Internship Progress Report

Le-Vu Tran

October 1, 2018

Contents

Chapter 1

Week 1 - June 7, 2018

1.1 Introduction

I have spent the majority of my time this week learning new things to support my knowledge about Deep Learning. I also tried to grasp the idea of Places365 framework and deploy it on my own computer as well as on the Google Colab.

1.2 EPFL Deep Learning Course

To gain more experience on Deep Learning area, I started learning EPFL Deep Learning course[1] but unfortunately, I made it through without any taken notes. So lately I decided to start learning from the beginning and taking note with Jupyter at the same time. Until now, I finished 5 main lectures.

Introduction and Tensor. The first lecture was about answering these questions what is deep learning, some history, what are the current applications. The lecturer also briefly introduced the most basic unit in PyTorch which was torch.Tensor; then using PyTorch to demonstrate linear regression problems.

Machine learning fundamentals. The lecturer taught me about the theories of Empirical risk minimization, capacity, bias-variance dilemma, polynomial regression, k-means and PCA.

Multi-layer perceptrons. The lecturer firstly taught about Linear classifiers, the perceptron concept, linear separability and feature extraction. Then we moved to Multi-Layer Perceptron, gradient descent and back-propagation tech-niques.

Convolutional Networks and Autograds. This lecture was about the Generalized acyclic graph networks; the torch.autograd technique to simplify our works; batch processing technique in training state. The lecturer also introduced the idea of convolutional layers and pooling layers as well as knowledge about kernel. After all, I was taught how to use *torch.nn.Module* from PyTorch to implement all of the cool things above practically.

Optimization. In this lecture, the lecturer taught me about the Cross-entropy, L1 and L2 penalty to train a neural network. Then I was taught several techniques to optimize our works during designing and training our neural network such as vanishing gradient, weight initialization, Xavier's rule and loss monitoring. Finally I was taght to implement everything I had learnt so far by using *torch.autograd.Function*.

During the learning process I had taken some notes and they could be found here on my GitLab.

1.3 CSAIL Places

1.3.1 Introduction

The Places dataset was designed following principles of human visual cognition could be used to train artificial systems for high-level visual understanding tasks, such as scene context, object recognition, action and event prediction, and theory-of-mind inference. Thing that made this dataset more special than others was that the semantic categories of Places are defined by their function: the labels represented the entry-level of an environment. For example the dataset had different categories of bedrooms, or streets.

1.3.2 Places Challenge

The goal of this challenge was to identify the scene category depicted in a photograph. The data for this task came from the Places dataset.

For each image, algorithms would produce a list of at most 5 scene categories in descending order of confidence. The quality of a labeling would be evaluated based on the label that best matches the ground truth label for the image.

1.3.3 Places365 Convolutional Neural Networks

Introduction

Places365[2] provided several pre-trained CNN models in 2 different frameworks which were Caffe[3] and PyTorch[4]. These models was train on the Places365-Standard dataset which had 1.8 million train images from 365 scene categories. With PyTorch framework, Places365 delivered 4 pre-trained model networks AlexNet[5], ResNet18[6], ResNet50 and DenseNet161[7] to work with.

Demo

Beside released CNN models, Places365 also provided Python code to run which could be found here on their GitHub. After deploying the demo, I got these results by running on different Convolutional Neural Networks models.

We could clearly see how differently each model actually looks at the input image. The ResNet18 model only focused on one small spot on top of the waterfall to make decision. The ResNet50 instead mainly focused on 3 different spots to make decision, so maybe that was the reason why the certainty of conclusions are significantly low. The DenseNet161 model focused exactly where the waterfall was and also provided good conclusions. So I thought we should only use 2 models, ResNet18 for simplification or DenseNet161 for better conclusions (and Class Activation Map).

Figure 1.1: Input image

Using ***ResNet18***, I got the scene category results of ***0.860 waterfall***, ***0.039 mountain***, ***0.023 canyon***, ***0.013 volcano***, ***0.011 hot spring***; scene attribute results of ***natural lights, open area, natural, rugged scene, climbing***; and the class activation map below.

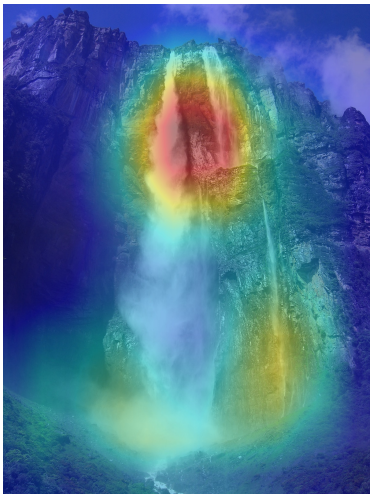


Figure 1.2: Class Activation Map Result on ResNet18

Using ***ResNet50***, I got the scene category results of ***0.129 waterfall***, ***0.055 dam***, ***0.041 fountain***, ***0.033 canyon***, ***0.030 cliff***; and the class activation map below.

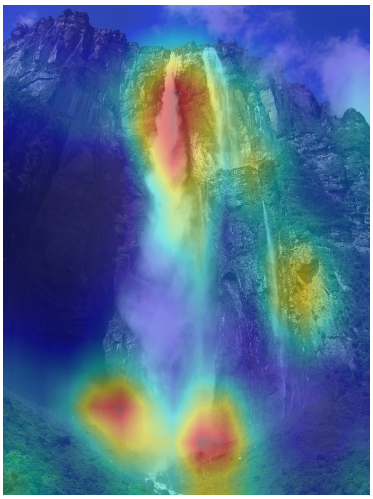


Figure 1.3: Class Activation Map Result on ResNet50

Using ***DenseNet161***, I got the scene category results of ***0.776 waterfall***, ***0.030 hot spring***, ***0.022 rainforest***, ***0.020 mountain***, ***0.018 volcano***; and the class activation map below.

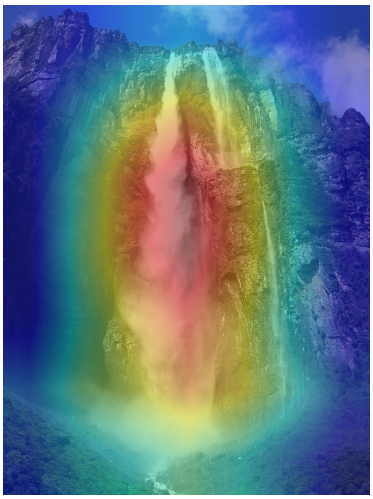


Figure 1.4: Class Activation Map Result on DenseNet161

1.3.4 Source code

In order to perform tests quickly and efficiently, I rewritten the code from CSAIL to support changing Models, Input type via arguments. My source code could found here on my GitHub.

```
usage: Places365 demo [-h] [-iu IMAGE_URL] [-vu VIDEO_URL]
                      [-if IMAGE_FILE] [-vf VIDEO_FILE]
                      [-id IMAGE_DIR] [-m MODEL]

optional arguments:
  -h, --help            show this help message and exit
  -iu IMAGE_URL, --image-url IMAGE_URL
                        Image URL (default: None)
  -vu VIDEO_URL, --video-url VIDEO_URL
                        Video URL (default: None)
  -if IMAGE_FILE, --image-file IMAGE_FILE
                        Image file (default: None)
  -vf VIDEO_FILE, --video-file VIDEO_FILE
                        Video file (default: None)
  -id IMAGE_DIR, --image-dir IMAGE_DIR
                        Image directory (default: None)
  -m MODEL, --model MODEL
                        PyTorch model: [alexnet, resnet18,
                                resnet50, densenet161]
                                (default: resnet18)
```

Figure 1.5: Usage helper

1.3.5 Class Activation Map

The authors proposed a simple technique to expose the implicit attention[8] of Convolutional Neural Networks on the image. The algorithm would highlighted the most informative image regions relevant to the predicted class. This work was published in CVPR'16. The authors also provided example code for testing purpose which could be found here on their Github.

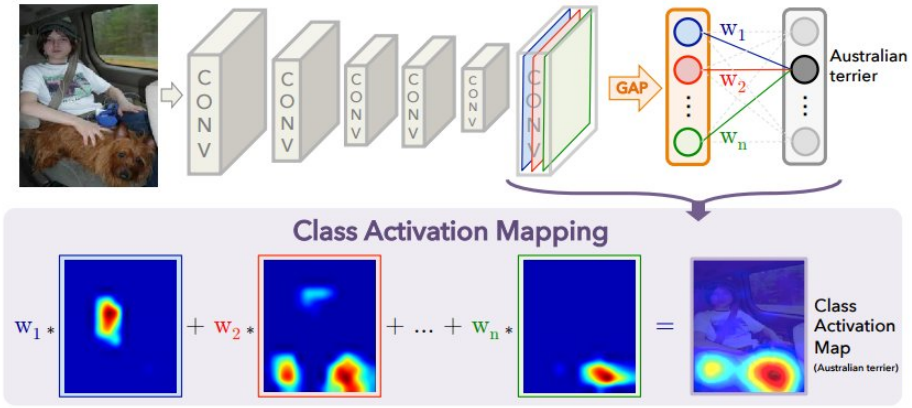


Figure 1.6: The framework of the Class Activation Mapping

The technique could be described as: for example we were running on the Resnet, first we had to capture the weight of the penultimate layer of the pre-trained network which could be the Softmax layer; secondly we forwarded our input image through the network; third we captured the output of the last layer right before Convolutional layers, in our case it was the Pooling layer; then we multiplied 2 factors above; finally we represent the result from the previous step onto our input image to get the Class Activation Map. This technique worked similarly for other Convolutional Neral Networks.

As mentioned above, the name of the last layer right before a bunch of Convolutional layers could be different in each Convolutional Neural Network. Its name could be *features* in the GoogleNet[9] and the DenseNet, *layer4* in the Resnet. Based on our purpose we had to choose the network and the right layers for the Class Activation Mapping algorithm to work properly.

By reading the implementation of these techniques proposed by the author, I had more experience about the architecture of a Torchvision model and how to explore a neural network. Using the *_modules* variable to explore the whole architecture (layer by layer) of that neural network and the *register-forward-hook* function to hook the parameters of a concrete layer during the forward pass; these two techniques helped me extracting the features of an input images. As the result, I was able to convert images to vectors by using any pre-trained Convolutional Neural Network (e.g. ResNet, Inception v3, DenseNet, ...).

Chapter 2

Week 2 - June 12, 2018

2.1 Introduction

I have spent the majority of my time this week trying to finish the EPFL Deep Learning Course EE559 as well as doing the miniproject number 1 from EPFL[10].

2.2 EPFL Deep Learning Course

This week I continued learning the EPFL course and I finished 2 more lectures. They were all complex but once I achieved the key ideas they turned out to be very fascinating.

Going deeper. The lecturer taught me about rectifiers (ReLU), dropout, batch normalization in theory and practical, the architecture of residual networks, some tips to gain good results in training state such as advanced weight initialization. The lecturer also taught me about differences in performance of GPU and CPU, programming techniques with torch.cuda in PyTorch.

Computer vision. The lecturer taught me about architectures of Deep networks for image classification such as AlexNet, VGGNet[11]; object detection algorithm YOLO[12], and semantic segmentation (FCN)[13], data-loaders, fine-tuning and neuro-surgery.

I had updated the taken notes and all of them could be found here on my GitLab.

2.3 Mini-project 1

Beside learning materials from course, I also tried implementing the mini-project 1 to gain more practical experience. The objective of this project was to train a predictor of finger movements from Electroencephalography (EEG) recordings[14]. The data and other materials also be provided by the course to help me understanding and doing the project better.

2.3.1 Problems

At the beginning, I made a big mistake that prevent me from achieving any goals which was using the Sigmoid function at the last layer of my CNN architecture. After days of training with all of my effort, the network did not gain anything and the loss was still a huge number. After searching on the internet and asking experience from my friend I figured out the problem. I fixed the architecture later and the network seemed to work well also.

2.3.2 Architecture

The data composed of 316 training recordings, and 100 test recordings, each composed of 28 EEG channels sampled at 1khz for 0.5s. Concretely, the train dataset had the shape of [316, 28, 50] and the test dataset has the shape of [100, 28, 50]. After examining the structure of dataset I started designing the appropriate Convolutional Neural Network. I tried to increase the channel from 28 to 56, then to 112 by two 1d convolutional layer and apply max pooling after each convolutional layer. After the two convolutional layers

the shape of data was [316, 112, 2]. Then I chose the nb-hidden of 100 and two fully-connected layer. Finally the output of my network was 2 numbers for classification purpose. I also used CrossEntropyLoss loss and SGD optimizer during my training process.

```
class Net(nn.Module):
    def __init__(self, nb_hidden):
        super(Net, self).__init__()
        self.conv1 = nn.Conv1d(28, 56, kernel_size=5)
        self.conv2 = nn.Conv1d(56, 112, kernel_size=5)
        self.fc1 = nn.Linear(224, nb_hidden)
        self.fc2 = nn.Linear(nb_hidden, 2)

    def forward(self, x):
        x = F.relu(F.max_pool1d(self.conv1(x), kernel_size=4, stride=4))
        x = F.relu(F.max_pool1d(self.conv2(x), kernel_size=3, stride=3))
        x = F.relu(self.fc1(x.view(-1, 224)))
        x = self.fc2(x)
        return x
```

Figure 2.1: Convolutional Neural Network architecture

2.3.3 Goals

With the learning rate of 1e-3 and after 5000 epochs. I got the train error of 0% (0/316) and the test error of 36% (36/100).

I also tried to apply dropout layer to my CNN architecture but the result was not change significantly. Maybe dropout layer had no effect on my CNN architecture, or I did not use it in the right way though. There was one thing that I was not try, the batch processing technique, I perhaps should try it later on my next convolutional neural networks.

2.3.4 Source code

All of my work could be found on my GitLab.

Chapter 3

Week 3 - June 19, 2018

3.1 Introduction

I have spent the majority of my time this week deploying, testing DLib[15] and YOLO framework.

3.2 DLib

3.2.1 Installation

To effectively manage different environments on my computer, I used Miniconda to install new frameworks. Here is the script to install DLib environment via Miniconda.

The installation script could be found in repository of colab script on my GitLab.

3.2.2 Face Detector

DLib provided a python example of Face Detector here for everybody to follow up. I referred to the example above and also made some changes to add bounding box to outputs.

DLib's Face Detector algorithm provided 2 thresholds to adjust by myself. The first threshold was the Upsampling threshold, which indicated how many times we wanted to upsample the image, that would make the input image bigger and allow us to detect more faces. The second threshold was the Detection threshold, where a negative value would return more detections and a positive value fewer.

Here were some results that I got by using DLib's Face Detector algorithm, with several Upsampling thresholds from 1 to 5, and two Detection thresholds which were the default threshold provided by DLib and my one of -1.

Figure 3.1: One person with straight face

In this case, algorithm worked well regardless of any Upsampling thresholds.

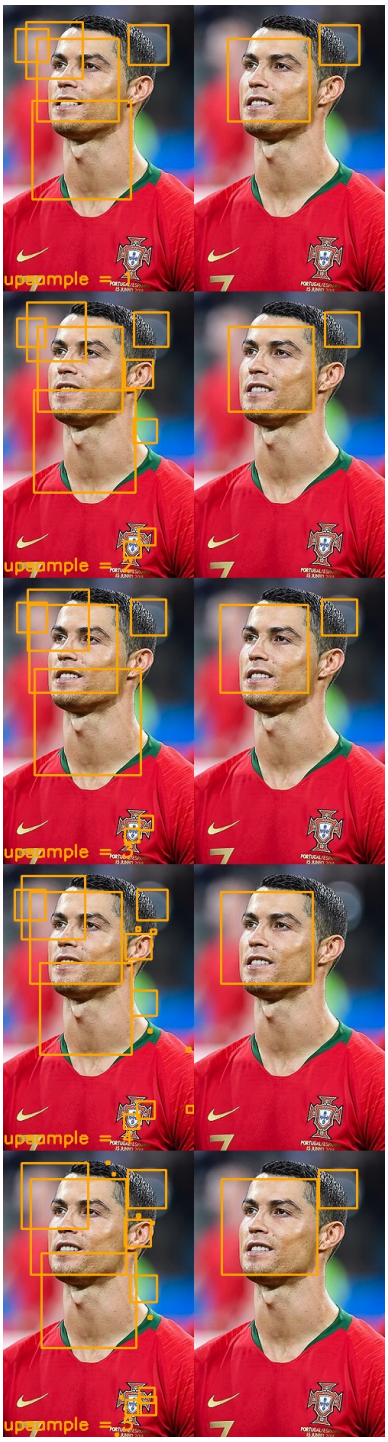


Figure 3.2: One person with side face

In this case, the only Upsampling threshold that worked well was 4 while the others also successfully detected the right face, but there was one redundant detection.

Figure 3.3: Multi people with resolution of 359x188

In this case, the default Upsampling threshold which was 1 completely failed to detect faces. When Upsampling threshold increased, DLib was being able to detect more faces, but the result was still not so good. We could see that DLib successfully detected potential faces, but after filtering the result was not so good as expect.

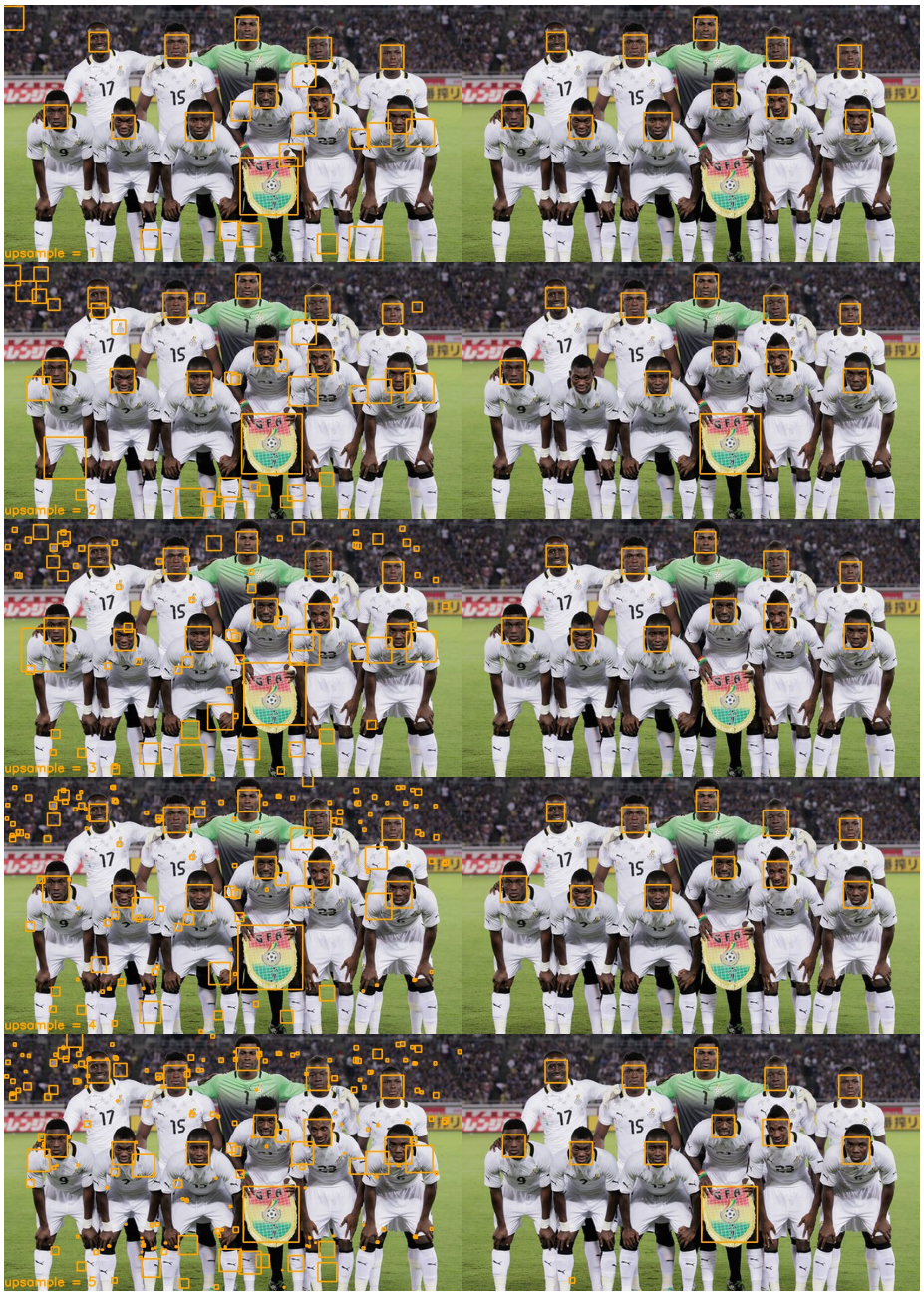


Figure 3.4: Multi people with resolution of 850x478

In this case, the default Upsampling threshold which was 1 already worked so good. But when Upsampling threshold increased, there were some cases that DLib wrongly detected the flag as a face.

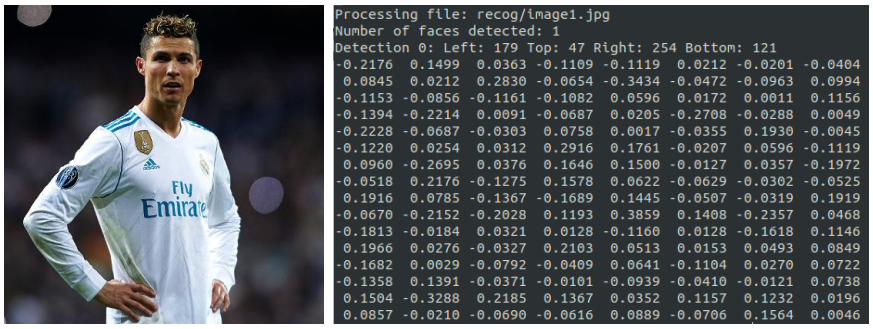
3.2.3 Face Recognition

DLib provides a python example of Face Recognition here to follow. For this algorithm I just focused on how to run it. DLib also provides trained facial shape predictor dataset here and trained recognition model here here to download.

The mission of this algorithm is to take input as an image and make output of a distinct 128 elements vector for each input person.

How to run:

```
python face_recognition.py shape_predictor.dat
dlib_face_recognition_resnet_model_v1.dat image_dir/
```



(a) Input (b) Output

Figure 3.5: Face recognition example

3.2.4 Problems

I could not find out any guide to build DLib with GPU support. So I don't know whether DLib supports GPU or not.

I don't know what to do next with the 128 elements vector provided by Face Recognition algorithm, I guess that I can use a k-d tree (k == 128) to make a recognition system.

How can we evaluate the Face Detection and Face Recognition performance? And how to test their correctness?

3.3 YOLO

3.3.1 Installation

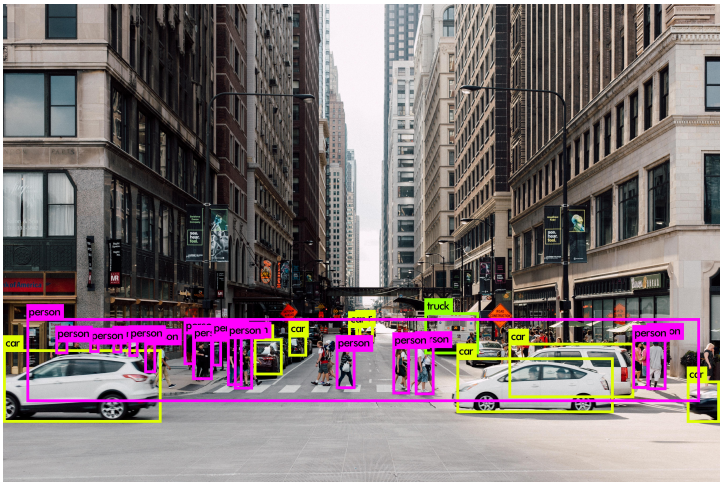
YOLO can be built with CUDA to run on GPU which is much faster than CPU. You must have CUDA installed on your machine to build YOLO with CUDA option. I already wrote a script to make the CUDA installation process easier.

There was an important note here, if I wanted to use the python example code provided by YOLO, I first must compile the YOLO framework to get the '.so' file, then I had to change the path to that file inside the python code (the file path must be full path, not relative path). And because the library itself was coded by C programming language, so every string type variable must be

transform to Byte form by inserting a character 'b' to the beginning of the string (outside the quotation mark).

3.3.2 Object Detection

(a) Input



(b) Output

Figure 3.6: YOLO Object Detection

3.3.3 Problems

I built successfully OpenCV from source, but I was not able to build YOLO with OpenCV so I could not test YOLO with video or real-time camera.

How to make a Java wrapper for YOLO to run it on Android devices?

Chapter 4

Week 4 - June 26, 2018

4.1 Introduction

I have spent the majority of my time this week learning CS231n[16] by Stanford University and try implementing a Convolutional Neural Network to classify images of CIFAR10 dataset.

4.2 CS231n

To gain more experience on Computer Vision area, I started learning Stanford CS231n course about Convolutional Neural Networks for Visual Recognition. Until now, I finished 7 main lectures.

Introduction. The first lecture was about the computer vision overview, the historical context and some course logistics.

Image Classification pipeline. The lecturer taught me about the data-driven approach, demonstrated fully K-nearest neighbor algorithm. Beyond that, the lecturer also briefly described the key idea of linear classification.

Loss Functions and Optimization.

1. Backpropagation and Neural Networks
2. Convolutional Neural Networks
3. Training Neural Networks I
4. Training Neural Networks II

All taken notes could be found here: GitLab

4.3 Image Classification using CIFAR10

4.3.1 CIFAR10 with AlexNet

I first tried CIFAR10[17] with AlexNet on PyTorch. The source code could be found here. But there were 2 problems, I did not know how to resize the input images to fit AlexNet's input requirement without using pytorch's DataLoader (from 32x32 to 224x224). But the training performance when using DataLoader was so poor. So it took me too much time to do anything. Finally I decided to drop it and started developing my own CNN architecture to work with CIFAR10.

4.3.2 CIFAR10 with DIY Net

Convolutional Neural Network Architecture

```
(features): Sequential (
    ( 0): Conv2d (in_channels=3, out_channels=16, kernel_size=3, padding=1)
    ( 1): ReLU (inplace)
    ( 2): Conv2d (in_channels=16, out_channels=16, kernel_size=3, padding=1)
    ( 3): ReLU (inplace)
    ( 4): MaxPool2d (kernel_size=2, stride=2)
    ( 5): Conv2d (in_channels=16, out_channels=32, kernel_size=3, padding=1)
    ( 6): ReLU (inplace)
    ( 7): Conv2d (in_channels=32, out_channels=32, kernel_size=3, padding=1)
    ( 8): ReLU (inplace)
    ( 9): MaxPool2d (kernel_size=2, stride=2)
    (10): Conv2d (in_channels=32, out_channels=64, kernel_size=3, padding=1)
    (11): ReLU (inplace)
    (12): Conv2d (in_channels=64, out_channels=64, kernel_size=3, padding=1)
    (13): ReLU (inplace)
    (14): MaxPool2d (kernel_size=2, stride=2)
)

(classifier): Sequential (
    (0): BatchNorm1d (1024)
    (1): Linear (1024 -> 2048)
    (2): BatchNorm1d (2048)
    (3): Linear(2048 -> 512)
    (4): BatchNorm1d(512)
    (5): Linear(512 -> 10)
)
```

Figure 4.1: Convolutional Neural Network Architecture

Result

With batch_size of 100, number of epochs of 100, learning_rate of 1e-2; we have the train accuracy of 100% (50000 / 50000) and the test error of 28.37% (2837 / 10000).

All my work and trained model could be found here.

Chapter 5

Week 5 - July 5 2018

5.1 Introduction

I have spent the majority of my time this week participating in the Summer School on Human - Computer Interaction and trying to improve the Convolutional Neural Network last week.

5.2 Summer School

This is a 1-week summer school to introduce the Human-Computer interaction field by Mr. Le Khanh Duy. In this summer school there are presentations by speakers such as Prof. Morten Fjeld, Prof. Shengdong Zhao, Prof Kening Zhu and Prof. Tran Minh Triet

Here were some certificates that I receive during Summer School.

5.3 Image Classification using CIFAR10

5.3.1 Convolutional Neural Network v2

I tried modifying some configurations on model v1 to achieve higher accuracy. I changed from BatchNorm[18] to DropOut. Here is my v2 architecture:

```
(features): Sequential (
    ( 0): Conv2d (in_channels=3, out_channels=16,
kernel_size=3, padding=1)
    ( 1): ReLU (inplace)
    ( 2): Conv2d (in_channels=16, out_channels=16,
kernel_size=3, padding=1)
    ( 3): ReLU (inplace)
    ( 4): MaxPool2d (kernel_size=2, stride=2)
    ( 5): Conv2d (in_channels=16, out_channels=32,
kernel_size=3, padding=1)
    ( 6): ReLU (inplace)
    ( 7): Conv2d (in_channels=32, out_channels=32,
kernel_size=3, padding=1)
    ( 8): ReLU (inplace)
    ( 9): MaxPool2d (kernel_size=2, stride=2)
    (10): Conv2d (in_channels=32, out_channels=64,
kernel_size=3, padding=1)
    (11): ReLU (inplace)
    (12): Conv2d (in_channels=64, out_channels=64,
kernel_size=3, padding=1)
    (13): ReLU (inplace)
    (14): MaxPool2d (kernel_size=2, stride=2)
)

(classifier): Sequential (
    (0): DropOut (0.5)
    (1): Linear (1024 -> 2048)
    (2): DropOut (0.5)
    (3): Linear (2048 -> 512)
    (4): DropOut (0.5)
    (5): Linear (512 -> 10)
)
```

I also increased the number of epochs to 300.

The training time increased from 18 mins to 55 mins, but I got the test error decreasing to 25.47% (decrease 4.49% compared to the v1 model).

5.3.2 Convolutional Neural Network v3

I continued modifying some configurations on model v2 and hoped for higher accuracy. I changed the learning rate, instead of using only one learning rate during training process, I tried 3 different learning rates which are:

- 1e-1 for epochs [0, 100)
- 1e-2 for epochs [100, 200)
- 1e-3 for epochs [200, 300]

The training time was not change, and I got the test error decreasing to 23.9% (decrease 1.57% compared to the v2 model).

And so far v3 model is the best one in term of the test error that I have.

5.3.3 Source code

All of my model architectures, detailed configurations, results and trained model could be found on my GitHub.

Chapter 6

Week 6 - July 12 2018

6.1 Introduction

I have spent the majority of my time this week grasping the key ideas of Generative Adversarial from CVPR-2017[19] and ICCV-2017[20].

6.2 CVPR-2017

Lecture video could be found here.

This lecture was taught by MingYu Liu, Jan Kautz and Julie Bernauer. Because the sound quality was so bad and this lecture was aimed at people that already have deep knowledge about what generative adversarial network is, so it was real hard for me to understand what they was saying.

I only captured a few things about generative adversarial network and all of my taken note could be found here on my GitHub.

6.3 ICCV-2017

Lecture videos could be found here.

This was a combination of multiple lectures:

1. Introduction to GAN[27]
2. Autoencoder GANs
3. Conditional GANs[24], StackGAN[25], StackGAN v2[26]
4. GANs in the Wild
5. Evaluating Generative Models
6. Domain Adversarial Learning[22][23]
7. Visual Synthesis and Manipulation with GANs
8. Do GANs learn the distribution?
9. Connections between adversarial training and RL
10. GANs as Learned Loss Functions

All of my taken note could be found here on my GitHub.

Chapter 7

Week 7 - July 19 2018

7.1 Introduction

I have spent the majority of my time this week grasping the key ideas of Generative Adversarial from CVPR-2018[21] and comparing the CVPR-2018 with CVPR-2017[19] and ICCV-2017[20].

7.2 CVPR-2018

Lecture videos could be found here.
This was a combination of multiple lectures:

1. Introduction to GAN[27] and SAGAN[28][29]
2. Paired Image-to-Image Translation[30][31]
3. Unpaired Image-to-Image Translation with CycleGAN[32][33]
4. Do GANs learn the distribution?
5. Learning Disentangled Representations with an Adversarial Loss
6. VAE-GAN Hybrids
7. Multimodal Unsupervised Image-to-Image Translation
8. Adversarial Domain Adaptation
9. Adversaries for Detection and Action[34][35]
10. Generative Adversarial Imitation Learning
11. Video Generation and Prediction

I only captured new things compared to ICCV-2017 and CVPR-2017 about generative adversarial network and all of my taken note could be found here on my GitHub.

7.3 CycleGAN Demo

[36]

7.3.1 horse2zebra

(a) Input (b) Output

Figure 7.1: Horse to zebra with black horse

I thought this was an easy input for CycleGAN to deal with and the result was quite good although the background environment changed well, the horse's tail was not realistic at all.

(a) Input (b) Output

Figure 7.2: Horse to zebra with brown horse

This test was extremely successful due to the correctness, the quality of the output. The results for this succeed maybe because the horse occupied most of the area of the input images and the pose of horse was really ideal.

(a) Input (b) Output

Figure 7.3: Horse to zebra with white horse

This test was failed. I did not know how to explain this case.

7.3.2 zebra2horse

(a) Input (b) Output

Figure 7.4: Zebra to horse case 1

The generated horse was good, but one more time the background environment was significantly changed.

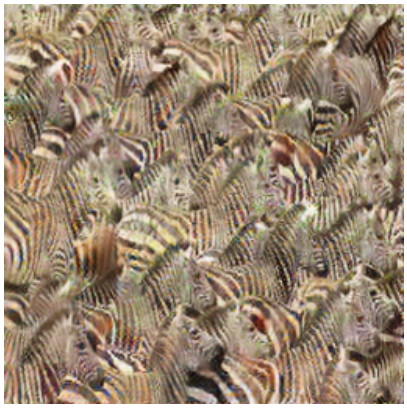
(a) Input

(b) Output

Figure 7.5: Zebra to horse case 2

The environment background was also changed, the horse's color was good but its rear legs were weird.

(a) Input

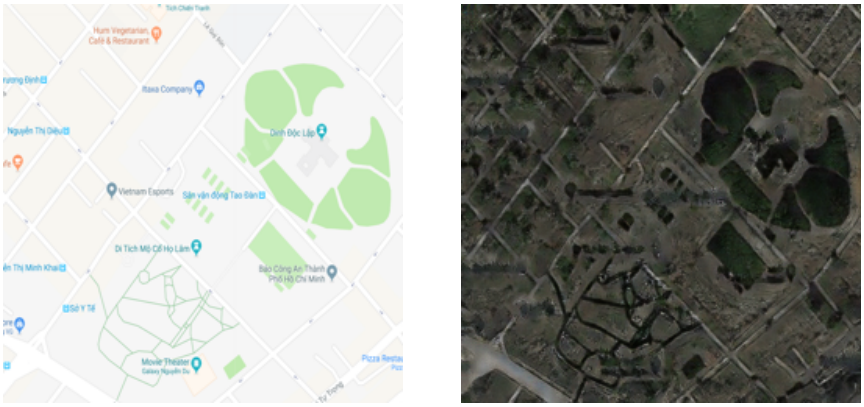


(b) Output

Figure 7.6: Zebra to horse case 3

This input was too hard for CycleGAN and I was not surprised when Cygle-GAN was completely failed in this case.

7.3.3 map2sat



(a) Input (b) Output

Figure 7.7: Map to satellite

I did not find interesting in this model. The result was not good at all. It just replace the green part of the map by ”tree” in the satellite, the white and yellow parts by ”land”. All of those stuffs were not impress me at all.

7.3.4 sat2map

(a) Input (b) Output

Figure 7.8: Satellite to map

This was a little bit better than the map2sat model, but I thought these two were still for entertaining purposes rather than serious situations in real life.

7.3.5 style_vangogh

(a) Input (b) Output

Figure 7.9: Vincent van Gogh style transfer

This paint was impressive, but I did not know such a thing about painting valuation so I could not tell whether this was Vincent van Gogh' styles or not.

7.4 Image Style Transfer using CNN[37]

This repository was a TensorFlow implementation of Image Style Transfer using Convolutional Neural Networks[38] and some other techniques[39][40]. The Neural Style algorithm synthesizes a pastiche by separating and combining the content of one image with the style of another image using Convolutional Neural Networks (CNN). These techniques were not supported or improved by GAN.

The content was one of my favorite pictures, the style image was ***The Starry Night***[41]. And the result was far from which I had imagined, it was so beautiful and high-detailed.

Watching this result I had a thought that why people need to "improve" the current CNN with GAN, the generated images' quality from GAN were much far from this.

(a) Content

(b) Style

(c) Result

Figure 7.10: Style transfer with one input style

Chapter 8

Week 8 - July 26 2018

8.1 Introduction

I have spent the majority of my time this week having a look at tasks from MultiMediaEval 2018[42], examining the dataset and considering appropriate strategies.

8.2 Medico Multimedia[43]

8.2.1 Description

The Medico Task tackled the challenge of predicting diseases based on multimedia data collected in hospitals with two additional requirements: perform the analysis efficiently and use as little data as possible for training.

Beside the 2 main tasks there were 2 more optional subtasks which were basic reporting required built system to generate summarization of a video in terms of how many different diseases and findings have occurred (e.g. if the same polyp occurs twice it should only be counted one time for the summary); and advanced reporting required the built system to automatically create a text-report for a physician for three video cases.

8.2.2 Dataset

MediaEval provided 2 different datasets which were ***Medico-2018-development-set*** and ***Medico-2018-development-set-features***.

The first one was the image-type dataset to work with, containing 5293 images with multiple resolutions, 16 classes. The image files were encoded using JPEG compression[44], extension of the image files was ".jpg".

The second one contained the extracted visual feature descriptors for all the image from the first dataset. The extracted visual features were stored in separate folders and files named accordingly to the name and the path of the corresponding image files. The extracted visual features were the global image features, namely: JCD, Tamura, ColorLayout, EdgeHistogram, AutoColorCorrelogram and PHOG. Each feature vector consists of a number of floating point values. The size of the vector depends on the feature. The size of the feature vectors are: 168 (JCD), 18 (Tamura), 33 (Color-Layout), 80 (EdgeHistogram), 256 (AutoColorCorrelogram) and 630 (PHOG). The extension of the extracted visual feature files was ".features".

After examining the dataset I had the result of number of images per class: ***polyps*** had 613 images, ***dyed-lifted-polyps*** had 457 images, ***ulcerative-colitis*** had 457 images, ***esophagitis*** had 444 images, ***normal-pylorus*** had 439 images, ***normal-z-line*** had 437 images, ***dyed-resection-margins*** had 416 images, ***normal-cecum*** had 416 images, ***retroflex-stomach*** had 398 images, ***stool-plenty*** had 366 images, ***colon-clear*** had 267 images, ***retroflex-rectum*** had 237 images, ***blurry-nothing*** had 176 images, ***stool-inclusions*** had 130 images, ***instruments*** had 36 images, ***out-of-patient*** had 4 images.

8.2.3 Models

ResNet[6]

My reconfigured ResNet model could be found on my GitHub.

8.3 Emotional Impact of Movies[45]

8.4 Media Memorability

8.4.1 Description

The Predicting Media Memorability Task[46] required participants to build systems predicting how memorable a video was, by computing a memorability score for each input video.

Beside the main task there were 2 more subtasks which were Short-term Memorability and Long-term Memorability[47]. The Short-term Memorability subtask involves predicting a short-term memorability score for a given video. The Long-term Memorability subtask involves predicting a long-term memorability score for a given video.

8.4.2 Dataset

Overview

The dataset was composed of 10,000 short soundless videos (named from video1.webm to video10000.webm); 8,000 elements (randomly picked) for the development set and the other 2,000 elements for the test set.

Each video came with its original title (might be useful to infer the memorability of video); its short-term memorability score; the number of annotations which was used to calculate its short-term memorability score; its long-term memorability score; the number of annotations which was used to calculate its long-term memorability score.

Data

All 8,000 videos were placed in /source folder and named with format of video{}.webm; their original titles were placed in dev-set_video-captions.txt. Each video was 7 seconds long and soundless.

Figure 8.1: video124 – children-help-their-mother-mix-ingredients

Figure 8.2: video9740 – flipping-meat-on-grill

Ground Truth

The videos came with initial memorability scores, defined as the percentage of correct detections by participants, for both short-term and long-term memory performances.

video	short-term_memorability	nb_short-term_annotations	long-term_memorability	nb_long-term_annotations
video10.webm	0.95	34	0.9	10
video100.webm	0.951	33	0.889	9
video10000.webm	0.832	33	1	13

Figure 8.3: Example of provided Ground Truth

Pre-computed Features

There were also total 7 pre-computed content descriptors in 2 different types which were video-dedicated feature and frame-based feature.

Video-decicated feature.

C3D spatio-temporal visual features[48], obtained by extracting the output of the final classification layer of the C3D model was trained on UCF101 dataset[49] (101 action categories), a 3-dimensional convolutional network proposed for generic video analysis; structured in a single list of numbers (dimension = 101).

HMP[50], the histogram of motion patterns for each video; structured in a single list of pairs of numbers with format: bin:number (dimension = 6075).

Frame-based feature (extracted on three key-frames - first, middle and last frames).

HoG descriptors (Histograms of Oriented Gradients)[51], calculated on 32x32 windows on a grey scale version of each frame; structured in single list of numbers on one line (dimension = depends on the image size).

LBP (Local Binary Patterns)[52], calculated for patches of 8x15 pixels; structured in a single list of numbers on one line (dimension = depends on the image size).

InceptionV3 features[53], correspond to the output of the fc7 layer of the InceptionV3 deep network; structured in a single list of pairs of numbers with format imagenet-class:activation (max dimension = 1,000).

ORB features[54], resulted from a fusion of FAST keypoint detector and BRIEF descriptor; structured in a list of keypoints and descriptors[55].

Color histograms, computed in the HSV space; structured in 3 lists (Red, Green, Blue) of 255 pairs with format bin:number.

These pre-computed features first seemed to be helpful but all of its concepts were so old compared to the year of 2018. And because the purpose of these pre-trained features was to facilitate participation from various communities so they all seemed to be not so useful for us.

Evaluation

The official evaluation metric for both subtasks would be the Spearman's rank correlation between the predicted memorability scores and the ground-truth memorability scores.

Chapter 9

Week 9 - August 2 2018

9.1 Introduction

I have spent the majority of my time this week taking part in the Media Memorability challenge from MediaEval 2018 with my teammate.

9.2 Feature Extracting

9.2.1 From Video to Frames

With 10,000 videos from the whole dataset, I used *ffmpeg* to extract frame by frame of each video; as the result, I got about 80,000 frames (images) from the dataset. At the beginning, I just only extracted 3 main frames of a video at 3 time intervals (beginning, middle amd end), but since the length of a video was only about 8 seconds, l so I decided to extract all 8 frames (1 frame per second).

This was the command line which I used to extract frames from a video. ***ffmpeg -i video1.mp4 -r 1 video1-%02d.jpg***. The ***-i*** indicated input file; ***-r*** was the number of frames per second I wanted to extract (e.g. *-r 1* meant 1 frame per second, *-r 0.1* meant 1 frame every 10 seconds); and the term ***%02d*** was used by ***ffmpeg*** to name the output frames.

9.2.2 Using TensorHub

TensorFlow[58] provided TensorHub which was a library for reusable machine learning modules. TensorHub maintained several modules to extract image feature vectors like Inception, MobileNet, ResNet and DenseNet.

I successfully extracted images' features but the result stucked in the variables themselves and I could not write those values to text files or numpy pickles. According to my research, I had to construct my code in a TensorFlow session to read those values, but I still got tons of errors by TensonFlow.

I would have to learn more about how to use TensorFlow correctly and try my luck again.

9.2.3 Using PyTorch

According to the comparison graph by TowardDataScience, I chose ResNet50 and Inception v3 to extract image feature because of their acceptable ratio of accuracy to complexity.

The length of feature vector of both ResNet and Inception v3 was 2048. For each video there was a corresponding numpy file (***.npy***) containing a 2d numpy array of size [8, 2018]. In order to load the numpy file into variable to perform computiation, I used ***video = numpy.load(filename)***; and ***input = numpy.stack(input, video)*** to concatenate multiple videos into one 3d array of size [8000, 8, 2048]. The concatenation was not expensive at all because it took only 5 seconds to concatenate 8,000 videos (avoid using git ***torch.cat()*** due to its poor performance).

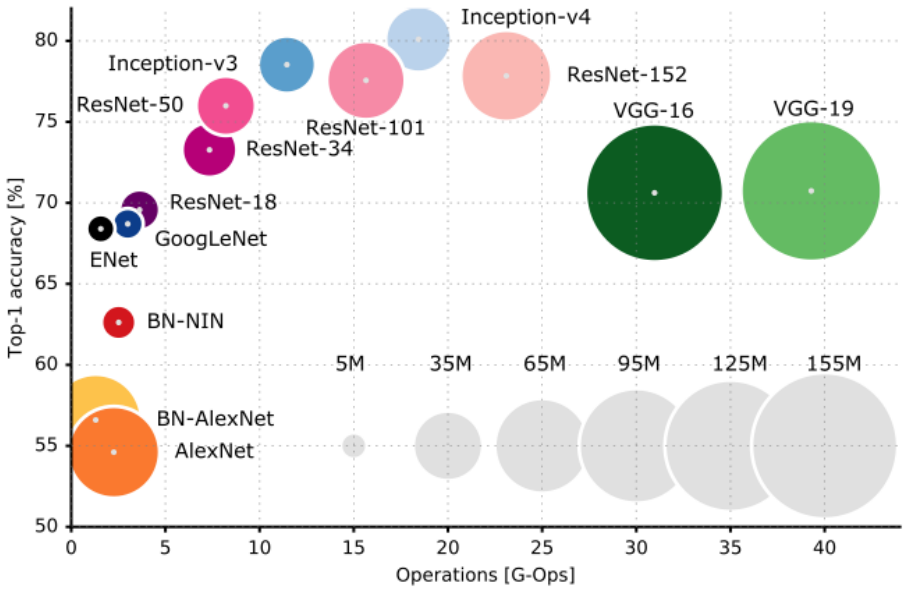


Figure 9.1: In-depth analysis and comparison of all networks[56]

9.3 Evaluation Metrics

9.3.1 Pearson Correlation Coefficient

One of the three evaluation metrics proposed by MultiMediaEval was Pearson Correlation Coefficient. It was the measure of the linear correlation between two variables X and Y. It had a value between +1 and -1, where 1 was total positive linear correlation, 0 was no linear correlation, and -1 was total negative linear correlation. Pearson correlation coefficient was the covariance of the two variables divided by the product of their standard deviations.

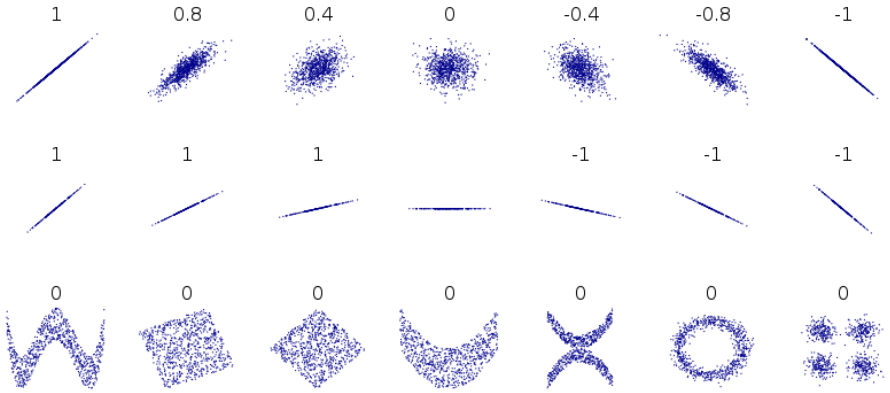


Figure 9.2: Pearson Correlation Coefficient of x and y for several (x, y) sets

9.3.2 Spearman's Rank Correlation Coefficient

The second evaluation metrics proposed was the Spearman's Rank Correlation Coefficient. It assessed how well the relationship between two variables could be described using a monotonic function. The Spearman correlation between two variables would be high when observations had a similar rank between the two variables, and low when observations have a dissimilar (or fully opposed for a correlation of -1) rank between the two variables. The Spearman correlation between two variables was equal to the Pearson correlation between the ranked values of those two variables; while Pearson's correlation assessed linear relation-ships, Spearman's correlation assessed monotonic relationships. The term ranked variables refered to the data transformation in which numerical or ordinal values were replaced by their rank when the data were sorted (e.g. given the numerical data 3.4, 5.1, 2.6, 7.3; the ranks of these data items would be 2, 3, 1 and 4 respectively).

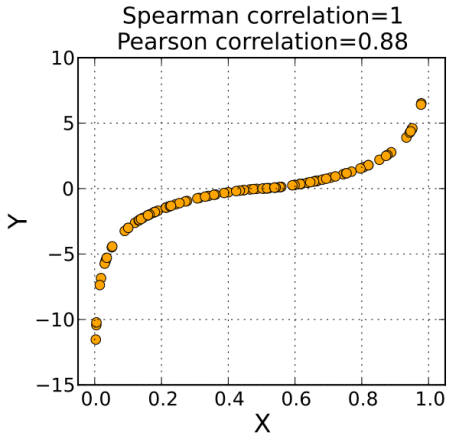


Figure 9.3: Spearman Correlation Coefficient of 1

9.3.3 Mean-Squared Error

The last evaluation metric was the Mean-Square Error between two variables. I did not think this metric could be helpful for my approaches.

9.3.4 Scipy Stats

Both Spearman's Rank Correlation Co-efficient and Pearson Correlation Coefficient were supported by Scipy.Stats. The usage of each metric could be found here and here.

9.3.5 Initiative

After learning about Spearman's Rank Correlation Coefficient, I figured out that I did not need to compute the memorability score for each video in this challenge. Another potential approach was to sort the video list for its memorability. For example, given three videos A, B and C with theirs grouth

truth memorability score of ***[0.2, 0.9, 0.4]***; I had to generate a list of ranking in term of their memorability starting with the lowest one which was ***[1, 3, 2]***. Computing Spearman's Rank Correlation Coefficient on ***[0.2, 0.9, 0.4]*** and ***[1, 3, 2]*** would give me the correlation of ***1.0***, which was the best score of this metric.

9.4 LSTM Network on Extracted Features

9.4.1 Architecture

I first try to connect one Long - Short Term Memory[57] cell with one fully-connected layer to classify between 2 classes. According to an example from PyTorch, they preferred training the network sample by sample, so I tried to do that too. And the network worked, so I tried to stack 8 Long - Short Term Memory cells to test (I thought that I had 8 images per video, so stacking 8 Long - Short Term Memory cells a time was a good choice). The figure below almost demonstrated correctly what I had done so far; the key difference was that at the final <end> gate, I also added one fully-connected layer of size [2048, 2] as a classifier.

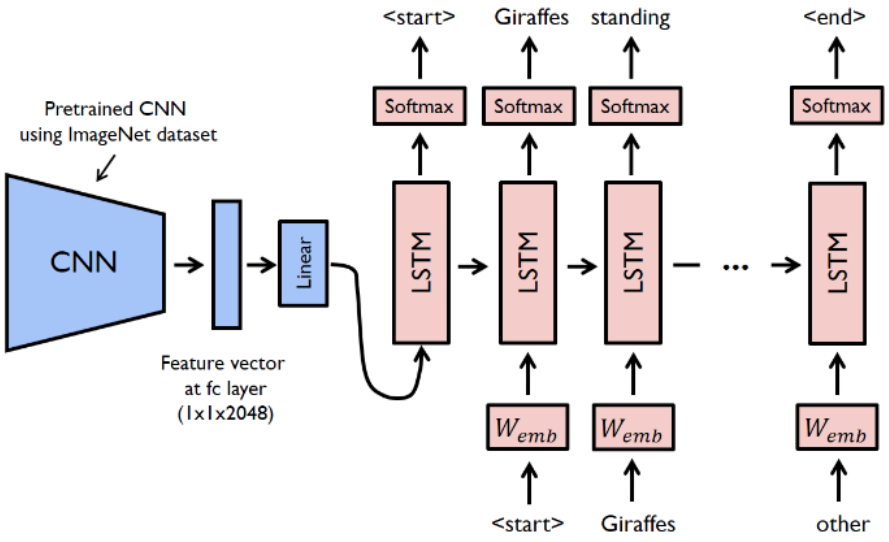


Figure 9.4: Long - Short Term Memory Network

In training process, there were some key differences such as I need to clear out the gradients before each input video (using *model.zero_grad()*) because PyTorch accumulated them. And I also need to clear out the hidden state of the Long - Short Term Memory, to detach the cell from its history on the last input video.

9.4.2 Problems

This was the first time I worked with Long - Short Term Memory cells. Although I had already obtained the main idea of Long - Short Term Memory, but in PyTorch I was so confused with the Long - Short Term Memory stacking, training, designing techniques. And I did not know how to evaluate my network, whether it was right or wrong, did I violate any rules or would it work effectively as I expected.

I did not think that my network was reliable. One epoch costed so much time, about 2.5 minutes, so if I wanted to train my network with 300 epochs it would take me 12.5 hours. I had trained my network for 150 epochs until Colab shut my session down. And even though after 150 epochs, I still got a 5-digit loss number (around 22,000).

Chapter 10

Week 10 - August 9 2018

10.1 Introduction

I have spent the majority of my time this week struggling with Long - Short Term Memory network, seeking for applicable video classification methods and figuring out what made an image memorable.

10.2 Video classification methods

Matt Harvey's paper[62] was about introducing some appropriate video classification methods. The author compared 5 different methods on UCF-101[49] to come up with the final results and his source code could be found here on his GitHub page. Here was the summary of 5 video classification methods introduced by the author.

Classify one frame at a time with a Convolutional Neural Network. Using this method meant we had ignored the temporal features of video and attempt to classify each video by looking at just a single frame. The most recommended technique for this method was using transfer learning to retrain a pre-trained Neural Network on different datasets such as ImageNet. We literally looked at each frame independently, and classified the entire video based solely on that one frame rather than looking at all the frames and doing any sort of averaging or max-ing. After training on UFC101, the author got the final test accuracy of 65% top 1.

Use a time-distributed Convolutional Neural Network, passing the features to an Recurrent Neural Network, in one network. The author suggested first using the Kera's TimeDistributed wrapper distribute layers of Convolutional Neural Network across an extra dimension - time; secondly, forwarding this wrapper through our Recurrent Neural Network. This model must be trained from scratch. After training, the author got the final test accuracy of 20%, which was much far from the baseline above.

Use a 3D Convolutional Neural Network. This seemed to be a reasonable method because 3D ConvsNet applied convolutions (and max pooling) in the 3D space, where the third dimension in our case was time. The author produced a network called C3D that achieved 52.8% accuracy on UFC101. After training, the author got the final test accuracy of 28%, which was also far from our baseline.

Extract features with a Convolutional Neural Network, pass the sequence to a separate Recurrent Neural Network. First, we ran every frame from every video through Inception (or some other networks), saving the output from the final pool layer of the network (feature). So we effectively chop off the top classification part of the network so that we ended up with a 2,048-d vector of features that we couls pass to our RNN. Second, we converted those extracted features into sequences of extracted features. The author stated the this model achieved better than the CNN-only results. The concretely accuracy number was 74%, which was significantly higher than our baseline.

Extract features from each frame with a Convolutional Network Network and pass the sequence to an Multi-Layer Perceptron. The author applied the same Convolutional Neural Network extraction process as in the previous method, but instead of sending each piece of the sequence to an Recurrent Neural Network, we would flatten the sequence and pass the new ***[40,***

2048] input vector into a fully connected network (a multilayer perceptron). After training, the author got the final test accuracy of 70%, which was higher than our baseline but lower than the previous method.

10.3 What makes an image memorable?

10.3.1 Introduction

After getting stuck in trying to improve the accuracy of my network architecture, I started researching which factors made an image or video memorable. I found a relevant paper in CVPR 2011 about this article which was ***What makes an image memorable?***[63].

Among the many reasons why an image might be remembered by a viewer, the authors investigated first the following factors: color, simple image features, object statistics, object semantics, and scene semantics.

10.3.2 Color and simple image features

The authors proposed a figure that demonstrate the correlation between memorability and basic pixel statistics. Mean hue was weakly predictive of memory: as mean hue transitions from red to green to blue to purple, memorability tended to go down (ρ = -0.16). This correlation might be due to blue and green outdoor landscapes being remembered less frequently than more warmly colored human faces and indoor scenes. Mean saturation and mean value, on the other hand exhibited even weaker correlations with memorability.

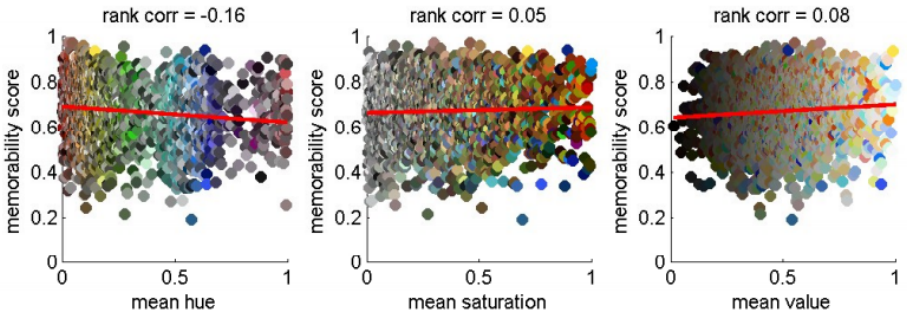


Figure 10.1: Correlation between Simple image features and Memorability.

These findings concorded with other work that had shown that perceptual features were not retained in long term visual memory[64]. In order to make useful predictions, more descriptive features were likely necessary.

10.3.3 Object statistics

Using LabelMe[65], each image in target set was segmented into object regions and each of these segments was given an object class label by a human user (e.g "person," "mountain," "stethoscope"). In this section, the authors quantified the degree to which the data could be explained by non-semantic

object statistics. The authors found that none of these statistics above made good predictions on their own.

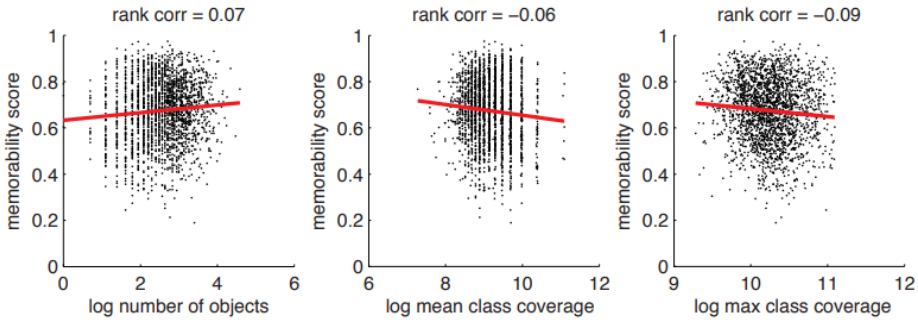


Figure 10.2: Correlation between Simple Object Statistics and Memorability 1.

Simple object statistics (log number of objects, log mean pixel coverage over present object classes, and log max pixel coverage over object classes) did not correlate strongly with memorability (ρ = 0.07, -0.06, and -0.09 respectively).

Beside that the authors also marginalized across classes to generate histograms of Object Counts, Object Areas and Multiscale Object Areas. And these factors' correlation were even smaller than those of the color and simple image features.

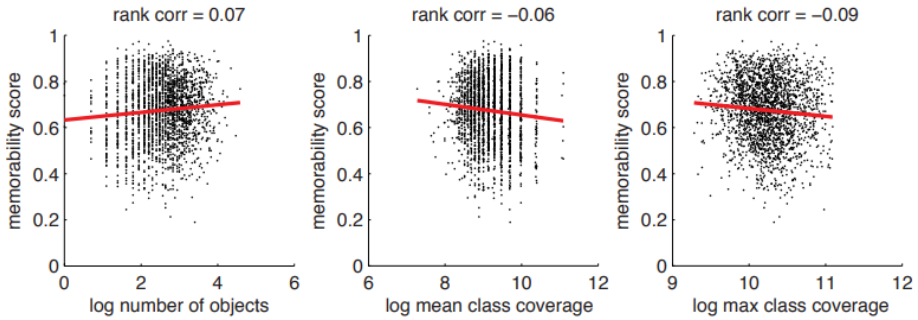


Figure 10.3: Correlation between Simple Object Statistics and Memorability 2.

10.3.4 Object and scene semantics

As demonstrated above, objects without semantics were not effective at predicting memorability. Using regression on the entire joint (object class, statistic), the authors proposed a comparision table of predicted versus measured memorabilities.

	Object Counts	Object Areas	Multiscale Object Areas	Object Label Presences	Labeled Object Counts	Labeled Object Areas	Labeled Multiscale Object Areas	Scene Category	Objects and Scenes	Other Humans
Top 20	68%	67%	73%	84%	82%	84%	84%	81%	85%	86%
Top 100	68%	68%	73%	79%	79%	82%	82%	78%	82%	84%
Bottom 100	67%	64%	64%	57%	57%	56%	56%	57%	55%	47%
Bottom 20	67%	63%	65%	55%	54%	53%	52%	55%	53%	40%
ρ	0.05	0.05	0.20	0.43	0.44	0.47	0.48	0.37	0.50	0.75

Figure 10.4: Comparision of Predicted versus Measured Memorabilities.

Even the Object Label Presences alone, which simply convey a set of semantic labels and otherwise did not describe anything about the pixels in an image, performed well above the authors' best unlabeled object statistic, Multiscale Object Areas ($\rho = 0.43$ and 0.20 respectively). Moreover, Scene Category, which just gave a single label per image, appeared to summarize much of what makes an image memorable ($\rho = 0.37$), and the best method was to combine both object and scene semantic information ($\rho = 0.50$). These performances supported the idea that object and scene semantics were a primary substrate of memorability.

10.3.5 Visualizing what makes an image memorable

Since object content appears to be important in determining whether or not an image would be remembered, the authors further investigated the contribution of objects by visualizing object-based memory maps for each image.

This visualization gave a sense of how objects contribute to the memorability of particular images. The authors additionally interested in which objects were important across all images so they estimated an object's overall contribution as its contribution per image, calculated as above, averaged across all test set images in which it appeared with substantial size. The authors sorted objects into an intuitive ordering: people, interiors, foregrounds, and human-scale objects tend to contribute positively to memorability; exteriors, wide angle vistas, backgrounds, and natural scenes tend to contribute negatively to memorability.

Figure 10.5: Objects sorted by their predicted impact on memorability.

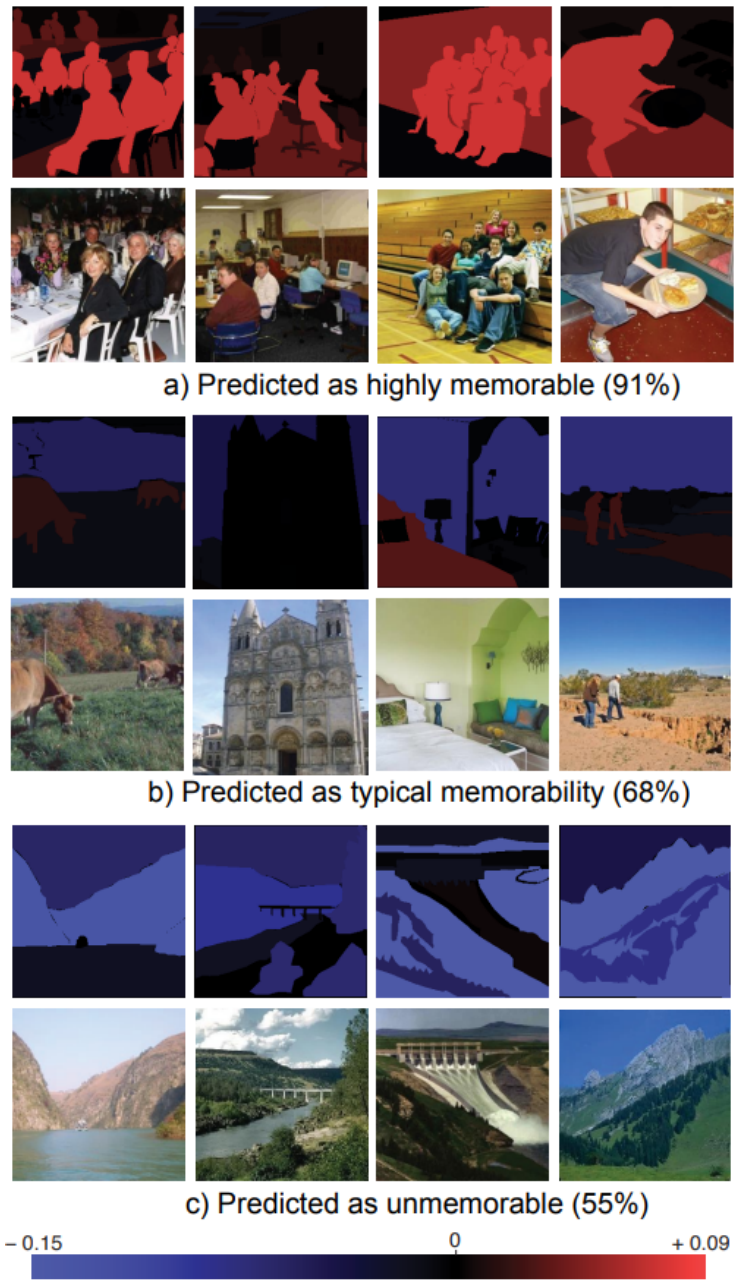


Figure 10.6: Visualization of how each object contributes to the memorability of sample images spanning a range of memorability predictions. In red were objects that contribute to higher predicted memorability and in blue were objects that contribute to lower predicted memorability. Brightness was proportional to the magnitude of the contribution.

10.4 PyTorch LSTM Classification Network

10.4.1 Architecture

I misunderstood between the concept of number of layers and number of Long - Short Term Memory (LSTM) cells so the training time was nearly 4 times longer than normal. For more details, the number of LSTM cells that PyTorch used was actually the quantity of sequences we feed into the LSTM network at once. So in my case it was 8 (because each video had 8 frames, and I feed all 8 frames once). The number of layers term in PyTorch meant how many times I wanted to stack the 8-LSTM-cell sequence to form a $\boldsymbol{8 \times num\_layers}$ grid of LSTM cells.

Each LSTM cell had multiple LSTM units, and the quantity of units was defined by ***hidden_size*** parameter. The figure below described correctly the Long - Short Term Memory model architecture in PyTorch.

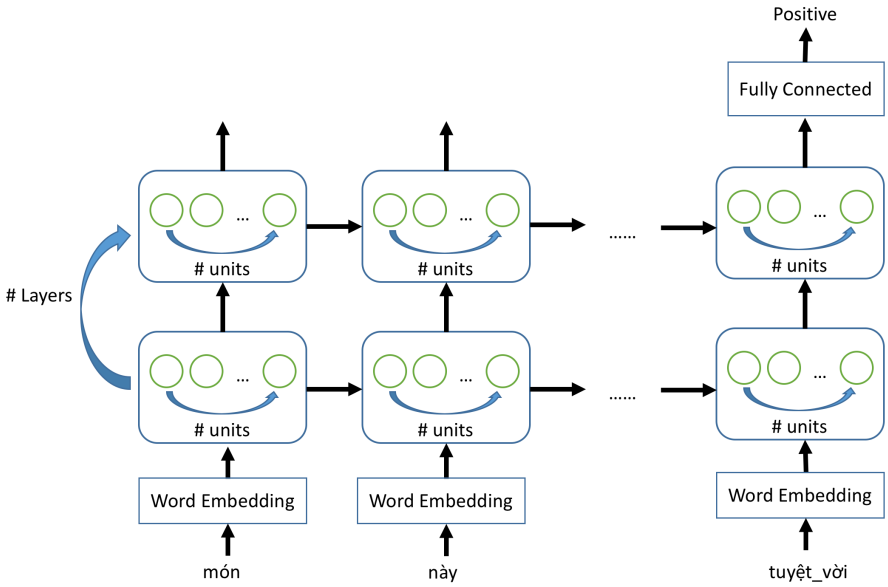


Figure 10.7: Long - Short Term Memory Network Architecture.

The smallest unit in the network above was the LSTM unit, multiple LSTM units formed one LSTM cell, each cell took responsibility for one input feature (multiple input features formed the input sequence). Multiple LSTM cells formed one LSTM layer. Stacking multiple LSTM layers formed one Stacked LSTM network.

10.4.2 Training on Dev-Set (extracted by ResNet50)

Strategy

The purpose of this challenge was to calculate the memorability score for each video, and this calculation was so ambiguous so I broke this problem into the classification problem of ***11*** categories. So after getting the classify results, I would devide them by 10 to get numbers which were similar to the scores

required by the challenge. My strategy was to splitted the provided Dev-Set for this challenge into two parts, since the Dev-Set had 8000 videos, I picked ***6000*** videos for training and ***2000*** for testing.

Version 1

My network architecture contained 2 parts; the first part was the Long - Short Term Memory part, took the sequence of input and gave a sequence of output; the second part was the Fully-connected layer acted as the classifier. I used a sequence of 8 LSTM cells took in input of size *[8, 2048]* and lately produced output of size ***[8, 8]***. Next, I flattened that array and passed it through the Fully-connected Layer to finally got the classification score for *11* classes. There was a figure below to demonstrate exactly how my architecture was.

I trained this architecture with ***200*** epochs and at the learning rate of ***1e-1***. But the result was not impressive to me at all. I think because the learning rate was too high for this kind of network. The loss and accuracy were completely turbulent over time.

The maximum accuracy was ***26%*** and the minimum value of loss was ***74*** but those values were not reliable.

This architecture took me ***3 hours*** to train 200 epochs (***54s*** per epoch).

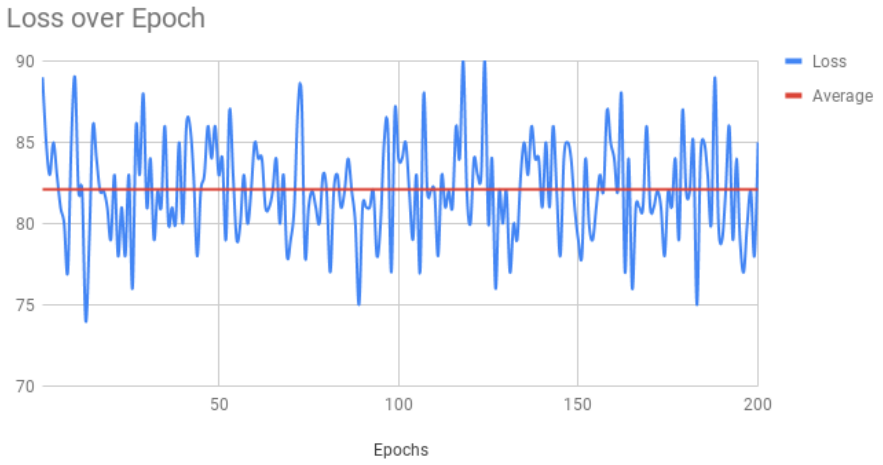


Figure 10.8: Loss over Epoch.

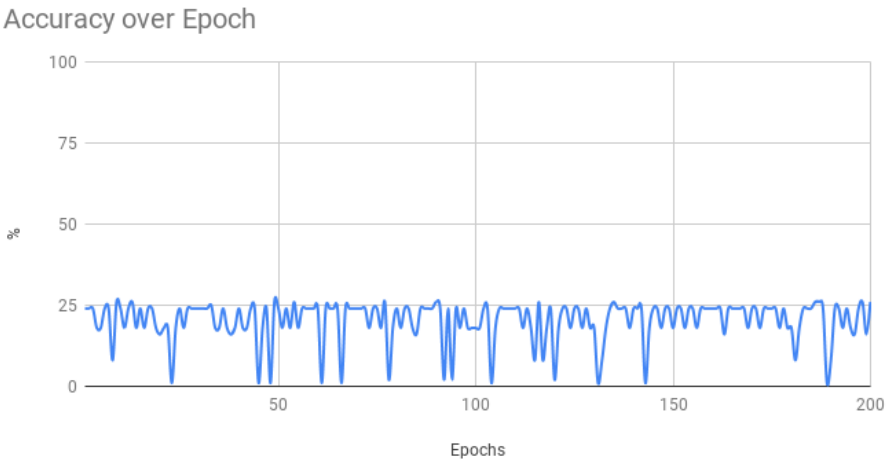


Figure 10.9: Accuracy over Epoch.

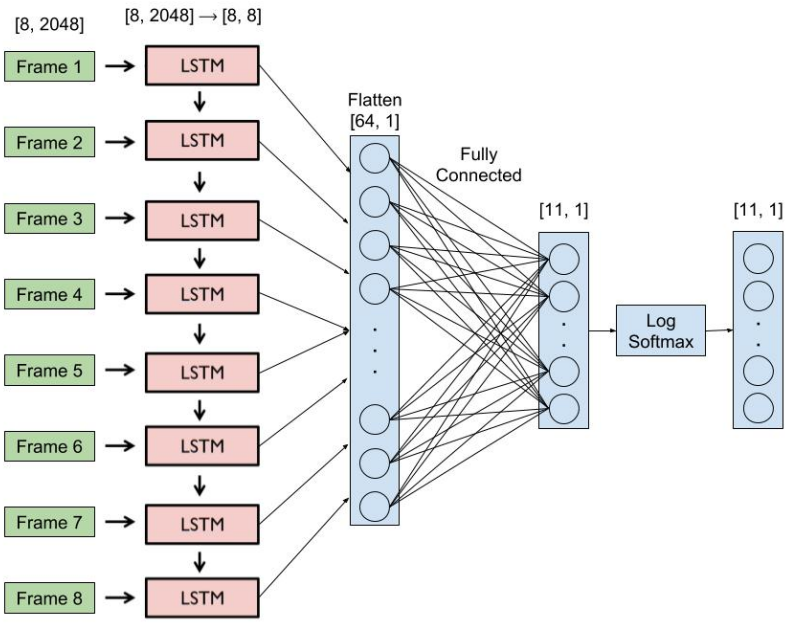


Figure 10.10: LSTM Classification v1.

Version 2

I had edited some parameter of my network architecture and change the the learning rate as well as the number of epochs. I changed the output size of Long - Short Term Memory cells from *[8, 8]* to ***[8, 1024]*** to determine whether I would have better results or not. I also decreased the learning rate to ***1e-5*** and the number of epochs to ***90*** (the number of epochs was actually 100 but I got my Colab session shut down and the loss was 0 anyway so I kept that number at 90).

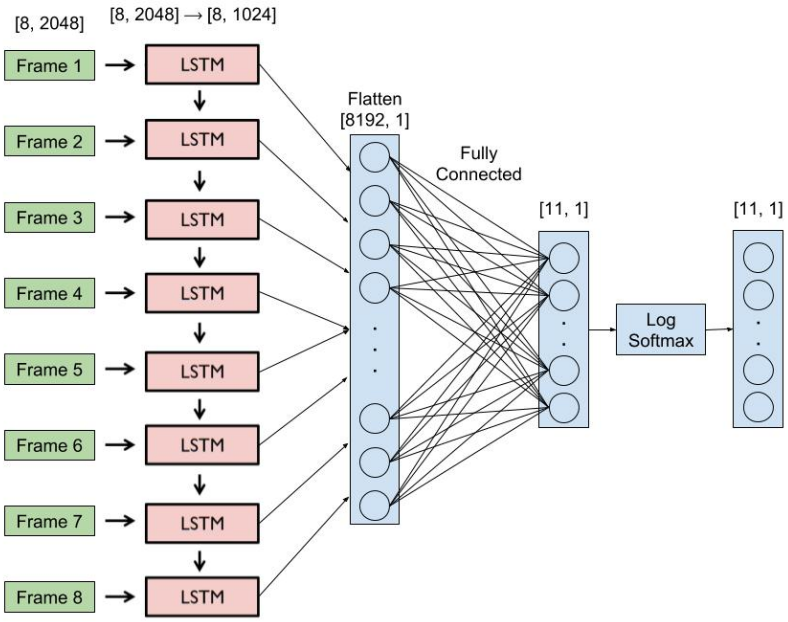


Figure 10.11: LSTM Classification v2.

I did not understand why after modifying the parameter of the network and learning rate, the initial loss value was drop from 89 to 1.8. The loss value was dropping over time as I expected, but unfortunately the accuracy was dropping too. At the time I got my loss value of 0, my test accuracy was just ***21.6%***. The maximum accuracy was also ***26%*** and the minimum value of loss was ***0*** but those values were not reliable. These figures below described the loss value and accuracy over epoch.

This architecture took me around ***2 hours*** to train 90 epochs (***1.3 mins*** per epoch).

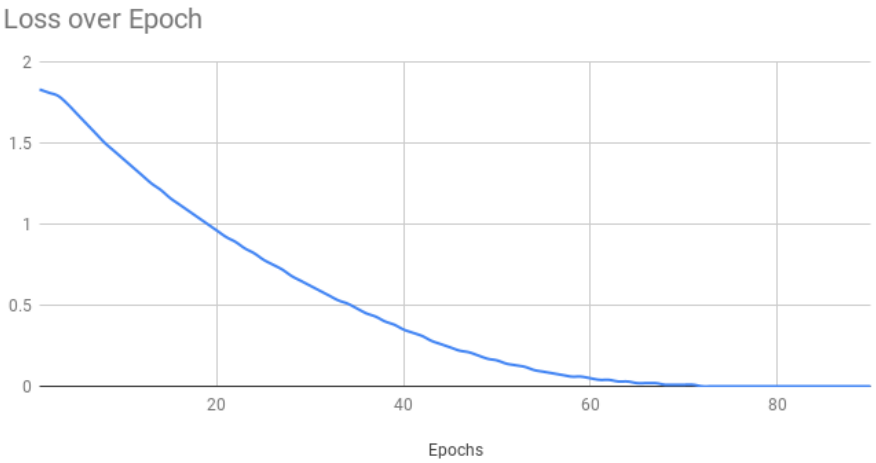


Figure 10.12: Loss over Epoch.

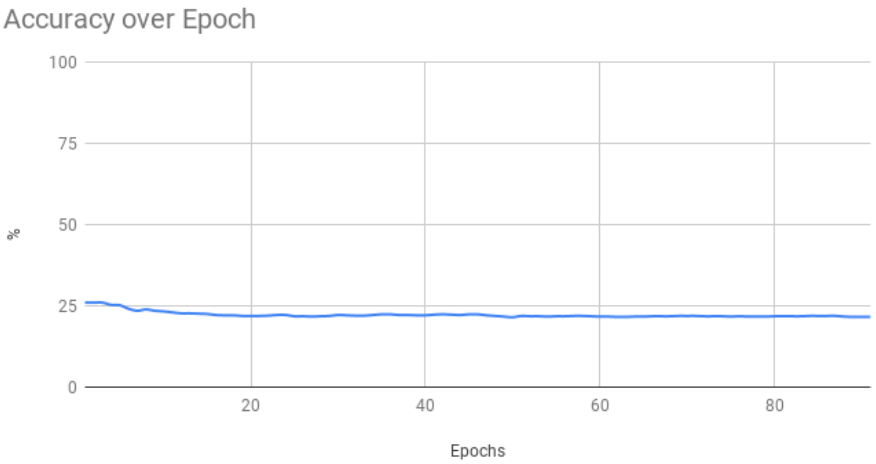


Figure 10.13: Accuracy over Epoch.

10.4.3 Training on LaMem-Set (extracted by Inceptionv3)

Introduction

Large - scale Image Memorability[61] was a dataset from MIT containing original 60000 images from diverse sources and corresponding memorability scores for memorability predicting purpose. The authors also introduced a Convolutional Neural Network with fine - tuned deep features that outperformed all other features by a large margin, reaching a rank correlation of 0.64, near human consistency (0.68). The authors could generate a memorability maps (likely Class Activation Map) for each image by using the analysis of the responses of the high - level CNN layers to shown which objects and regions were positively, and negatively, correlated with memorability.

Strategy

I used the same architecture as the Dev-Set version v2 above to test this dataset. As the dataset had 55000 images when I downloaded it, I splitted this dataset into two parts, 45000 images for training and 10000 images for testing. I also have to change the number of Long - Short Term Memory cells in my network architecture to 1 because I had only one image at a time. I used Tensorflow's pre-trained Inceptionv3 convolutional neural network to convert those images above into feature vectors and then used them as the input of my network.

Version 1

I used the learning rate of ***1e-5*** and the number of epochs of ***20*** (it was a small number because this dataset was huge and took so much time to deal with, and my purpose was just to test my network architecture). This figure below demonstrated how my network architecture was.

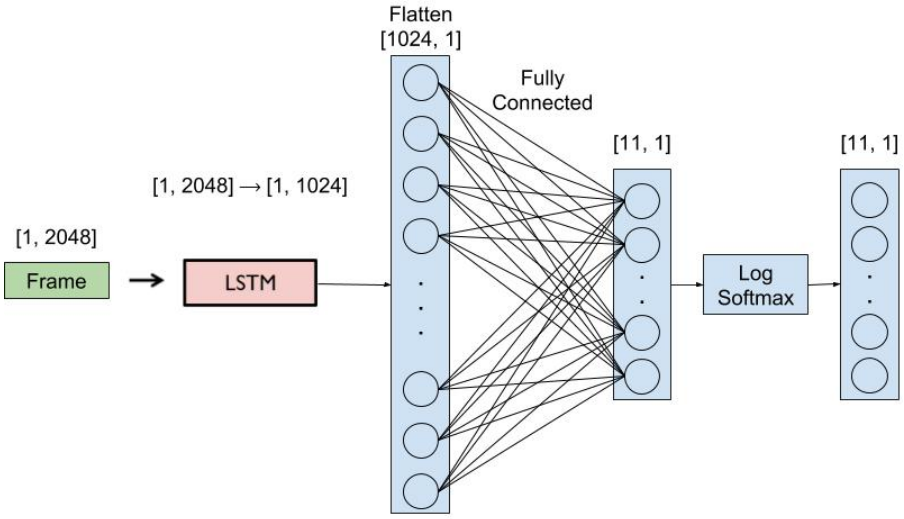


Figure 10.14: LSTM Classification for LaMem v1.

Because I only trained my network on this dataset for 20 epochs so the loss value was nearly remain from the initialization. In the other hand, the accuracy was too turbulent also. So I guess that my network architecture had some defects so it was not stable at all. The maximum accuracy was ***31%*** and the minimum value of loss was ***16.26***. These figures below described the loss valueand accuracy over epoch.

It took me around ***1.8 hours*** to train 20 epochs (***5.5 mins*** per epoch).

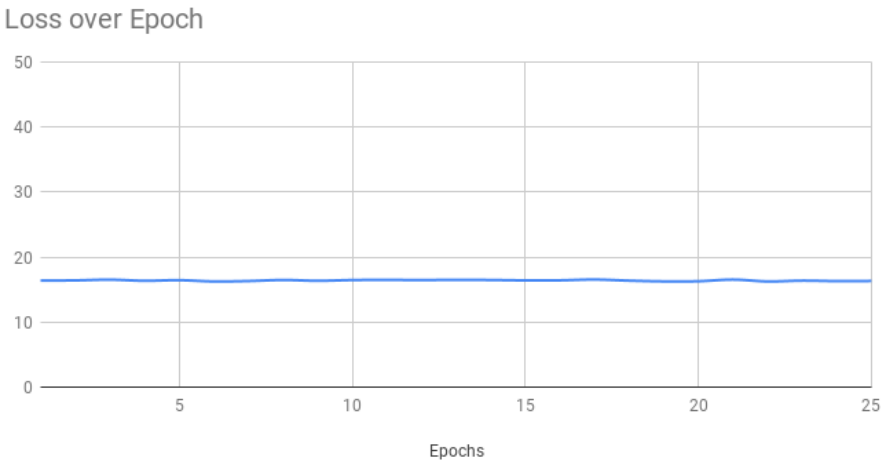


Figure 10.15: Loss over Epoch.

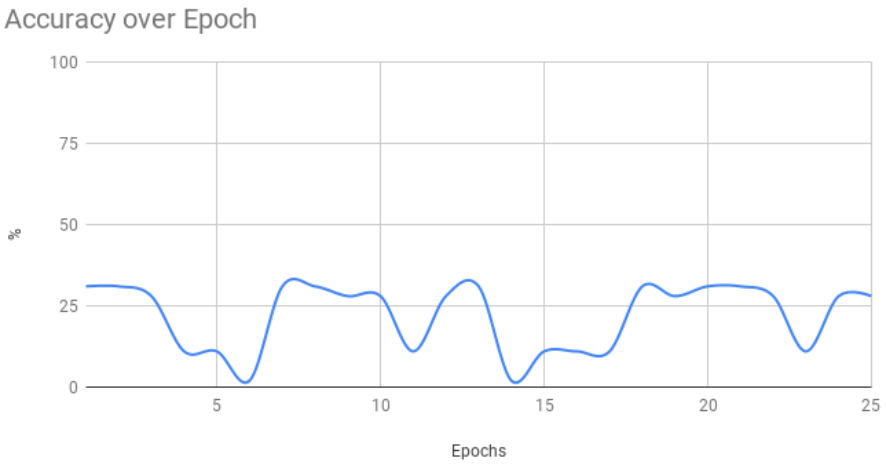


Figure 10.16: Accuracy over Epoch.

Chapter 11

Week 11 - August 16 2018

11.1 Introduction

I have spent the majority of my time this week trying my Long - Short Term Memory network with features extracted from Places365's pre-trained ResNet18, learning how to implement LSTM in TensorFlow.

11.2 Places365 Caption

This week I also run Places365 on images to analyze scene categories and attributes. And I concatenated 10 most relevant attributes structuring caption for a concrete image. Finally we tried to used that caption as input of any Convolutional Neural Network to predict images' memorabilities. I rewrote the source code and it could be found here on my GitLab.

11.3 TensorFlow LSTM Classification Network

11.3.1 Architecture

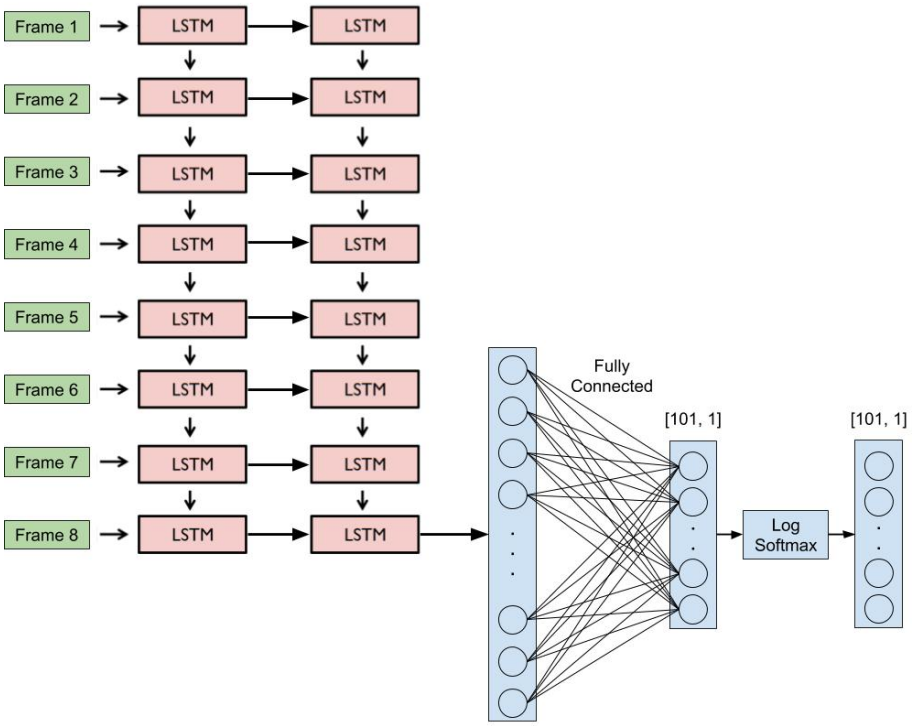


Figure 11.1: Long - Short Term Memory Network Architecture.

This week I tried to implement Long - Short Term Memory network using TensorFlow library. I finally figured out what was the definition of graph and session term in TensorFlow. I also learnt some new TensorFlow functionalities

for example Graph, Session, DropoutWrapper, BasicLSTMCell, MultiRNNCell, AdamOptimizer and argmax.

I re-designed my LSTM network using TensorFlow library and the figure below would demonstrate clearly how my network was. I used a stack of 2 LSTM cells sequences and only took the last LSTM cell's output for my Fully - connected layer. In each version I changed some hyper parameters to figure out whether I could increase the accuracy of not.

11.3.2 Training on Dev-Set (extracted by ResNet50)

Strategy

This times I broke the problem into the classification problem of 101 categories. Because I thought this dataset was pretty small for deeplearning so I splitted the provided Dev-Set for this challenge into two parts, since the Dev-Set had 8000 videos, I picked ***7000*** videos for training and ***1000*** videos for testing.

Version 1

In this version, I trained a batch of size ***128*** at once, with ***512*** LSTM units for each LSTM cell and ***7000*** iterations.

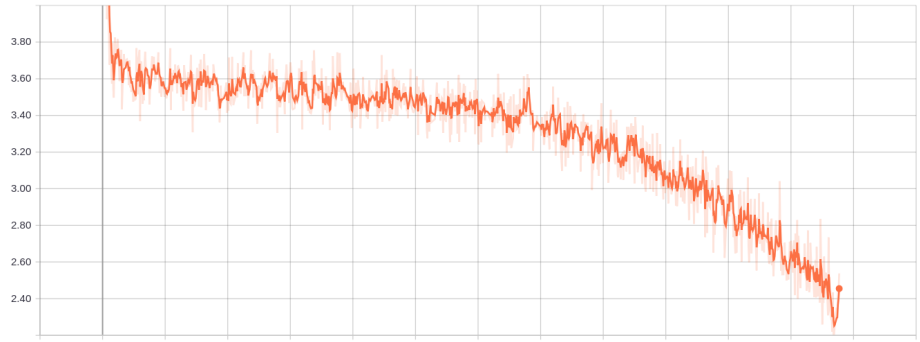


Figure 11.2: Loss over Epoch.

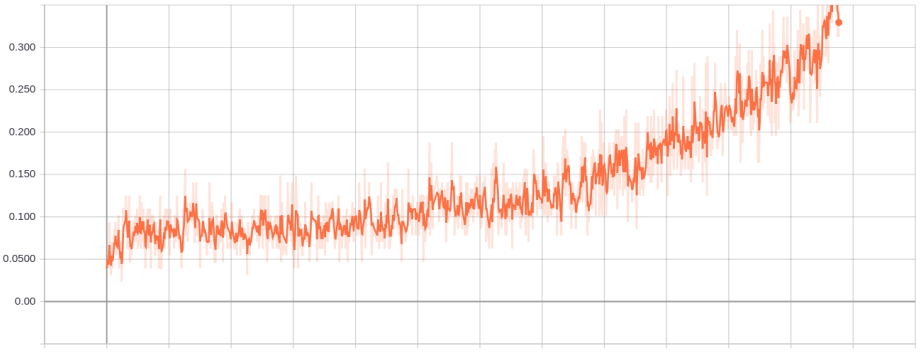


Figure 11.3: Accuracy over Epoch.

After 7000 iterations, the loss value was ***2.5*** and the accuracy value was ***0.3***. But from my perspective view, the loss and accuracy were both turbulent so lately I moved to another version with some hyper-parameter changed. After performing test on the ***train set***, I got the Spearman Rank Correlation of ***0.37***; on ***test set***, the Spearman Rank Correlation was just ***0.05***.

Version 1.1

In this revamp version, I trained a batch of size ***512*** at once, with ***1024*** LSTM units for each LSTM cell and ***7000*** iterations.

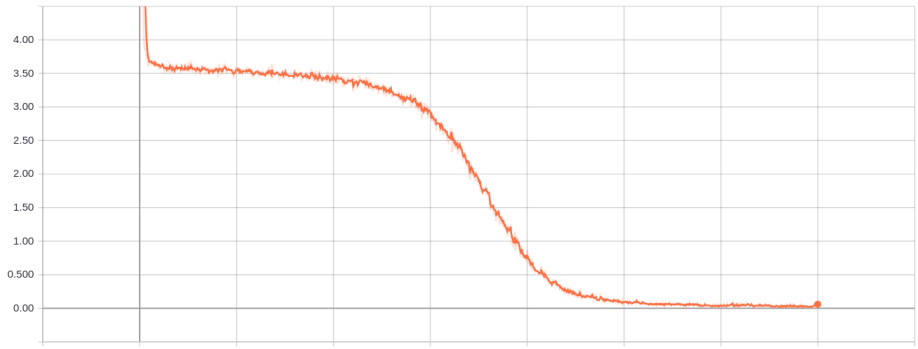


Figure 11.4: Loss over Epoch.

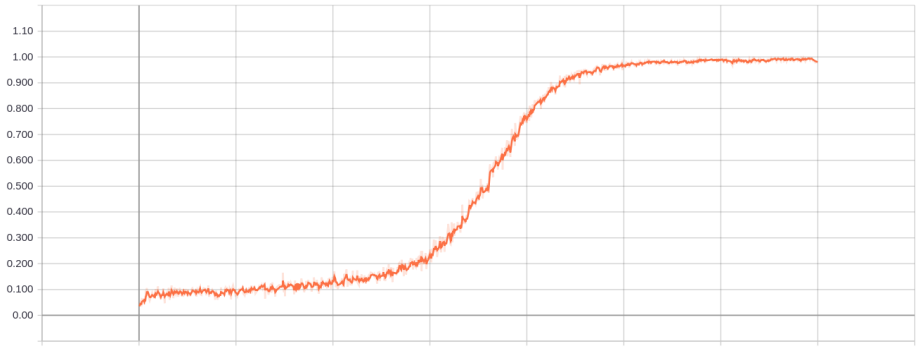


Figure 11.5: Accuracy over Epoch.

After 7000 iterations, the loss value was ***0.06*** and the accuracy value was ***0.98***. But from my perspective view, the loss and accuracy were both turbulent so lately I moved to another version with some hyper-parameter changed. After performing test on the ***train set***, I got the Spearman Rank Correlation of ***0.98***; on ***test set***, the Spearman Rank Correlation was just ***0.03***. This result seemed to be the consequence of overfitting. I thought 7000 iterations was not a good number.

11.3.3 Training on Dev-Set (extracted by Places365)

Strategy

I also broke the problem into the classification problem of 101 cate-gories. I first passed all of my input through Places365's pre-trained ResNet18 network and used those features as input of my LSTM network. The features extracted by ResNet18 had the dimension of ***512***. I aslo picked ***7000*** videos for training and ***1000*** videos for testing too.

Version 1

In this version, I trained a batch of size ***128*** at once, with ***256*** LSTM units for each LSTM cell and ***7000*** iterations.

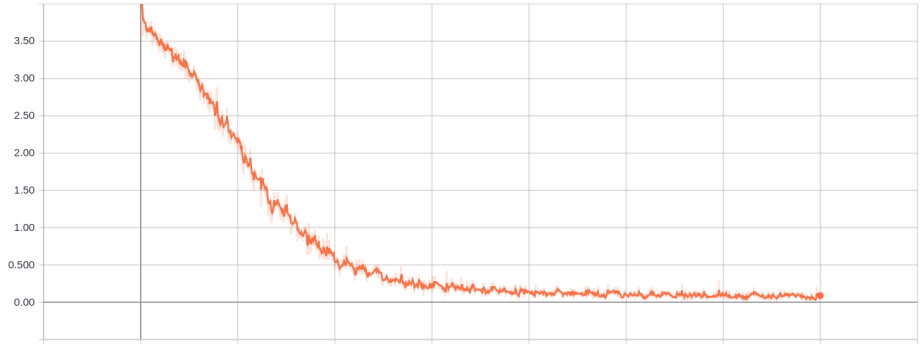


Figure 11.6: Loss over Epoch.

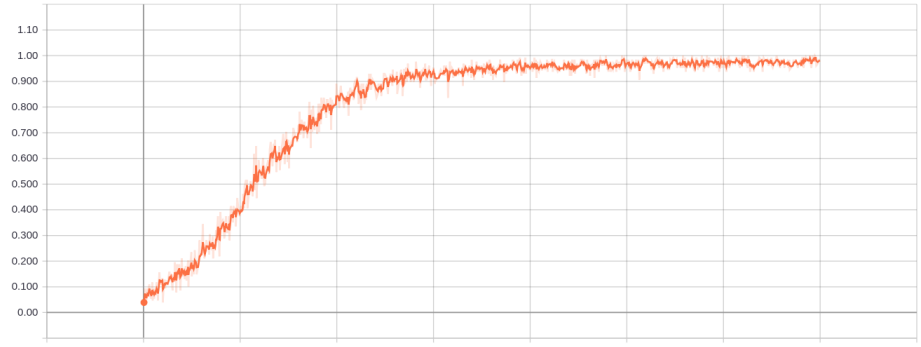


Figure 11.7: Accuracy over Epoch.

After 30000 iterations, the loss value was ***0.09*** and the accuracy value was ***0.98***. But from my perspective view, the loss and accuracy were both turbulent so lately I moved to another version with some hyper-parameter changed. After performing test on the ***train set***, I got the Spearman Rank Correlation of ***0.97***; on ***test set***, the Spearman Rank Correlation was just ***0.07***. This result also seemed to be the consequence of overfitting. I should decrease the number of iterations in the next version.

Chapter 12

Week 12 - August 23 2018

12.1 Introduction

I have spent the majority of my time this week testing the Spearman Rank correlation, Topological idea and grasping the idea of AMNet[66] by Jiri Fajtl.

12.2 Spearman Rank Correlation Experiments

This week I tested the Spearman Rank Correlation computation on groundtruth and some values inherited from the groundtruth itself.

Keep the Groundtruth by itself. I calculated the correlation between the groundtruth and itself to validate my reasoning that it would be 1. These were the results; ***Spearman's Correlation*** value of ***1.0***, ***Pearson's Correlation*** value of ***1.0***, ***Mean Squared Error*** value of ***0.0***.

Divide the Groundtruth by 2. I divided the groundtruth by 2 and calculated the correlation between them. These were the results; ***Spearman's Correlation*** value of ***1.0***, ***Pearson's Correlation*** value of ***1.0***, ***Mean Squared Error*** value of ***0.16***.

Ordinal Ranking. For example I had 6 elements ***[0.45, 0.25, 0.1, 0.2, 0.25, 0.15]***, the ordinal ranking of these 6 elements would be ***[6, 4, 1, 3, 5, 2]***. This ranking technique did not care about equal values, it just ranked the elements by their values and their index (for equal values). I calculated the correlation between the groundtruth and its ordinal ranking. These were the results; ***Spearman's Correlation*** value of ***0.999***, ***Pearson's Correlation*** value of ***0.966***, ***Mean Squared Error*** value of ***21319943.72***. Because the ranks were far from the groundtruth so it was reasonable for the mean squared error value to be a huge number.

Max Ranking. Given 6 elements as the example above, the max ranking of these 6 elements would be ***[6, 5, 1, 3, 5, 2]***. This ranking technique based on the ordinal ranking, but it treated equal values as the same ranks and their ranks would be the maximum one in term of their ordinal ranks. I calculated the correlation between the groundtruth and its max ranking. These were the results; ***Spearman's Correlation*** value of ***1.0***, ***Pearson's Correlation*** value of ***0.968***, ***Mean Squared Error*** value of ***22609763.39***. As this ranks also made the predict results completely differed from the groundtruth so the mean squared error was also a huge number.

Dense Ranking. Given 6 elements as the example above, the dense ranking of these 6 elements would be ***[5, 4, 1, 3, 4, 2]***. This ranking technique treated equal values as the same ranks and kept moving on without making any gaps in the ranking sequence. I calculated the correlation between the groundtruth and its dense ranking. These were the results; ***Spearman's Correlation*** value of ***1.0***, ***Pearson's Correlation*** value of ***0.992***, ***Mean Squared Error*** value of ***12969.23***. Because this ranking technique did not create any gaps so it was much more efficient (in term of memory) and the mean squared error was also smaller than the 2 ranking techniques' above.

After testing these ranking techniques I had a conclusion that the ***Dense Ranking*** would work best because its Pearson's Correlation value was higher and mean squared error value was much smaller than 2 other ranking techniques above. So it would be a reasonable choice for the Topological Sort idea which would be presented next.

12.3 Topological Sort Network

12.3.1 Architecture

This was actually my mentor's idea and I also confirmed its feasibility after experiments on the Spearman Rank Correlation computation above. My network took in 2 videos at once, concatenated them and pass through several Fully-connected layers to get determine which video was more memorabible than the other one. After comparing all two-video pairs I would use Topological Sort algorithm to retrieve the ranking sequence. Spearman Rank Correlation would be calculated on this ranking sequence and the groundtruth.

12.3.2 Training on Dev-Set (extracted by ResNet50)

Strategy

I splitted the provided Dev-Set for this challenge into three parts, since the Dev-Set had 8000 videos, I picked ***6000*** videos for training, ***1000*** videos for validating and the last ***1000*** videos for testing.

Version 1

In this version I used 2 hidden layers which size were ***2048*** and ***128***, I also added 3 dropout layers (probability of ***0.75*** each) between each Fully-connected layers. The figure below demonstrated my version 1 network.

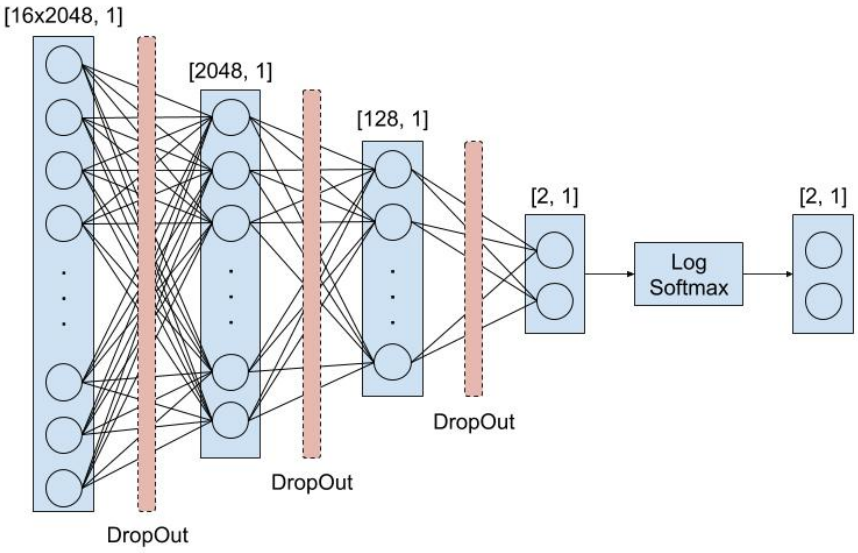


Figure 12.1: Topological Sort Network Network Architecture.

I trained this network with ***1000*** epochs. Each epoch I randomly picked a batch of ***128*** videos and compared each video with others (it meant I had average ***16384*** comparisons each epoch). The loss value after 1000 epochs was ***0.078*** and the highest validate accuracy was ***54.68%***.

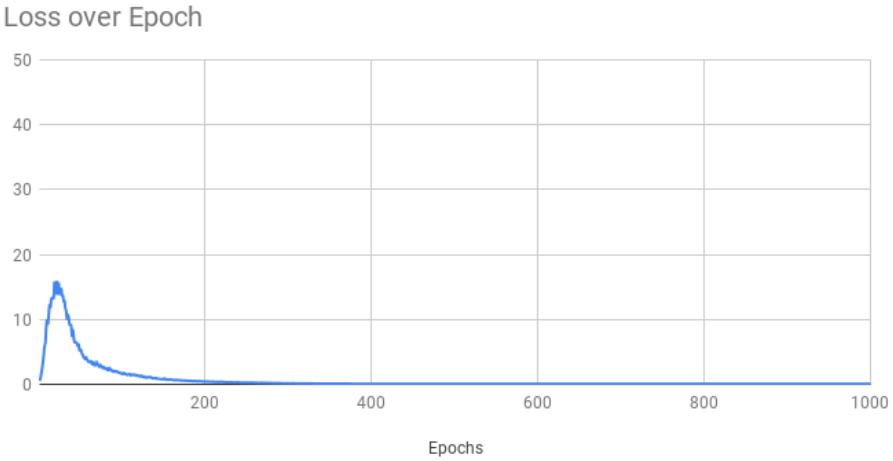


Figure 12.2: Loss over Epoch (on Train set).

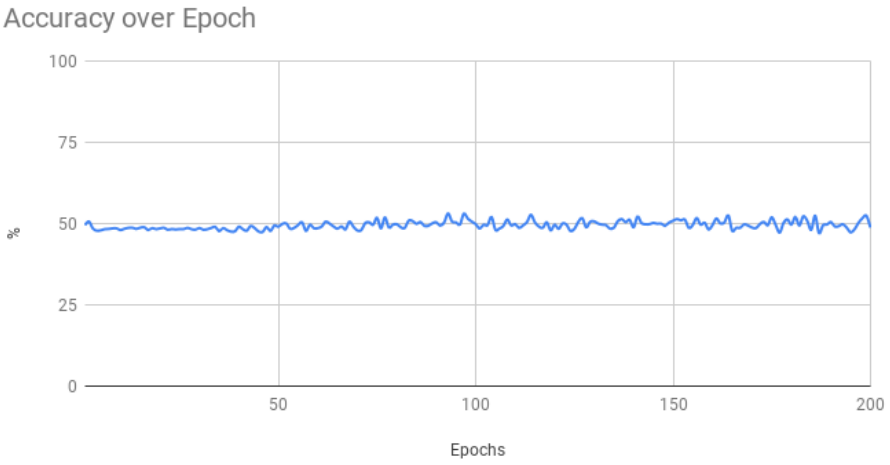


Figure 12.3: Accuracy over Epoch (on Validate set).

Then I used this trained model to calculate all comparisons between videos and used those values for Topological Sort algorithm to get the final ranking sequence. But the Spearman Rank correlation value was only ***0.01***. So I thought this idea was not a reasonable approach.

12.4 AMNet

12.4.1 Introduction

In this paper the authors presented the design and evaluation of an end-to-end trainable, deep neural network with a visual attention mechanism for memorability estimation in still images. The author analyzed the suitability of transfer learning of deep models from image classification to the memorability task. Further on the authors studied the impact of the attention mechanism on the memorability estimation and evaluate their network on the SUN Memorability and the LaMem datasets. Their network outperforms the existing state of the art models on both datasets in terms of the Spearman's rank correlation as well as the mean squared error, closely matching human consistency.

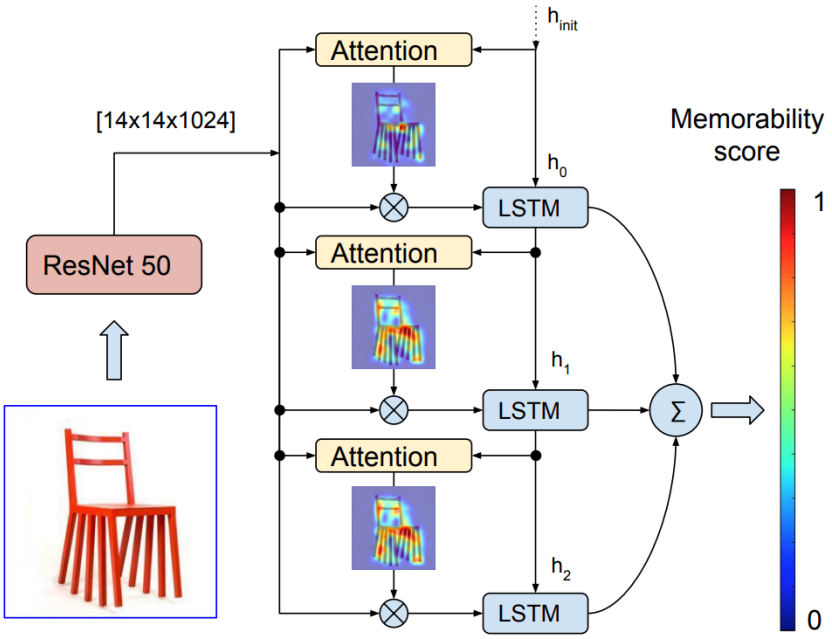


Figure 12.4: AMNet iteratively generates attention maps linked to the image regions correlated with the memorability. After three iterations the memorability scores were added and presented on the output.

The AMNet estimated the image memorability by taking a single image $\boldsymbol{X}$ and generating a memorability score $\boldsymbol{y}$. The process of memorability estimation was summarized in algorithm presented in the figure above. The authors proposed all required mathematical equations (Eq.3, Eq.4, Eq.6, ...) in their paper for implement purpose. The figure below demonstrated the detailed process of memorability estimation.

procedure MEMORABILITY(X) ▷ $y = f(\boldsymbol{X})$
  $\mathbf{x} = get\_cnn\_features(\boldsymbol{X})$ ▷ ResNet50 fwd pass
  $\mathbf{h}_0 = f_{init_c}(\mathbf{x})$ ▷ Eq. 12
  $\mathbf{c}_0 = f_{init_h}(\mathbf{x})$ ▷ Eq. 12
  $lstm\_init(\mathbf{h}_0, \mathbf{c}_0)$
  $y = 0$
  for $t = 0$ to T **do** ▷ at $t = 0 \rightarrow \mathbf{h}_t = \mathbf{h}_0$
    $\mathbf{e} = f_{att}(\mathbf{x}, \mathbf{h}_t)$ ▷ Eq. 8
    $\boldsymbol{\alpha} = softmax(\mathbf{e})$ ▷ Eq. 6
    $\mathbf{z} = []$
    for $i = 0$ to L **do** ▷ for all locations, Eq. 4
      $\mathbf{z} = \mathbf{z} + \alpha_i \mathbf{x}_i$ ▷ $\mathbf{z} \in \mathbb{R}^D$
    $\mathbf{h}_t, \mathbf{c}_t = lstm\_step(\mathbf{z}, \mathbf{h}_t, \mathbf{c}_t)$ ▷ Eq. 3
    $y = y + f_m(\mathbf{h}_t)$ ▷ Eq. 11
  return y ▷ Memorability score $[0, 1]$

Figure 12.5: Summarized AMNet Algorithm.

The authors also proposed their custom loss function in their paper. Their loss function composed of two terms. The first term represented the mean squared error between the groundtruth and predicted image memorability. In order to encourage the attention model to explore all image regions over all time steps, the authors added a second term which performed a joint penalty as a function of activations of all attention maps in the LSTM sequence, which was already introduced by Kelvin Xu[67].

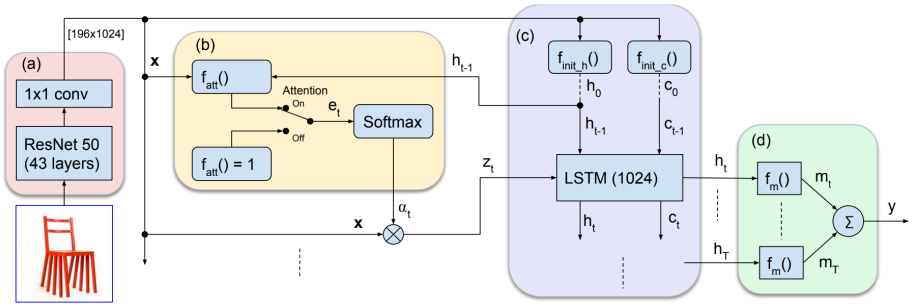


Figure 12.6: Detailed AMNet Algorithm.

The authors claimed that the entire model was fully differentiable and trained end-to-end with the ADAM[68] optimizer with a fixed learning rate 10^3. The input image feature vector was extracted from the 43^{rd} layer of the RestNet50 with dimensions $[14 \times 14 \times 1024]$. The ResNet50 was trained for image classification on the ImageNet dataset and its weights were not updated during the AMNet training.

Evaluation on the LaMem dataset was performed by training one model on each of the five random splits as suggested by the authors and then reporting the final memorability rank correlation and MSE , averaged over the results from five corresponding test datasets.

In Figure 1.7 below the author showed that the AMNet model with the active attention achieves $\rho = 0.677$, or a 5.8% improvement over the best known method MemNet[61]. Even without attention the AMNet outperforms prior

work by 3.6% which demonstrated that the pretrained, deep CNN with their recurrent and regression network layers still achieve high accuracy.

Method (LaMem dataset)	$\rho \uparrow$	MSE $\downarrow$
AMNet	**0.677**	**0.0082**
AMNet (no attention)	0.663	0.0085
MemNet [18]	0.64	NA
CNN-MTLES [14] (different train/test (50/50) split)	0.5025	NA

Figure 12.7: Average Spearman's rank correlation ρ and MSE over 5 test splits of the LaMem dataset.

12.4.2 Training on LaMem-Set (extracted by Inceptionv3)

Strategy

Instead of using the input image feature vector of dimensions [14 × 14 × 1024] and having the sequence length of 14 × 14 as AMNet, I used the input image feature vector of dimensions [1 × 2048] and my sequence length was 1. I did not want to complicate my loss function, I just used the mean squared error only. I splitted the LaMem-Set for into two parts, the total number of images was ***55000***, I picked ***45000*** images for training, and ***10000*** images testing.

Version 1

In this version I had my input size of ***2048***, hidden size of ***1024***, ***3*** stacking LSTM layer and as mentioned above, my sequence length of ***1***. My ADAM optimizer's learning rate was ***1e-4***. I trained this version with ***200*** epochs and saved the snapshot which had the highest correlation value on test set. The loss and the correlation value both behaved stably.

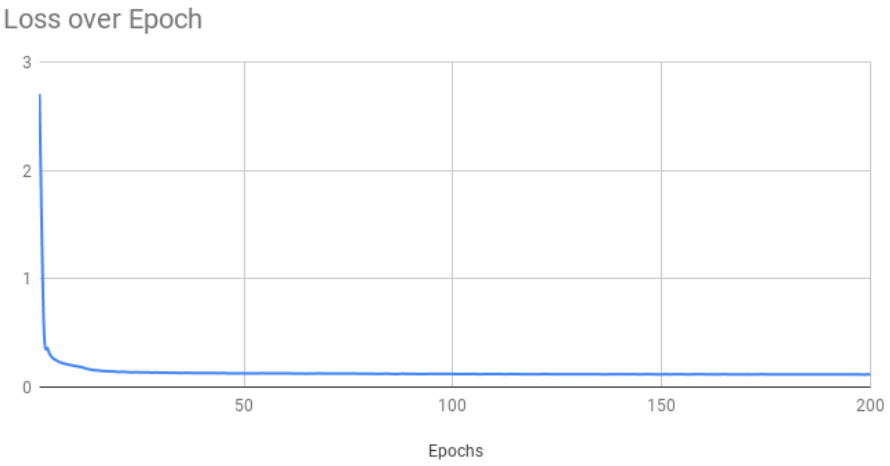


Figure 12.8: Loss over Epoch (on Train set).

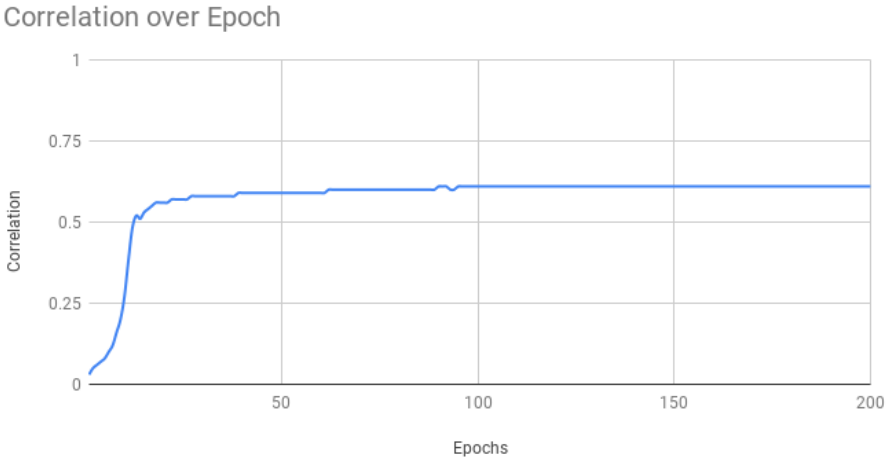


Figure 12.9: Correlation over Epoch (on Test set).

The loss and correlation value stopped significantly changing from the epoch 100^{th} so I thought it only needed 100 epochs to converge. After 200 epochs, my loss value was around ***0.117*** and my final correlation value on test set was ***0.61***.

Lately, I used this pre-trained model to calculate the correlation value on the Dev-Set but predictably, the result was absolutely terrible, it was not even a positive number.

12.4.3 Training on Dev-Set (extracted by ResNet50)

Strategy

I used the same input strategy as the previous version that I trained on LaMem-Set. I splitted the provided Dev-Set for this challenge into three parts, since the Dev-Set had 8000 videos, I picked ***6000*** videos for training, ***1000*** videos for validating and the last ***1000*** videos for testing.

Version 1

I kept the input size and hidden size as the same as the previous version that I trained on LaMem-Set. This times I changed the number of stacking LSTM layer to ***1*** and the sequence length to ***8***. Finally was the ADAM optimizer with learning rate of ***1e-4***.

After ***100*** epochs I luckily got the loss decreasing but the correlation was not stable. My loss value was around ***0.42*** and the best correlation value on validate set I got was only ***0.15***.

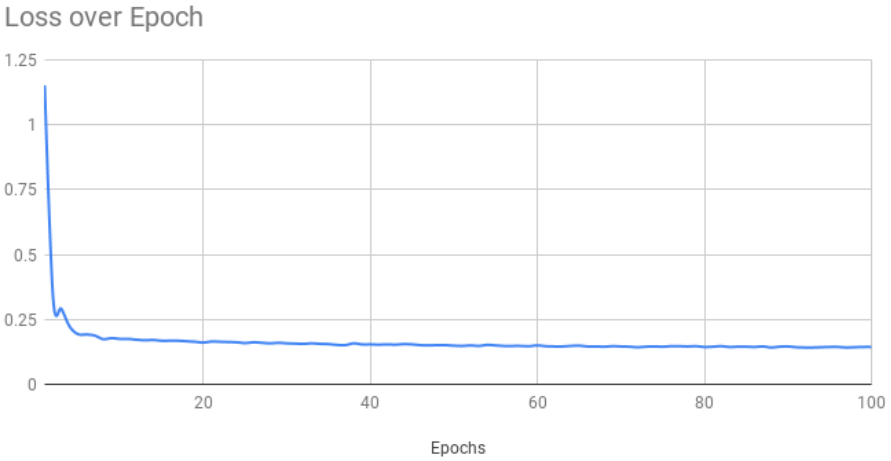


Figure 12.10: Loss over Epoch (on Train set).

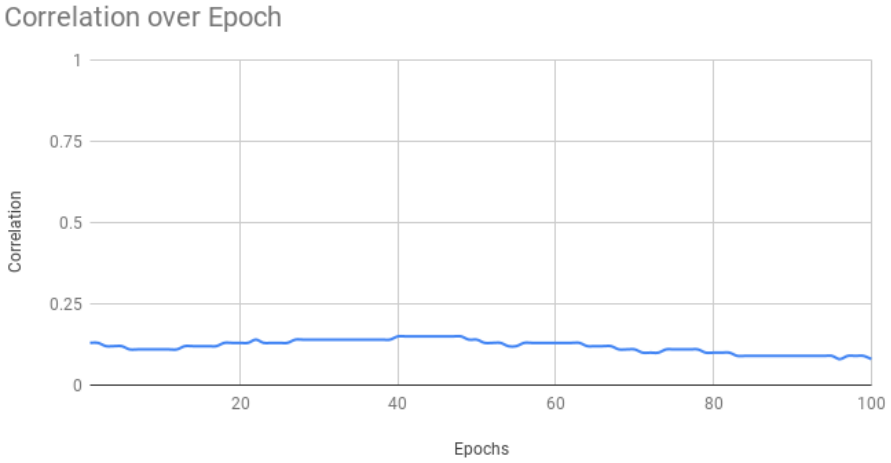


Figure 12.11: Correlation over Epoch (on Validate set).

Lately, I used this pre-trained model to calculate the correlation value on the test set but unfortunately, the result was just only ***0.001***.

12.5 Conclusion on Long-Term Memorability Set

From the beginning till this time, I only focused on the long-term memorability set and I still could not make anything worked. At this state, I thought I finally figured out why all of my effort was just wasting time, because the long-term memorability dataset was not reliable at all.

Chapter 13

Week 13 - August 30 2018

13.1 Introduction

I have spent the majority of my time this week adding attention loss to my AMNet and testing it with features extracted by ResNet50.

So far I had just only worked with the Long-term subtask and I figured out that the Long-term Memory dataset was nothing but useless garbage. So till that moment I would start working with the Short-term subtask.

13.2 AMNet

13.2.1 Attention Loss

The model which I had used last week even though deployed the attention concept but I did not used the attention loss beside the normal Mean-squared error. So this week I tried to implement the attention loss in my network and hope it would perform better.

13.2.2 Training on Dev-Set (extracted by ResNet50)

Strategy

I used the same input strategy as the last week versions that I had trained. I splitted the provided Dev-Set for this challenge into three parts, since the Dev-Set had 8000 videos, I picked ***6000*** videos for training, ***1000*** videos for validating and the last ***1000*** videos for testing.

Version 2

I kept the input size and hidden size as the same as the previous version that I had trained on Dev-Set last week. This times I changed the number of stacking LSTM layer to ***1*** and the sequence length to ***8***. As mentioned above, I tried to conduct the same loss function as the one proposed by the authors which was a combination of mean squared error and attention penalty. There was one new hyper-parameter gamma which specified the impact of the attention penalty. I used its value of ***1e-5***. Finally was the ADAM optimizer with learning rate of ***1e-4***.

After ***1000*** epochs I luckily got the loss decreasing but the correlation was not stable. My loss value was around ***0.156*** and the best correlation value on validate set I got was only ***0.23***.

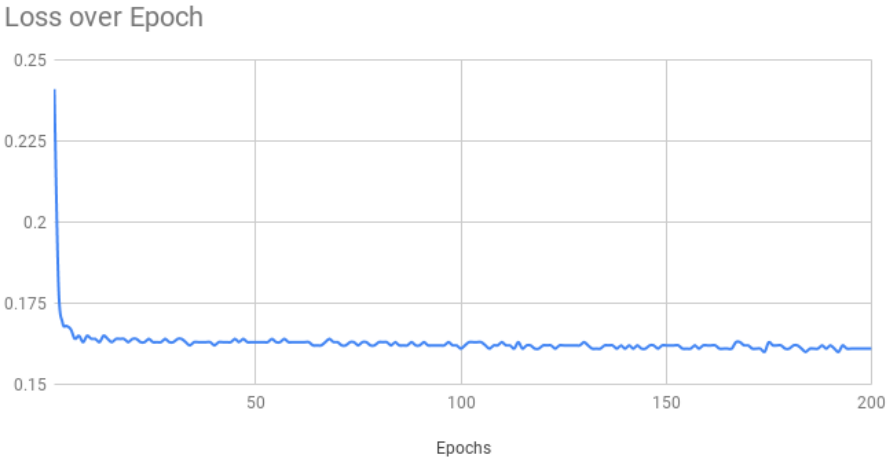


Figure 13.1: Loss over Epoch (on Train set).

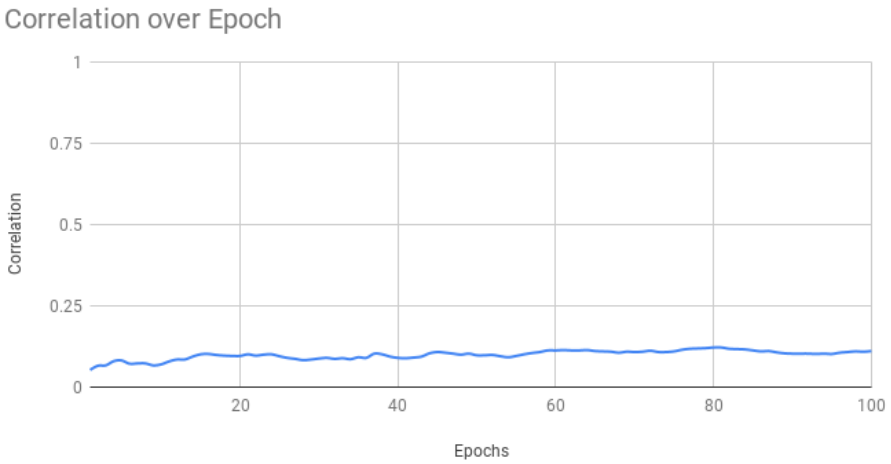


Figure 13.2: Correlation over Epoch (on Validate set).

Lately, I used this pre-trained model to calculate the correlation value on the test, the correlation was a little bit lower, which was ***0.18***.

13.3 Conclusion

At this moment I started thinking that my extracted features were the problem. After testing my extracted features with my teammate Long-Short term memory network, my final conclusion was my ResNet50 extracted features were the problem causing my correlation horrible.

Chapter 14

Week 14 - September 6 2018

14.1 Introduction

I have spent the majority of my time this week improving my Long - Short Term Memory network (PyTorch version), trying to extract videos' features by using TensorHub Module[69] and TensorFlow for Poets[70]. As mentioned before, till this week I would only work with the Short-term subtask so all my experiments from this moment were performed on the Short-term memorability scores.

14.2 Feature Extracting

14.2.1 Using TensorHub

In week 9, I had already try to extract images' features but at that moment I misunderstood the concept of how TensorFlow actually worked. After learning more about how to use TensorFlow this week I finally figured out how to use TensorHub to extract features correctly.

I used TensorHub to extract images' features with pretrained Inception V3 network. As mentioned by TensorHub, the checkpoint exported into the module was inception-v3-2016-08-28/inception-v3.ckpt downloaded from TensorFlow-Slim's pre-trained models. Its weights were originally obtained by training on the ILSVRC-2012-CLS dataset for image classification (Imagenet).

All my code could be found on my Colab notebook. I also normalized the input image by subtracting it by ***128*** then dividing it by ***128*** (IMPORTANT step). This module took me more than one hour to extract 8000 videos (8 frames for each video), which were around 64000 frames total.

14.2.2 Using TensorFlow for Poets

TensorFlow provided a tutorial called TensorFlow for Poets teaching how use transfer learning, which meant starting with a model that had been already trained on another problem, then retrain it on a similar problem.

At the 4th step Retraining the network, they described correctly how their retrain script worked. I changed the architecture argument to Inception V3 to use it as pretrained model to extract bottleneck values.

One of the very first phase of their retrain script was to analyzed all the images and calculated the bottleneck values for each of them. The term bottleneck here was actually indicated the term feature. After this phase I got a folder containing all images' bottleneck values, the next job I had to do was concatenating them all to work later.

14.3 PyTorch LSTM Classification Network

14.3.1 Architecture

Beside using the same architecture as I had used in week 9, I added 2 Fully-Connected Layers separately to initialize the hidden state ***h0*** and cell state ***c0*** of the LSTM cells based on the input. The motivation behind this change was

when I thought using a randomly initialized hidden state and cell state was nothing but a terrible idea because it made no sense at all.

These 2 Fully-Connected Layers I mentioned above turned the input size of [batch-size, sequence-length, input-size] (e.g. [1000, 8, 2048]) to hidden state and cell state sizes of [num-layers, batch-size, hidden-size] (e.g. [3, 1000, 1024]).

14.3.2 Training on Dev-Set (extracted by Inception V3 with TensorFlow for Poets)

Strategy

Till this moment I would use this strategy for every new networks. I splitted the provided Dev-Set for this challenge into three parts, since the Dev-Set had 8000 videos, I picked ***6000*** videos for training, ***1000*** videos for validating and the last ***1000*** videos for testing.

Version 2.1

As described above in the architecture section, I used my version 2 architecture powered by some Fully-Connected Layers so I called this 2.1. I trained this architecture with only ***100*** epochs and at the learning rate of ***1e-4***. My network converged with only around 50 epochs and the loss value seemed to decreasing stably.

The best correlation value on Validate set was ***0.4757*** and the minimum loss value was ***0.011***. As the loss would continue decreasing, the correlation value also decreased, so I thought it was over-fitting.

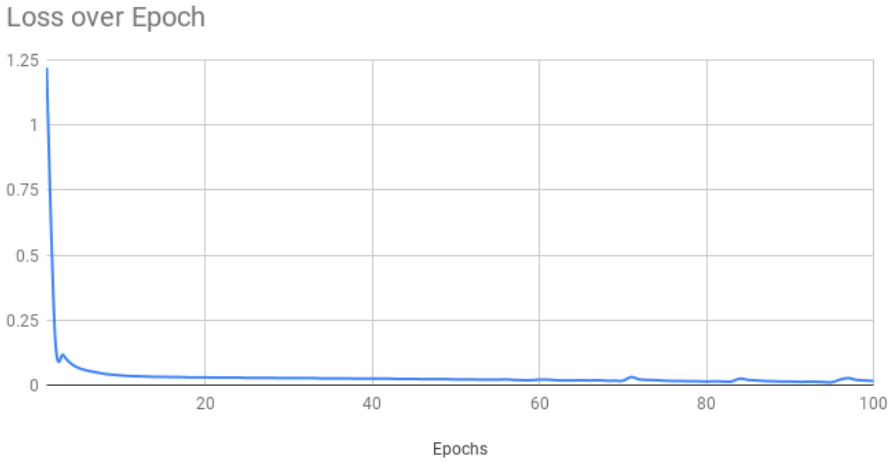


Figure 14.1: Loss over Epoch (on Train set).

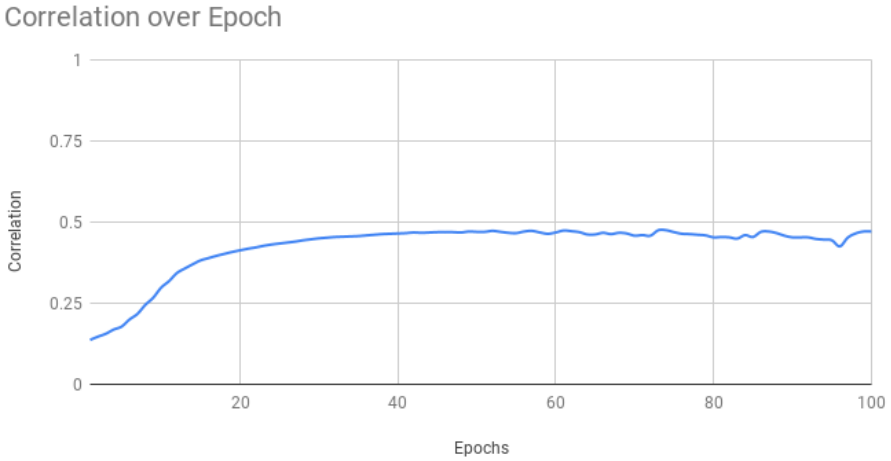


Figure 14.2: Correlation over Epoch (on Validate set).

Lately, I used this pre-trained model to calculate the correlation value on the test, the correlation was a little bit lower but acceptable, which was ***0.4591***.

14.3.3 Training on Dev-Set (extracted by Inception V3 with TensorHub)

Version 2.1

I trained this architecture with only ***100*** epochs and at the learning rate of ***1e-4***.

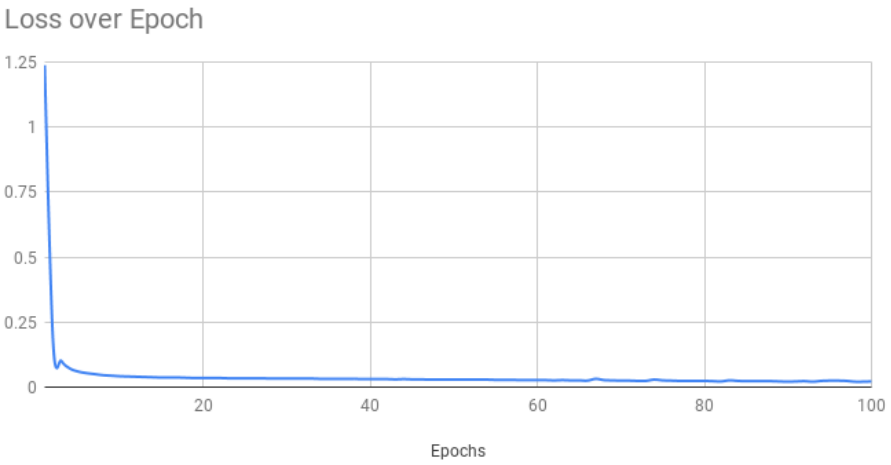


Figure 14.3: Loss over Epoch (on Train set).

The best correlation value on Validate set was ***0.3585*** and the minimum loss value was ***0.021***. As the loss would continue decreasing, the correlation

value also decreased, so I thought it was over-fitting.

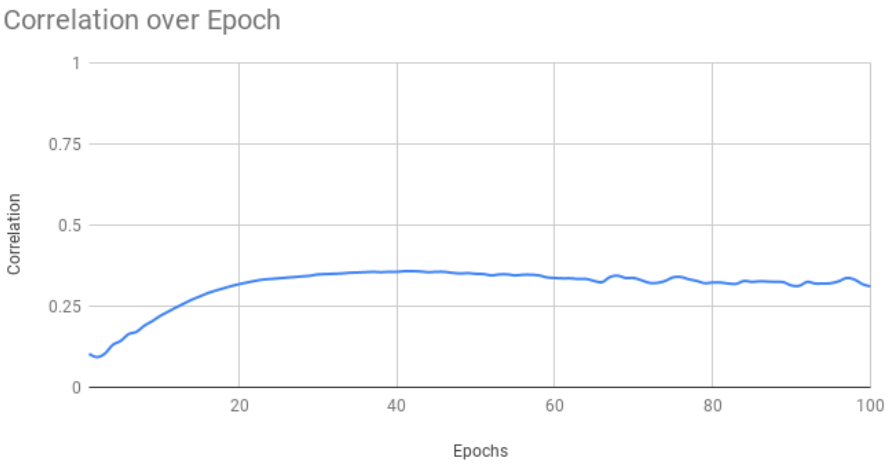


Figure 14.4: Correlation over Epoch (on Validate set).

Then I used this pre-trained model to calculate the correlation value on the test, the correlation was a little bit lower but acceptable, which was ***0.2989***.

Comment

I did not normalized the images before extracting their feautures in the previous week, and the value of correlation on Validate set was only 0.2x; the value of correlation on Test set was even worse (0.01x).

In this week I tried normalizing the images and my features performed much more better. The results was described above in the sub section above.

Chapter 15

Week 15 - September 13 2018

15.1 Introduction

I have spent the majority of my time this week trying my LSTM with others features extract with multiple methods and re-implementing the extracting features part of TensorFlow for Poets to build up the full network version (Extracting Feature and LSTM in a single network)

15.2 PyTorch LSTM Classification Network

So far I had tried many different methods to extract features and used them as input for my LSTM network, but I found that the method that providing the best features for my LSTM network was the pretrained Inception V3 model from 2015 by Google TensorFlow for Poets.

This week I tried normalizing the images before extracting their features with PyTorch's pretrained models. But the results were just the same as TensorHub, they all could not be better than the TensorFlow for Poets' features.

15.3 Full Network

15.3.1 Feature Extracting Layer

Based on the example code by TensorFlow for Poets, I implemented my own version.

Step 1. Retrieving Inception V3 model detail information such as where to download it, bottleneck layer's name, size of extracting features, input images' size required by model, mean and standard deviation for normalizing input images.

Step 2. Using the model link provide at step 1 to download and extract the model on disk.

Step 3. Generating default TensorFlow Graph[71], grasping the tensors of resizing input images, computing bottleneck values from the model downloaded at step 2. Then I had to add both tensors above to the generated Graph for further usages.

Step 4. Creating tensor to decode input images from string type to float types and another tensor to reshape and normalize decoded images from the size information got from step 1. Then I had to add both tensors above to the generated Graph for further usages.

Step 5. Examining all input videos, caching their bottleneck values on disk and loading them into training dataset at the same time.

15.3.2 Long-Short Term Memory Layer

This Long-Short Term Memory layer was backed by Keras, took in the feature vectors from first part above and outputted a sequence of 8 vectors (for each video).

The input size of this layer was equivalent to the size of the feature vectors (of length 2048) for each video which was totally [8, 2048], the drop out indicating fraction of the units to drop for the linear transformation of the inputs was 0.5, the hidden size indicating dimensionality of the output space was set to be 1024

or 2048 based on preference. The output size of this layer would be [8, 1024] or [8, 2048] based on how I configed the hidden size. One more notable thing was that I only used one LSTM layer which meant there was no stacking here.

15.3.3 Time-Distributed Layer

This Time-Distributed layer was also backed by Keras, took in the output sequence of vectors from LSTM layer and outputted the memorability score of a video.

The input size of this layer was [8, 1024] or [8, 2048] based on the previous layer. This layer applied the Dense layer to each timestep independently, outputted a vector of size [8, 1] for each input indicating the memorability score through out every timestep (every frame).

The last layer attached to this Time-Distributed layer was Lambda layer. Its purpose was to extract only the last memorability score (from [8, 1] to [1, 1]) of the Time-Distributed's output.

15.3.4 Cost Function and Optimizer

The official cost function of this model was Mean-Squared Error, and the offical optimizer was Adam optimizer with default learning-rate of 1e-3.

Bibliography

[1] École Polytechnique Fédérale de Lausanne, *EE-559 - Deep Learning.*

[2] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, A. Torralba, *Places: A 10 million Image Database for Scene Recognition*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2017.

[3] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor, *Caffe: Convolutional Architecture for Fast Feature Embedding*, arXiv:1408.5093, 2014.

[4] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, *PyTorch.*

[5] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks.*

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*, arXiv:1512.03385, 2015.

[7] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger, *Densely Connected Convolutional Networks*, arXiv:1608.06993, 2016.

[8] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba, *Learning Deep Features for Discriminative Localization*, arXiv:1512.04150, 2015.

[9] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Scott Reed, *Going Deeper with Convolutions*, arXiv:1409.4842, 2014.

[10] École Polytechnique Fédérale de Lausanne *EE-559 - Deep Learning*, *Mini-project 1: Prediction of finger movements from EEG recordings.*

[11] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, arXiv:1409.1556, 2014.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, *You Only Look Once: Unified, Real-Time Object Detection.*

[13] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, arXiv:1311.2524, 2013.

[14] G. C. Benjamin Blankertz and K.-R. R. Muller, *Towards brain computer interfacing*, *Neural Information Processing Systems (NIPS)*, 2002.

[15] Davis E. King, *DLib*.

[16] Stanford University, CS231n - Convolutional Neural Networks for Visual Recognition.

[17] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *CIFAR-10 dataset*.

[18] Sergey Ioffe, Christian Szegedy *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*.

[19] Conference on Computer Vision and Pattern Recognition 2017.

[20] International Conference on Computer Vision 2017, *2017.thecvf.com*.

[21] Conference on Computer Vision and Pattern Recognition 2018.

[22] Eric Tzeng, Judy Hoffman, Kate Saenko, Trevor Darrell, *Adversarial Discriminative Domain Adaptation*, arXiv:1702.05464, 2017.

[23] Artem Rozantsev, Mathieu Salzmann, Pascal Fua, *Beyond Sharing Weights for Deep Domain Adaptation*, arXiv:1603.06432, 2016.

[24] Mehdi Mirza, Simon Osindero, *Conditional Generative Adversarial Nets*, arXiv:1411.1784 2014.

[25] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas, *Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks*, arXiv:1612.03242, 2016.

[26] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas, *StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks*, arXiv:1710.10916, 2017.

[27] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, *Generative Adversarial Networks* arXiv:1406.2661, 2014.

[28] Han Zhang, Ian Goodfellow, Dimitris Metaxas and Augustus Odena, *Self-Attention Generative Adversarial Networks*, arXiv:1805.08318, 2018.

[29] Takeru Miyato, Toshiki Kataoka, Masanori Koyama and Yuichi Yoshida, *Spectral Normalization for Generative Adversarial Networks*, arXiv:1802.05957, 2018.

[30] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros, *Paired Image-to-image translation using Conditional GAN*, arXiv:1611.07004, 2016.

[31] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro, *High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs*, CVPR 2018.

[32] Jun-Yan Zhu, Taesung Park, Phillip Isola, Alexei A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, ICCV 2017.

[33] Zhu, Jun-Yan and Zhang, Richard and Pathak, Deepak and Darrell, Trevor and Efros, Alexei A and Wang, Oliver and Shechtman, Eli, *Toward Multimodal Image-to-Image Translation*, arXiv:1711.11586, Nov 2017.

[34] Xiaolong Wang, Abhinav Shrivastava, Abhinav Gupta, *A-Fast-RCNN: Hard Positive Generation via Adversary for Object Detection*, CVPR, 2017.

[35] Ross Girshick, *Fast R-CNN*, arXiv:1504.08083, 2015.

[36] Jun-Yan Zhu, *pytorch-CycleGAN-and-pix2pix*.

[37] Cameron Smith, *neural-style-tf*.

[38] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, *Image Style Transfer Using Convolutional Neural Networks*.

[39] Manuel Ruder, Alexey Dosovitskiy, Thomas Broxn, *Artistic style transfer for videos*, arXiv:1604.08610, 2016.

[40] Leon A. Gatys, Matthias Bethge, Aaron Hertzmann, Eli Shechtman, *Preserving Color in Neural Artistic Style Transfer*, arXiv:1606.05897, 2016.

[41] The Starry Night by Vincent van Gogh, 1889.

[42] Multimedia Evaluation 2018, *mediaeval2018*.

[43] Medico: The 2018 Multimedia for Medicine Task.

[44] Joint Photographic Experts Group.

[45] The 2018 Emotional Impact of Movies Task, *emotionalimpact*.

[46] The 2018 Predicting Media Memorability Task, *memorability*.

[47] James L McGaugh, *Memory-a Century of Consolidation*, Science 287, 5451 (2000), 248-251.

[48] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, *Learning Spatiotemporal Features with 3D Convolutional Networks*, ICCV 2015.

[49] Khurram Soomro, Amir Roshan Zamir and Mubarak Shah, *UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild*, CRCV-TR-12-01, November 2012.

[50] ALMEIDA, Jurandy; LEITE, Neucimar J.; TORRES, Ricardo da S, *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011. p. 3673-3676.

[51] Dalal, N. and B. Triggs, *Histograms of Oriented Gradients for Human Detection*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1 (June 2005), pp. 886–893.

[52] DC. He and L. Wang (1990), *Texture Unit, Texture Spectrum, And Texture Analysis*, Geoscience and Remote Sensing, IEEE Transactions on, vol. 28, pp. 509 - 512.

[53] SZEGEDY, Christian et al, *Rethinking the inception architecture for computer vision*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016. p. 2818-2826.

[54] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski, *Orb: an efficient alternative to sift or surf*, In Computer Vision (ICCV), 2011, IEEE International Conference on, 2011.

[55] OpenCV *Keypoints and Descriptors tutorial*.

[56] Eugenio Culurciello, TowardsDataScience, *Neural Network Architectures*.

[57] Sepp Hochreiter; Jürgen Schmidhuber, *Long Short-Term Memory*, Neural Computation.

[58] Google's AI organization, *TensorFlow*.

[59] Ronan Collobert, Koray Kavukcuoglu, Clement Farabet, *Torch*.

[60] Skymind, *From word to embeddings*.

[61] Aditya Khosla, Akhil S. Raju, Antonio Torralba and Aude Oliva, *Understanding and Predicting Image Memorability at a Large Scale*, International Conference on Computer Vision, 2015.

[62] Matt Harvey, Coastline Automation, *Five video classification methods implemented in Keras and TensorFlow*.

[63] Phillip Isola, Jianxiong Xiao, Antonio Torralba and Aude Oliva, *What makes an image memorable?*, IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[64] T. Konkle, T. F. Brady, G. A. Alvarez, and A. Oliva, *Conceptual distinctiveness supports detailed visual long-term memory for realworld objects*, JEP:G, 2010.

[65] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman, *LabelMe: a database and web-based tool for image annotation*, IJCV, 2008.

[66] Jiri Fajtl, Vasileios Argyriou, Dorothy Monekosso, Paolo Remagnino, *AMNet: Memorability Estimation with Attention*, arXiv:1804.03115, 2018.

[67] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio, *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*, arXiv:1502.03044, 2015.

[68] Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980, 2014.

[69] Google, *TensorFlow Hub Module*.

[70] Google, CodeLabs, *TensorFlow for Poets*.

[71] Google, TensorFlow, *Dataflow Graph*.