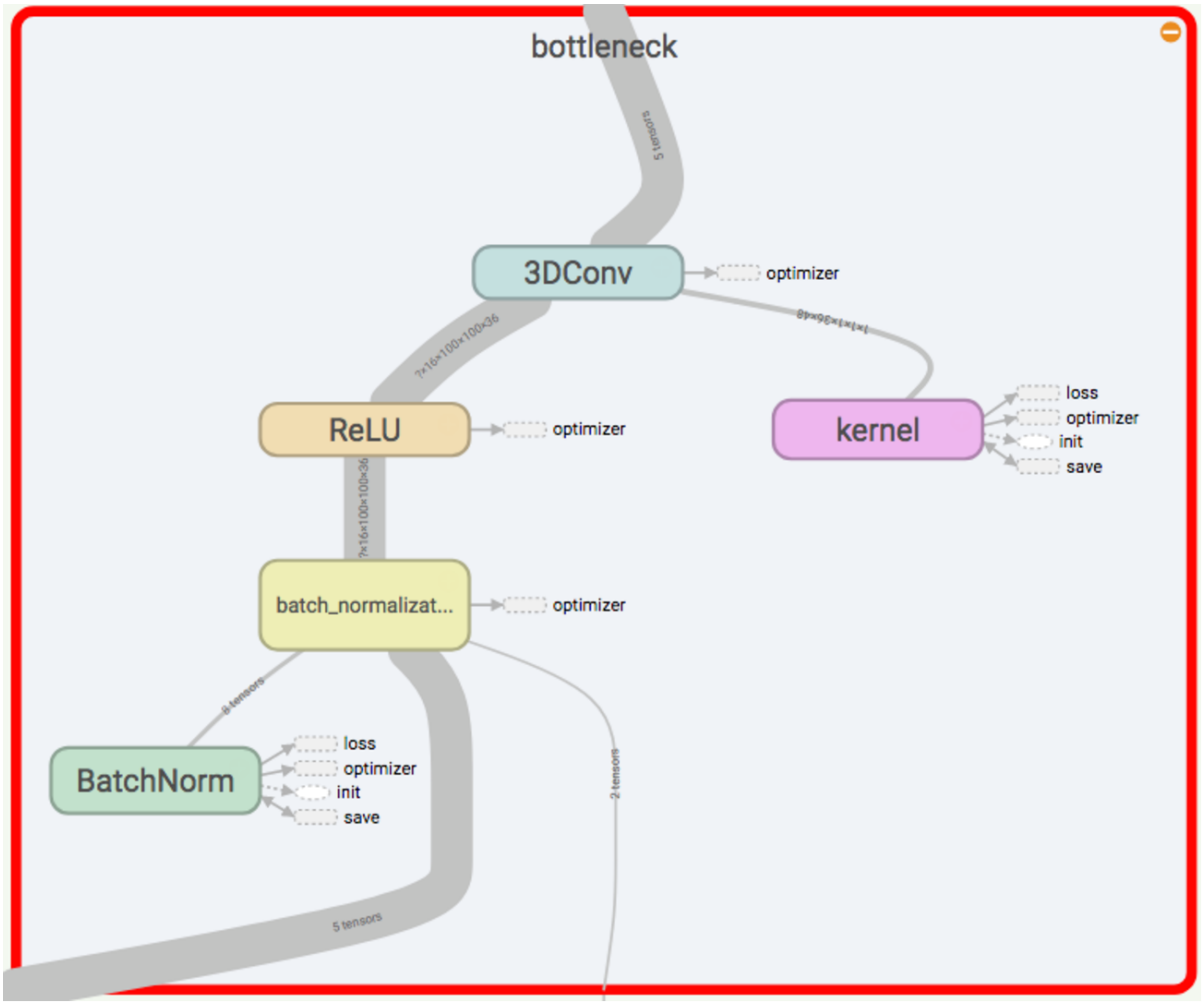


Figure 4.8: The bottleneck layer inside block layer in Figure 4.6.

The **block** (Figure 4.6) consists of two layers, with each layer taking the output from the previous layer as the input, and processing the data with two functions, the bottleneck function (Figure 4.8) and the composite function (Figure 4.7). Both of these functions have BN, ReLU, and 3D convolution operations; they are almost identical to each other, except that the 3D convolution operation inside the bottleneck function has a kernel size 1 instead of 3 in the composite function. The concat operation inside each layer concatenates the input of this layer and the output from the composite function along the last dimension to create the output of this layer.

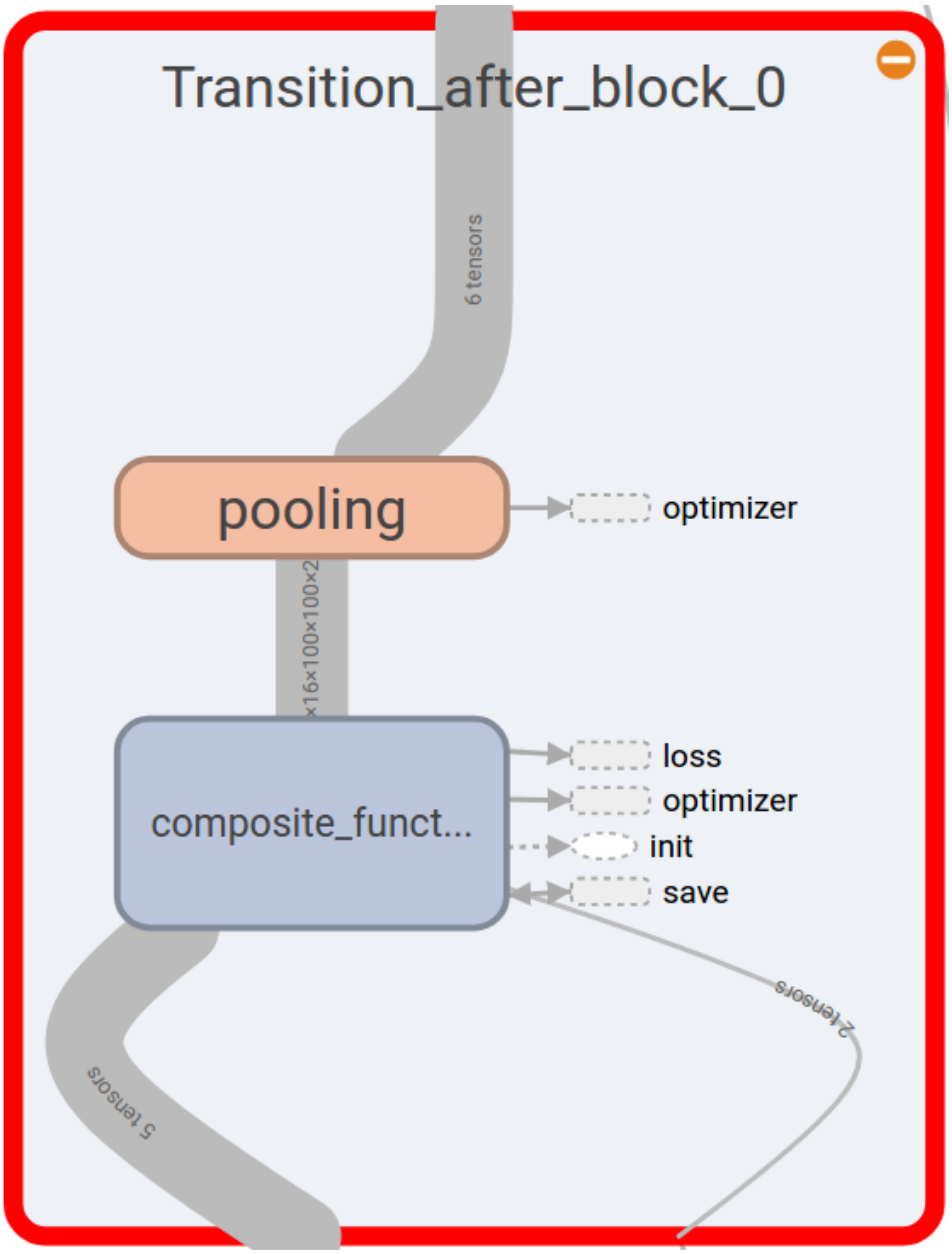


Figure 4.9: The Transition layer after each block of the general 3D-DenseNet-BC network.

The **transition** (Figure 4.9) layer after each block has a composite function, followed by a 3D average pooling operation. The composite function is the same as the composite function inside the layer we described above. The 3D pooling operation has a kernel with size $1 * 2 * 2$ (depth * height * width) at the first transition layer and a kernel with size $2 * 2 * 2$ at the second transition layer.

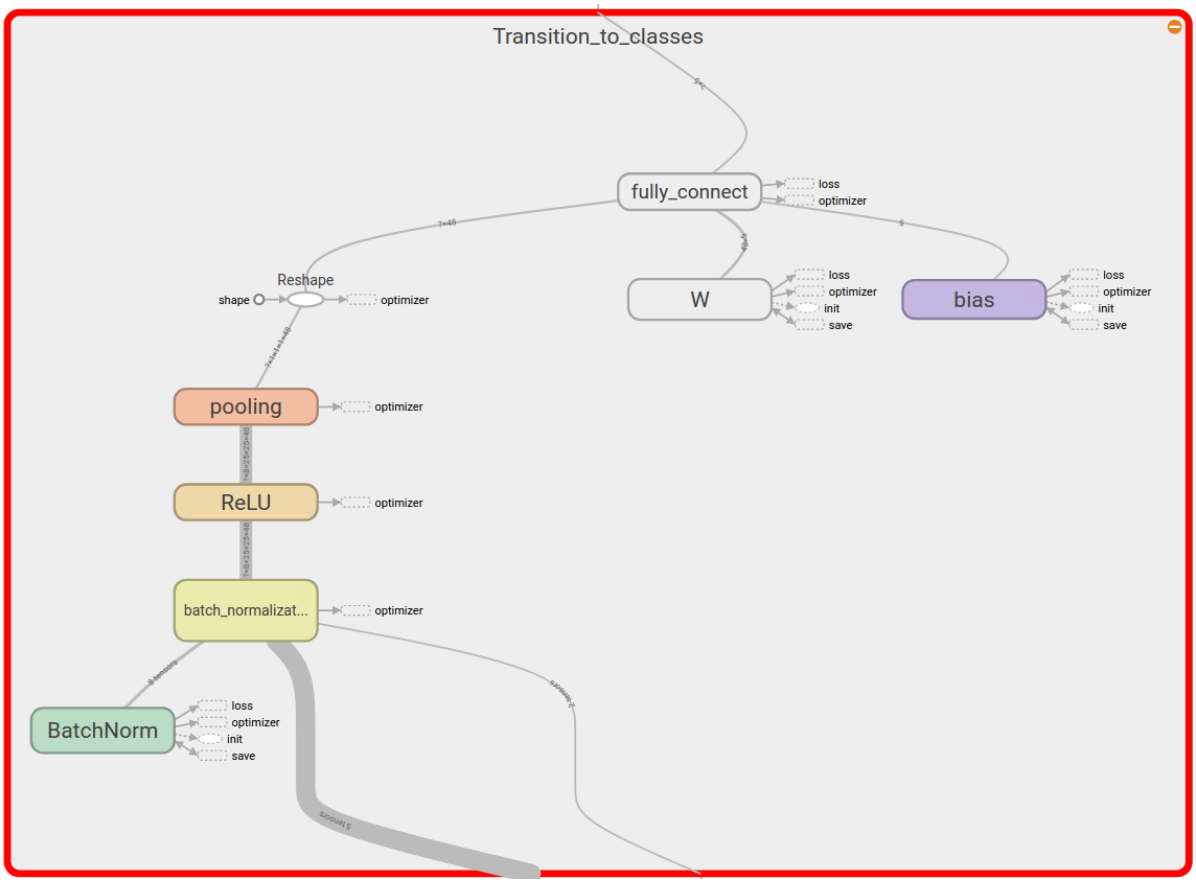


Figure 4.10: The Transition layer after the last block of the general 3D-DenseNet-BC network.

Transition to classes (Figure 4.10) is the transition layer right after the last block of the network. It takes the output from the last block as input and performs BN, ReLU, and 3D pooling operations sequentially. The 3D pooling operation will flatten the data into a 1-D array and use this array as an input for the fully connected layer. This fully connected layer connects the 1-D array and the number of classes to produce the score of each class.

Softmax and prediction operations normalize the score of each class and produce the prediction result.

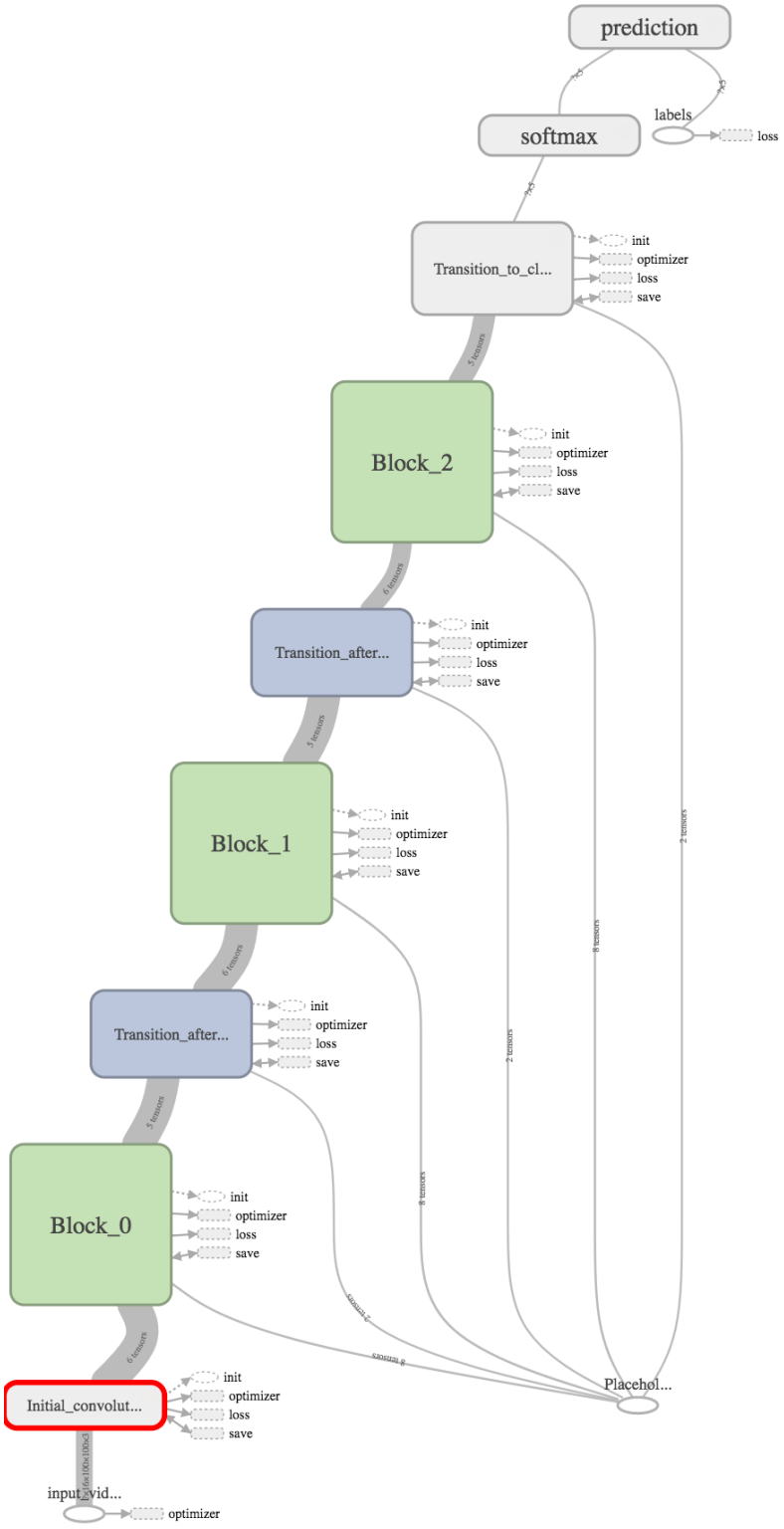


Figure 4.11: TensorFlow main graph of the general 3D-DenseNet-BC network.

4.5 Classification Results

We train 3D-DenseNets with different depths, d, and growth rates, k. The main results on the MERL shopping and KTH datasets are shown in Table 4.2. We use **boldface** to highlight the best result on both datasets.

Method	Growth Rate (k)	Depth (d)	Params	MERL	KTH
(1)General 3D-DenseNet	12	20	0.6M	75.54%	-
(2)General 3D-DenseNet-BC	12	20	0.1M	75.59%	-
(3)General 3D-DenseNet-BC	12	20	0.1M	-	87.01%
(4)General 3D-DenseNet-BC	12	40	0.4M	76.98%	-
(5)General 3D-DenseNet-BC	24	20	0.4M	77.32%	-
(6)General 3D-DenseNet-BC	24	30	0.9M	-	**90.56**%
(7)General 3D-DenseNet-BC	24	30	0.9M	-	89.59%
(8)General 3D-DenseNet-BC	24	40	1.4M	78.22%	-
(9)Lite 3D-DenseNet-BC	12	20	0.1M	70.46%	-
(10)Lite 3D-DenseNet-BC	24	40	1.5M	73.31%	-
(11)Lite 3D-DenseNet-BC	24	40	1.5M	**84.32%**	-
(12)Lite 3D-DenseNet-BC	24	40	1.5M	78.35%	-
(13)Lite 3D-DenseNet-BC	24	40	1.5M	78.63%	-
(14)Lite 3D-DenseNet-BC	24	40	1.5M	79.91%	-
(15)Lite 3D-DenseNet-BC	24	100	5.1M	72.50%	-

Table 4.2: Mean accuracy precision (%) on the MERL shopping and KTH datasets. d denotes the network depth, and k is the growth rate. The number at the beginning of each network is the reference number that corresponds to the number in Table 4.1. In addition, all experiment graphs can be found in Appendix A. The best results are in **bold**.

Method	MERL	KTH
Two-Stream [64]	65.21%	-
MSB-RNN [69]	80.31%	-
Niebles et al. [55]	-	81.50%
Jhuang et al. [35]	-	**91.70**%
(6)General 3D-DenseNet-BC	-	90.56%
(11)Lite 3D-DenseNet-BC	**84.32%**	-

Table 4.3: Comparison of performance of our 3D-DenseNet with previous action detection methods on the MERL shopping and KTH dataset. Mean Average Precision (%) is reported. The number at the beginning of each network is the reference number that corresponds to the number in Table 4.2. The best results are in **bold**.

Testing. Notice that our testing dataset for MERL follows the same splitting rate as that in [69]. MERL has 41 subjects in total. Subjects 1 to 20 have 60 videos, and they belong to the training set. Subjects 21 to 26 belong to the validation set, and subjects 27 to 41 belong to the testing set. For the KTH dataset, we use the first 16 subjects as the training set and the last 9 ones as the testing set.

Accuracy. According to Table 4.3, our best result on the MERL dataset performs better than the existing state-of-the-art method by 4%. Notably, our network takes only the RGB image as input instead of the four different channels shown in MSB-RNN [69]. Our method also achieves a competitive result (90.56%) on the KTH dataset and has the potential to reach a higher accuracy by testing different configurations of the network.

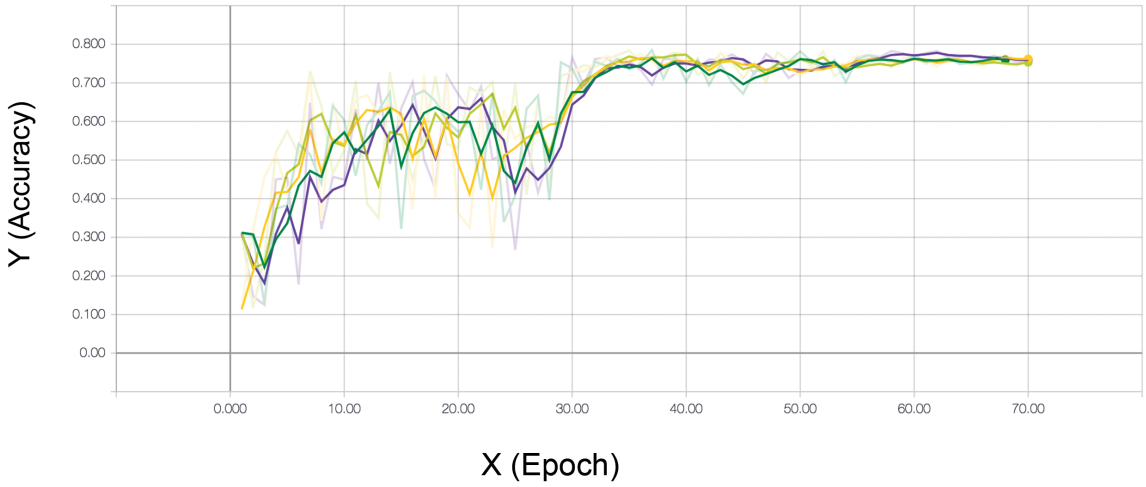


Figure 4.12: The MERL dataset: The test mean accuracy precision of all general 3D-DenseNet-BC networks. green (reference 2 in Table 4.2): $k = 12, d = 20$. lime (reference 5 in Table 4.2): $k = 24, d = 20$. violet (reference 8 in Table 4.2): $k = 24, d = 40$. yellow (reference 4 in Table 4.2): $k = 12, d = 40$.

Capacity. By analyzing Table 4.2, we can observe the general trend that 3D-DenseNets perform better as k and d increase (Figure 4.12). We believe that this is primarily due to the growth in model capacity, which is the parameters column in Table 4.2. Examples can be found in the MERL column. In MERL, the mean accuracy precision increases from 75.59% to 76.98% and finally to 78.22% as the number of parameters increases from $0.1M$, over $0.4M$ to $1.4M$. In the KTH dataset, we notice a similar trend. This observation shows that 3D-DenseNets can use the increased representational power of larger and deeper models.

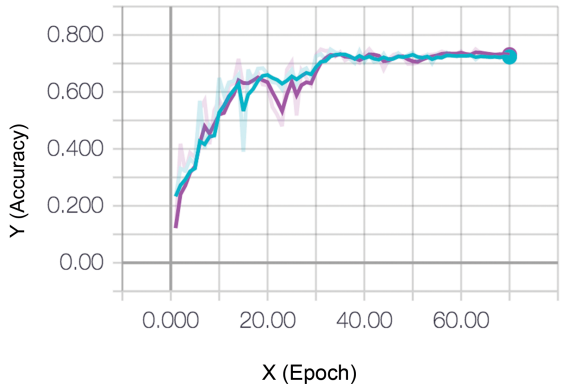


Figure 4.13: The MERL dataset: The test mean accuracy precision of lite 3D-DenseNet-BC with $k = 24, d = 40$ (violet: reference 10 in Table 4.2) and lite 3D-DenseNet-BC with $k = 24, d = 100$ (cyan: reference 15 in Table 4.2). The x axis represents the number of epochs, and the y axis represents the mean accuracy precision.

Overfitting. In the MERL dataset, we notice that the accuracy of the lite 3D-DenseNet-BC model decreases from 73.31% to 72.50% as the number of parameters increases from $1.5M$ to $5.1M$. This result is probably caused by overfitting. However, the accuracy difference is less than 1% (Figure 4.13) compared with the $3x$ difference in the number of parameters, indicating that the 3D-DenseNet bottleneck and compression layers are likely an effective way to reduce overfitting.

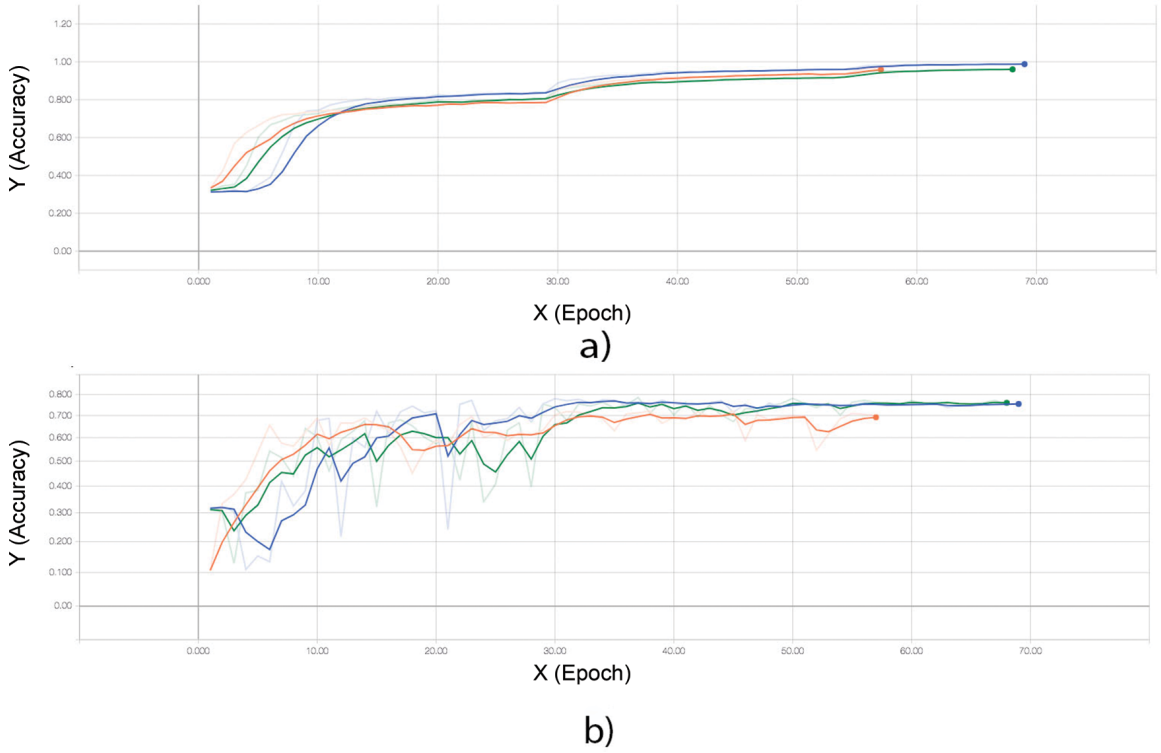


Figure 4.14: The MERL dataset: Comparison between the general 3D-DenseNet (blue: reference 1 in Table 4.2), the general 3D-DenseNet-BC (green: reference 2 in Table 4.2), and lite 3D-DenseNet-BC (orange: reference 9 in Table 4.2). The x axis represents the number of epochs and the y axis represents the mean accuracy precision. a) is the training accuracy and b) is the testing accuracy.

Parameter Efficiency. The general 3D-DenseNet and general 3D-DenseNet-BC both have the same growth rate and depth. However, bottleneck and compression reduce the number of parameters from $0.6M$ to $0.1M$ while keeping the same level of accuracy as shown in Figure 4.14 b. The blue line (general 3D-DenseNet) and the green line (general 3D-DenseNet-BC) almost overlap with each other in terms of accuracy. This observation shows that bottleneck and compression layers can increase parameter efficiency and reach a higher accuracy with the same level of parameters.

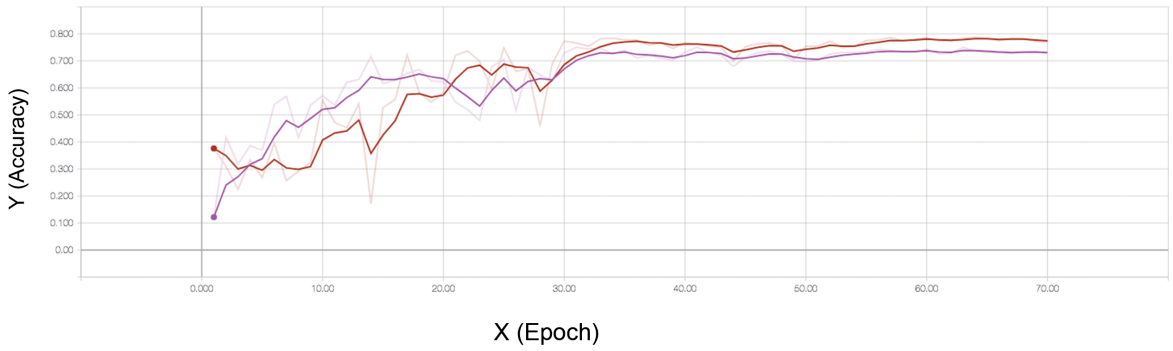


Figure 4.15: The MERL dataset: Comparison between two lite 3D-DenseNets with different crop sizes. Both of the lite 3D-DenseNets have the same depth (40) and growth rate (24). The red line is lite 3D-DenseNet-BC with $(256, 256)$ crop size, and the violet is lite 3D-DenseNet-BC with $(128, 128)$ crop size.

Information loss. In Figure 4.14, we observe that the general 3D-DenseNet-BC (green) and lite 3D-DenseNet-BC (orange) have the same depth and growth rate but their accuracies are quite different. We believe that this occurs because lite 3D-DenseNet has a 3D convolution layer with a stride of $1 * 2 * 2$, followed by a 3D pooling layer at the beginning of the network, which reduce the size of the input too early and cause information loss. To avoid information loss, we can also increase the crop size of the input image. In Figure 4.15, when we change the crop size from $(128, 128)$ to $(256, 256)$, the performance increases by around 5%.

It is worth noting that our program is implemented in TensorFlow. All models need to be loaded on the GPU memory before we train our network. The size of the GPU memory limits the possibility of testing a deeper networks or a larger growth rate. More extensive hyper-parameter searches may further improve the performance of 3D-DenseNet. In addition, we also repeat some experiments and find that the accuracy result tends to fluctuate slightly ($\pm 1\%$).

Chapter 5

Summary and Conclusion

5.1 Summary

The approach presented in this thesis is drawn from the DenseNet architecture from [32] and expands it to 3D-DenseNet by adding a temporal dimension to all the convolution and pooling layers in DenseNet for action recognition. The direct connections between any two layers enable 3D-DenseNet to scale to hundreds of layers with no optimization difficulties. Our experiment focuses on the MERL shopping dataset. By testing multiple configurations, 3D-DenseNet has been able to performs better than the state-of-the-art method on the MERL shopping dataset and achieves competitive results on the KTH dataset. In addition, our method can not only tackle human shopping actions, but also has the potential to apply on other actions, such as abnormal behaviors, for security monitoring.

5.1.1 Limitations

In general, 3D-DenseNet performs better than the state-of-the-art method on the MERL shopping dataset, but has a rigid temporal structure. We need to predefine the sequence length. This process may not work well on some actions that happen within a small amount of time or have different speeds. Another limitation of 3D-DenseNet is the important requirement of GPU memory because the depth of our model is significantly larger than general deep networks, and our input source is video clips instead of images. We need to use a small batch size and resize the video so that we can successfully train our model.

5.1.2 Future works

One potential future work for 3D-DenseNet is trying a deeper networks. DenseNet [32] has reached 250 layers on image classification. According to our experiment, 3D-DenseNet tends to yield a consistent improvement in accuracy with an increasing number of depth and growth rate, without performance degradation or overfitting. Another future work on 3D-DenseNet can be the use of a multi-stream network [69] as input, such as optical flow or audio clues. The state-of-the-art method [69] of the MERL shopping dataset proposes a multi-stream network that uses four different streams of information to boost the performance significantly. The first and second streams of [69] are the original RGB image and segmentation of RGB image. The third and fourth streams of [69] are the pixel trajectory and segmentation of pixel trajectory.

References

[1] Saad Ali and Mubarak Shah. "Human action recognition in videos using kinematic features and multiple instance learning". In: *IEEE transactions on pattern analysis and machine intelligence* 32.2 (2010), pp. 288–303.

[2] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. "Sequential deep learning for human action recognition". In: *International Workshop on Human Behavior Understanding*. Springer. 2011, pp. 29–39.

[3] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[4] Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. "Actions as space-time shapes". In: *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*. Vol. 2. IEEE. 2005, pp. 1395–1402.

[5] Aaron F. Bobick and James W. Davis. "The recognition of human movement using temporal templates". In: *IEEE Transactions on pattern analysis and machine intelligence* 23.3 (2001), pp. 257–267.

[6] Oren Boiman and Michal Irani. "Detecting irregularities in images and in video". In: *International journal of computer vision* 74.1 (2007), pp. 17–31.

[7] Matthew Brand, Nuria Oliver, and Alex Pentland. "Coupled hidden Markov models for complex action recognition". In: *Computer vision and pattern recognition, 1997. proceedings., 1997 ieee computer society conference on*. IEEE. 1997, pp. 994–999.

[8] Alexia Briassouli, Vagia Tsiminaki, and Ioannis Kompatsiaris. "Human motion analysis via statistical motion processing and sequential change detection". In: *EURASIP Journal on Image and Video Processing* 2009.1 (2009), p. 652050.

[9] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Return of the devil in the details: Delving deep into convolutional nets". In: *arXiv preprint arXiv:1405.3531* (2014).

[10] Navneet Dalal and Bill Triggs. “Histograms of oriented gradients for human detection”. In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[11] Navneet Dalal, Bill Triggs, and Cordelia Schmid. “Human detection using oriented histograms of flow and appearance”. In: *European conference on computer vision*. Springer. 2006, pp. 428–441.

[12] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. “Imagenet: A large-scale hierarchical image database”. In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 248–255.

[13] Piotr Dollár, Vincent Rabaud, Garrison Cottrell, and Serge Belongie. “Behavior recognition via sparse spatio-temporal features”. In: *Visual Surveillance and Performance Evaluation of Tracking and Surveillance, 2005. 2nd Joint IEEE International Workshop on*. IEEE. 2005, pp. 65–72.

[14] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. “DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition.” In: *Icml*. Vol. 32. 2014, pp. 647–655.

[15] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. “Long-term recurrent convolutional networks for visual recognition and description”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 2625–2634.

[16] Gianfranco Doretto, Alessandro Chiuso, Ying Nian Wu, and Stefano Soatto. “Dynamic textures”. In: *International Journal of Computer Vision* 51.2 (2003), pp. 91–109.

[17] Alexei A Efros, Alexander C Berg, Greg Mori, Jitendra Malik, et al. “Recognizing Action at a Distance.” In: *ICCV*. Vol. 3. 2003, pp. 726–733.

[18] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. “Convolutional two-stream network fusion for video action recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1933–1941.

[19] Ken-ichi Funahashi and Yuichi Nakamura. “Approximation of dynamical systems by continuous time recurrent neural networks”. In: *Neural networks* 6.6 (1993), pp. 801–806.

[20] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks." In: *Aistats*. Vol. 15. 106. 2011, p. 275.

[21] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[22] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. "Bidirectional LSTM networks for improved phoneme classification and recognition". In: *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005* (2005), pp. 753–753.

[23] S Gross and M Wilber. "Training and investigating residual nets". In: *Facebook AI Research, CA.[Online]. Avilable: http://torch. ch/blog/2016/02/04/resnets. html* (2016).

[24] Yan Guo, Gang Xu, and Saburo Tsuji. "Understanding human motion patterns". In: *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*. Vol. 2. IEEE. 1994, pp. 325–329.

[25] Chris Harris and Mike Stephens. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[28] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[29] David Hogg. "Model-based vision: a program to see a walking person". In: *Image and Vision computing* 1.1 (1983), pp. 5–20.

[30] Yuxiao Hu, Liangliang Cao, Fengjun Lv, Shuicheng Yan, Yihong Gong, and Thomas S Huang. "Action detection in complex scenes with spatial and temporal ambiguities". In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 128–135.

[31] Feiyue Huang and Guangyou Xu. "Viewpoint insensitive action recognition using envelop shape". In: *Computer Vision–ACCV 2007* (2007), pp. 477–486.

[32] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks". In: *arXiv preprint arXiv:1608.06993* (2016).

[33] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[34] H Jhuang, H Garrote, E Poggio, T Serre, and T Hmdb. "A large video database for human motion recognition". In: *Proc. of IEEE International Conference on Computer Vision*. Vol. 4. 2011, p. 6.

[35] Hueihan Jhuang, Thomas Serre, Lior Wolf, and Tomaso Poggio. "A biologically inspired system for action recognition". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. Ieee. 2007, pp. 1–8.

[36] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. "3D convolutional neural networks for human action recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013), pp. 221–231.

[37] Gunnar Johansson. "Visual perception of biological motion and a model for its analysis". In: *Perception & psychophysics* 14.2 (1973), pp. 201–211.

[38] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. "Large-scale video classification with convolutional neural networks". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2014, pp. 1725–1732.

[39] Alexander Klaser, Marcin Marszałek, and Cordelia Schmid. "A spatio-temporal descriptor based on 3d-gradients". In: *BMVC 2008-19th British Machine Vision Conference*. British Machine Vision Association. 2008, pp. 275–1.

[40] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009).

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[42] Ivan Laptev. "On space-time interest points". In: *International journal of computer vision* 64.2-3 (2005), pp. 107–123.

[43] Ivan Laptev and Tony Lindeberg. "Local descriptors for spatio-temporal recognition". In: *Spatial Coherence for Visual Motion Analysis*. Springer, 2006, pp. 91–103.

[44] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[45] Jingen Liu, Jiebo Luo, and Mubarak Shah. "Recognizing realistic actions from videos "in the wild"". In: *Computer vision and pattern recognition, 2009. CVPR 2009. IEEE conference on*. IEEE. 2009, pp. 1996–2003.

[46] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[47] David Marr and Herbert Keith Nishihara. "Representation and recognition of the spatial organization of three-dimensional shapes". In: *Proceedings of the Royal Society of London B: Biological Sciences* 200.1140 (1978), pp. 269–294.

[48] Marcin Marszalek, Ivan Laptev, and Cordelia Schmid. "Actions in context". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE. 2009, pp. 2929–2936.

[49] Michael Mathieu, Camille Couprie, and Yann LeCun. "Deep multi-scale video prediction beyond mean square error". In: *arXiv preprint arXiv:1511.05440* (2015).

[50] Hongying Meng, Nick Pears, and Chris Bailey. "A human action recognition system for embedded computer vision application". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–6.

[51] Ross Messing, Chris Pal, and Henry Kautz. "Activity recognition using the velocity histories of tracked keypoints". In: *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 104–111.

[52] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černock, and Sanjeev Khudanpur. "Extensions of recurrent neural network language model". In: *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. IEEE. 2011, pp. 5528–5531.

[53] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. "Reading digits in natural images with unsupervised feature learning". In: *NIPS workshop on deep learning and unsupervised feature learning*. Vol. 2011. 2. 2011, p. 5.

[54] Juan Carlos Niebles, Chih-Wei Chen, and Li Fei-Fei. "Modeling temporal structure of decomposable motion segments for activity classification". In: *European conference on computer vision*. Springer. 2010, pp. 392–405.

[55] Juan Carlos Niebles, Hongcheng Wang, and Li Fei-Fei. "Unsupervised learning of human action categories using spatial-temporal words". In: *International journal of computer vision* 79.3 (2008), pp. 299–318.

[56] Ramprasad Polana and Randal Nelson. "Low level recognition of human motion (or how to get your man without finding his body parts)". In: *Motion of Non-Rigid and Articulated Objects, 1994., Proceedings of the 1994 IEEE Workshop on*. IEEE. 1994, pp. 77–82.

[57] Deva Ramanan and David A Forsyth. *Automatic annotation of everyday movements*. Computer Science Division, University of California, 2003.

[58] Kishore K Reddy and Mubarak Shah. "Recognizing 50 human action categories of web videos". In: *Machine Vision and Applications* 24.5 (2013), pp. 971–981.

[59] Anthony J Robinson and F Failside. "Static and Dynamic Error Propagation Networks with Application to Speech Coding." In: *NIPS*. 1987, pp. 632–641.

[60] Mikel D Rodriguez, Javed Ahmed, and Mubarak Shah. "Action mach a spatio-temporal maximum average correlation height filter for action recognition". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8.

[61] Karl Rohr. "Towards model-based recognition of human movements in image sequences". In: *CVGIP: Image understanding* 59.1 (1994), pp. 94–115.

[62] Sreemanananth Sadanand and Jason J Corso. "Action bank: A high-level representation of activity in video". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1234–1241.

[63] Christian Schuldt, Ivan Laptev, and Barbara Caputo. "Recognizing human actions: A local SVM approach". In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 3. IEEE. 2004, pp. 32–36.

[64] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE Transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[65] Paul Scovanner, Saad Ali, and Mubarak Shah. "A 3-dimensional sift descriptor and its application to action recognition". In: *Proceedings of the 15th ACM international conference on Multimedia*. ACM. 2007, pp. 357–360.

[66] Feng Shi, Robert Laganiere, and Emil Petriu. "Gradient boundary histograms for action recognition". In: *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*. IEEE. 2015, pp. 1107–1114.

[67] Feng Shi, Robert Laganière, and Emil Petriu. "Local Part Model for Action Recognition". In: *Image Vision Comput.* 46.C (Feb. 2016), pp. 18–28. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2015.11.010. URL: http://dx.doi.org/10.1016/j.imavis.2015.11.010.

[68] Karen Simonyan and Andrew Zisserman. "Two-stream convolutional networks for action recognition in videos". In: *Advances in neural information processing systems*. 2014, pp. 568–576.

[69] Bharat Singh, Tim K Marks, Michael Jones, Oncel Tuzel, and Ming Shao. "A multi-stream bi-directional recurrent neural network for fine-grained action detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1961–1970.

[70] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A dataset of 101 human actions classes from videos in the wild". In: *arXiv preprint arXiv:1212.0402* (2012).

[71] Richard Souvenir and Justin Babbs. "Learning the viewpoint manifold for action recognition". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–7.

[72] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. "Unsupervised learning of video representations using lstms". In: *International Conference on Machine Learning*. 2015, pp. 843–852.

[73] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Training very deep networks". In: *Advances in neural information processing systems*. 2015, pp. 2377–2385.

[74] Lin Sun, Kui Jia, Dit-Yan Yeung, and Bertram E Shi. "Human action recognition using factorized spatio-temporal convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4597–4605.

[75] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning". In: *International conference on machine learning*. 2013, pp. 1139–1147.

[76] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*. 2014, pp. 3104–3112.

[77] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

[78] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. "Learning spatiotemporal features with 3d convolutional networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4489–4497.

[79] Hirofumi Uemura, Seiji Ishikawa, and Krystian Mikolajczyk. "Feature Tracking and Motion Compensation for Action Recognition." In: *BMVC*. 2008, pp. 1–10.

[80] Gül Varol, Ivan Laptev, and Cordelia Schmid. "Long-term temporal convolutions for action recognition". In: *arXiv preprint arXiv:1604.04494* (2016).

[81] Subhashini Venugopalan, Huijuan Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, and Kate Saenko. "Translating videos to natural language using deep recurrent neural networks". In: *arXiv preprint arXiv:1412.4729* (2014).

[82] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103.

[83] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. "Action recognition by dense trajectories". In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 3169–3176.

[84] Heng Wang, Muhammad Muneeb Ullah, Alexander Klaser, Ivan Laptev, and Cordelia Schmid. "Evaluation of local spatio-temporal features for action recognition". In: *BMVC 2009-British Machine Vision Conference*. BMVA Press. 2009, pp. 124–1.

[85] Liang Wang and David Suter. "Informative shape representations for human action recognition". In: *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*. Vol. 2. IEEE. 2006, pp. 1266–1269.

[86] Ying Wang, Kaiqi Huang, and Tieniu Tan. "Human activity recognition based on r transform". In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8.

[87] Chris Whiten, Robert Laganiere, and Guillaume-Alexandre Bilodeau. "Efficient action recognition with MoFREAK". In: *Computer and Robot Vision (CRV), 2013 International Conference on*. IEEE. 2013, pp. 319–325.

[88] Geert Willems, Tinne Tuytelaars, and Luc Van Gool. "An efficient dense and scale-invariant spatio-temporal interest point detector". In: *Computer Vision–ECCV 2008* (2008), pp. 650–663.

[89] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, Xiangyang Xue, and Jun Wang. "Fusing multi-stream deep networks for video classification". In: *arXiv preprint arXiv:1509.06086* (2015).

[90] Junji Yamato, Jun Ohya, and Kenichiro Ishii. "Recognizing human action in time-sequential images using hidden markov model". In: *Computer Vision and Pattern Recognition, 1992. Proceedings CVPR'92., 1992 IEEE Computer Society Conference on*. IEEE. 1992, pp. 379–385.

[91] Xing Yan, Hong Chang, Shiguang Shan, and Xilin Chen. "Modeling video dynamics with deep dynencoder". In: *European Conference on Computer Vision*. Springer. 2014, pp. 215–230.

[92] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. "Beyond short snippets: Deep networks for video classification". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 4694–4702.

[93] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

Appendices

Appendix A

Experiment result

In the chapter, we append the training accuracy, validation (testing) accuracy during training, training loss and validation (testing) loss of all 3D-DenseNets shown in Table 4.2.

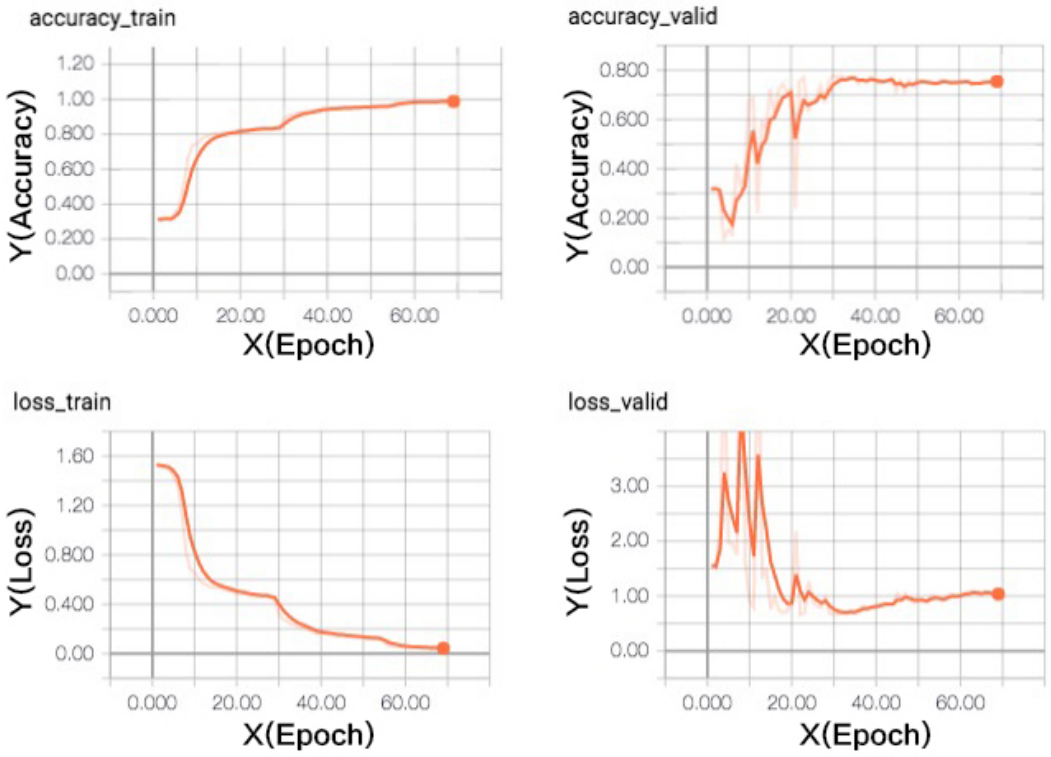


Figure A.1: The result of 3D-DenseNet with reference number 1 in Table 4.2

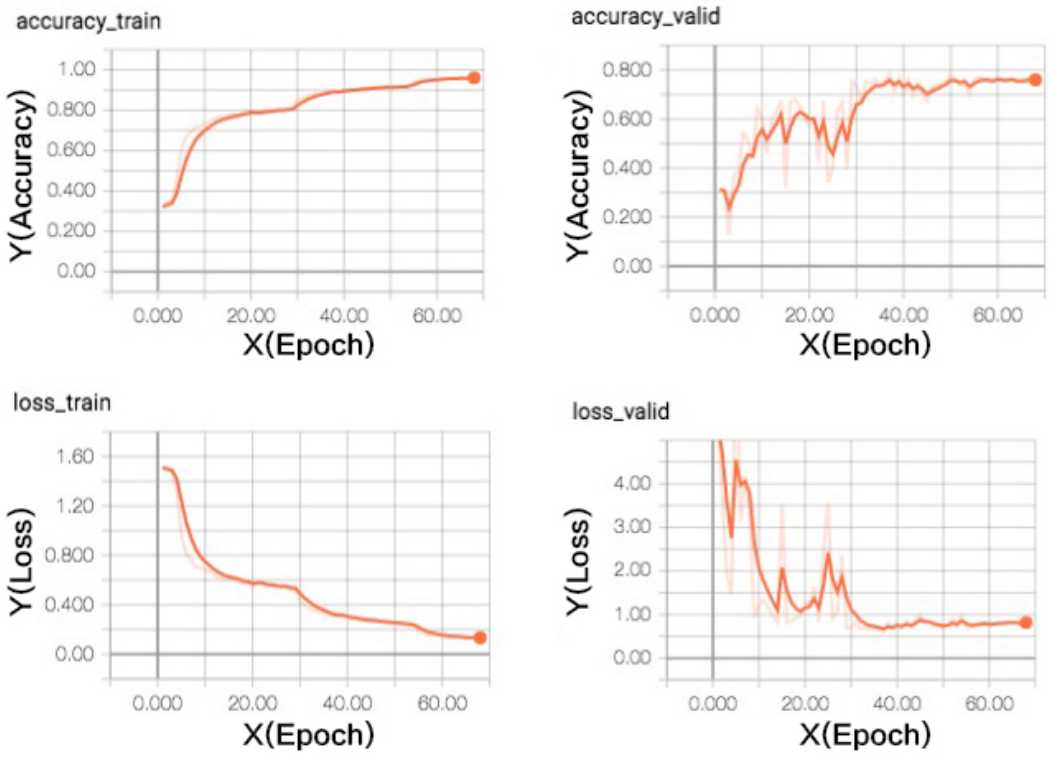


Figure A.2: The result of 3D-DenseNet with reference number 2 in Table 4.2

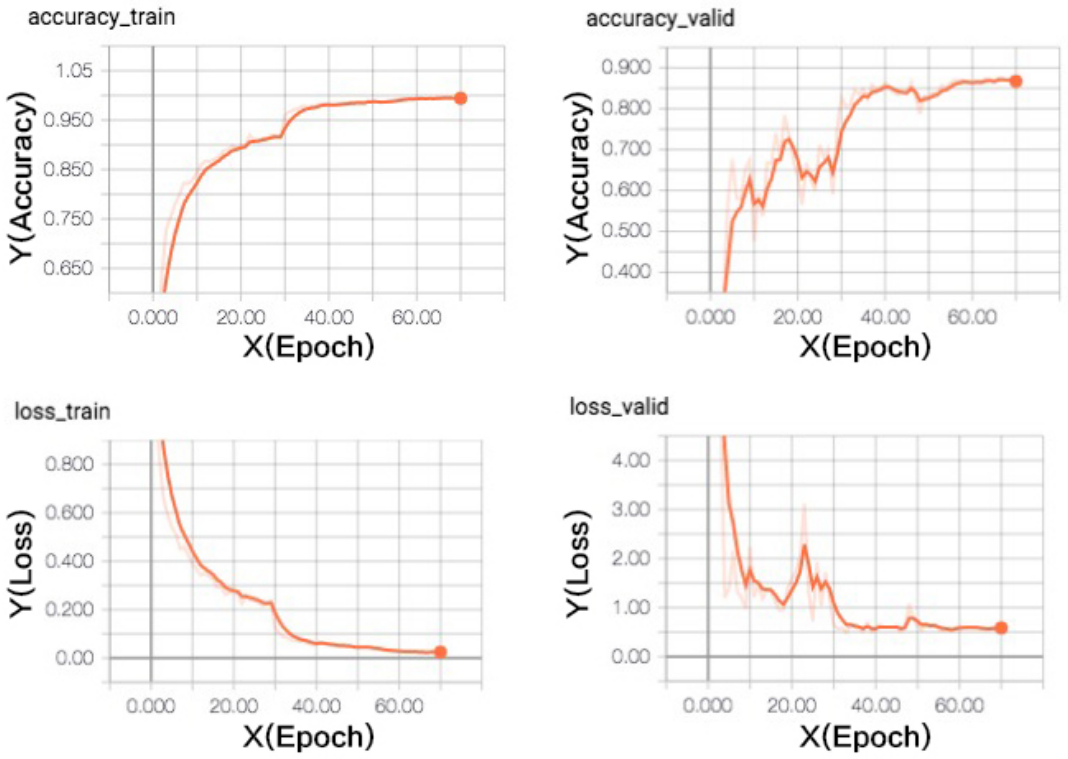


Figure A.3: The result of 3D-DenseNet with reference number 3 in Table 4.2

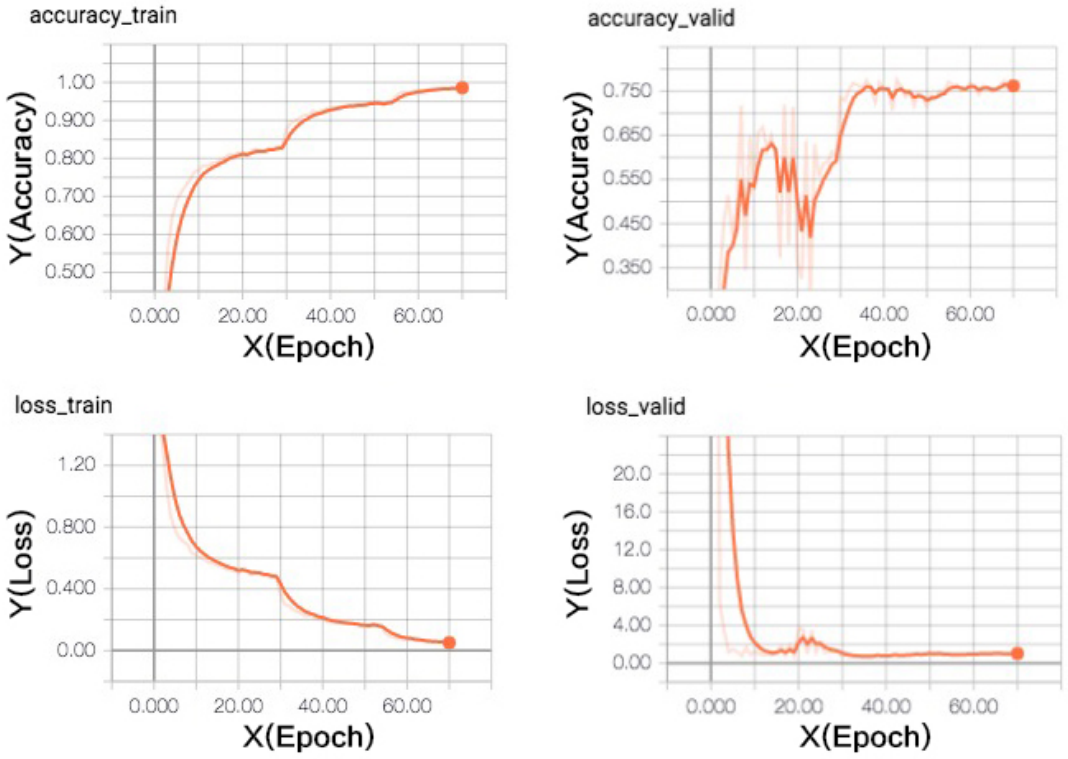


Figure A.4: The result of 3D-DenseNet with reference number 4 in Table 4.2

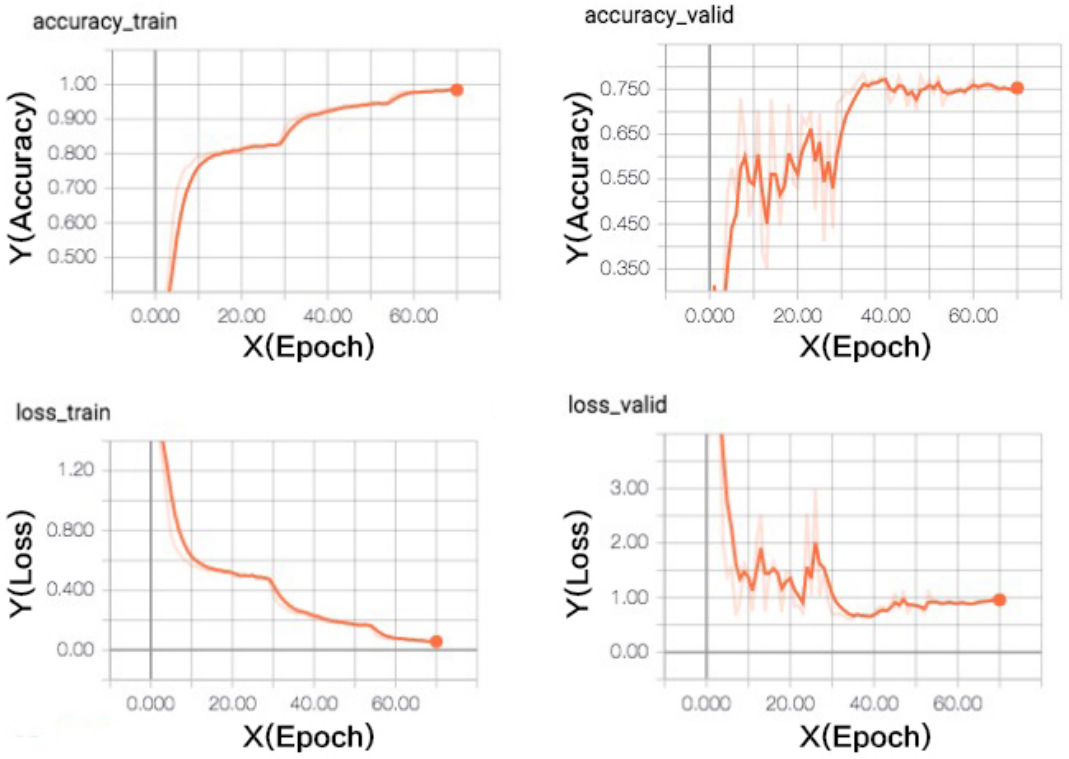


Figure A.5: The result of 3D-DenseNet with reference number 5 in Table 4.2

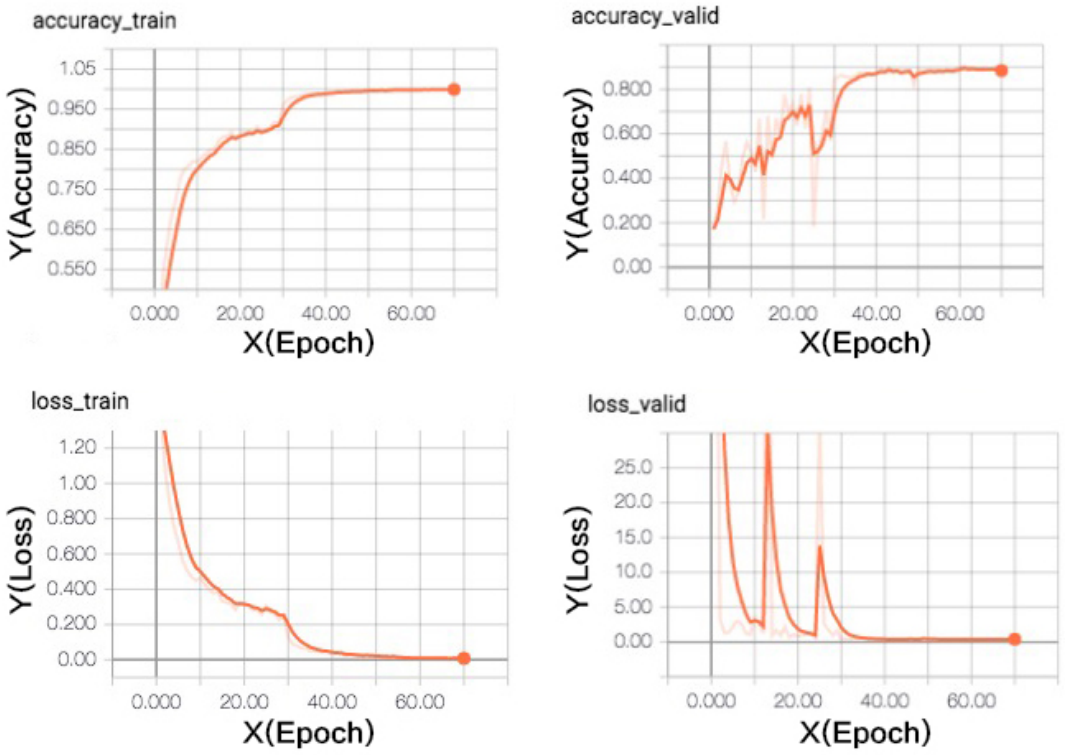


Figure A.6: The result of 3D-DenseNet with reference number 6 in Table 4.2

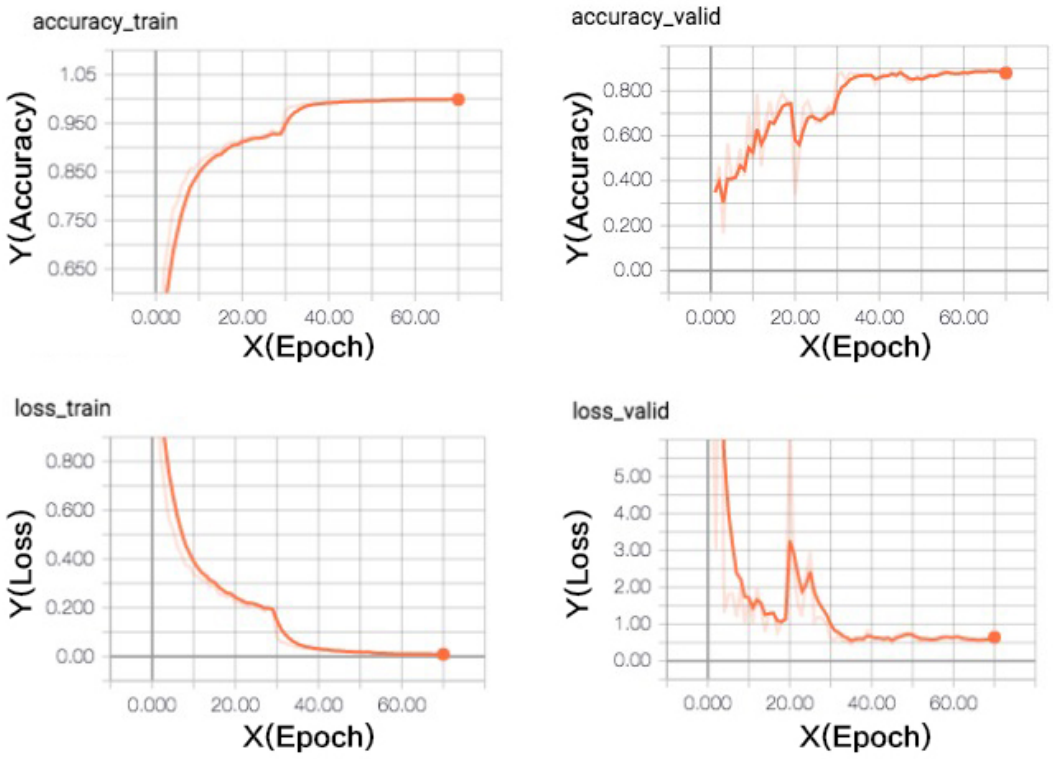


Figure A.7: The result of 3D-DenseNet with reference number 7 in Table 4.2

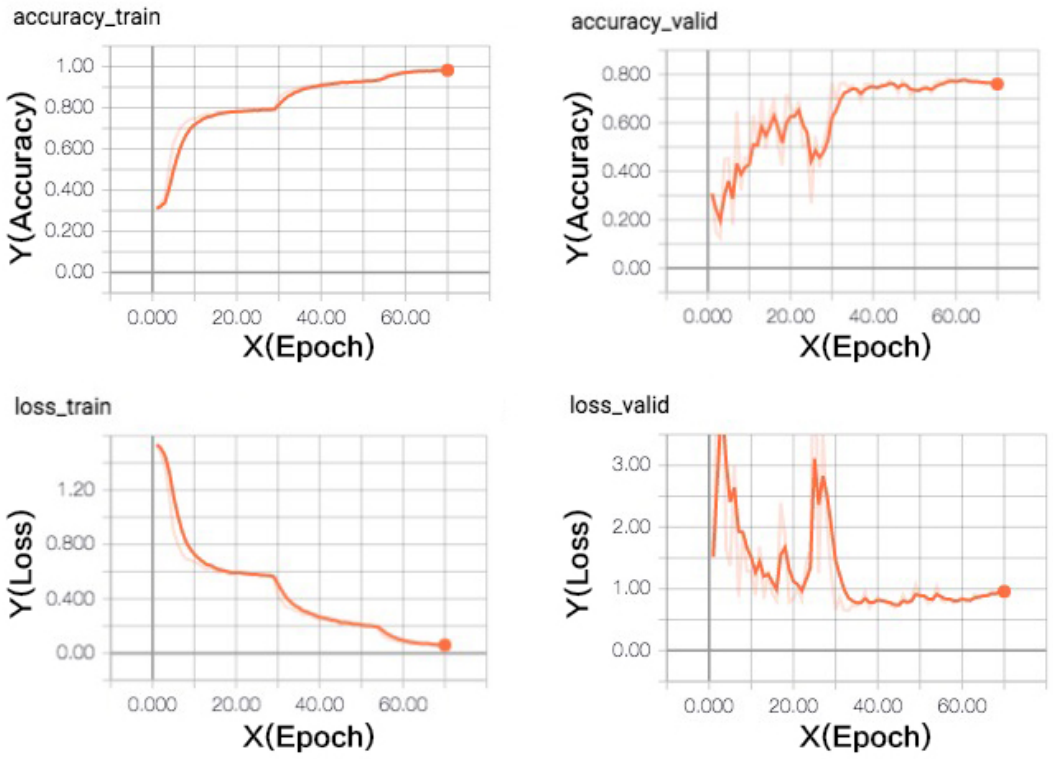


Figure A.8: The result of 3D-DenseNet with reference number 8 in Table 4.2

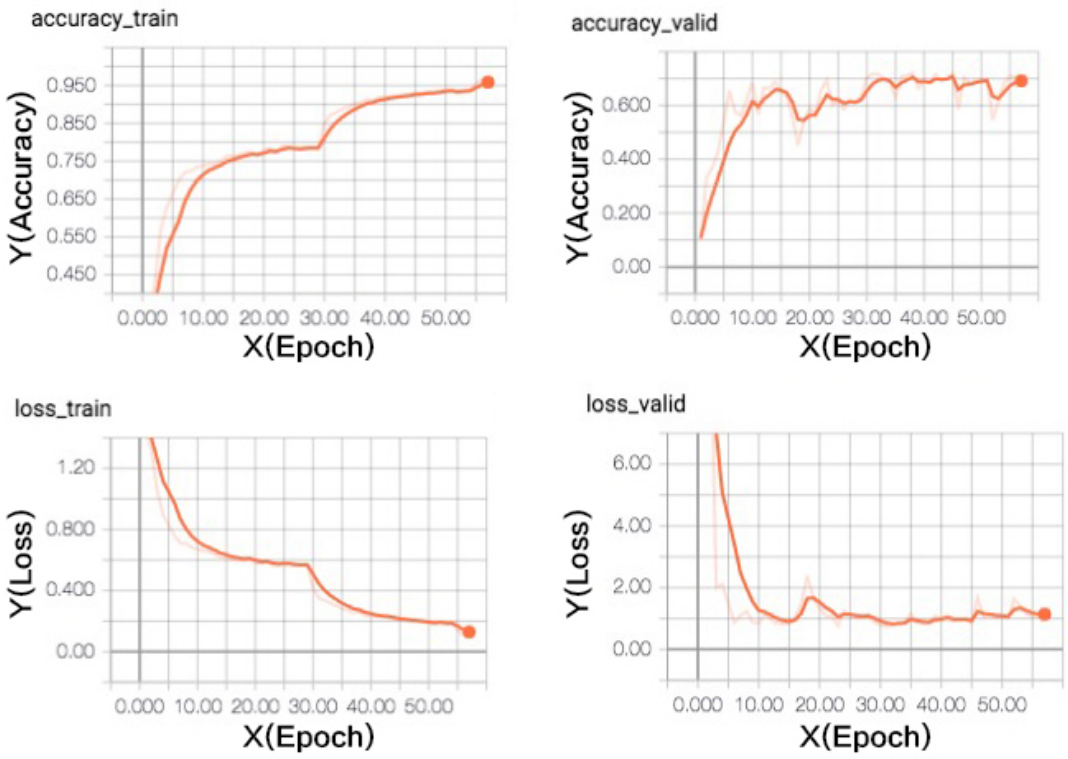


Figure A.9: The result of 3D-DenseNet with reference number 9 in Table 4.2

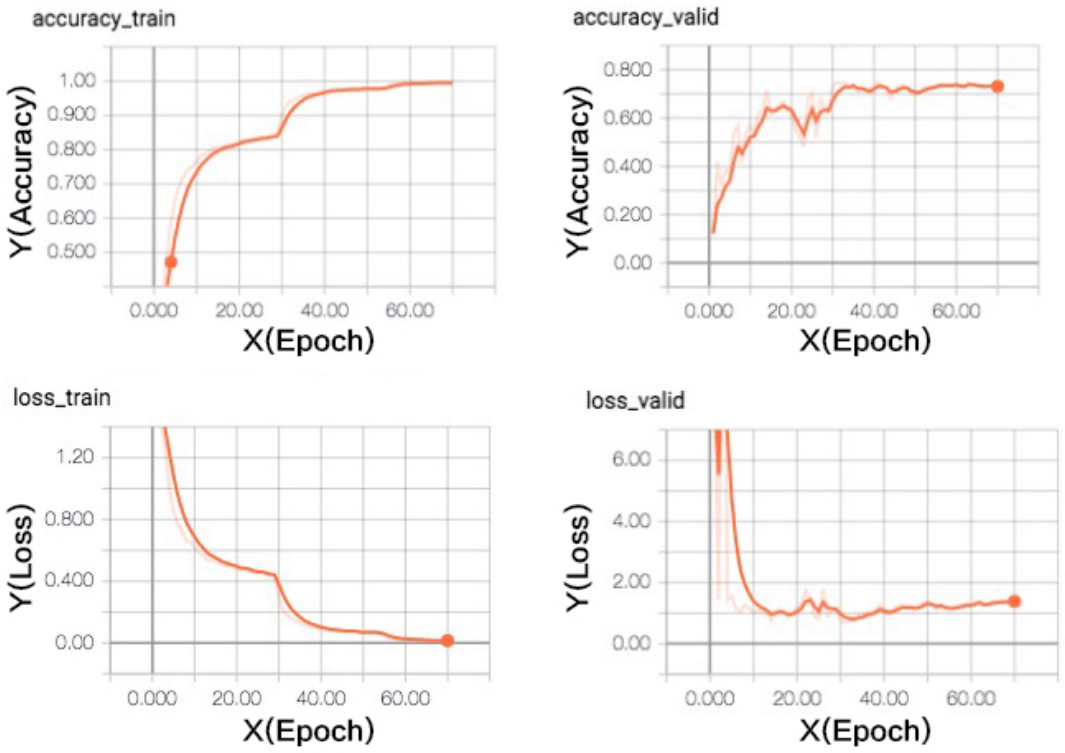


Figure A.10: The result of 3D-DenseNet with reference number 10 in Table 4.2

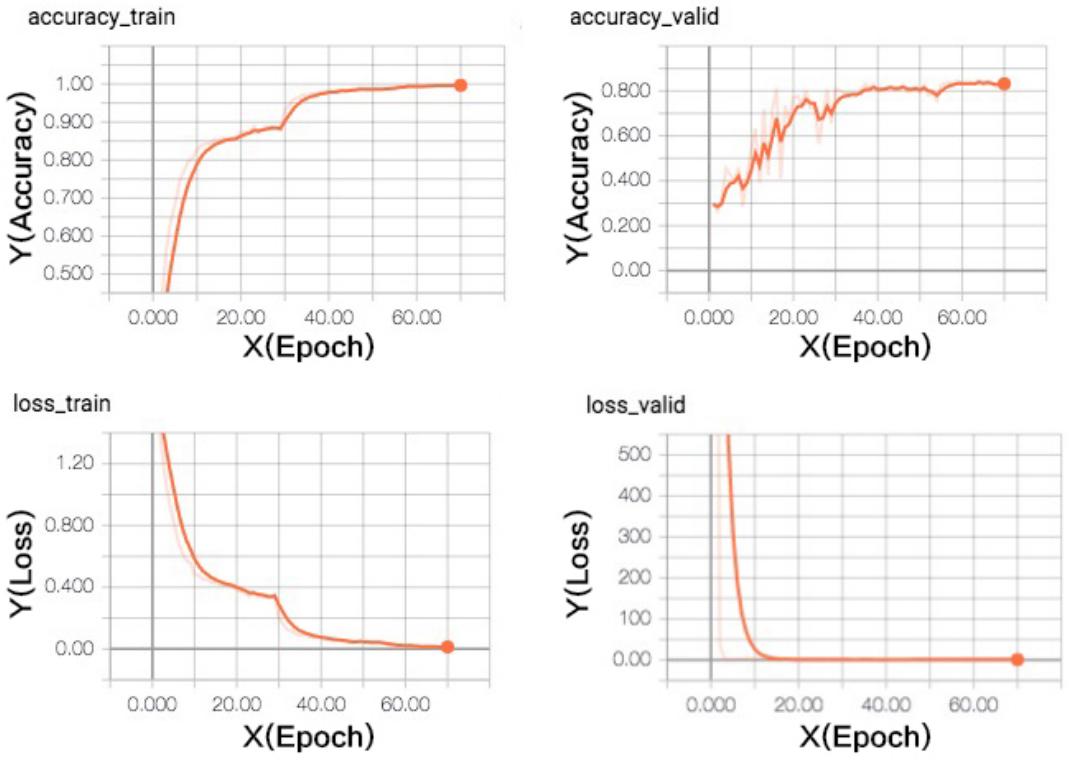


Figure A.11: The result of 3D-DenseNet with reference number 11 in Table 4.2

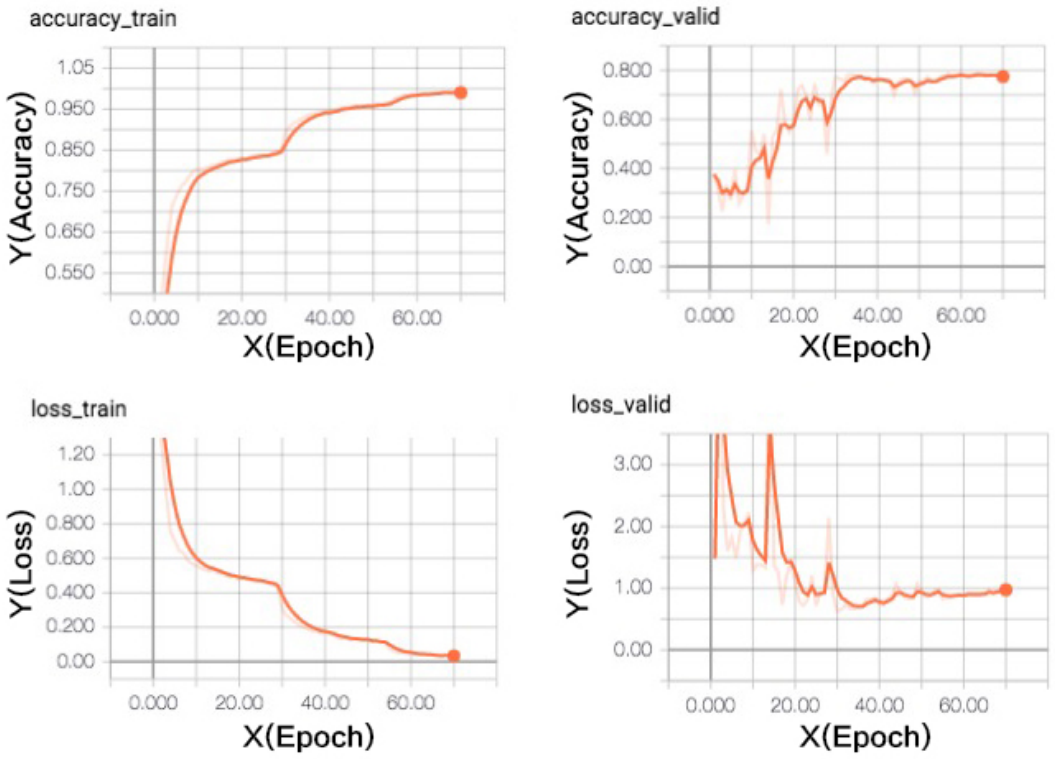


Figure A.12: The result of 3D-DenseNet with reference number 12 in Table 4.2

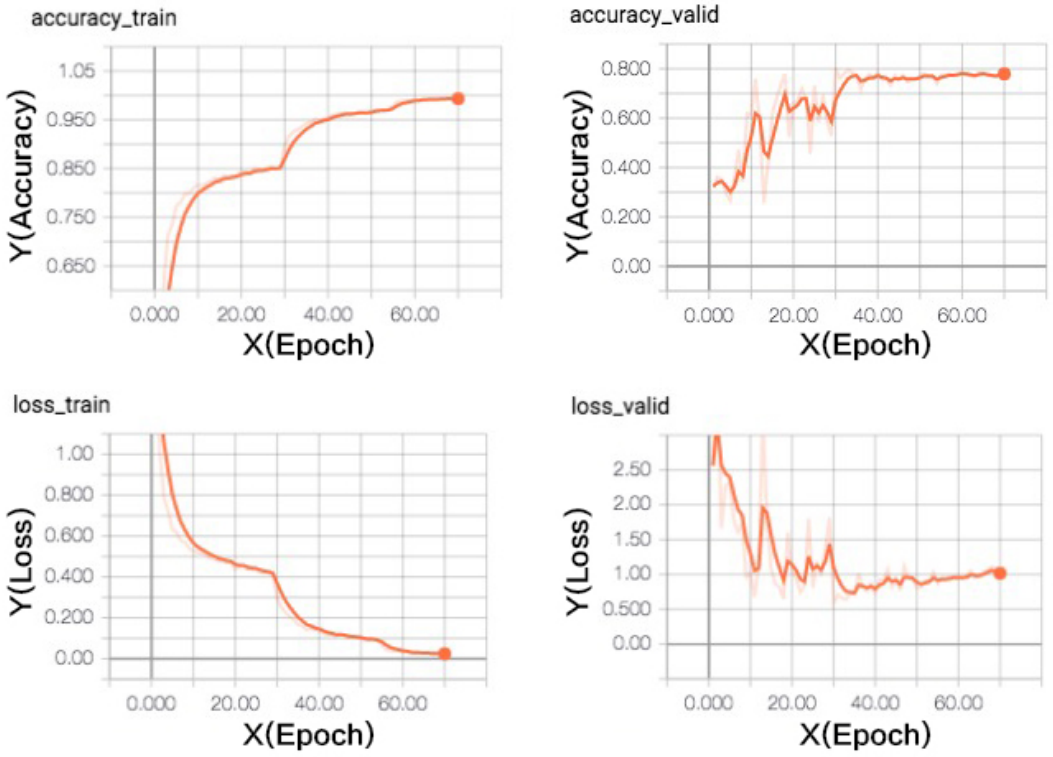


Figure A.13: The result of 3D-DenseNet with reference number 13 in Table 4.2

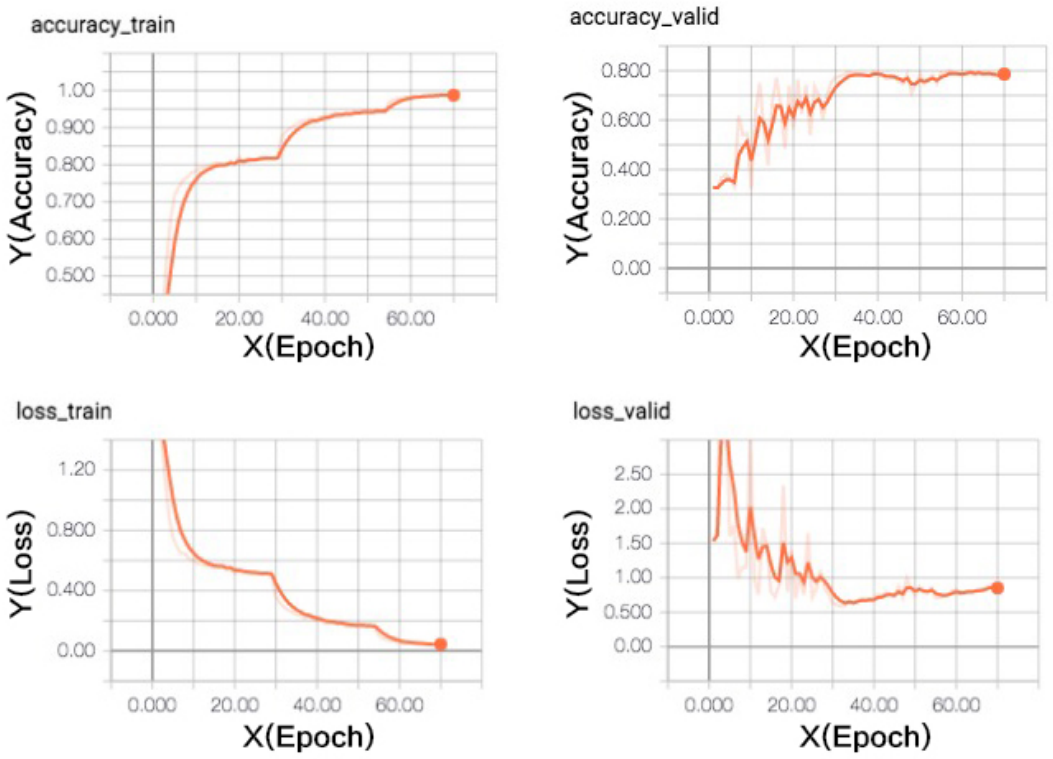


Figure A.14: The result of 3D-DenseNet with reference number 14 in Table 4.2

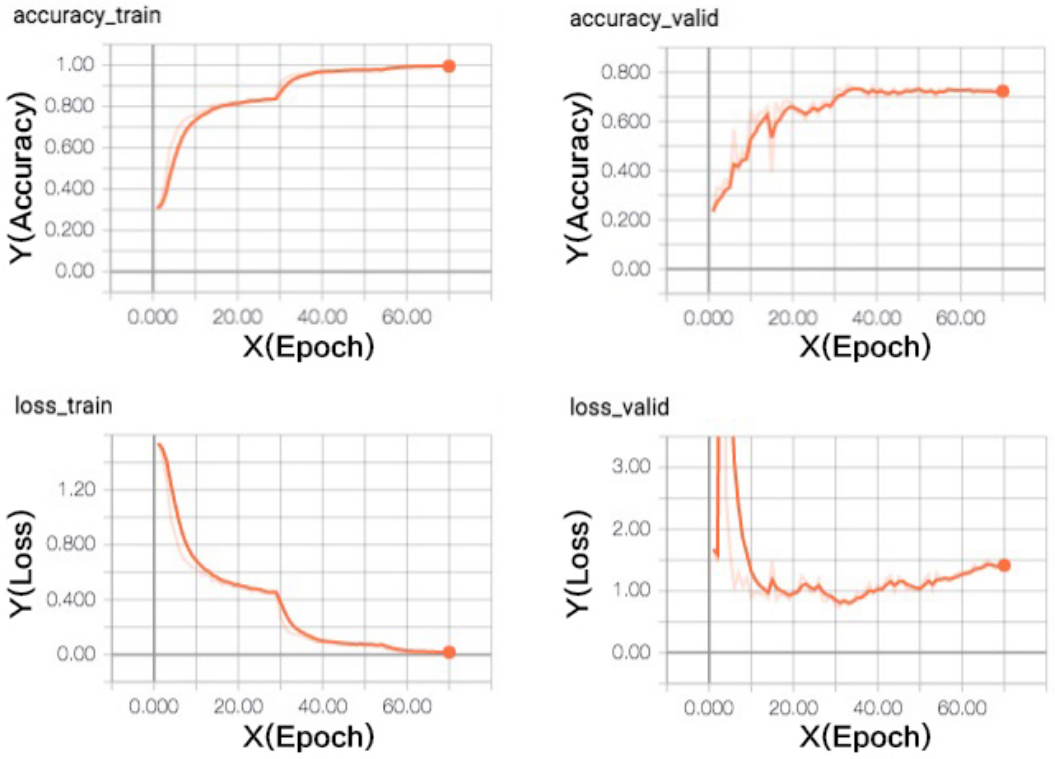


Figure A.15: The result of 3D-DenseNet with reference number 15 in Table 4.2

Appendix B

Code implementation readme

Reference link: https://github.com/frankgu/3d-DenseNet

B.1 3D-DenseNet with TensorFlow

Expand the Densely Connected Convolutional Networks DenseNets to 3D-DenseNet for action recognition (video classification):

1. 3D-DenseNet - without bottleneck layers
2. 3D-DenseNet-BC - with bottleneck layers
3. Each model can be tested on such datasets:
    - KTH

- MERL
- UCF101

A number of layers, blocks, growth rate, video normalization and other training params may be changed trough shell or inside the source code.

B.1.1 Pre-request libraries

- python2
- tensorflow 1.0
- opencv2 for python2

B.1.1.1 Step 1: Data preparation (UCF dataset example)

1. Download the UCF101 (Action Recognition Data Set).
2. Extract the UCF101.rar file and you will get ../UCF101/<action_name>/<video_name.avi> folder structure.
3. Use the ./data_prepare/convert_video_to_images.sh script to decode the UCF101 video files to image files.
4. Use the ./data_prepare/convert_images_to_list.sh script to create/update the train,test.list according to the new UCF101 image folder structure generated from last step (from images to files).
    - run ./data_prepare/convert_images_to_list.sh .../UCF101 4, this will update the test.list and train.list files (number 4 means the ratio of test and train data is 1/4)
5. Copy/Cut the test.list and train.list files to the root of video folder (../UCF101).

B.1.2 Step 2: Train or Test the model

Optional arguments	Description
-h, –help	show this help message and exit
–train	Train the model
–test	Test model for required dataset if pretrained model exists.
–model_type	What type of model to use (default: DenseNet3D-BC)
–growth_rate	Depth of whole network, restricted to paper choices (default: 40)
–dataset	Path to the dataset
–total_blocks	Total blocks of layers stack (default: 3)
–keep_prob	Keep probability for dropout.
–gpu_id	Specify the gpu ID to run the program
–weight_decay	Weight decay for optimizer (default: 0.0001)
–nesterov_momentum	Nesterov momentum (default: 0.9)
–reduction	reduction Theta at transition layer for DenseNet3D-BC models
–logs	Write tensorflow logs
–no-logs	Do not write tensorflow logs
–saves	Save model during training
–no-saves	Do not save model during training
–renew-logs	Erase previous logs for model if exists.
–not-renew-logs	Do not erase previous logs for model if exists.

Table B.1: Program optional arguments

- Check the training help message in Table B.1
  - python run_dense_net_3d.py -h
- Example: Train and test the program
  - python run_dense_net_3d.py -train -test -ds path/to/video_folder