Deep Action Classification: Toolset for Benchmarking and Visualizing Action Classification Models

BY

KHAW TATT JUEN

A REPORT (FINAL YEAR PROJECT II)

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2019

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **Deep Action Classification: Toolset for Benchmarking and Visualizing Action Classification Models**

Academic Session: MAY 2019

I _____KHAW TATT JUEN_____

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

_____ _____

(Author's signature) (Supervisor's signature)

Address:
3D-5-6, N-Park
Jalan Batu Uban, Glugor
11700 Penang

_____
Supervisor's name

Date: _____ **Date**: _____

Deep Action Classification: Toolset for Benchmarking and Visualizing Action Classification Models

BY

KHAW TATT JUEN

A REPORT (FINAL YEAR PROJECT II)

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2019

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Deep Action Classification: Toolset for Benchmarking and Visualizing Action Classification Models** " is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :

Name : Khaw Tatt Juen

Date :

ACKNOELEDGEMENTS

A sincere thanks and appreciation to my supervisor, Dr. Tan Hung Khoon, who has offered me an opportunity to engage a deep learning related task. Also to my moderator, Ts. Lai Siew Ching, and Dr. Tan, who both have given me adequate amount of supports and advices on this project, and provided me access to computing machines with necessary capabilities, which has made this project possible.

A special thanks to a former professor in FICT, UTAR, Prof. Zen Chen, who has introduced me, and intrigued my interest on field of computer vision, who also guided me discovering my true interest. Finally, thanks for the continuous love and support from my families and friends throughout my study.

ABSTRACT

Ever since the outstandingly success of AlexNet in ImageNet image classification competition, deep network architectures have become a mainstream approach on investigating computer vision problems. Action/Video classification is not an exception from this transformation. Computer vision communities never stop innovate and putting effort on designing and experimenting with varieties of deep action network architectures. Among these architectures, two-stream is one of the well-known base architecture that has shaped many of the state-of-the-arts, such as R2P1D and I3D.

Despite the facts that the community has actively contributed in publishing new works on action classification models, there are lacking of tools on comparing the training and evaluation performance between each models statistically. Furthermore, works on visualizing action classification models are lacking attention from community and rarely spotted among researchers' publication, hence, limiting the access of gaining insight on the learning behavior of action classification networks.

Therefore, inspired by ConvNetJS, which is an online experimenting and visualizing tools on deep image classification models by Karpathy, this project proposes a comprehensive toolset with the same concept, but on action classification models. It is designed for simple research works and academic purposes, which supports performance benchmarking and visualization on varying two-stream architectures of deep action network.

Up-to-date project source code and descriptions are available on
https://github.com/juenkhaw/flask_fyp

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

H	Spatial Height
W	Spatial Width
L	Temporal Length
k	Spatial kernel size
d	Temporal kernel size

LIST OF ABRREVATIONS

1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
BoF	Bag of Features
C3D	Three-dimensional Convolutional Network
CMD	Command Prompt
ConvNet	Convolutional Neural Network
FICT	Faculty of Information Communication and Technology
FYP	Final Year Project
GBP	Guided Back-Propagation
GoogLeNet	Inception Network
GPU	Graphics Processing Unit
GradCAM	Gradient-Weighted Class Activation Mapping
GUI	Graphical User Interface
I/O	Input / Output
I3D	Inflated Three-dimensional Convolutional Network
iDT	Improved Dense Trajectories
IoU	Intersection over Union
LR	Learning Rate
R2P1D	(Two Plus One)-dimensional Convolutional Network
ReLU	Rectified Linear Unit
ResNet	Residential Network
RGB	Red-Green-Blue Image Channels
SGD	Stochastic Gradient Descent
Softmax	Normalized Expotential Function
SVM	Support Vector Machine

CHAPTER 1 INTRODUCTION

Ever since the remarkable success of deep network, *AlexNet* (Krizhevsky, Sutskever, & Hinton, 2012) in ImageNet still-image classification challenges (Russakovsky*, et al., 2015), computer vision researchers and communities have become increasingly aware of deep learning techniques. Throughout the years after that, advancement on deep learning on image classification has been actively driven by innovative researchers, from *VGGNet* with small filters (Simonyan & Zisserman, 2015), *GoogLeNet* with network inception (Szegedy, et al., 2015), *ResNet* with residual learning (He, Zhang, Ren, & Sun, 2016), to *DenseNet* that connects every layers in feed-forwarding (Huang, Liu, Van Der Maaten, & Weinberger, 2017). Every bit of them has gradually lifting performance of still-image deep networks up close to human-level performance.

On the other hand, action, or video classification, remains as a challenging task for deep learning researcher communities. Different from still-image, video contains temporal information that provides additional clues on motion. Therefore, action classification networks learn on not only the spatial space (RGB frames), but also the temporal space (motion), which often leads to longer training time and hinders network performance with its additional dimension of learnable parameters included. Despite of its difficulty, researcher community has never stopped their ambition on exploring solutions to tackle this domain of computer vision. Throughout the recent years, researchers have published a diversified variety of network architectures, and actively enriching our understanding on this domain.

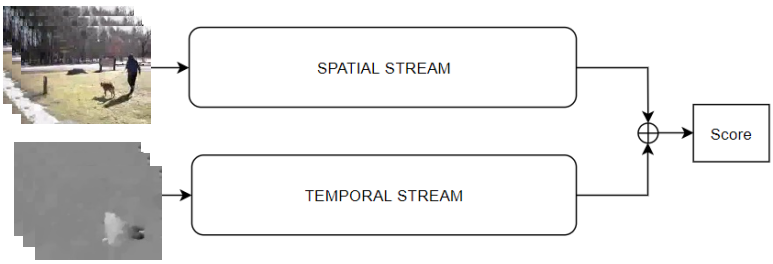


*Figure 1-1: **Overview architecture diagram of a conventional two-stream deep action network.** Each stream learns on its respective modalities (RGB frames and optical flows) and predicts scores. Scores are typically fused by weighted averaging or SVM classifier.*

Among these publications, two-stream network, as introduced in (Simonyan & Zisserman, 2015), is one of the well-known base architectures that shapes many of the state-of-the-arts proven to have robust performance, including *I3D* (Carreira & Zisserman, 2017), *R(2+1)D* (Tran, et al., 2018), and *DTPP* (Zhu, Zhu, & Zou, 2018). Two-stream network learns on two aspects, RGB frames and optical flows, of videos independently, leading to two stream

networks trained on different, respective modalities of input volume, and yielding two sets of classification scores. These scores are to be fused to produce a final score set using common method like weighted averaging. Despite the unity of two-stream networks, the family of two-stream action network itself has a great variance between each architectures in terms of network structures, input pre-processing, and training preferences. These differences are to be further discussed in *Chapter 2*.

To truly study and evaluate these network architectures, one of the most effective is to train our own version of models, experiment, and compare with peer networks with distinct architecture. However, processes of benchmarking and comparison between models with distinct architectures are often cumbersome and inconvenient, due to issues such as: (1) Each of these networks often requires dedicated implementation of data pre-processing, training, and testing modules. (2) The metric on performance measurement could be different in terms of testing videoset, input pre-processing, and evaluation method, which a direct compare based on reported accuracy thorough the papers are often inaccurate and inconvincible if authors do not experiment with the benchmark model in interest. (3) Visualization on action networks are often lacking of attention from communities and its implementation is highly dependent to the network structure, which is often dissimilar among architectures.

To resolve these problems, this project proposes a comprehensive unified toolset, which could train, evaluate, visualize variety of deep action classification models, and statistically compare between models with the same evaluation metrics applied. This project is mainly inspired by the concept of *ConvnetJS*, by Karpathy, that offers online deep learning experimenting toolset on image classification that allows training on all sort of conventional image classification network. Therefore, it encourages this project to replicate this idea and applies onto domain of video/action recognition, and aims to generalize the training, evaluating, and visualizing process to be compatible with most sort of conventional two-stream action networks, regardless of its internal network structure.

For the rest of the report, *Chapter 2* discusses on the advancement of action classification network over time from hand-crafted models to deep two-stream models, and published works or applications that serve similar purposes as the proposed toolset. Internal structure and functionalities of the proposed applications are described in *Chapter 3*, while *Chapter 4* is about experimenting on *R2P1D* models using the proposed toolset, and a brief analysis on the experiment outcome with help of output from the toolset.

CHAPTER 2 LITERATURE REVIEW

Before the era of deep learning, action recognition models were primarily dominated by feature-based and handcrafted approaches. Those were the time where action recognition models rely on manually extracted features to learn spatio-temporal representations. Different from deep network nowadays, it is shallow and suffers from limitation on learning complex mapping, it is when the classification performance came to a bottleneck, and deep learning brought an evolutional advancement. The more researchers shifted their focus onto developing deep representations, the clearer the fact that deep networks are outperforming feature-based models, with its automatic feature learning capability. The barrier of feature-based models is hence, resolved and performance continues to triumph as researchers pursue discovery in refining and standardizing deep action recognition networks.

In this chapter, first section discusses briefly on differences between feature-based models and deep representations, followed by the advancement of deep learning paradigms over time in action classification. Second section introduces a listing of existing applications with similar ideas and how are them going to be different from the proposed program.

2.1 Action Classification Models: Evolution over Time

2.1.1 Network Depth: Shallow to Deep

Handcrafted representations were once pioneer in the domain of computer vision, which infer action by reasoning specific and representative clues or patterns from high dimensional space-time features extracted from videos using certain descriptors. Therefore, its robustness is often dependent on effectiveness of feature descriptors and noise reduction strategies, as realistic videos data often contain camera motion and severe motion blur, which could reduce effectiveness of feature extractors from capturing accurate motion information.

iDT (Wang & Schmid, 2013) is a renowned state-of-the-art feature-based model, which measures camera motion explicitly and effectively suppresses background features by removing outlier trajectory features through estimation with homography technique. With the histograms of cleaned features, it is then transformed into a *BoF* representation and fed into a *SVM* classifier for reasoning.

Feature-based models often require careful consideration on selection of feature descriptor, encoder, and noise suppression techniques in order to obtain decent performance. Different from that, deep Convolutional Neural Networks (*ConvNet*) facilitate end-to-end

automated and hierarchical feature learning mechanism on input volume, which eliminates necessity on manual feature extraction. With multiple hidden layers, modelling higher complexity of spatio-temporal representation is possible with non-linearity through activation functions. However, action recognition *ConvNet* requires (1) extensive computational power and (2) huge amount of training videos to maximally optimize millions of its trainable parameters, which both of them was not widely available back then, and was truly hindered its advancement.

Multiresolution *ConvNet* (Karpathy, et al., 2014) is proposed to tackle limitation on computing resources without sacrificing its performance on reasoning spatio-temporal features from low quality images. The strategy is to downsample resolution of input images and introduces a *multi-res* two-stream architecture, much like predecessor to two-stream in (Simonyan & Zisserman, 2015), which to be discussed in the next section, but trains on only a single modality (RGB) of videos. These two streams are *Fovea stream* that learns on centre patch cropped from frames with its resolution reserved; while *Context stream* receives blurred frames with its global context maintained. For time space, *slow fusion* model is proposed to progressively converges multiple instances of *multi-res* bottom layers that train on evenly extracted video clips to simulate temporal reasoning.

2.1.2 Convolutional Neural Networks: 2D to 3D

Compared to feature-based state-of-the-art mode, *iDT*, *Multi-Res* mode with *slow fusion* is significantly underperforming, which suggested context changes across RGB frames are solely insufficient for effective temporal reasoning. To circumvent that, **Two-Stream *ConvNet*** (Simonyan & Zisserman, 2015), a multi-modalities two-stream network, proposes *Temporal stream* that trains on stack of optical flows that depict motion between frames, making it a clip-level representation; while *Spatial stream* as a frame-level representation that trains on only single RGB frame. Its multi-modalities property provides *ConvNet* access to another perspective from which to gain diversified insight on the temporal representation. Its overall performance is on par with *iDT*, and unquestionably better than single-modality models.

In terms of temporal connectivity, *Spatial stream* is merely a image classification model, whereas for *Temporal stream* and *Multi-res* models, they collapse input temporal space instantly at the first layer of convolution, leaving the rest consists only spatial operations. Such implementation hampers the modelling on fine-level motion patterns with such aggressively huge respective field on temporal dimension. *Slow-fusion* model is an exception among them,

which attempts to facilitate time-space modelling up to only three layers, and yields promising result that motivate researchers to enhance temporal connectivity by incorporating more 3D convolution operations.

As computing machines have become increasingly robust and available in recent years, researchers started employing more 3D convolution operations into action classification models. ***C3D*** (Tran, Bourdev, Fergus, Torresani, & Paluri, 2015) is a full 3D *ConvNet*, which learns spatio-temporal features simultaneously by applying 3D kernels in each of its convolutional layers, convolutes input volume spatially and temporally, produces a 3D activation output with temporal signal preserved. With sufficient input frames and gentler growth on temporal receptive field, *C3D* could perform temporal reasoning deeper than ever, of which most of the 2D *ConvNets* and fusion models are not capable. However, *C3D* does not yield satisfying result due to: (1) Enormous amount of learnable parameters with additional dimension of time, which leads to convergence difficulty. (2) Data-hungry, which requires huge videoset to optimize its parameters effectively. (3) Omits the benefit of transfer learning. Despite of its weaknesses, *C3D* was still a great attempt and motivated researches on refining architectures while preserving its ability of simultaneous spatio-temporal feature learning.

2.1.3 Two-Stream Networks: State-of-the-Arts

I3D (Carreira & Zisserman, 2017) and ***R2P1D*** (Tran, et al., 2018) are both state-of-the-arts two-stream models adapted from *C3D* model, and yield decent performance with their own refined architectures.

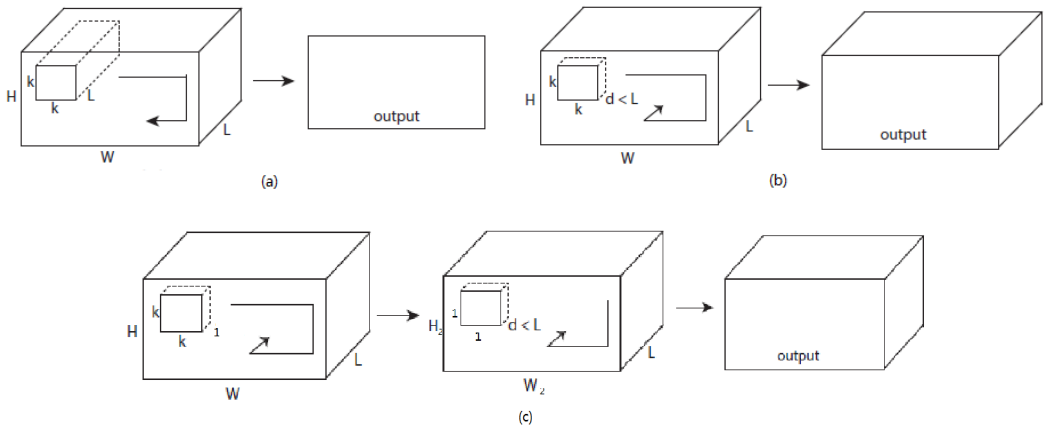


*Figure 2-1: **2D, 3D, and 2.5D Convolutional Operation.** (a) 2D Convolution collapses temporal space of input volume and outputs a 2D volume. (b) 3D Convolution with filter temporal dimension shorter than input volume, results in a 3D volume. (c) 2.5D Convolution introduced in R2P1D that factorizes a 3D Convolution into a 2D spatial convolution and a 1D temporal convolution, which contributes much fewer parameters and outputs with temporal signal preserved. Image is retrieved and adapted from (Tran, Bourdev, Fergus, Torresani, & Paluri, 2015)*

I3D exploits transfer learning by applying maturely pre-trained image recognition models. The idea is that a 2D filter from a 3D equivalent version of an image classification network would share the same spatial size with 2D filters in 2D image network, which suggests applying an inflated version of pre-trained weights from a 2D model is viable. As *I3D* filters has an additional temporal dimension of parameters, a 2D weightage matrices have to be repeatedly duplicated across temporal space before transferring them into a 3D filters. The benefit of transfer learning is clearly observed after fine-tuning with videoset.

On the other hand, *R2P1D* decomposes a 3D convolutional operation into a 2D spatial convolution followed by a 1D temporal convolution. This architecture aims to maximally reduce the number of learnable parameters spawn in a full 3D convolution, and hence results in more feasible optimization. By decomposing 3D kernels, the effect of dimensionality curse shall be minimized and the network would not be extremely data-hungry as *C3D*. This method also doubles effect of non-linearity, as activations are applied after each spatial and temporal convolution, which helps strengthening complexity of the model.

2.2 Existing Toolsets: How are them Different from Proposed Toolset

Comparing and evaluating multiple architectures of action network at once is never an easy task, as each network has its own unique training preferences, architectural design, and input pre-processing procedure. Therefore, to cope with these variances, the toolset and module design has to be generalized to work with all these model differences, by either argument passing or modularize network structure to allow flexible network building. It is a common practice that researchers hosted their benchmarking programs on Github, the section below investigates on some of them that are closely related to this project.

As mentioned in previous chapter, this project is mainly driven by the idea of ConvnetJS by Karpathy, a deep learning JavaScript library that runs deep models entirely on only user browser with no installation and GPU required. One of its toolset facilitates real-time training, evaluating, monitoring, and visualizing on image classification models. It is a perfect example that tends to generalize the deep learning toolset such as trainer, tester, and visualizer to be compatible with varying and customized image classification network architectures. Its modules are discussed further in each respective sections.

2.2.1 Models Training Toolset

ConvnetJS image classification toolset supports training customizable image network on MNIST (LeCun, Bottou, Bengio, & Haffner, 1998) and CIFAR-10 imageset (Krizhevsky

& Hinton, 2009). Network definition is encoded as a list of JSON texts conveying layers type and relevant parameters, which will later be complied into an actual network by its JS library. Users could change the structure of network by specifying layer types with JSON in the editor provided, and start a new trainer. During training, real-time training stats and loss graphs are displayed on the panel for user to observe how well the training went over iterations. Users could also tune the hyperparameters such as learning rate, momentum, etc. in the middle of training, and its effect will start reflecting on the graph.

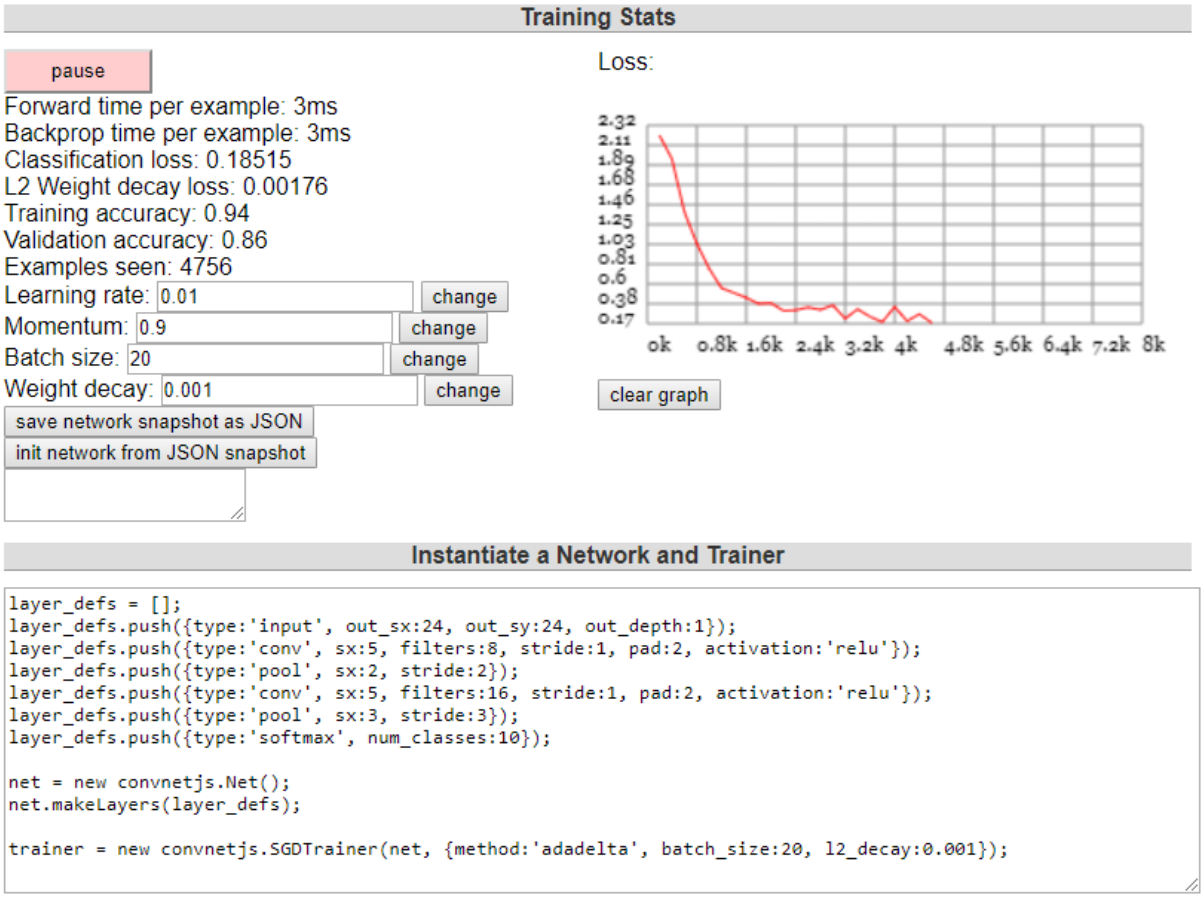


Figure 2-2: ***Interface of ConvnetJS training image classification network with MNIST.*** *Upper part shows real-time training stats and loss graph for current training task. Lower part shows an editor that allows users to customize their own network architecture using JS library provided by the author. Image retrieved from https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html.*

It is one of the few deep model benchmarking applications that provide a GUI for users to interact, which does not require skills on working with CMD. Unlike ConvnetJS, most of the benchmarking programs hosted by the researcher, they tend to keep its simplicity and avoid overhead on loading interfaces.

R2P1D action network benchmarking program by its authors facilitates a model zoo that houses a variety of network architectures with similar genre. The selection of network

architecture involves R2P1D, RxD, and MCx, which all of them share the same base architecture of ResNet (He, Zhang, Ren, & Sun, 2016). Users get to parse relevant parameters to the trainer script in order to initiate training on the model selected with other preferences specified. The design on network components is highly modularized and reusable, which is generally a good practice to ease network assembly, especially for experiment on network block arrangements, each blocks could be easily swapped in and out. Balance on network granularity

Based on the observation on these two distinct toolsets, compensation point has to be made in between user interactivity and swiftness of CMD. Therefore, for the proposed toolset, the toolset itself is designed to be an independent backend library, where two interface programs for CMD and browser are then attached to it respectively. The GUI of proposed toolset would tend to portray the interface on ConvnetJS, as it is a rare sample of deep models benchmarking toolset with GUI. Instead of displaying monotonic loss and accuracy graphs, proposed toolset would instead compare training stats of multiple peer models and presented in graphs. For network compilation, proposed toolset would follow fashion of customizable network architecture from ConvnetJS rather than restricting variety of networks with predefined network blocks, as what R2P1D program does.

2.2.2 Network Visualization Toolset

Network visualization has been a useful tool among computer vision researchers to gain insight on how a deep network performs inference and particular features and patterns on input space that excite the network in classification. One of the renowned network visualization tool, DeepVis by (Yosinski, Clune, Nguyen, Fuchs, & Lipson, 2015) generalizes visualization technique to be compatible with any sort of image classification network that users defined. Similar to ConvnetJS, it visualizes all filters with activations in a particular selected layer and displays them all at once on the interface. Besides classifying images in the dataset, it allows live network visualization where input are real-life images captured through webcams.

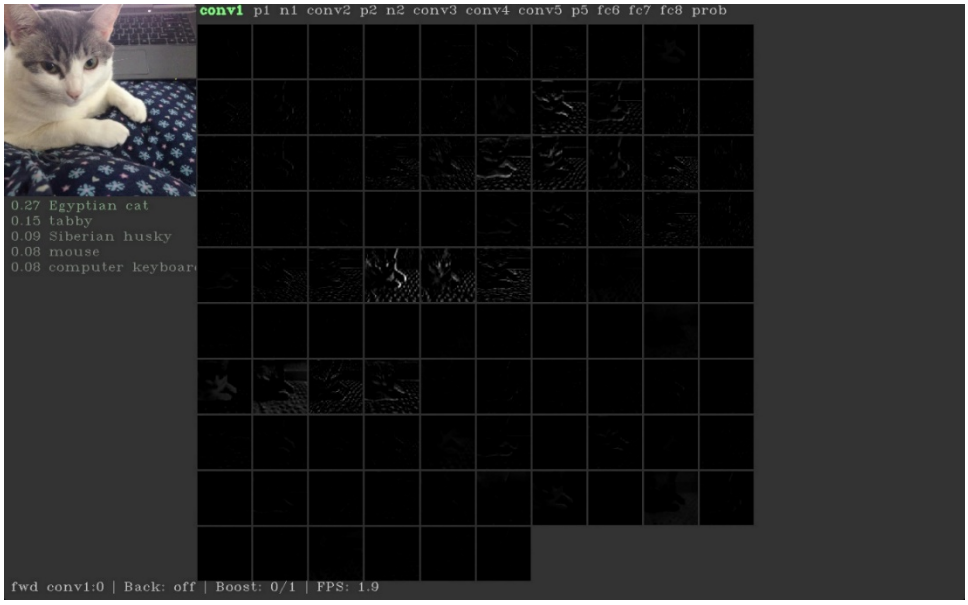


Figure 2-3: ***Interface of DeepVis toolset.*** *Live network visualization mode classifies real-time images obtained from webcam, visualizes gradient maps at particular network depth and updates them in real-time. Image retrieved from https://github.com/yosinski/deep-visualization-toolbox.*

Another instance of similar visualization toolset hosted on Github is CNNVis by (Ozbulak, 2019). Different from DeepVis that utilises same visualization techniques onto variety of network architectures, CNNVis tends to facilitate pool of visualization tools on specific networks such as AlexNet and VGG from Pytorch model zoo. Despite that, slight modification on the source code would make that toolset compatible with other network architecture. CNNVis incorporates an extensive variety of network visualization techniques such as guided backpropagation, saliency maps, GradCAM, etc., which could run independently from each other based on the users preference.

These visualization toolsets have offered this project insight on designing visualization module for the proposed toolset. In terms of approach, this toolset tends to follow more onto ConvnetJS and DeepVis, which the visualization techniques are applicable to possibly all variety of image network architectures. In terms of output display, DeepVis output would be too overwhelmed for users and not computationally feasible for action network with 3D filters. Therefore, proposed toolset would follow the approach in CNNVis where output is produced on request basis based on the input clip, target label, and target layer. Last but not least, these existing visualization toolsets are all meant for 2D image networks, visualization techniques deployed in these applications require modification for them to be workable with 3D action networks.

CHAPTER 3 METHODOLOGY

The proposed program would be a replica version of ConvnetJS specialises on action recognition network benchmarking and visualizing. It provides comprehensive toolset on training, evaluating and visualizing on variety of action network architecture based on the user-defined preferences. The toolset supports running on both CMD and browser. The backend programs of this toolset are written in Python language and Pytorch as deep learning library. Interfaces are written to provide linking between backend programs and display terminals (CMD and browser). For CMD, interface program parses user-defined parameters into backend program and run on specified module; while for the browser version, graphical interfaces are rendered and coordinated using FLASK web framework library. Therefore, the overview architecture of this toolset consists of one backend program that takes care of deep learning tasks, with two interface programs linked to it that handle I/O between users and the backend through CMD and browser respectively.

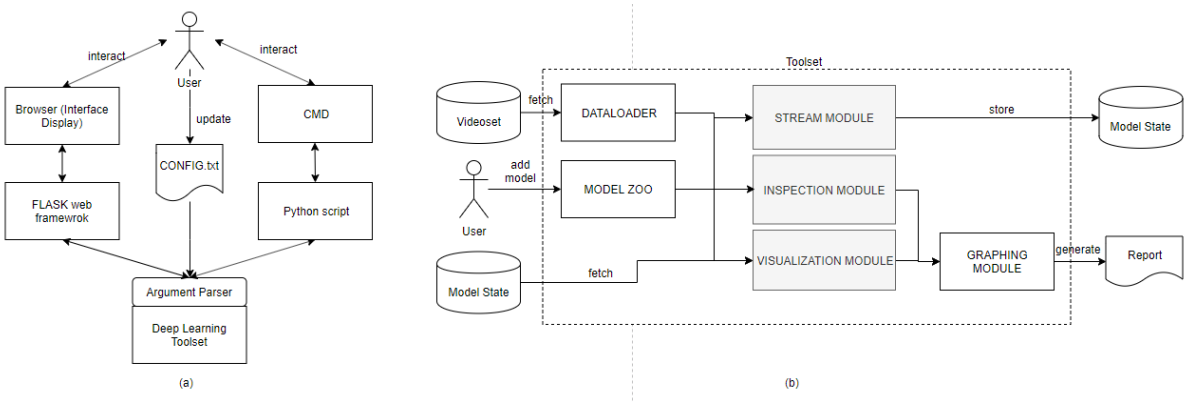


*Figure 3-1: **Architectural view of the proposed toolset.** (a) Diagram depicts how users could interact with the toolset. CONFIG.txt is an external configuration text file where users could modify to include dataset and network defined by them. (b) Internal structure of the toolset and interaction with outside environment, with three main modules, and other utilities to support generalization of processes.*

This toolset consists of three main functions: (1) Training and testing on conventional two-stream action network and real-time comparison on training stats between models. (2) Inspection on each models performance with statistical output. (3) Visualization on action network that provides insight on how the network identifies and infers action. The rest of this chapter are about a closer look up on functionalities and implementation of each main modules.

3.1 Network Trainer and Tester

To realise the concept of generalized training and testing procedure, this module is designed to be cope with any sort of conventional two-stream action networks that accept RGB

frames or optical flows as input and output a vector of Softmax scores, as well as any pre-computed videoset. Network trainer trains and validates the model and outputs checkpoint model state and training stats at the end of each epoch, while interface updates the progress and stats on the display. Tester evaluates the models with common methods that are popular among publications (10-clips and 10-crops) and reports accuracies and confusion matrices in its output. The following section shows the implementation of these functionalities.

3.1.1 Network and Dataset Generalization and Plug-in

Network customization enables users to build their own network with Pyotch modules, which requires them to have basic knowledge on Pytorch modules and tensor processing. Fortunately, Pytorch modules are designed to be stackable, modularizable and easy integrity between each blocks and another, which is effortless for apprentice users to learn, and encourages the design of a unified network template interface that creates abstraction over the network implementation.

Network template is designed to be flexible and workable with any sort of two-stream network architectures by structuring them into a uniform format that toolset modules can recognize and work on. This idea is similar to the way instantiating network in Keras. The template interface is implemented with additional methods such as freezes and returns output tensors at designated layers. If users intend to train their own network, they shall define their network in a script using the interface class prepared by the toolset, and register it. Registration is done in an external configuration JSON text file, by adding an entry in network section of this files, toolset shall recognize this network and to be appeared as an option in trainer panel.

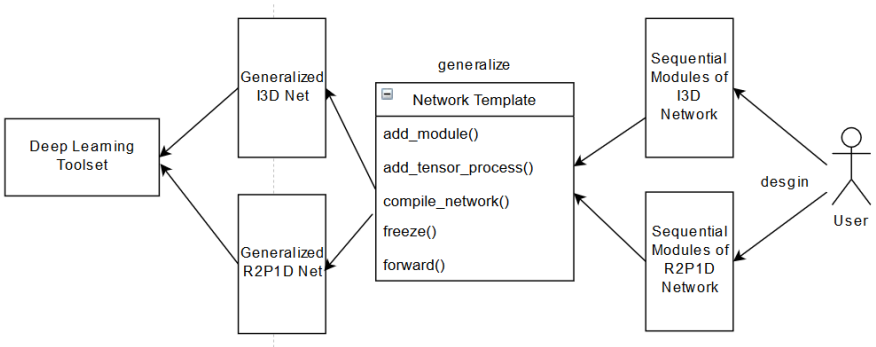


Figure 3-2: ***Explanatory diagram on how network template generalizes user-defined network architecture.*** *Network template interface class compiles two-stream network regardless of its internal structure into a standard network object that toolset modules could work on.*

For dataset, this toolset also attempts to generalize input pre-processing and distribution modules on any sort of videoset, by using Data Loader module. However, this is a much more

challenging task comes with many restrictions in order to maintain the unity: (1) Optical flows has to be pre-computed beforehand, so that users could employ flow extraction algorithms based on their own preference. (2) RGB frames and optical flow must be saved in a series of JPEG images and contained in folder named with its respective video name. (3) Training, validation, and testing I/O mapping listing and class label listing has to be explicitly defined. The pre-processing procedure shall be generalized only by restraining format and structure of videosets. However, exceptional two-stream architecture like *Hidden-I3D* (Zhu, Lan, Newsam, & Hauptmann, 2018) generates their own optical flow input through encoder network where its loss function is considered in optimization on flow stream network, which could not possibly incorporate with this toolset yet. Similar to network plug-in, manual registration of dataset entry in configuration file is necessary for the toolset to recognize it.

3.1.2 Training and Testing Panel

Setting up trainer and tester module is rather easy. Passing relevant arguments through CMD would be sufficient to start the training/testing, while GUI version offers web form elements for easy selection on each options. Besides selecting network and dataset, it also supports applying pre-trained model for transfer learning, data augmentation preferences, settings on SGD optimizer and LR scheduler, and debugging mode that allows the network to run on only a small batch of data extracted from original videoset to verify network validity. For tester, this toolset implements two commonly used evaluation metrics: (1) 10-clips that samples 10 clips uniformly across a video, and (2) 10-crops that applies 10-clips, and for each clips, crops out 4 corners patches and 1 center patch, and flips them vertically to form another 5 patches, making 100 testing clips for each video. Besides, users could also choose to resume previous training session for more epochs by loading a checkpoint model in the output folders.

It is a common practice where researchers test its network usability and validity by feeding it with small chunks of dataset. It is to ensure that the network would overfit to that chunk of data, and examines its ability to converge. Debugging mode comes with two option: (1) Extract only few samples from the beginning regardless of labels. (2) Extract few samples uniformly over labels. Based on user preferences, they could examine their networks before initiating actual training on full batch of videoset.

Figure 3-3: ***Conceptual interface on training panel.*** *Upper part shows design on panel that displays real-time loss and accuracy graph while comparing with training stats of its peer models. Lower part depicts design on expected statistical output on validation performance, by showing averaged accuracy of each class label and top mislabelled label.*

During the training, progress on current batch and epoch is shown on the display panel, with at least one epoch completed, the panel would start to display latest loss and accuracy graphs, along with training stats of the other trained models if specified in the setup. For validation results, it shows details on averaged accuracy or scores of classification performance on each action labels, along with top mislabelled actions to imitate a confusion matrix partially. By applying different metrics and sorting, users get to observe particular action class that network good and bad at classifying, and through that, analysis on network ability on capturing context changes and motion information would easily be justified with statistical output as such.

3.2 Network Outcome Inspection

Models benchmarking involves one base model as a standard performance for other peer models to compare and evaluate. Network Inspection module emphasizes reporting on the training and evaluation performance of the base and each competing peer models. To ensure fairness and validity, comparisons could only be made in between models trained on the same dataset split. It only makes sense to evaluate network ability of spatio-temporal features modelling on same set of datasets that carry the same clues and patterns for models to learn.

Figure 3-4: ***Conceptual design on confusion matrix display.*** *Upper part shows part of an actual confusion matrix to be generated using Plotly graphing library. Lower part shows another display mode that informs averaged performance difference on each class label between base model and peer models.*

Users get to select one base model and multiple peer models in order to review the report. For the base model, a chart of confusion matrix would be constructed for its testing result using Plotly-graphing toolset library, which also helps plotting the line graphs throughout this program. Through confusion matrix or graphical output on the panel, users could observe most misclassified label for each action labels, as well identify averaged differences on accuracy and score in classifying each action labels between models to gain insight on genre of class labels on which the stream network perform particularly good or bad.

With these reliable graphical and statistical output information on peer models performance, users would be confidently investigate strong and weak spots between each stream modality and network architectures, with the same evaluation metrics applied throughout the models.

3.3 Network Visualization

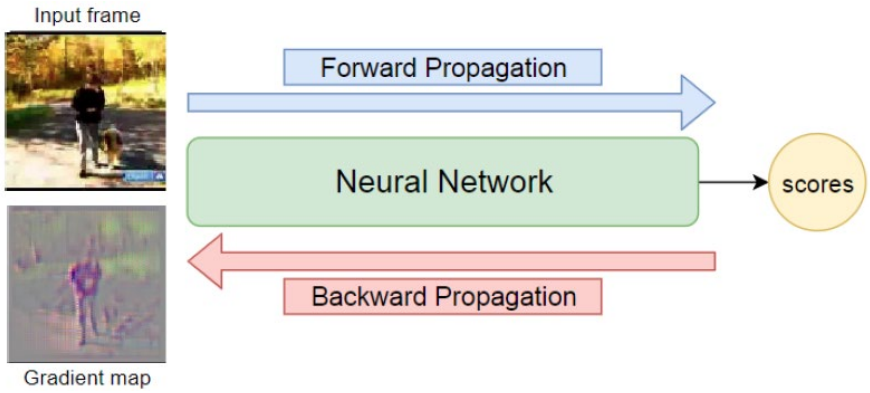


*Figure 3-5: **Conceptual diagram that explains network visualization technique.** Forward propagation predicts scores from input frames and backward propagation transmits gradients throughout the network until reaching input space.*

One way to study further on effectiveness on feature learning of a network is through visualization techniques. In deep learning, visualization is about investigating gradient maps back-propagated to the input space to find out what particular aspects of input space triggers hidden neurons and eventually classify that specific action label. Different from vanilla back-propagation, these visualization techniques involve adjustment and modification on the gradient maps propagating back to a target layer to suppress noise gradient from saturated or non-activated neurons that de-excites the network from classifying a particular action.

This module deploys Guided Backpropagation (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2014) and Grad-CAM (Selvaraju, et al., 2017) visualization techniques to visualize an action network. GBP depicts fine-grained low-level pixel space of features on the input frames that particularly excite the network; while GradCAM output are label-discriminative

and layer-specific that broadly localizes the area on the input space (high-level) that is decisive to network prediction.

However, these techniques are adequately experimented on only 2D image networks and proven its effectiveness. Therefore, this project also attempts to replicate and adapt these techniques to be workable with 3D action networks. Furthermore, with a unified network template, which mentioned in section above, this visualization module is also generalized to be coped on both RGB and Flow stream networks from any conventional two-stream architecture.

To extract the visualized gradient maps, the network shall first perform forward propagation on a clip selected by users all the way to the end (Softmax layer). Then, the module shall zero out all output tensor except for the score for the specified label class, and only proceed to back-propagation. GBP could only visualize the gradient maps back to the input space, while GradCAM could visualize gradient maps extracted at the end of any convolutional layers. For GradCAM, this module allows selection of multiple target layers for visualization, so that users could observe them all at once.

CHAPTER 4 EXPERIMENT AND ANALYSIS

This chapter is about conducting a small experiment on training, evaluating and comparing action classification models using the proposed program, and simple analysis on the statistical output produced by the program. Action network state-of-the-arts, R2P1D and I3D are selected to be the candidate architectures in this experiment. These two architectures are highly distinct in terms of their network structure and ideology, which respective base architectures are Inception-v1 (Ioffe & Szegedy, 2015) and ResNet (He, Zhang, Ren, & Sun, 2016) respectively. Therefore, this pair would form an interesting comparison on their classification performance and effectiveness on feature modelling. However, due to time constraint and the fact which I3D takes significantly longer time to train, architectural comparison is left for future works.

In this chapter, first section describes about network implementation and training setup and preferences, second section is about experimenting R2P1D network on freezing layers for transfer learning and comparison on performance between streams with different modalities.

4.1 Experiment Setup

All training is through transfer learning from respective pre-trained model. Stochastic Gradient Descent (SGD) is used in all training, with initial learning set at 0.01 that reduces by 0.1 when validation loss reaches a plateau with patience step set to 10 epochs.

R2P1D network is initialized with model pre-trained with Kinetics videoset (Carreira & Zisserman, 2017). Experiment on freezing point is conducted to identify an optimal layer depth to carry on optimization with new videoset. Each experiment lasts for 45 epochs. For I3D, model pre-trained with Charades videoset (Sigurdsson, et al., 2016) is considered. I3D training freezes network up to Mixed_3c and lasts for 25 epochs.

For input pre-processing, spatial and temporal jittering (random cropping) are applied onto both network architectures, and random flipping is applied on only I3D networks. Input frames are scaled to spatial size of 128*171 and cropped randomly to patches with size of 112*112 for R2P1D, while I3D receives input frames with spatial size of 224*224 randomly cropped from frames, which its smaller video side is resized to 256 pixels beforehand. Temporal length of each training clips are 16 and 64 for R2P1D and I3D respectively. Normalization is applied to zero-centre the intensity values to achieve facilitate normal distribution on activation values.

Trainer settings mentioned above are all available in the setup, while R2P1D and I3D are built-in networks and ready to train.

Trainings are all done on split 1 of *UCF-101* videoset (Soomro, Zamir, & Shah, 2012), a compilation of 13,320 *YouTube* videos including actions that involve: (1) Human-object interaction, (2) Body-motion only, (3) Human-human interaction, (4) Playing musical instruments, and (5) Sport activities. The source of precomputed sets of RGB frames and optical flows are available on Github hosted by (Feichtenhofer, Pinz, & Zisserman, 2016), while the train/val/test split files are obtained from (Le, 2018).

4.2 Stream Comparison: R2P1D RGB and Flow

This section is about analysing performance differences between RGB and Flow streams with the same network architecture of R2P1D. Before that, several trainings are done individually on each modalities to select the best candidate for comparison.

4.2.1 Training RGB and Flow

For experiments with RGB stream, this project starts with applying random flipping as additional augmentation on training data, which the original authors do not consider, to observe whether it is beneficial for model generalization. Turns out, the outcome is not satisfying as expected. A negligible reduction on validation loss is observed on the stream applied random flipping, but its accuracy is saturated as the stream without. The videoset itself could have been included with adequate variation on presentation of actions that play in different directions, which making itself already vertically augmented in the nature. Therefore, random flipping is not considered for the rest of the trainings.

Moving on to identifying optimal freezing points for each respective modalities, which is the most important hyperparmeter in transfer learning tasks. The extent of necessity for network optimization and fit to the new dataset could be determined through training several instances of stream network with different freezing points. Therefore, this demo experiments with three different depths of the network as freezing points on RGB and Flow stream individually.

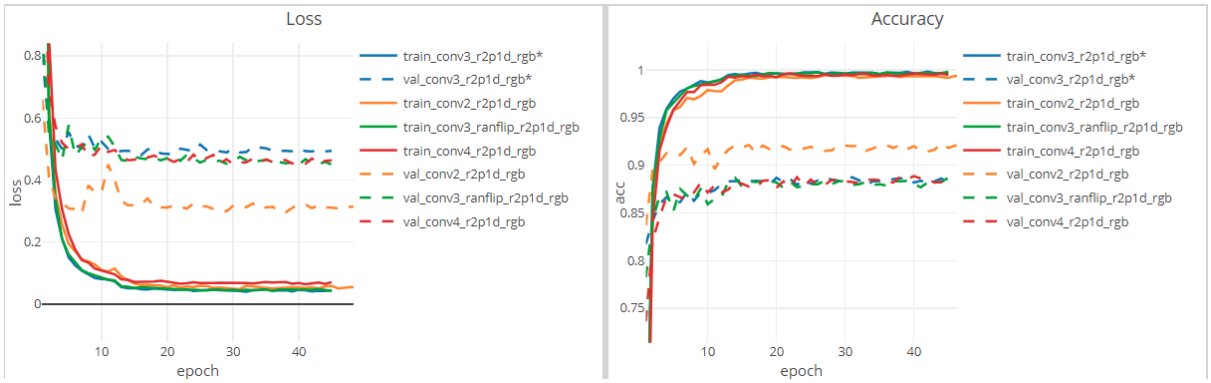


Figure 4-1: ***Loss and accuracy graphs for R2P1D RGB stream models.*** *Each models are trained with different freezing points. One of the two instances of Conv3_x models (green lines) applies random flipping.*

As for the outcome for RGB stream in *Figure 4-1*, it is obvious that network with fewer frozen layers (up to Conv2_x) outperforms other peers by around 3.5% in validation accuracy over the last few epochs. It conveys that spatio-temporal features learnt (from Kinetics) in middle-low layers might not generalizable to UCF-101, and hence requires optimization. Further reduction on frozen layers could be beneficial, at the same time, also susceptible to overfitting issue.

For Flow stream, networks should generally be more difficult to converge than RGB stream, which optical flows often contain irrelevant noise due to severe camera motion. Hence, mean subtraction (Simonyan & Zisserman, 2015) is applied onto each optical flows input volume to minimize global motion and preserve only displacements originated from the action-in-interest.

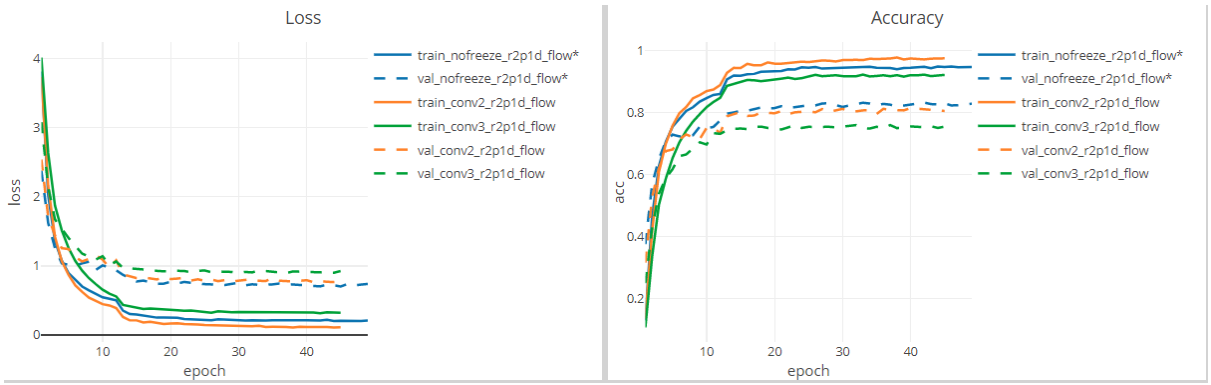


Figure 4-2: ***Loss and accuracy graphs for R2P1D Flow stream models.*** *Each models are trained with different freezing points.*

For freezing points, only low and middle depth of layers are considered due to network convergence issue. From the graph output at *Figure 4-2*, the performance of each models are distinctive from each other, which model with no frozen layers apparently yields the most satisfying result compared to its peers with frozen layers. The outcome proves the hypothesis

on convergence difficulty, which requires the entire network to be fully optimized to achieve state-of-the-art performance by adapting both class-agnostic and class-discriminative features with the new dataset.

4.2.2 RGB and Flow with Visualization

Model	Pretrained Dataset	Freezing Point	Testing Accuracy on Split 1 of UCF-101	
			10-clips Top-1	10-clips Top-5
R2P1D-RGB	Kinetics	Conv2_x	**94.34%**	**99.29%**
R2P1D-RGB with random flipping	Kinetics	Conv3_x	90.67%	99.02%
I3D-RGB	Charades	Mixed3c	89.27%	98.23%
R2P1D-Flow	Kinetics	None	**90.51%**	**97.99%**
R2P1D-Flow	Kinetics	Conv2_x	85.36%	96.14%
R2P1D-TwoStream	Kinetics	Conv2_x (RGB) None (Flow)	**95.61%**	**99.31%**

*Table 4-1: **Performance of models on split 1 of UCF-101 testing set.** (RGB and FLOW) Only two most promising RGB and Flow models are selected among peers from previous experiments for evaluation based on their generalizability and validation accuracy. (I3D-RGB) Includes as a benchmarking model to show performance difference between models with distinct architecture. (Two-Stream) Scores from each stream models are averaged with equal weightages.*

From previous section, RGB stream with freezing layers up to Conv2_x and Flow stream with no freezing layers are the most promising models, and they shall be the ideal benchmarking representative models for each respective modalities. Cross-referencing confusion matrices (or other relevant representation) and mean accuracy difference between peer models could help gaining insight on strength and weak spots of a particular stream model in classifying certain categories of action. Visualization techniques could be used for further analysis on understanding how a model recognizes (or mislabels) particular actions.

From the training stats in the previous section, Flow stream is inferior to RGB stream in validation and network convergence. Therefore, this section would be more towards analysing on root cause of this noticeable performance gap in the perspective of Flow stream.

*Figure 4-3: **Listing of class label on which RGB stream performs significantly better than Flow stream.** For each item, (Left diagram) Top bar shows accuracy of RGB stream on classifying clips labelled with that action, while the bar below shows accuracy difference with Flow stream, (Right diagram) Top action mislabelled by Flow stream on each particular label class.*

The image above shows a compilation of action label that Flow stream particularly worse at classifying, which some of the labels are mostly impeccably recognizable by RGB stream. Through these data, Flow stream particularly messes up with rough assumptions could be made are that: (1) Motion on hand movements could be ambiguous and not distinctive in the form of optical flows, leading the model misclassifying Haircut, Brushing Teeth, and Shaving Beard with similar other actions involving hands. (2) Action-in-interests appear insignificantly on the frames (too small), such as Field Hockey Penalty and Shotput, which network could not sufficiently model fine-grained details on motion of these actions. (3) Dynamic action, like Skiing, with non-static background hinders quality of optical flows.

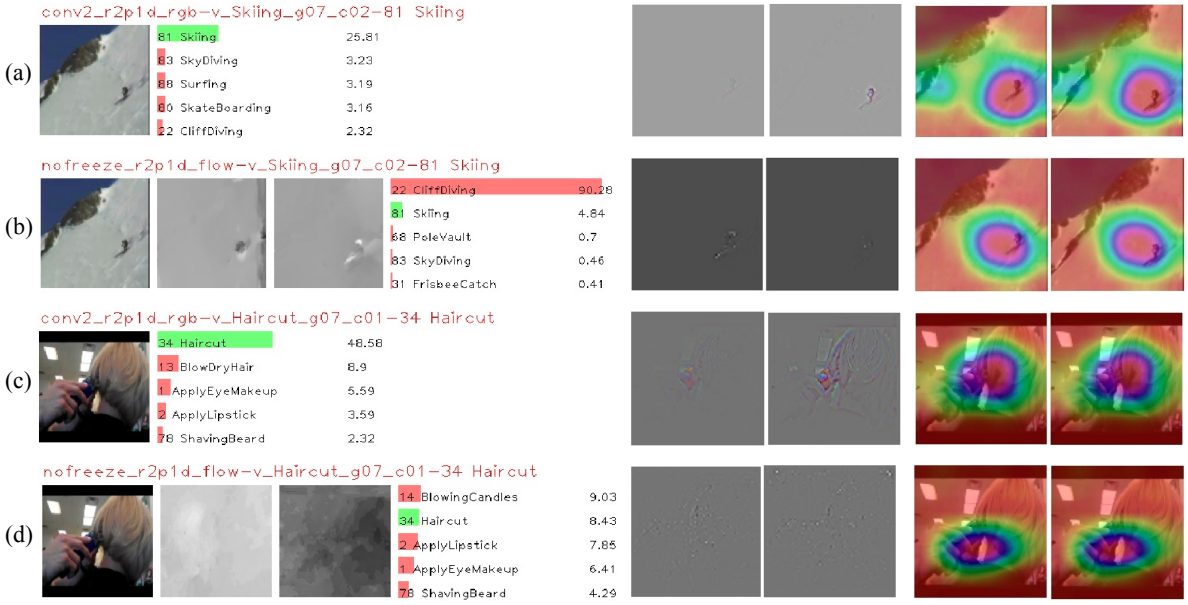


*Figure 4-4: **Visualization on Skiing and Haircut clips with RGB and Flow streams.** From left to right, each row shows prediction accuracy, GBP gradient maps and GradCAM for the first and the last frame of the clip. (a) and (c) are visualized from RGB stream, while (b) and (d) are from Flow stream.*

For further clue seeking, visualization is done on both stream classifying on testing clips labelled with these confusing actions. Through the result, observation is that Flow stream is a dedicated classifier infers based solely on temporal motion encoded on the human/object-in-interests, unlike RGB stream that has access to abundant context changes on background and inanimate objects. From the heatmaps of Skiing action, RGB stream tends to excite on snowfield in the background besides the skier, while Flow stream focuses solely on the area around action-performer. Similar phenomena is observed in Haircut action as well, where RGB stream recognizes the hair (inanimate object) besides the moving hand, which Flow stream could barely localize.

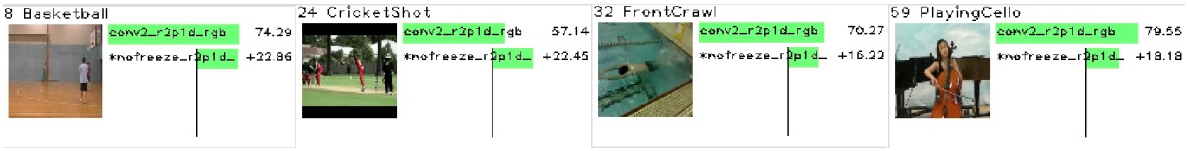


Figure 4-5: ***Listing of class label on which Flow stream performs significantly better than RGB stream.*** *Top bar displays accuracy of RGB stream on classifying clips labelled with that particular clips, while bar below shows accuracy difference with Flow stream.*

In contrary, Flow stream does perform better than RGB stream in classifying certain actions where background context could be misleading for RGB classifier. For instance, Cricket Shot is often mislabelled as Cricket Bowling in RGB stream due to that: (1) These two actions often occur on grass fields, which the context itself does not offer distinctive appearance for the network to differentiate between them. (2) Motion across temporal space provides more informative clues than the frame context. Therefore, optical flows become explicitly useful in classification when the network failed to correctly infer frame context (classifying Playing Cello as Playing Violin), which also explains the significant performance boost obtained by complementary fusion of these two modalities.

4.2.3 Visualizing Hierarchical Learning Model

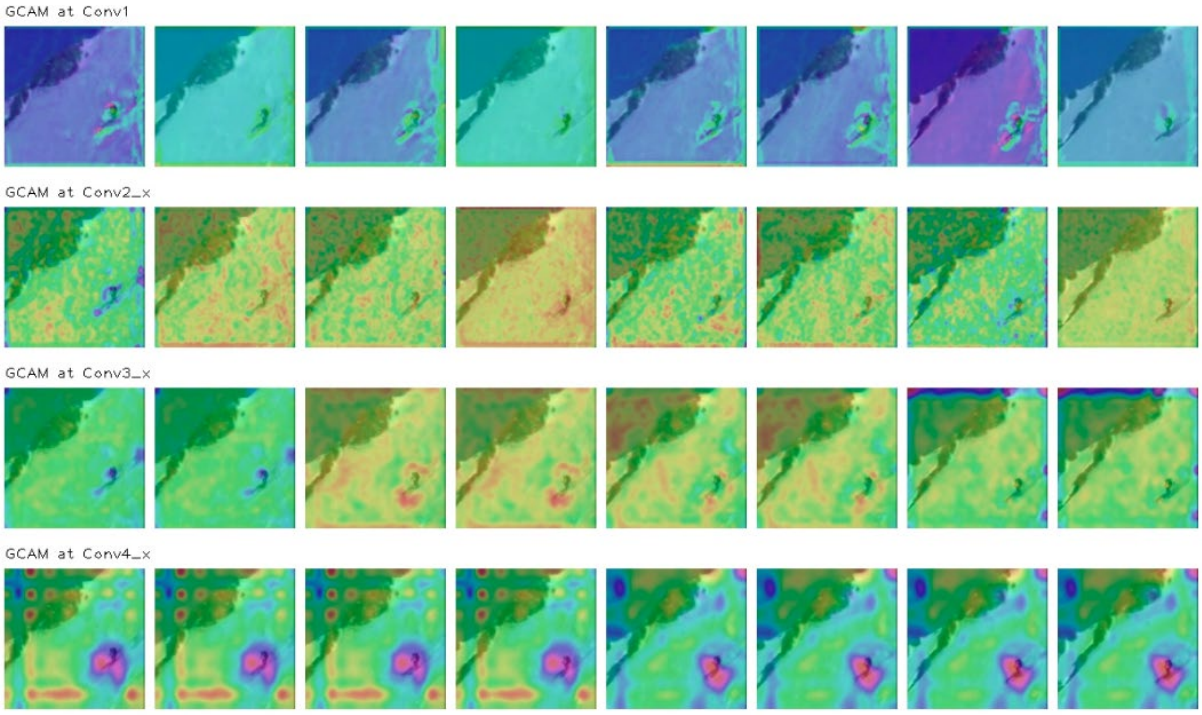


Figure 4-6: ***GradCAM output at different depth of Flow stream with Skiing clip.*** *Top to bottom rows indicates shallower to deeper layers of network.*

Through visualizing at gradients maps back-propagated at different depth of layers on the Skiing clip from section, the hierarchical learning property of Flow stream are revealed accordingly on the output.

Shallow counterparts of the stream tends to excite at everywhere in the input space, as they filter on rather simple and class-agnostic features (edges, corners, dots) that could commonly occur on the input space. Second and third rows represent as middle layers, where features learnt are getting complex and structured, which can be observed from the transitions where molecules of active gradients at started to group, and centralise on only objects or contexts associating with the action-in-interest. At the same time, excitement on background contexts started to weaken, as it becomes increasingly trivial for inferencing at deeper layers.

Localization gradually takes place at Conv4_x convolutional block, where features are mostly class-discriminative (specialized filters for only certain actions). Strong magnitude of activation on areas associated with the action diminishes influences from other areas, and eventually ends up as the gradient map at the last layer, where a smooth blob surrounds only the area with action-in-interest that particularly dictates the network prediction.

CHAPTER 5 CONCLUSION

In summary, this project attempts to replicate an ideology similar to image recognition toolset of ConvnetJS onto domain of action classification by developing a deep learning toolset that provides training, benchmarking, and visualizing modules specialized on action classification network. This toolset is suitable for academic purpose and simple research, which eases users on formulating and conducting experiments on different network models and hyperparameter sets. Module generalization allows trainer, inspection, and visualization tools to run independently on the network architecture and dataset, which indicates a great reusability of module.

For reporting, this toolset provides sufficient statistical and graphical output on training stats, validation and testing performance, and visualization on gradient maps for users to conduct analysis and make reliable comparison between models. This report demonstrates a simple analysis on the toolset output thorough experimenting with R2P1D and I3D networks. With these reporting tools, users would gain better intuition on how an action network extracts spatio-temporal features and identifies an action, and leading users a bit closer to understand nature of an action network.

Despite of these comprehensive toolset, there are still plenty of rooms for this toolset to expand and improve. If this project happens to be carried on in the future, here is a list of suggestions on continue developing this toolset:

(1) Incorporates action localization network into the toolset, by developing its relevant generalized network template, data loader, and evaluation metrics such as IoU.
(2) Further extend functionalities on network template to be coped with non-conventional action classification network, such as Flow stream of Hidden I3D, which generates own optical flows before feeding into classification network.
(3) Supports fusion module such as weighted averaging and SVM classifier, which fuses two stream models and reporting on fusion result.
(4) Investigates a GradCAM algorithm specialized for action network that preserve the temporal information, so that output heatmap would be differentiable across temporal depth on each respective input frames.
(5) Improves memory management and optimizes memory usage, deallocates memory whenever a variable or tensor is no longer in use, which could be helpful in making the program more feasible to run on low-end machines.

REFERENCES

Carreira, J., & Zisserman, A. (2017). Quo vadis, action recognition? A new model and the kinetics dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6299-6308).Feichtenhofer, C., Pinz, A., & Zisserman, A. (2016). Convolutional Two-Stream Network Fusion for Video Action Recognition. Conference on Computer Vision and Pattern Recognition (CVPR). Retrieved from https://github.com/feichtenhofer/twostreamfusion

Feichtenhofer, C., Pinz, A., & Zisserman, A. (2016). Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1933-1941).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). *Densely connected convolutional networks.* Proceedings of the IEEE conference on computer vision and pattern recognition.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). *Large-scale Video Classification with Convolutional Neural Networks.* CVPR.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images* (Vol. 1, No. 4, p. 7). Technical report, University of Toronto.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Le, D. (2018). Pseudo-3D Residual Networks. *Github*. Retrieved from https://github.com/naviocean/pseudo-3d-pytorch

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

Ozbulak, U. (2019). PyTorch CNN Visualizations. *Github.* Retrieved from: https://github.com/utkuozbulak/pytorch-cnn-visualizations

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... & Berg, A. C. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 618-626).

Sigurdsson, G. A., Varol, G., Wang, X., Farhadi, A., Laptev, I., & Gupta, A. (2016, October). Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision* (pp. 510-526). Springer, Cham.

Simonyan, K., & Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems* (pp. 568-576).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Soomro, K., Zamir, A. R., & Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*.

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2014). Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1-9).

Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 4489-4497).

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (pp. 6450-6459).

Wang, H., & Schmid, C. (2013). Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision* (pp. 3551-3558).

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

Zhu, J., Zhu, Z., & Zou, W. (2018, August). End-to-end video-level representation learning for action recognition. In *2018 24th International Conference on Pattern Recognition (ICPR)*(pp. 645-650). IEEE.

Zhu, Y., Lan, Z., Newsam, S., & Hauptmann, A. (2018, December). Hidden two-stream convolutional networks for action recognition. In *Asian Conference on Computer Vision*(pp. 363-378). Springer, Cham.

APPENDICES

A. Plug-in Customized Network into Toolset

This section provides a tutorial on building and running user-defined network in the toolset.

1. Create a new python script in toolset directory: /network.
2. In the scripts, import *torch* library, and optionally *numpy* and other libraries that users might use.

```
import torch
import numpy as np
```

3. Import *TemplateNetwork* and any other module blocks that users found useful from *network.module* library.

```
from network.module import TemplateNetwork
```

4. (Optional) Users could build their own module blocks using class inherited from *nn.Module* before compiling the final network. It helps providing abstraction on inner structure of network architecture, and simplifying presentation of network structure.

```
class SpatioTemporalResModule(nn.Module):
	def __init__(self, …):
		# Define sequential modules for this block here
	def forward(self, …):
		# Define forward propagation here
```

5. Define the network class inherited from *TemplateNetwork* interface, on which the toolset can recognize and process.

```
class R2P1D18Net(TemplateNetwork):
```

6. In the constructor, calls *add_module* function to append module or module blocks into the network class. The function parameters are layer name in *string*, and module object in *nn.Module.*

```
self.add_module('Conv2_x', SpatioTemporalResModule(…))
```

7. (Optional) If output tensors from previous layer require processing (e.g. reshape) before feeding to the next layer, calls *add_inter_proces()* function. The function parameters are name of next layer in *string*, and dict of function objects and its respective set of parameters.

```
self.add_inter_process('Linear', {torch.Tensor.view:{'size':(-1,512)}})
```

8. Calls *compile_module()* upon finish building the network at end of the constructor.

```
self.compile_module()
```

9. Save the script, proceed to main directory of the toolset and locate *BASE_CONFIG.txt*.
10. Add a new entry in the *network* section, by including attributes:
    a. *module* : Name of the script, without .py

b. *class* : Name of the network class

c. *endpoint* : List of layer names

```
"r2p1d-18":
        {
        "module":"r2p1d",
        "class":"R2P1D18Net",
        "endpoint":["Conv1", "Conv2_x", "Conv3_x", "Conv4_x", "Conv5_x", "Avgpool", "Linear", "Dropout", "Softmax"]
        },
```

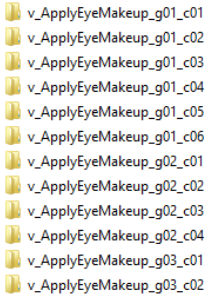
B. Plug-in Videoset

The section explains requirement on the format of videosets and procedure to register them.

1. Make sure the videoset folder fulfils the criterias below:
    a. RGB frames are extracted from actual videos beforehand and Optical flows are pre-computed from actual videos.
    b. All frames must be in *.jpg* format.
    c. Structure of the videoset folder as such:

```
dataset
|--rgb
   |--[all RGB frames]
|--flow
   |--u
      |--[all u Optical Flows]
   |--v
      |--[all v Optical Flows]
|--classInd.txt
|--trainList.txt
|--valList.txt (Optional)
|--testList.txt
```

    d. Frames must be categorized by saving them into each respective folders named with the name videos from which they are extracted as such:



2. Generates a *classInd.txt* that stores mapping between label index and name, and place it at main directory of the videoset as such:

```
1 ApplyEyeMakeup
2 ApplyLipstick
3 Archery
4 BabyCrawling
5 BalanceBeam
6 BandMarching
7 BaseballPitch
8 Basketball
9 BasketballDunk
10 BenchPress
```

3. Generates text files of train, test, and optionally validation set of mapping between video name and its respective ground truth label as such:

```
v_ApplyEyeMakeup_g11_c01 0
v_ApplyEyeMakeup_g11_c02 0
v_ApplyEyeMakeup_g11_c03 0
v_ApplyEyeMakeup_g11_c04 0
v_ApplyEyeMakeup_g11_c05 0
v_ApplyEyeMakeup_g12_c01 0
v_ApplyEyeMakeup_g12_c02 0
v_ApplyEyeMakeup_g12_c03 0
v_ApplyEyeMakeup_g12_c04 0
v_ApplyEyeMakeup_g12_c05 0
v_ApplyEyeMakeup_g12_c06 0
v_ApplyEyeMakeup_g13_c01 0
v_ApplyEyeMakeup_g13_c02 0
v_ApplyEyeMakeup_g13_c03 0
v_ApplyEyeMakeup_g13_c04 0
v_ApplyEyeMakeup_g13_c05 0
v_ApplyEyeMakeup_g13_c06 0
```

   Notes the difference on index between *classInd.txt* (starts at 1) and split files (start at 0)

4. (Optional) Users could generate more than one set of train/val/test split files.
5. Register the videoset in *dataset* section of BASE_CONFIG.txt with the following attributes:
   a. *label_num* : Count of class label
   b. *base_path* : Location of videoset on the drive
   c. *split* : Count of train/val/test set
   d. *label_index_txt* : Name of the label index-name mapping file
   e. *train/val/test_txt* : List of name of train/val/test files

```
"UCF-101":
        {
        "label_num" : 101,
        "base_path" : "C:\\Users\\Juen\\Desktop\\Gabumon\\Blackhole\\UTAR\\Subjects\\FYP\\dataset\\UCF-101",
        "split" : 3,
        "label_index_txt" : "classInd.txt",
        "train_txt" : ["ucf_trainlist01.txt", "ucf_trainlist02.txt", "ucf_trainlist03.txt"],
        "val_txt" : ["ucf_validationlist01.txt", "ucf_validationlist02.txt", "ucf_validationlist03.txt"],
        "test_txt" : ["ucf_testlist01.txt", "ucf_testlist02.txt", "ucf_testlist03.txt"]
        },
```

DEEP ACTION CLASSIFICATION

Tools for Benchmarking and Visualizing Classification Models

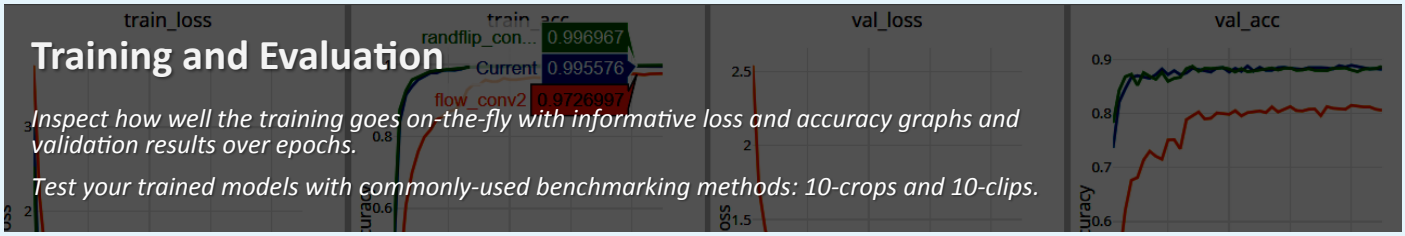
Khaw Tatt Juen (1704754)

Supervisor

Dr. Tan Hung Khoon

Experimenting with your own deep action network

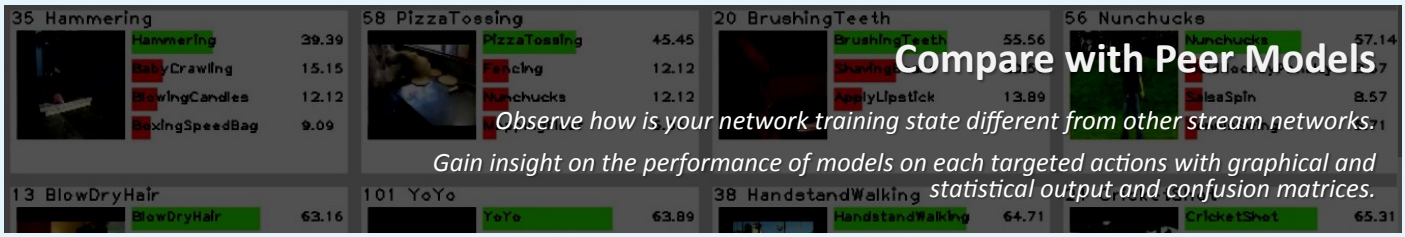
*A comprehensive toolset for academy and simple research works on **benchmarking and analysing** variety of **two-stream** deep action classification models.*



Training and Evaluation

Inspect how well the training goes on-the-fly with informative loss and accuracy graphs and validation results over epochs.

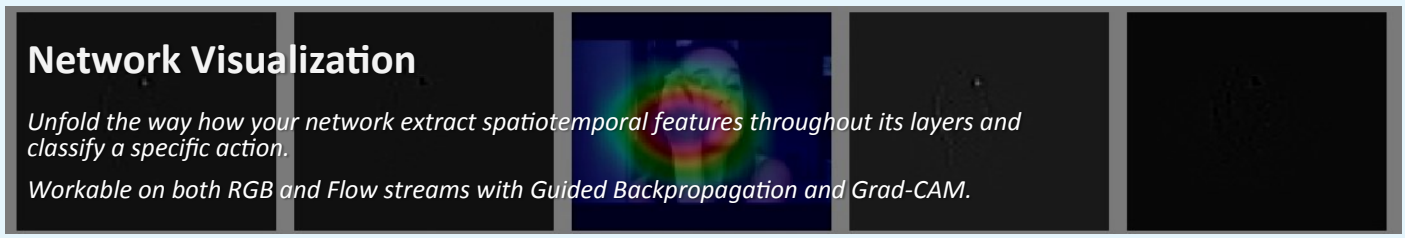
Test your trained models with commonly-used benchmarking methods: 10-crops and 10-clips.



Compare with Peer Models

Observe how is your network training state different from other stream networks.

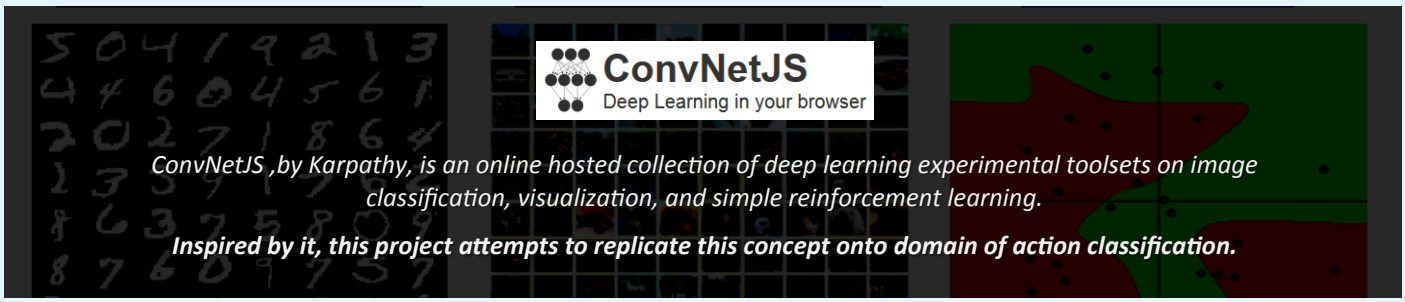
Gain insight on the performance of models on each targeted actions with graphical and statistical output and confusion matrices.



Network Visualization

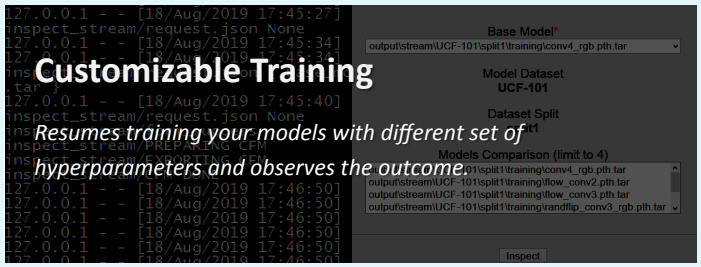
Unfold the way how your network extract spatiotemporal features throughout its layers and classify a specific action.

Workable on both RGB and Flow streams with Guided Backpropagation and Grad-CAM.



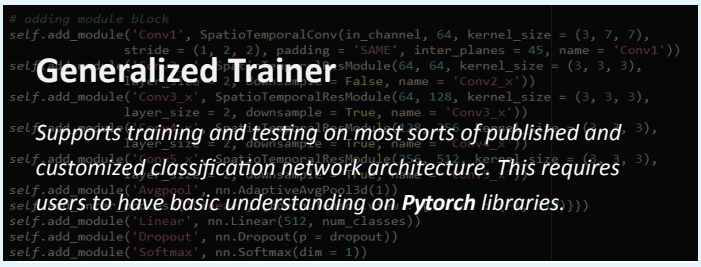
ConvNetJS ,by Karpathy, is an online hosted collection of deep learning experimental toolsets on image classification, visualization, and simple reinforcement learning.

Inspired by it, this project attempts to replicate this concept onto domain of action classification.



Customizable Training

Resumes training your models with different set of hyperparameters and observes the outcome.

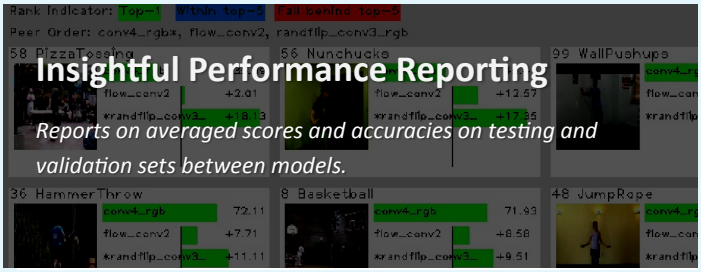


Generalized Trainer

*Supports training and testing on most sorts of published and customized classification network architecture. This requires users to have basic understanding on **Pytorch** libraries.*

Dataset Plug-in

Supports pre-processing and distribution of two-stream precomputed videosets as long as it is registered in configuration.



Insightful Performance Reporting

Reports on averaged scores and accuracies on testing and validation sets between models.

Turnitin Originality Report

Processed on: 23-Aug-2019 02:27 +08
ID: 1162323724
Word Count: 6213
Submitted: 2

FYP2-Deep_Action_Classification_Toolset_for_B... By Tatt Juen KHAW

Similarity Index	Similarity by Source	
< 1%	Internet Sources:	0%
	Publications:	0%
	Student Papers:	0%

include quoted include bibliography excluding matches < 8 words download print mode: quickview (classic) report Change mode

<1% match (publications)
Rahib H. Abiyev, Murat Arslan. "Head mouse control system for people with disabilities", Expert Systems, 2019

Ever since the remarkable success of deep network, AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) in ImageNet still-image classification challenges (Russakovsky*, et al., 2015), computer vision researchers and communities have become increasingly aware of deep learning techniques. Throughout the years after that, advancement on deep learning on image classification has been actively driven by innovative researchers, from VGGNet with small filters (Simonyan & Zisserman, 2015), GoogLeNet with network inception (Szegedy, et al., 2015), ResNet with residual learning (He K. , Zhang, Ren, & Sun, 2016), to DenseNet that connects every layers in feed-forwarding (Huang, Liu, Van Der Maaten, & Weinberger, 2017). Every bit of them has gradually lifting performance of still-image deep networks up close to human-level performance. On the other hand, action, or video classification, remains as a challenging task for deep learning researcher communities. Different from still-image, video contains temporal information that provides additional clues on motion. Therefore, action classification networks learn on not only the spatial space (RGB frames), but also the temporal space (motion), which often leads to longer training time and hinders network performance with its additional dimension of learnable parameters included. Despite of its difficulty, researcher community has never stopped their ambition on exploring solutions to tackle this domain of computer vision. Throughout the recent years, researchers have published a diversified variety of network architectures, and actively enriching our understanding on this domain. Among these publications, two-stream network, as introduced in (Simonyan & Zisserman, 2015), is one of the well-known base architectures that shapes many of the state- of-the-arts proven to have robust performance, including I3D (Carreira & Zisserman, 2017), R(2+1)D (Tran D. , et al., 2018), and DTPP (Zhu, Zhu, & Zou, 2018). Two-stream network learns on two aspects, RGB frames and optical flows, of videos independently, leading to two stream networks trained on different, respective modalities of input volume, and yielding two sets of classification scores. These scores are to be fused to produce a final score set using common method like weighted averaging. Despite the unity of two-stream networks, the family of two-stream action network itself has a great variance between each architectures in terms of network structures, input pre-processing, and training preferences. These differences are to be further discussed in Chapter 2. To truly study and evaluate these network architectures, one of the most effective is to train our own version of models, experiment, and compare with peer networks with distinct architecture. However, processes of benchmarking and comparison between models with distinct architectures are often cumbersome and inconvenient, due to issues such as: (1) Each of these networks often requires dedicated implementation of data pre-processing, training, and testing modules. (2) The metric on performance measurement could be different in terms of testing videoset, input pre-processing, and evaluation method, which a direct compare based on reported accuracy thorough the papers are often inaccurate and inconvincible if authors do not experiment with the benchmark model in interest. (3) Visualization on action networks are often lacking of attention from communities and its implementation is highly dependent to the network structure, which is often dissimilar among architectures. To resolve these problems, this project proposes a comprehensive unified toolset, which could train, evaluate, visualize variety of deep action classification models, and statistically compare between models with the same evaluation metrics applied. This project is mainly inspired by the concept of ConvnetJS, by Karpathy, that offers online deep learning experimenting toolset on image classification that allows training on all sort of conventional image classification network. Therefore, it encourages this project to replicate this idea and applies onto domain of video/action recognition, and aims to generalize the training, evaluating, and visualizing process to be compatible with most sort of conventional two-stream action networks, regardless of its internal network structure. For the rest of the report, Chapter 2 discusses on the advancement of action classification network over time from hand-crafted models to deep

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATON COMMUNICATION AND TECHNOLOGY

Full Name(s) of Candidate(s)	Khaw Tatt Juen
ID Number(s)	1704754
Programme / Course	CS
Title of Final Year Project	Deep Action Classification: Toolset for Benchmarking and Visualizing Action Classification Models

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: _____ % **Similarity by source** Internet Sources: _____ % Publications: _____ % Student Papers: _____ %	
Number of individual sources listed of more than 3% similarity: _____	

Parameters of originality required and limits approved by UTAR are as follows:
(i) Overall similarity index is 20% and below, and
(ii) Matching of individual sources listed must be less than 3% each, and
(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor
Name: _____

Date: _____

Signature of Co-Supervisor
Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	
Student Name	
Supervisor Name	

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Cover
	Signed Report Status Declaration Form
	Title Page
	Signed form of the Declaration of Originality
	Acknowledgement
	Abstract
	Table of Contents
	List of Figures (if applicable)
	List of Tables (if applicable)
	List of Symbols (if applicable)
	List of Abbreviations (if applicable)
	Chapters / Content
	Bibliography (or References)
	All references in bibliography are cited in the thesis, especially in the chapter of literature review
	Appendices (if applicable)
	Poster
	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report. _____ (Signature of Student) Date:	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. _____ (Signature of Supervisor) Date: