Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

MASTER THESIS

TITLE: Shot Boundary Detection based on Deep Convolutional Neural Network

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Chenjia Zhao

ADVISOR: Francesc Tarrés, Arnau Raventos Mayoral

DATE: August, 31st 2018

Abstract

Shot boundary detection (SBD) is the process of automatically detecting the boundaries between shots in the videos, which is an important pre-processing step for video analysis, such as indexing, browsing, summarization and other content-based operations. Nowadays with the continuously growing video data, traditional technologies based on low-level features of the frames (such as color histogram) no longer fits the requirements, not only from the speed point of view, but also the accuracy.

Knowing that convolutional neural networks (CNN) become unprecedented popular in image processing these years, due to it’s powerfulness in feature analysis and classification, we implement a fully convolutional neural network based on the paper created by Michael Gygli, which is a 3-dimensional neural network. Our purpose is to detect the middle shot boundary in every 10 frames, that is to say, to estimate if there is shot boundary between the 5th and the 6th frame out of 10. While implementing the neural network architecture several modifications have been tested and proposed.

Thus, we created a proprietary dataset with thousands of frames and generated different transitions such as cuts and gradual transitions, to put them into our 3-dimensional network for training. The advantage of fully convolutional networks is that allows to use a large temporal context without the need of repeatedly processing frames.

By testing with our evaluation dataset, in the end of the project we got a satisfactory result with the accuracy approximately 95%. And we found the weakness of the network through analysis of each kind of shot transition. We hope that our efforts for the project will contribute to the investigation of shot detection strategies based on convolutional neural network.

ACKNOWLEDGMENT

I'd like to take this opportunity to express my gratitude to my advisor Francesc and Arnau, they guided me to improve little by little in this project and helped me solved some key problems, such as the initialization of weights and the construction of cloud servers. Without their help, I barely could not finish my project favourably.

I am also grateful to all the members of the faculty and staff in UPC, Castelldefels, who have provided me a lot of help and encouragement during my one year's study.

Last but not least, I would give my thanks to my parents and friends for their support, which helped me to finish my study successfully.

CONTENTS

INTRODUCTION

The main purpose of the project is to construct a neural network in order to detect if there is a shot boundary in the middle of the input 10 frames. Our work is mainly based on the paper [1], to realise a similar function.

We extracted our training and evaluate dataset from more than 70 American series manually and from which we made an augmented dataset for our network training and testing.

This document first explains what is shot boundary detection and current algorithms for its automatic detection. Later it introduces neural network as a tool to solve this problem. Our project used a 3D fully convolutional neural network to acquire change of information due to motion. The second chapter is the implementation detail of our neural network, including its structure, feature map, etc, and the next chapter explains in detail how we made the dataset. In the end, the forth chapter is some testing results of our network, in which we analysed the result with the strong point and weakness of our network. In the conclusion we summarized our work on this project, what we've contributed our effort and what we still need to improve, together with some considerations of sustainability and ethical considerations.

1. Introductions of the Basic Concepts

1.1. Introduction to Shot Boundary Detection

Video is gradually becoming the main stream in propagating information thanks to the development of Internet, as a result, an increasingly amount of video contents are generated and transmitted world-widely every day, even each hour. To combat the information explosion, it is essential to analyse and understand these videos for various purposes such as: research, recommendation and ranking, the later ones have a tightly relationship with Big-Data analysis, which is also an important tool of data analysis used in many research fields, and of course, has tremendous economic value.

Videos have been studied for so many years by different communities in computer vision field, some tasks like: action recognition, motion detection and video retrieval are closely-connected with shot boundary detection. A shot in a video is a series of interrelated consecutive pictures taken contiguously by a single camera and representing a continuous action in time and space [6], and shot boundary indicates the boundary of the frames where the interrelated contents start and end. Shot boundary detection (named SBD afterwards) is an indispensable process step of video manipulation.

A shot is a sequence of frames shot uninterruptedly by one camera. There are several video transitions usually used in film editing to juxtapose adjacent shots [7]. In the context of shot transition detection they are usually grouped into two types: Sharp transition and Gradual Transition (see Fig. 1.1 [2]). Sharp transition is a sudden transition from one shot to another, which are also known as hard cuts or simply cuts. We can see from Fig. 1.1 that the first two frames belong to one shot and the third frame belongs to another shot. Gradual transitions are also often known as soft transitions and can be of various types: Wipes, Fades, Dissolves, Semi-transparent, etc. In this kind of transitions the two shots are combined using chromatic, spatial or spatial-chromatic effects which gradually replace one shot by another [7], thus the transit process may contain more than one frames.

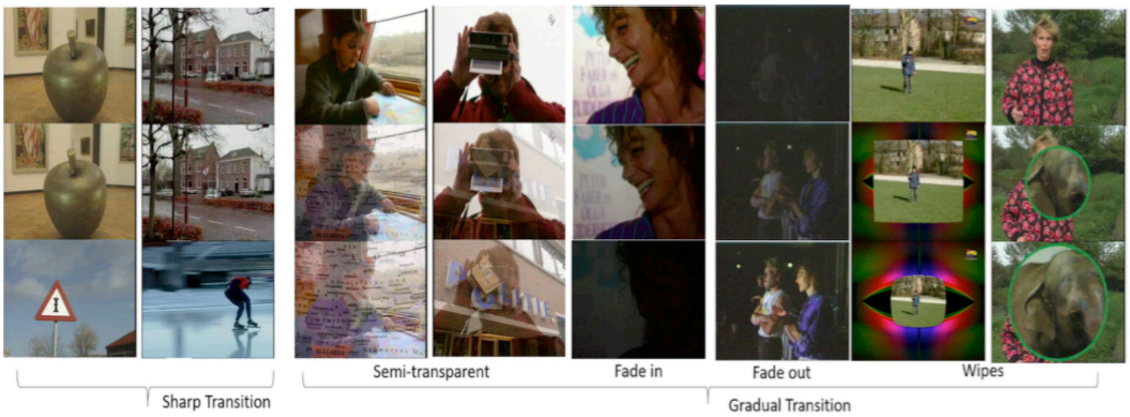


Fig. 1.1 Shot transitions are classified into two main categories: sharp and gradual. Gradual transitions are further classified into soft and wipes. Soft include semi-transparent, fade in and fade out. Wipes are the most ill-defined form of transitions.

For SBD, it is required to deal with different types of transitions not only the hard cuts but those gradual transitions such as fades and dissolves. Although shot detection appears to be a simple task for a human being, for the fact that human sense of vision is unbelievably advanced. However, it is a non-trivial task for computers. Shot detection would be a trivial problem if each frame of a video was enriched with additional information about when and by which camera it was taken. Possibly no algorithm for cut detection will ever be able to detect all cuts with certainty, unless it is provided with powerful artificial intelligence [8].

While most algorithms achieve good results with hard cuts, many fail with recognizing soft cuts. Hard cuts usually go together with sudden and extensive changes in the visual content while soft cuts feature slow and gradual changes. A human being can compensate this lack of visual diversity with understanding the meaning of a scene. While a computer assumes a black line wiping a shot away to be 'just another regular object moving slowly through the on-going scene', a person understands that the scene ends and is replaced by a black screen [8].

Also, the ‘flashes’ as well as some light changes always happen in the videos, which also bring challenges to SBD.

Traditional SBD methods depend on a set of low level features, such as colour or edge histogram, in conjunction with simple models like SVMs (Support Vector Machines), whose results normally are hard to be satisfying due to several reasons. For example, they don’t perform well when dealing with gradual transitions, because the gradual transitions don’t show apparent colour change between frames. And also, the video editors always try to make shot cut more inconspicuous in order that the contents look more consistent, which also brings big challenges to SBD algorithms. Due to the fact that videos show strong variation in content and motion speed, the blur caused by fast motions is always falsely considered as shot change.

To improve SBD methods and encouraging researches, the TRECVid initially launched benchmarking activities of SBD challenges for several years, and supported evaluation of the task where a large variety of SBD techniques from different research groups worldwide were benchmarked each year on the same video using the same scoring mechanisms and with the same manually created ground-truth[3], and these actually obtained good results. Nevertheless, problems haven't been solved yet, at the meantime, new technologies about images are developing continuously, which push us to move on to chase for better solutions.

1.2. Introduction to convolutional neural network

Convolutional neural network is a kind of network that is most successful in practical applications. It is specially used to process lattice structure data. For example, image data can be seen as two-dimensional lattice data composed of pixels. Unlike mathematics, in machine learning, convolution is the local feature multiplied by the corresponding weight, and then to be accumulated.

In 1958, two neurobiologists, Hubel and Wiesel, conducted an early study of the visual cortex and eventually discovered the secret of the Primary Visual Cortex (V1) of the mammalian visual system.

This visual cortex has three important properties:

- The V1 layer is arranged in the air like a net. When light passes through only the lower part of the retina, the general area corresponding to V1 enters into excitement.
- It contains many simple cells that map linearly to small areas of the image, which is called 'Localized Receptive Field'. The convolutional feature extraction unit of convolutional network also mainly simulates the nature of simple cells.
- At the same time it also contains many complex cells that detect features in simple cells and have invariant detection capabilities for small translations of features, which is the source of inspiration for the 'Pooling unit' in the convolutional network.

From the point of view of machine learning, convolution brings two important ideas: Sparse Connectivity and Parameter Sharing. The former improves generalization performance by reducing the number of parameters while improving efficiency and the latter uses the same parameters to extract the same kind of features, greatly reducing the number of parameters that need to be stored.

1.2.1 Fully-convolutional neural networks

Inspired by deep learning breakthroughs in image processing domain, where rapid progress has been made in the past few years in feature learning [4], many convolutional network models have been made for extracting image features. The fundamental goal in applying deep learning to computer vision is to remove the cumbersome, and ultimately limiting, feature selection process and convolutional network takes advantage of the fact that we're analysing images, and sensibly constrains the architecture of the deep network so that we drastically reduce the number of parameters in our model [5].

Fig. 1.2 [14] shows the traditional structure of convolutional neural network, which consist of convolution layers, max-pooling processes and fully-connected layers. Here the green 'convolution' area includes convolutional layers and max-pooling processes and the blue area includes only fully-connected layers. The function of convolutional layers is to extract high-dimensional features and poolings reduce the size of feature map. As it is showed in the figure, the height and width of the feature map have been reduced every step after a pooling process and the depth grows in the first few steps by learning more features. Fully-connected layers, similar to deep-learning networks, are used for weight training, in the end select the most probable results using Softmax.

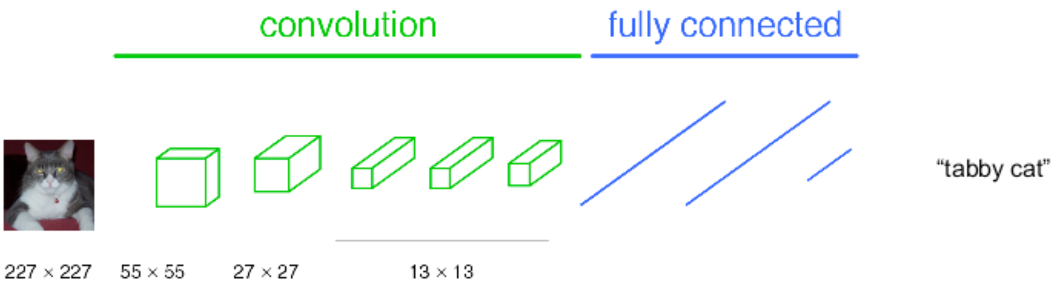


Fig. 1.2 Normal structure of convolutional neural network

The fully-convolutional structure shown below in the Fig. 1.3 [14]. Just like the literal meaning, it is a structure only contains convolutional layers all the time, with stride or max-pooling, but without fully-connected layers. Here in the end the height and width of the feature map is the 1/32 times of the original image by poolings or strides.

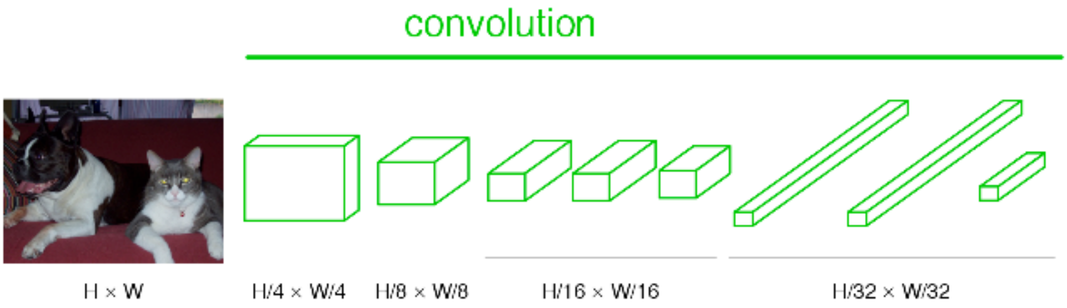


Fig. 1.3 Fully-convolutional structure

Fully-convolutional networks can have a process called Upsampling (marked in Fig. 1.4 [14]), which is a backward strided convolution process. That is to say, to increase the size of the feature map, step by step, until it reaches the original size, thus realize an end-to-end, pixel-wise prediction. Generally, there are 3 types of upsampling, FCN-32s, FCN-16s and FCN-8s, which can increase the height and width by 32 times, 16 times and 8 times respectively. The author in paper [9] explained the process in detail, and he made a conclusion that FCN-8s works better than FCN-32s and FCN-16s. In the same way we can also have FCN-4s or FCN-2s, but the author gave us a specific conclusion that once it belongs FCN-8s, the network cannot be optimized.

In the end we can apply Softmax to estimate the probability of each category. It is a pixel-wised estimation, because in the end the output image would be a probability estimation where larger the values of corresponding pixels, more probable that they belong to this specific category.

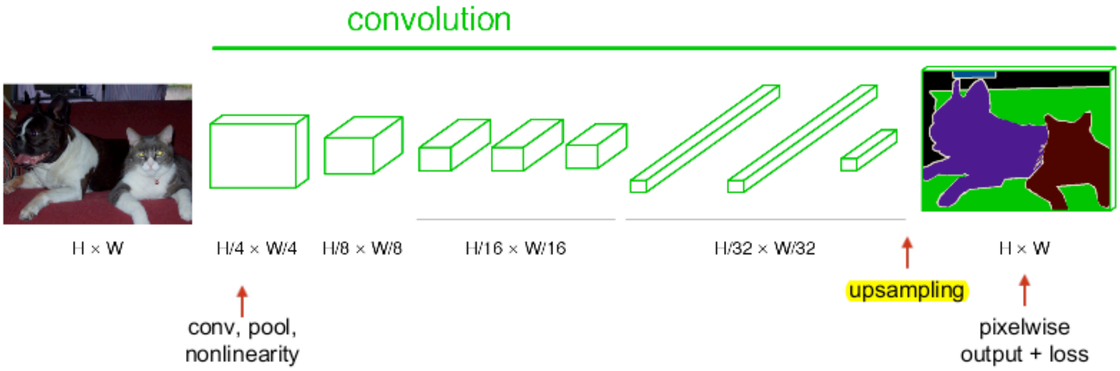


Fig. 1.4 Upsampling process in fully-convolutional network

The contribution of fully convolutional networks is that realize an end-to-end classification by learning with only convolutional layers, thus fully convolutional versions of existing networks predict dense outputs from arbitrary-sized inputs. Both learning and inference are performed whole-image-at-a-time by dense feed-forward computation and back-propagation (see Fig. 1.5) [9].

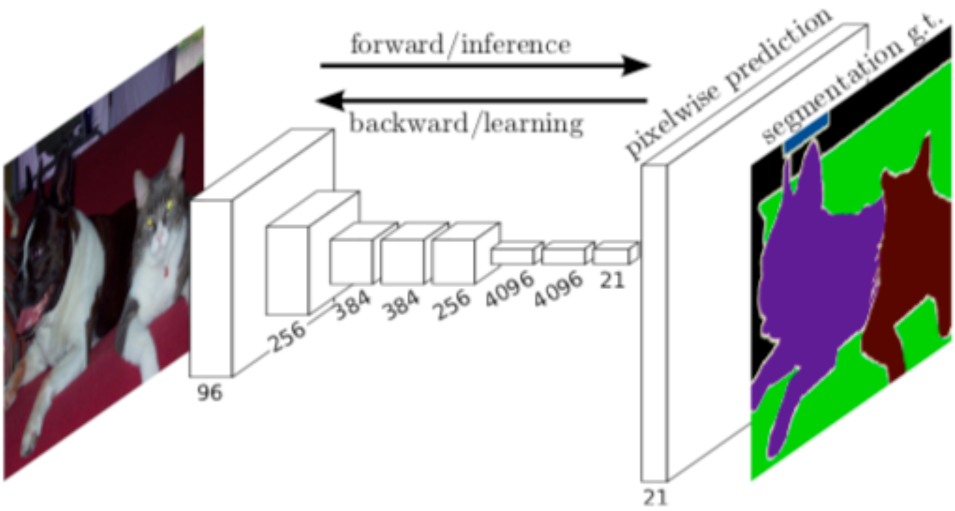


Fig. 1.5 Fully convolutional network can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

However, fully-convolutional networks have some shortcomings, one of them is that it uses relatively superficial features. For the reason that some upsample operation may add the pooling feature value of the upper layer, which may lead to insufficient use of high-dimensional features. Also, if we need upper layer features, we must pay more attention to the change of image size. One situation is that if the images of test set are too much larger or smaller than that of training set, effects of fully-convolutional networks wouldn't be so that good.

Anyway, the paper [9] written by J. Long and E. Shelhamer provides us a new method for semantic segmentation, which inspired the appearance of some outstanding semantic CNNs, for example the SegNet from Cambridge and symmetric back-forward convolutional network of Hyeonwoo Noh. In our project, we also used fully-convolutional network, without upsample process, without repeat process of the frames.

1.2.2 3D convolutional networks

Current SBD techniques are classified into two main categories: spatial-only and spatio-temporal analysis based [2]. The former estimates the temporal profile by comparing only spatial features such as colour histograms, edges, mutual information and entropy, etc, which can generate conservative detection accuracy with fast processing speed. Networks built on this method are known as 2D ConvNets.

Spatio-temporal techniques use optical flow to make detection more robust to scene and camera motions, which are known as 3D ConvNets. Compared to 2D ConvNet, 3D ConvNet is well-suited for spatio-temporal feature learning. Due to the fact that more than one frame may be included in one shot transition, containing the temporal features, 3D ConvNets approach is a better choice.

Fig. 1.6 illustrates the difference, 2D convolution applied on an image will output an image, 2D convolution applied on multiple images (treating them as different channels [10]) also results in an image [11]. Hence, 2D ConvNets lose temporal information of the input signal right after every convolution operation. Only 3D convolution preserves the temporal information of the input signals resulting in an output volume.

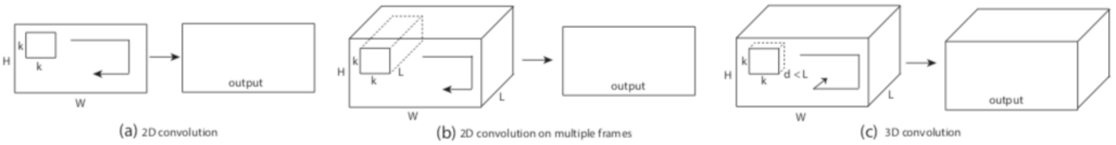


Fig. 1.6 2D and 3D convolution operations.

a) Applying 2D convolution on an image results in an image. b) Applying 2D convolution on a video volume (multiple frames as multiple channels) also results in an image. c) Applying 3D convolution on a video volume results in another volume, preserving temporal information of the input signal.

A deconvolutional method has been explained in [15], that how does 3D ConvNets learn internally [15]. As it is tested by [11], we observed that C3D starts by focusing on appearance in the first few frames and tracks the salient motion in the subsequent frames [11]. We took an example (see Fig.1.7) from the experiment of Tran's work [11], which shows clearly how the 3D networks learn when facing a series of frames with motion changes.

In the first example, we can see the first row are composed of 16 continuous frames with motion change of a woman, and the second row is the visualization of the deconvolution method act on feature maps with highest activations projected back to the image space [11], which clearly shows that in the first few frames the feature focuses on the whole person and later it tracks the motion of the pole vault performance over the rest of the frames. It is better viewed from colour screen.

Similar to this, the second example also shows that in the beginning 3D ConvNet focuses on the eye area and later it moves to the area around the eyes while applying the makeup.

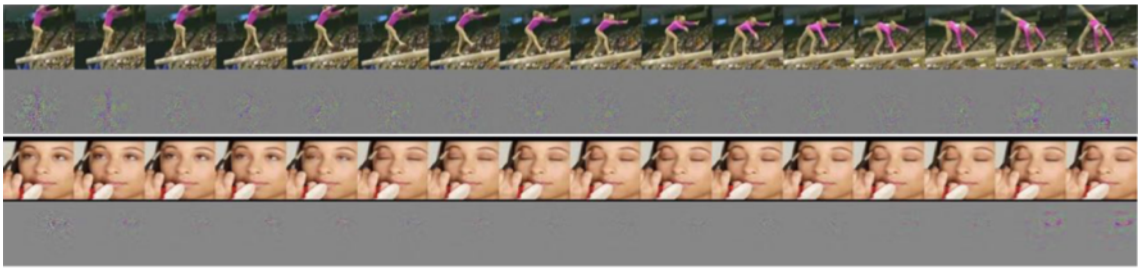


Fig. 1.7 Motion learning in 3D ConvNets

From this experiment we can see that C3D differs from standard 2D ConvNets in that it selectively attends to both motion and appearance, and it is definitely a proper course to take for analysing shot boundary, as it contains obvious motion change along the time between two shots.

1.3. Research framework

Regarding to the problems of SBD we've mentioned, considering the outstanding merits of 3D ConvNets, we constructed a 3D ConvNet structure using fully convolutional layers to implement the project based on the paper created by Michael Gygli [1]. Considering that there are some shortcomings in their work, for instance, it's not sensible enough to motion blur, falsely detected partial hard cut, etc, we made our own improvement in the training dataset for a better training. We also considered to modify the network architecture if necessary and optimize the parameter selection for our extended training dataset. At the same time, we used the visualization tool Tensorboard to visualize our work, which is really helpful to check problems in the neural network at the beginning.

1.4. Structure arrangement

The second chapter of this thesis states the network in detail, including the network architecture, feature maps and implementation details. The third chapter presents the dataset we've used for training and testing. The results are shown in the fourth chapter, followed by performance analysis. In the end, is the conclusion. The whole network has been implemented in python environment: Spyder, with the open source software library: TensorFlow. The details of the dataset and the code of this project have been attached in annexes, specified in file diretories.

2. Neural Network Implementations

We propose shot boundary detection as a binary classification problem [1]. The objective is to correctly predict if a frame is part of the same shot as the previous frame or not. Each frame-prediction is based on a context of 10 frames. We will decide if there is boundary between the 5th and the 6th frames, no matter hard cuts or gradual transitions.

2.1. Network architecture

Fig. 2.1 shows our fully-convolutional architecture, predicts frame-accurate labels directly from pixels. The advantage of this kind of architecture is that by using a model that is fully convolutional in time, we can increase the input size and thus make e.g. 11 predictions by analysing 20 frames or 91 predictions by analysing 100 frames, etc., thus minimizing redundant computation [1].

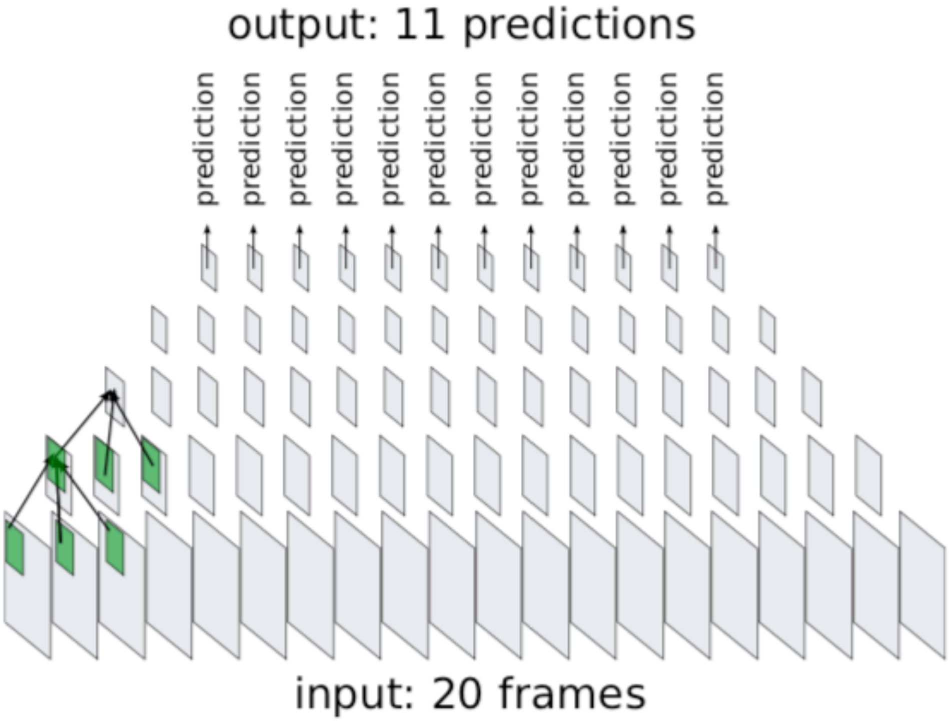
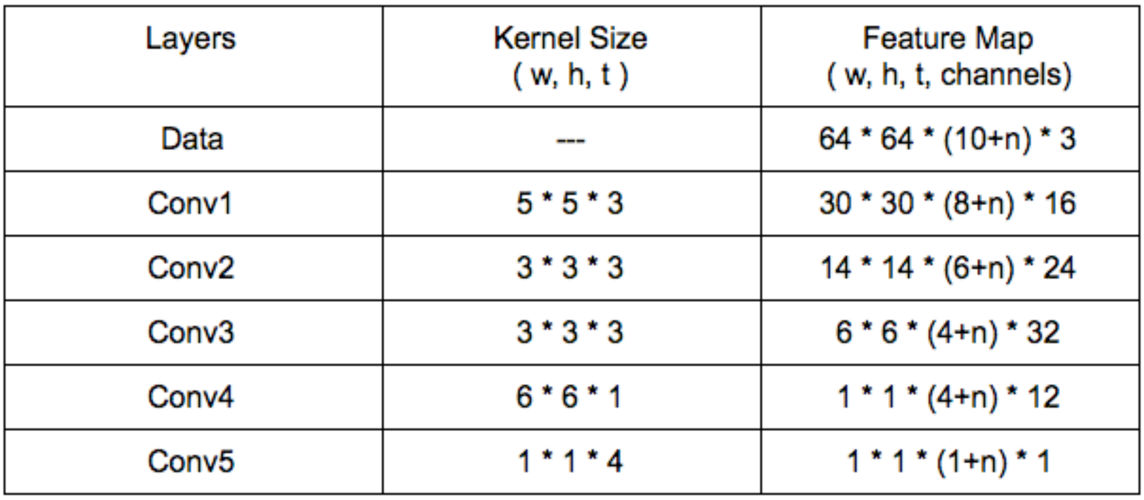


Fig. 2.1 Our fully convolutional architecture

By providing 20 frames, the network could predict labels for frames from 6th to 16th, thus making redundant computation unnecessary. This allows to obtain large speedups at inference.

Our network architecture is shown below in Table 1 in detail. In total we have 5 layers and all our layers are fully convolutional and each is followed by a ReLU non-linearity [1]. For the convenience of fully convolutional architecture, we are able to increase the input size by n, thus reusing the shared parts of the convolutional feature map and improving efficiency.

Table 2.1. Network architecture



Layers	Kernel Size (w, h, t)	Feature Map (w, h, t, channels)
Data	---	64 * 64 * (10+n) * 3
Conv1	5 * 5 * 3	30 * 30 * (8+n) * 16
Conv2	3 * 3 * 3	14 * 14 * (6+n) * 24
Conv3	3 * 3 * 3	6 * 6 * (4+n) * 32
Conv4	6 * 6 * 1	1 * 1 * (4+n) * 12
Conv5	1 * 1 * 4	1 * 1 * (1+n) * 1

Since our network is a 3D ConvNets, we refer the input feature map with a size of w * h * t * channels, where w and h are width and height of the frames, t is the length in number of frames, and channels refer to the number of channels, respectively. We also refer 3D convolution kernel size by w * h * t, where the w and h are kernel spatial size and t refers to kernel temporal depth.

We use a small input resolution of 64*64 RGB frames for efficiency and since such low resolution are often sufficient for scene understanding [13].

2.1.1 Comparison of hidden neuron output units

We defined our convolutional layers with xavier_initializer of the weight and with zero bias. This initializer is designed to keep the scale of the gradients roughly the same in all layers. In uniform distribution this ends up being the range: x = sqrt(6. / (in + out)); [-x, x] and for normal distribution a standard deviation of sqrt(2. / (in + out)) is used (defined in [22]).

Each convolutional layer is followed by an activation function using as hidden neuron output, and Rectified Linear Unit (ReLU) is always considered as an excellent hidden neuron output unit, due to the fact that Sigmoid and Tanh are easy to become saturated.

Fig. 2.2 shows the shape of Sigmoid and its derivative. And Tanh has the similar shape with Sigmoid, shown in Fig. 2.3, with the difference that it squashes real number to range between [-1,1].

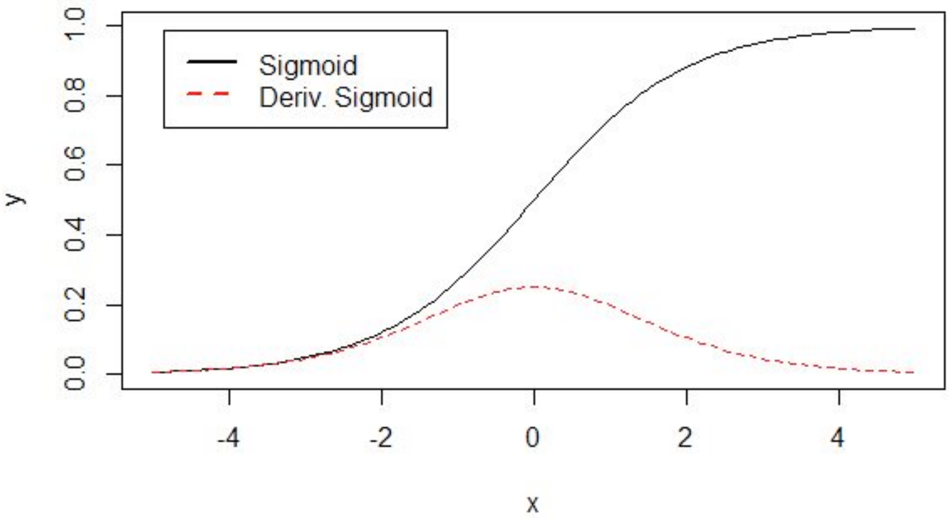


Fig. 2.2 Sigmoid and its derivative.

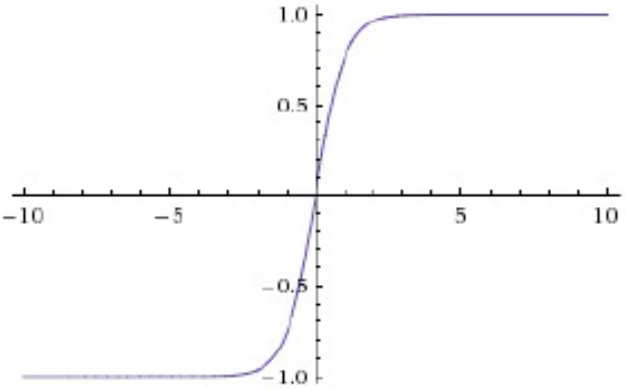


Fig. 2.3 Shape of Tanh

Which is shown in Fig. 2.2 is that when the input value is very large or very small, the gradients of Sigmoid and Tanh will be near to 0. That is to say, when we are trying to modify the weights in order to learn, we may nearly get zero gradient, so it cannot learn more. This situation is like a cup of salt water near to saturation. Thus, it’s quiet important to pay attention to the initial value of the parameters, in order to avoid saturation.

Since ‘gradient disappearance problem’ is the nightmare for network learning, ReLU is really a good choice to combat it. The shape of ReLU shows in Fig. 2.4 and (2.1-1&2) are the expressions of ReLU.

$$f(x) = x \quad \text{when } x >= 0 \qquad \textbf{(2.1-1)}$$

$$f(x) = 0 \quad \text{when } x < 0 \qquad \textbf{(2.1-2)}$$

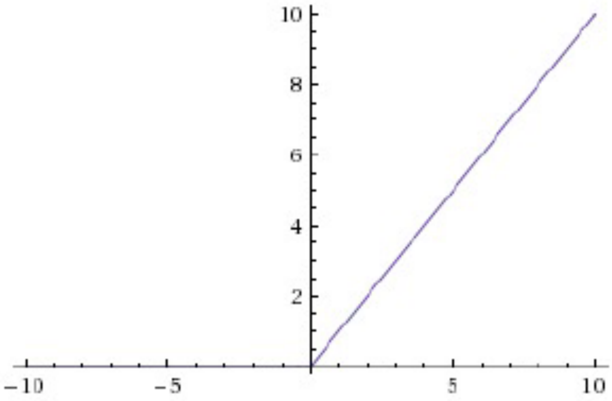


Fig. 2.4 ReLU shape

Comparing to Sigmoid and Tanh function, ReLU has mainly three differences:

- Unilateral inhibition: Function value equals to 0 when input less than 0.
- Relatively wide excitement border: When input is more than 0, the function is linear with derivative equals to 1 until infinity. Unlike Sigmoid, whose excite border is very narrow.
- Sparse activation: it only selectively responds to a small part of the input signal, and a large amount of signals are deliberately shielded, as it inhibits the other side.

In 2001, Attwell et al. speculated that neuronal coding work is sparse and distributed based on observational learning of brain energy expenditure. In 2003, Lennie et al. estimated that only 1 to 4% of neurons were activated simultaneously in the brain, further indicating the sparseness of neuron work. Thus, ReLU is similar to the working principle of human neurocritical layer, which can improve the accuracy of learning and extract sparse features better and faster.

From this point of view, after the initialization of the weights, the traditional Sigmoid function has nearly half of the neurons activated at the same time, which is inconsistent with the study of neuroscience, and it will cause great problems for deep network training.

However, there is no such perfect function, ReLU also has its weakness. Such as ‘no derivative’ at the point ‘0’, cause its left derivative is 0 and right derivative is 1, also we cannot train the negative axle. So, between many years researchers gave some improvements to ReLU, such as Leaky ReLU and Absolute Value Rectification. There is one called Softplus, it’s like a smooth version of ReLU, whose shape shown below in Fig. 2.5, comparing to ReLU with green line (Softplus) and blue line (ReLU).

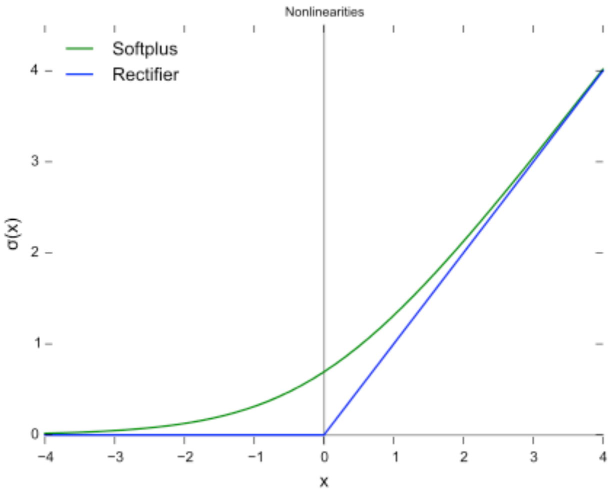


Fig. 2.5 Shape of Softplus and ReLU.

$$\text{Softplus}\ (x) = \ln(\ 1 + e^{x}\ ) \quad \textbf{(2.2)}$$

With the expression of (2.2), we can get the derivative at x=0. Comparing to ReLU, we can get the derivative for every input value and it's not easy to be saturated neither comparing to Sigmoid. It is smoother than ReLU, but has the same function property. Nevertheless, in 2011, Glorot compared Softplus and ReLU and found that the later one is better [20]. With the fact that 'Practice is the sole criterion for testing truth', normally it's not encouraged to use Softplus, although it seems perfect in theory, but not good in practice.

From our point of view, Softplus doesn't work well maybe because the lack of 'Sparse activation', which is really important in biological nerve.

Last but not least, there is no prove that ReLU is suitable for all kinds of networks, we need to choose suitable function for different tasks via lots of trials. For instance, in recurrent neural networks, we don't often use piecewise linear activation function like ReLU, but tend to use Sigmoid or Tanh, with the knowing of saturation. Thus, the choice of hidden neuron is a kind of hyperparameter, referring to variables based on empirical, which need to be tested in real case.

2.2. Details of implementation

2.2.1 Feature map and strides

According to the network structure proposed in [1], it only contains convolutional layers, without max-pooling layers to reduce the dimension. And for each layer, we used a stride of 2. In C3D the shape of the stride should be [1, 2, 2, 2, 1], which hasn't been clarified in [1], but it has been explained in Tensorflow website [12]. We used a stride = [1, 1, 2, 2, 1] for the reason that our input has the dimension of [10, 64, 64, 3], the first parameter refers to the 10 frames in one snippet, which we don’t want to reduce or decrease it. Although it hasn’t been clarified in [1], we believe that we took the same decision. So here we set our stride equals to [1, 1, 2, 2, 1] instead of [1, 2, 2, 2, 1] in order to keep the first dimension.

Fig. 2.6 is an illustration of a filter’s stride hyperparameter [5] for extracting features, where the filter has three different weights. If the stride=1, we get the full convolution computed in the figure. For example, here ‘-7’ is computed by (3*1+(-4) *2+2*(-1)) and we get five outputs after that. When the stride=2, we don’t make the fully compute, instead, it gets convolution computation at the distance of two, which is equivalent to convolutional computation with a decimation by a factor of 2. In the case of stride=1, if we sample it with a distance of 2, we’ll get the same result as we sample all of them when stride of two. But the computation complexity increases a lot if we make full convolution, which is a waste of computer resources.

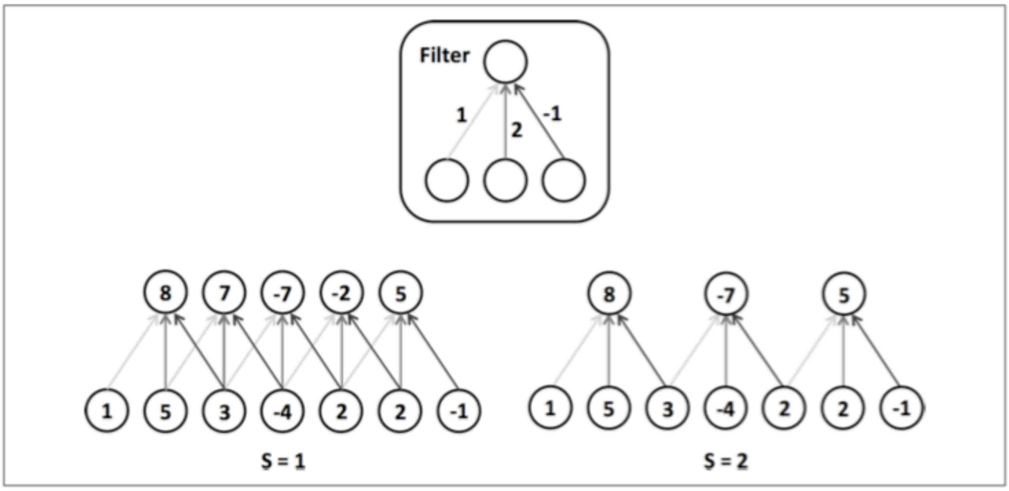


Fig. 2.6 Illustration of strides

2.2.2 Comparison of classifiers: Softmax vs. Sigmoid

For the output unit of the neuron network, we usually calculate probability when dealing with classify tasks, the most widely used are Sigmoid units or Softmax units. From function level (target category prediction) these two functions are the same. However, they are obviously different from mathematics point of view, which play a vital role in deep learning and the other research fields.

The expression of Sigmoid shown in (2.3):

$$f(x) = (1 + e^{-x})^{-1} \quad \textbf{(2.3)}$$

This function takes any range of real numbers and the returned output value is in the range of (0, 1). It has a ‘S curve’, shown in Fig. 2.7, clearly that Sigmoid can be used for two-level classification.

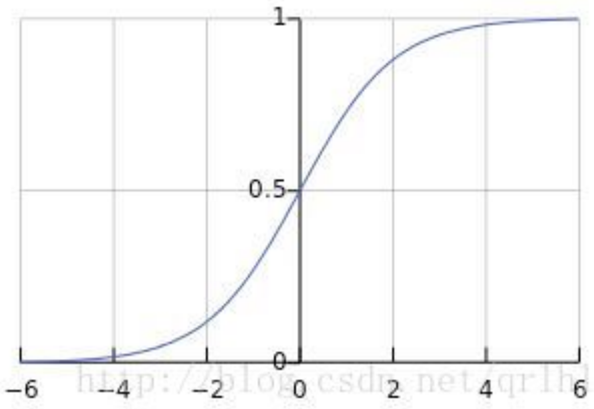


Fig. 2.7 Shape of Sigmoid

From a mathematical point of view, the Sigmoid function has a large signal gain in the central region and a small signal gain in the two side regions, which has a good effect on the feature space mapping of the signal. From the perspective of neuroscience, the central region resembles the excitement of neurons, and the two regions resemble the inhibition states of neurons. Therefore, in the aspect of neural network learning, the key features can be pushed to the central area and the non-key features can be pushed to both sides.

The Softmax function is also often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression [16].

Which is different from Sigmoid is that the domain of Softmax is a vector, it returns probabilities with the sum equal to 1. The Softmax function calculates the probability distribution of more than two events. In general, this function will calculate the probability of each target category in all possible target classes.

The expression of Softmax shown in (2.4):

$$f(z_j) = \frac{e^{z_j}}{\sum_{i=1}^{n} e^{z_i}} \tag{2.4}$$

It can be seen from the formula that if one z_j is larger than the other z, then the mapped component (the probability of target category) will be close to 1 and the others will be close to 0. We can see that when n=2, the expression will be exactly the same with that of Sigmoid, thus we can say that the Softmax function can be viewed as a generalization of the Sigmoid function, it can be used for multi-classify, while Sigmoid can only classify two classes. We can also understand Softmax as it maps K-dimensional vectors to another K-dimensional vector. In terms of communication, if the Sigmoid function is MISO, Softmax is the MIMO Sigmoid function. [19]

In this project we chosen Sigmoid function for classification unit, considering that we only have two labels: ‘0’ and ‘1’, which refer to ‘no boundary in the middle' and 'boundary in the middle' respectively. Thus, it's suitable for us to use Sigmoid. In the paper of Michael Gygli [1], they used Softmax for classification, suppose that they may made their labels as '[0,1]' and '[1,0]' to represent 'no boundary in the middle' and 'boundary in the middle', or vise versa. From function level, these two functions for our project are exactly the same, depend on our label form, we chosen Sigmoid.

2.2.3 Loss function

We need a cost function to estimate the extent of inconsistency between the model's predicted value and the real value. The gradient in the gradient descent refers to the partial derivative of the cost function to each parameter. The direction of the partial derivative determines the direction of the parameter decline in the learning process. The learning rate determines the step size of each step change. The derivative and learning rate can be updated using the Gradient Descent Algorithm.

To train our model, we used cross-entropy loss, which minimize with vanilla stochastic gradient descent. As we explained in the previous paragraphs, Sigmoid and Softmax units are easy to saturate, which apparently shows in their curves. Thus, we need to provide it a relatively large gradient to overcome the saturation when it appear an 'error'. That's why normally in neuron networks we use cross-entropy as cost function by default.

Here we define cross-entropy cost function as:

$$C = -\frac{1}{n}\sum_{x}[yIna + (1-y)In(1-a)] \qquad \textbf{(2.5)}$$

for one neuron with multiple inputs and one output, where y refers to expecting output and a is the real output. In the case of Sigmoid:

$$\alpha = \sigma(z) = (1 + e^{-z})^{-1} \text{ where } z = w * x + b \qquad \textbf{(2.6)}$$

And the partial derivative of weight and bias when we use Sigmoid:

$$\frac{\alpha C}{\alpha wj} = \frac{1}{n}\sum_{x} xj\,(\sigma(z) - y) \qquad \textbf{(2.7)}$$

$$\frac{\alpha C}{\alpha b} = \frac{1}{n}\sum_{x} xj\,(\sigma(z) - y) \qquad \textbf{(2.8)}$$

We can see from the expression is that for cross-entropy, the renewal of the weight is depend on the difference between expecting output and the real output (σ(z)-y). If there is a big difference, weight will renew faster, and in the opposite situation, the other way around. This is a really good character in deep learning, and it can overcome the problem of cost function that weights update too slowly.

2.2.4 Optimizer

For optimizer we used Momentum-Based Optimization instead of Stochastic Gradient Descent (SGD), which is also widely used in deep learning. The difference between Momentum and SGD is that in SGD, how long we go in one step depend on the simply multiplication of gradient and learning rate, but in Momentum algorithm, how long we go in one step depend on previous velocity and current strength, which refers to gradient.

Similar to physics, we use the variable v to express the velocity, indicating the direction and rate of the parameter's movement in the parameter space, and the negative gradient of the cost function indicates the force of the parameter's movement in the parameter space. According to Newton's laws of motion, momentum equals mass times velocity, while in Momentum, we assume that the quality of the unit 1, so the speed v can be directly used as momentum. We also introduce hyper-parameter β, whose value ranges between [0,1], is used to adjust the attenuation of the previous gradient.

The following expression (2.9) is the updated velocity, where α is initial learning rate and β is initial momentum parameter. And w is the updated parameter for the next step.

$$V = \beta v - \alpha \square w \quad \mathbf{(2.9)}$$

$$W = w + v \quad \mathbf{(2.10)}$$

One way to think about how we might tackle this problem is by investigating how a ball rolls down a hilly surface. Driven by gravity, the ball eventually settles into a minimum on the surface, but for some reason, it doesn’t suffer from the wild fluctuations and divergences that happen during gradient descent. Why is this the case? Unlike in stochastic gradient descent (which only uses the gradient), there are two major components that determine how a ball rolls down an error surface. The first, which we already model in SGD as the gradient, is what we commonly refer to as acceleration. But acceleration does not single-handedly determine the ball’s movements. Instead, its motion is more directly determined by its velocity. Acceleration only indirectly changes the ball’s position by modifying its velocity [5].

Velocity-driven motion is desirable because it counteracts the effects of a wildly fluctuating gradient by smoothing the ball’s trajectory over its history. Velocity serves as a form of memory, and this allows us to more effectively accumulate movement in the direction of the minimum while cancelling out oscillating accelerations in orthogonal directions. Velocity is used to accumulate parameter gradients for each round of training, and the larger the β, the greater the influence of the previous gradient on the current training gradient. Assume that the direction of each training gradient is the same, just as the ball rolls down from the slope, but due to the presence of the attenuation factor β, the ball does not always accelerate down, but reaches the maximum speed and after with uniform speed forward. In practice, the commonly used β value can be 0.5, 0.9, 0.99, whichever is appropriate.

2.2.5 Learning rate decay

In practice, we always need to decrease the learning rate by the increase of training steps. The reason is that some algorithm may introduce source noise, and this kind of noise is caused by the special nature of individual data [17]. Thus, even though we are around the optimal solution, the noise will not disappear and causes oscillation near the optimal solution. To eliminate or mitigate this situation, we try to minimize the learning rate around the optimal solution.

For the learning rate decay, the most common method used is Exponential_decay, the advantage of this method is that the convergence speed is faster and simpler. Similar to that, we have Piecewise_constant, Polynomial_decay, Natural_exp_decay, etc. These several methods are not much different, for they are mainly based on exponential attenuation. There exist a problem is that in the beginning, the learning rate falls rapidly, thus in complex problems, it may lead to rapid convergence to local minimums without exploring a certain range of parameter space.

What we used in the project is Cosine_decay. It is a new strategy proposed in the last year (2017) [21] by Ilya Loshchilov and Frank Hutter. The basic shape of whom is the cosine function, shown in Fig. 2.8, where the red line refers to alpha=0.3 and blue line refers to alpha=0.0, with alpha the minimum learning rate we can accept.

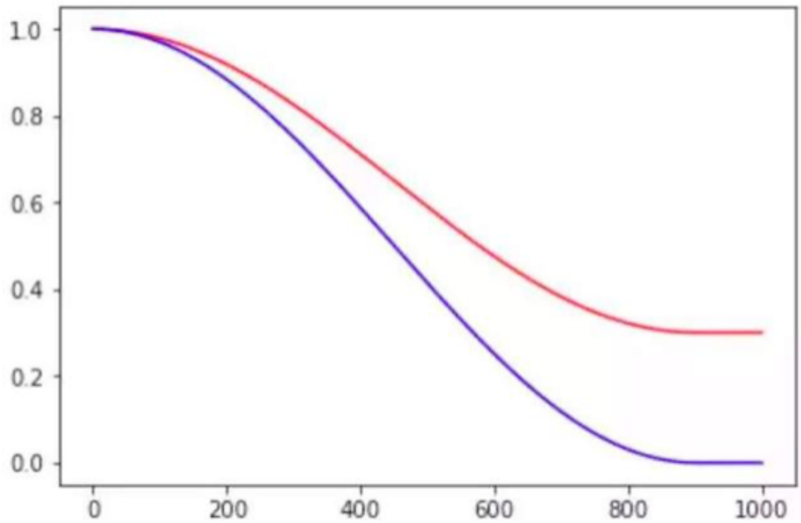


Fig. 2.8 Cosine_decay function shape

These four steps below are the calculations for decayed learning rate, where ‘decay_steps’ is the total iterations we’ll take and ‘global_step’ refers to in which step we are. We took alpha equals to 0.0 in our work.

global_step = min(global_step, decay_steps) **(2.11)**

cosine_decay = 0.5 * (1 + cos (pi * global_step / decay_steps)) **(2.12)**

$$\text{decayed} = (1 - \text{alpha}) * \text{cosine\_decay} + \text{alpha} \quad \textbf{(2.13)}$$

$$\text{decayed\_learning\_rate} = \text{learning\_rate} * \text{decayed} \quad \textbf{(2.14)}$$

3. Dataset Preparation

The dataset has been generated in a two steps' process. First, we've collected and verified hard cuts in real case from our material, after that we've generated other types of transitions applying the algorithms indicated in paper [1], that is our augmented dataset.

3.1. Extract shot boundaries in real case

To obtain the dataset large enough to train the network, we created dataset using randomly selected 70 episodes from more than 20 American series (details in Annex A). Each episode has a text file attached, indicating the number of frames that should be a boundary, detecting with the traditional method. And we verified them manually the selected episodes, to see if it is correct or no, we labelled the errors of the provided text file and to use them to make our accurate dataset.

Proofed by facts that most of the judge errors of algorithm occur in the period of introduction, for the reason that the images and letters are vibrating all the time, thus influence the judgement of algorithm. Another common error is the flash, or the changing of light colour, which heavily influence recognition results. Due to the fact that most of the series we chosen are criminals, and cases occur mainly at night or obscure places, once there is a gunshot, the specific frame would be very bright, thus leads to judge error. Also, the police car with red and blue lights, whose the colour will fill all the image thus could be judged as different shots.

In the process of manual verification and re-annotation, if there is a boundary not in the middle place (between the 5th and 6th, we labelled them as 'no boundary', because we want to train our network to learn that the boundary is in the middle. If it is a dissolve or fade transition, we need to see if the boundary is in the middle of the transition or no.

In regard to the weakness mentioned in Michael's paper [1], we paid more attention on motion blur and background change. We marked these kinds of change as 'no transition', to put them into our network for training, in order to make the network resistive to them.

In our case, one special situation is that one single frame has been divided into 2 or 3 parts, each part contains a scene. Once one of the parts changes, we judged it as a shot boundary for the fact that there is a boundary of one shot, although the whole image frame hasn't change that much. However, this may bring some defects in the network learning, for example, some partial change of the background can be detected as shot change, which will be an error in the future prediction.

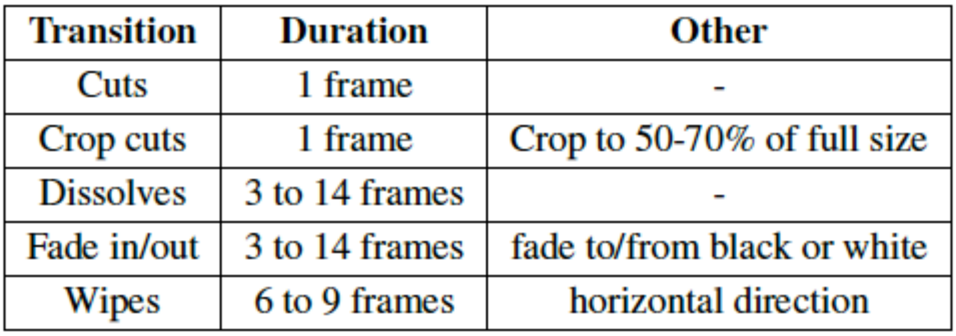
In this part we have 35072 snippets as input data, which is or dataset in real case, including ‘transition’ (‘1’) and ‘no transition’ (‘0’). Each input has a shape (10*64*64*3). And we divide 10% of it as our test dataset for the network to make evaluation and to calculate accuracy.

3.2. Augmentation of the dataset

3.2.1 Shot boundary generation

For the better training of the network, we need to generate more dataset to provide a large variety or to make the network resist to a specific kind of transition. In general, we need to generate five kinds of transition: hard cuts, crops, dissolves, fade ins/outs and wipes. Table 3.1 [1] gives us a specification how these different transitions generated.

Table 3.1 Transitions references



Transition	Duration	Other
Cuts	1 frame	-
Crop cuts	1 frame	Crop to 50-70% of full size
Dissolves	3 to 14 frames	-
Fade in/out	3 to 14 frames	fade to/from black or white
Wipes	6 to 9 frames	horizontal direction

- Cuts. Its duration is one frame, which means among the 10 frames, the 5th and the 6th are from different shots respectively, thus they are obviously different. These cuts are easy to generate, just to take five continuous frames from one shot and another five from another shot and make an input of 10 frames.

- Crop cuts. This transition also with a duration of one frame. In order to generate this cut, we took 10 continuous frames from one single shot, from the 6th one, we made a crop cut, up to 50%-70% of full image size. Although the cropped sizes are randomly chosen, in one snippet, these five cropped frames have the same cropped size and the same cropped position. With this method, we made new input snippets by connecting the five cropped frames with the five original frames.

- Dissolves. It is a kind of transition that linearly interpolates between shots, with a duration of 3 to 14 frames. To generate this, we took two samples from different shots, each sample contains 10 continuous frames. And then we generated numpy arrays according to the duration, one array contains 10 elements ranging from 0 to 1, with which we combine the two samples by sequence to make linear interpolations, in the end returned one new snippet containing 10 new frames, that is a new dataset.

- Fade in/out. This transition is similar to dissolves, but it interpolates linearly from one frame to a frame of single colour (normally black or white), also with a duration from 3 up to 14 frames. The method to generate these transitions are quite similar to that of dissolves, only to use a black or white frames as one sample to generate linear interpolations.

- Wipes. This is a type of transition that one shot is moving out while the other shot is moving in, typically in horizontal direction, with a duration of 6 to 9 frames. For an on-going frame, one part is a part of a frame from previous shot, the other part comes from a frame of the next shot. These two parts form an on-going frame, proportionally, depend on its duration. To generate this, we also made numpy arrays, each has 10 elements valuing from 0 to 1, proportionally according to the duration. The same with dissolves we used two samples taken from different shots, to crop the frames respectively according to the proportion indicating in the array, and append the corresponding part together to generate a new snippet.

It’s important to understand that here the duration with the number of frames means that how many frames are on-going. Taking wipe transition as an example, if the duration is 6 frames, which means that among the 10 input frames, there are 6 frames that contain two parts, and we kept the transition in the middle by correctly making numpy arrays.

Since we already have the dataset taken from the episodes, we could use them to generate our new data without reading new frames again. Most of the transitions require us to take samples with 10 continuous frames, we could use the snippets that with the label ‘0’. However, it will arise a problem is that, snippets labelled with ‘0’ not only refer to 10 continuous frames, but also include the case that there is a boundary not in the middle, using that must bring some errors in the dataset.

Thus, we decided to take samples with label ‘1’, which contain 10 frames each, with a boundary in the middle, and we are sure that the first five frames are continuous and the last five the same, as we assume that most of them are hard cuts. We can take any two of them to make a new hard cut, shown in Fig.3.1.

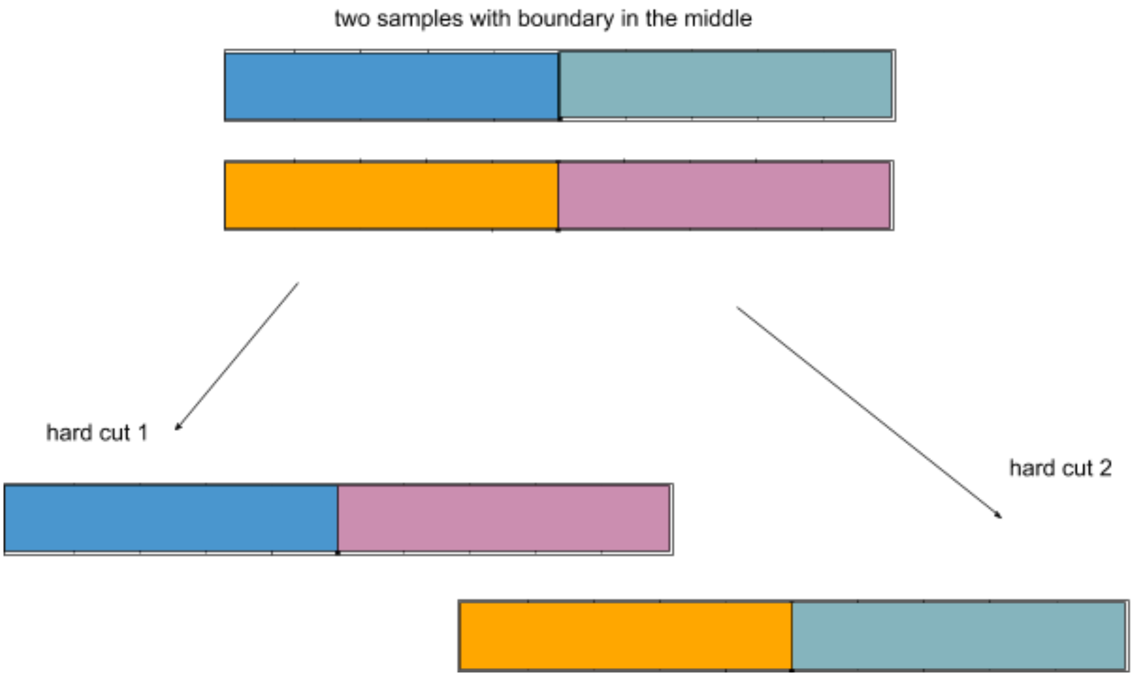


Fig. 3.1 Hard cuts generation

With this method, we considered that we may mislead the network by putting the first parts of the hard cuts always in the first, so the network may learn that the last frame of the first part is the boundary place, not the position between the 5th and 6th. Thus, we've made another kind of hard cuts shown in Fig. 3.2, in which we exchanged the position of two parts of the cut, to avoid such problems.

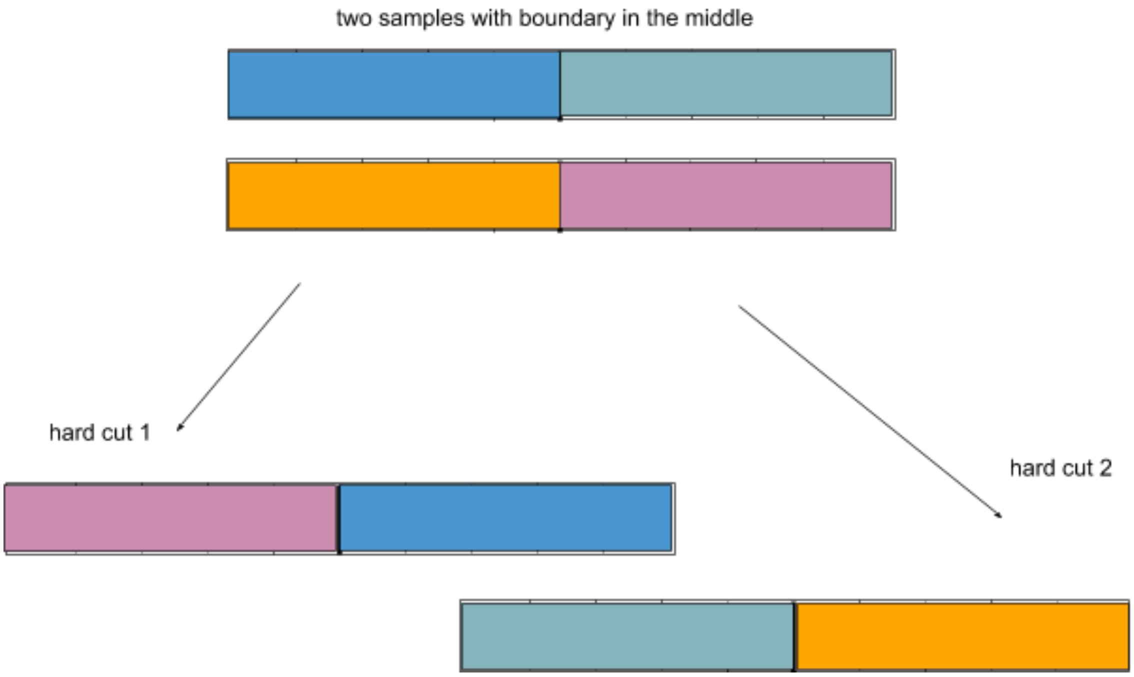


Fig. 3.2 Hard cuts generation without misleading

Fig. 3.3 shows some samples generated by our algorithm, where the first 10 frames is a new data and the next 10 frames is another. And then we used the new generated hard cuts to make the other transitions.

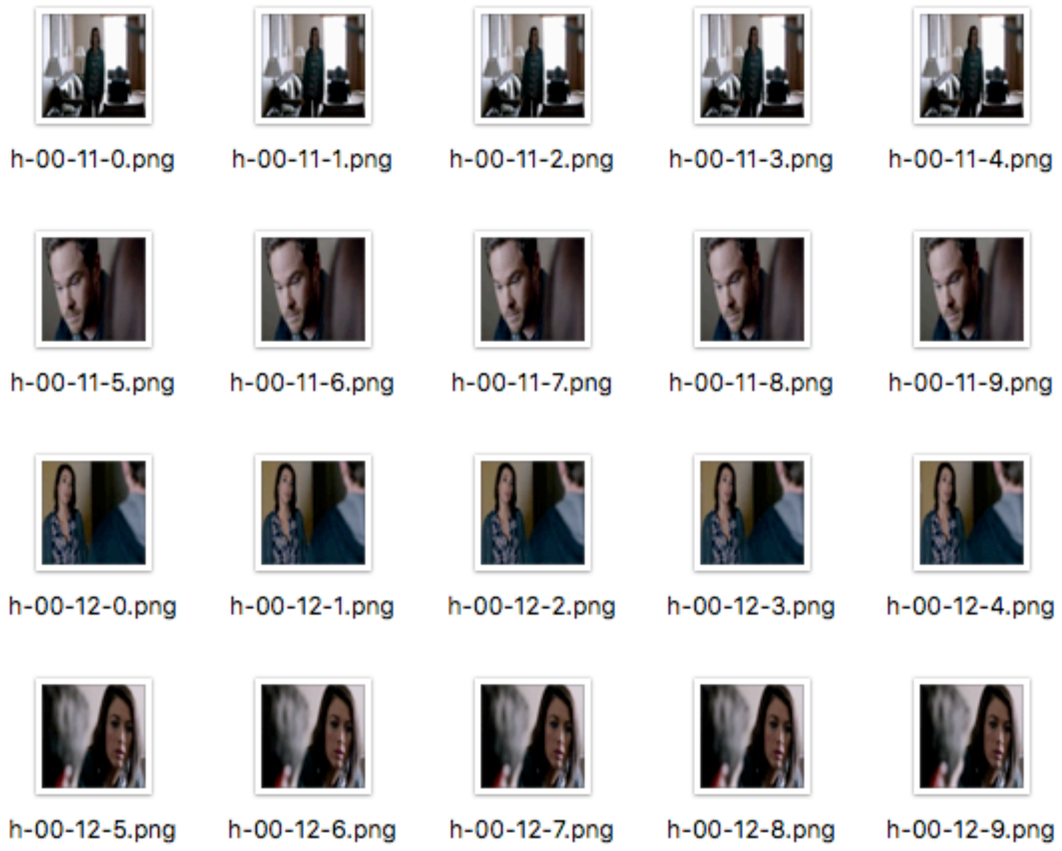


Fig. 3.3 Generated hard cuts

In order to make the other transitions, we need to make samples of 10 frames with ‘no transition’ first to suit our algorithms (here 'no transition' refers to continuous 10 frames without boundary in any place). There are many methods to make 10 continues frames, in our project we use three methods to generate ‘no transition’: Mirror, Copy and Double.

For the mirror method, we took the first five to make five mirror images, to be attached in front of the sample, together with the original five frames, we can make one new sample x, containing 10 continuous frames. The same with that we can make another continuous sample y using the other five frames in this snippet. Fig. 3.4 indicates the way to generate continuous samples.

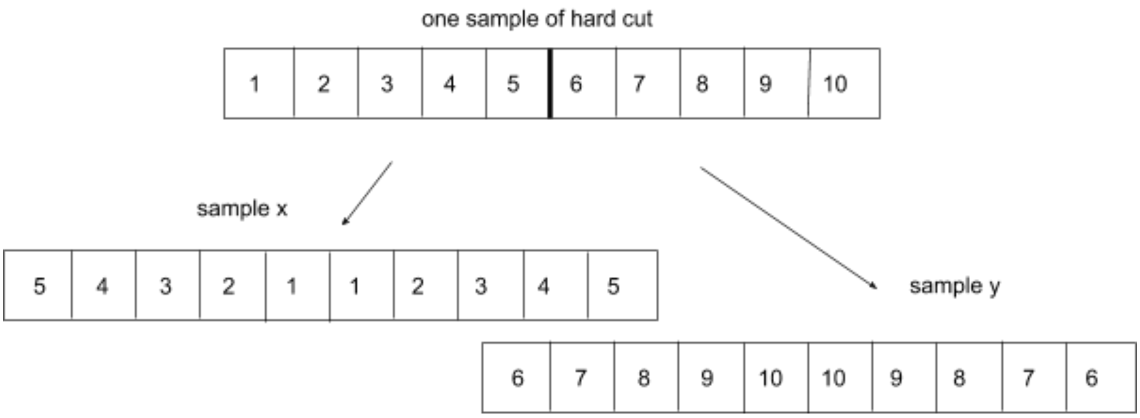


Fig. 3.4 Mirror method to make continuous samples

Copy method is simple, for one sample taken, we only used the first frame and the 10th frame and copy them to make continuous samples x and y, shown in Fig. 3.5.

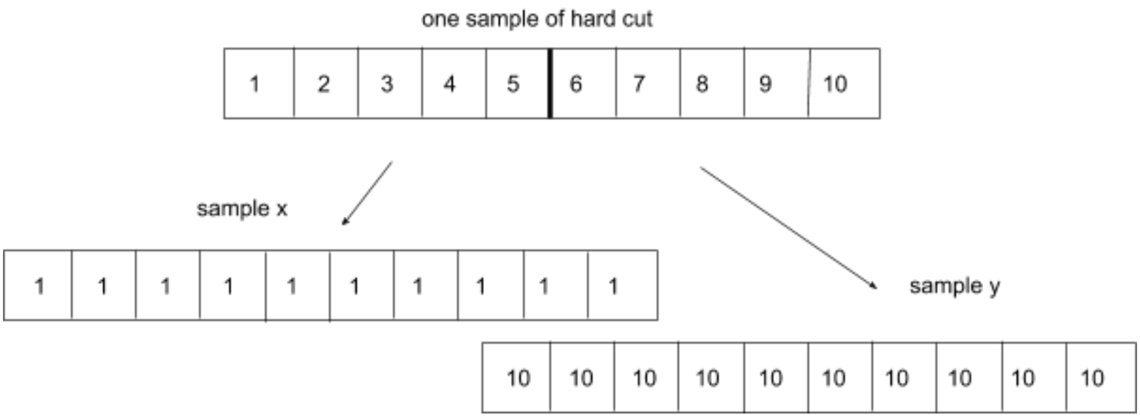


Fig. 3.5 Copy method to make continuous samples

And the double method is that for every frame in a hard cut, we made a copy, it’s like we doubled every frame to make a sample of 20, and cut them into two (See Fig. 3.6).

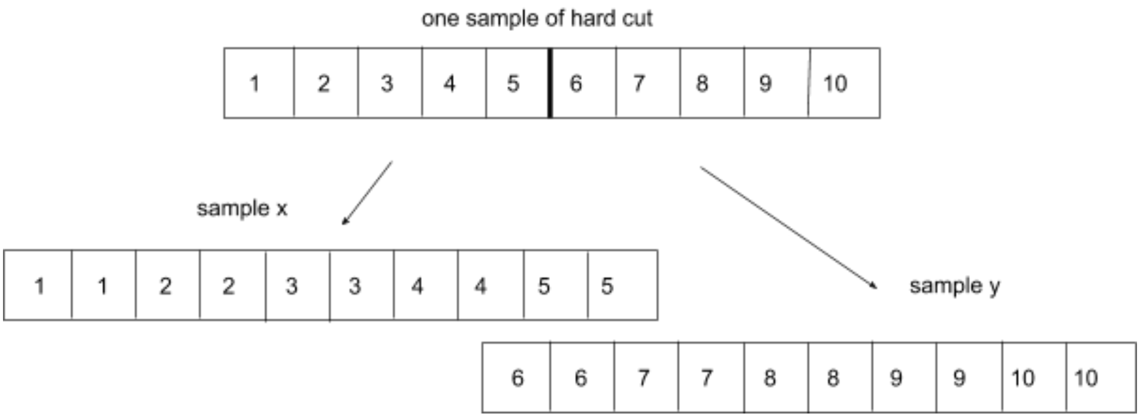


Fig. 3.6 Double method to make continuous samples

Now we can use our newly-made ‘no transition’ samples to generate the other transitions. For the crops, we can make a crop to the last half of sample x (or sample y) and keep the first five frames to form a crop cut. Fig. 3.7 gives us an example of crop cut.

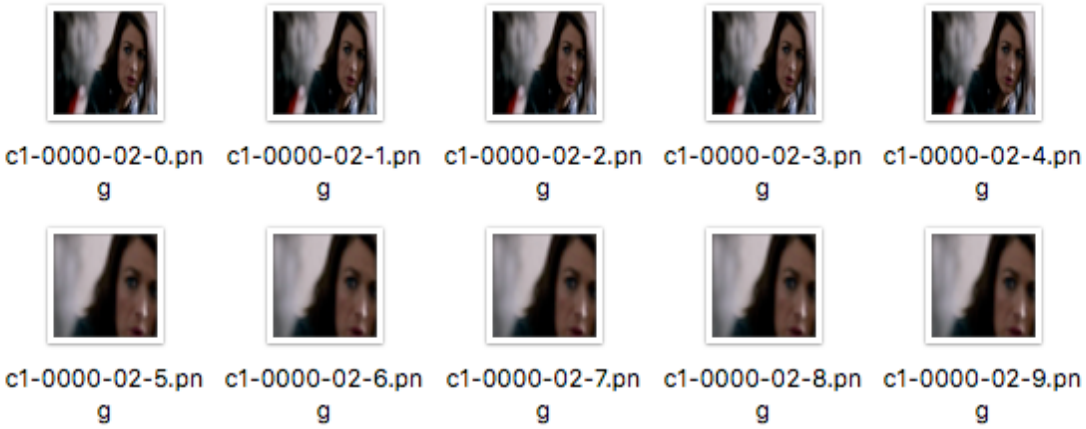


Fig. 3.7 Generated crop cut

In the case of dissolves and fade ins/outs, sample x and sample y can be used as two continuous samples to generate linear interpolations, and for the wipes, these two samples also can be used. Fig. 3.8.a provides us a sample of dissolve we generated, with a duration of. Fig. 3.8.b shows two samples of fade in and fade out respectively and Fig. 3.8.c gives us a sample of wipe.

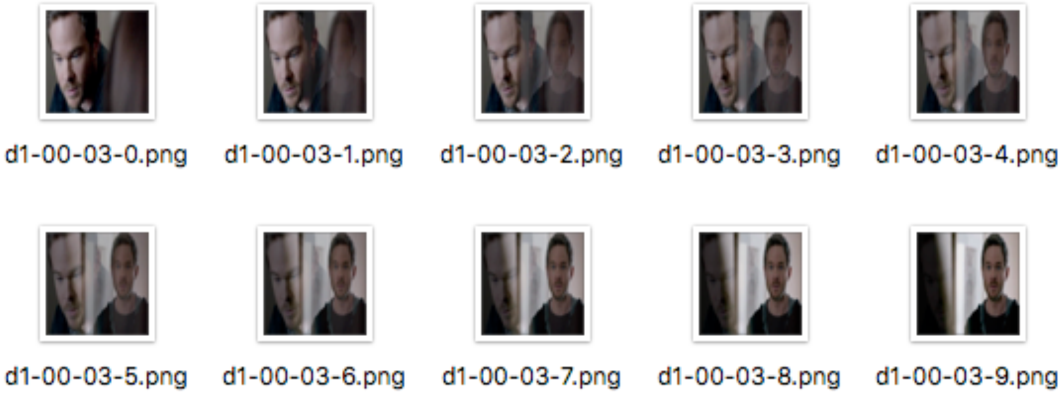


Fig. 3.8.a Generated dissolve transition

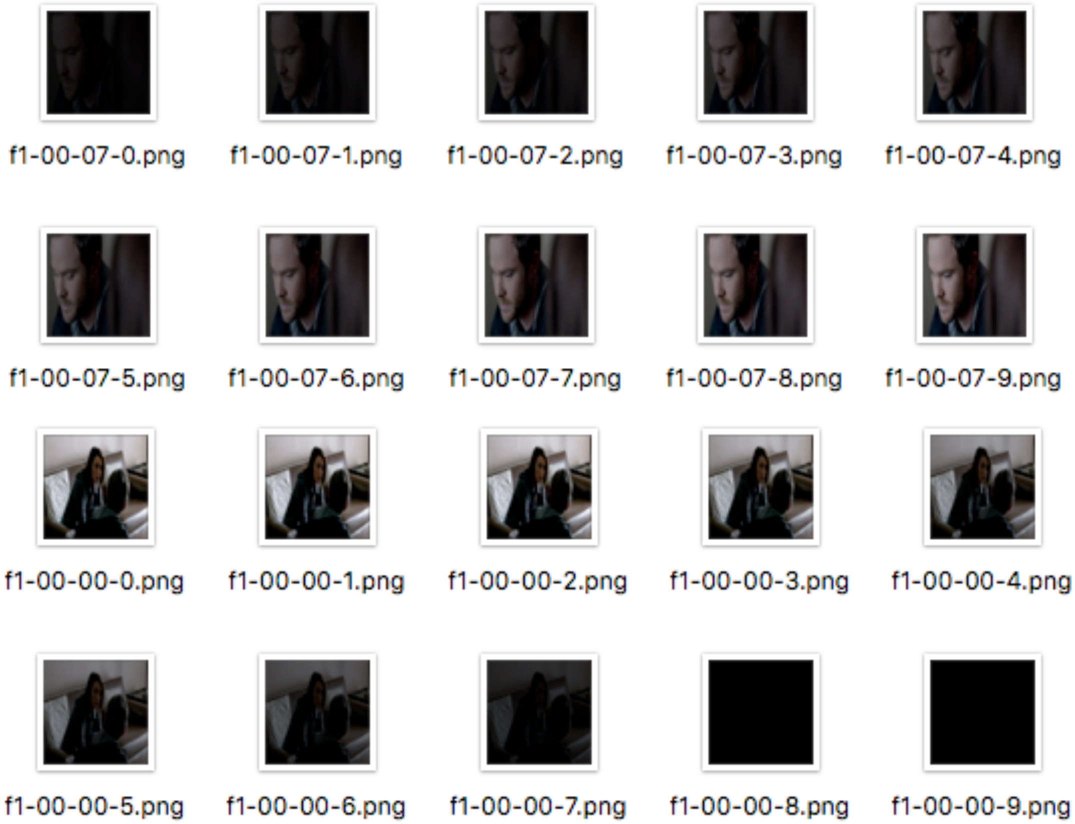


Fig. 3.8.b Generated fade-in & fade-out transitions

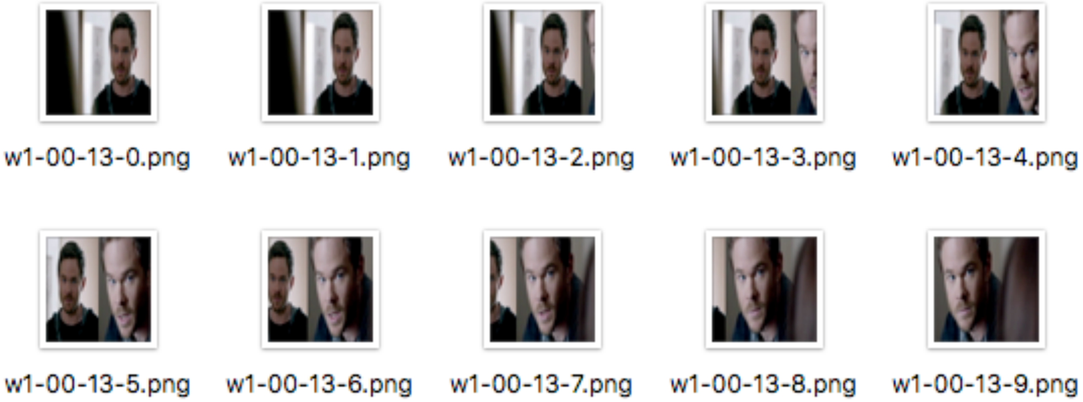


Fig. 3.8.c Generated wipe transition

Also, the sample x and sample y can be used as new ‘no transition’ data (continuous samples), which will be similar to the original datasets. Fig. 3.9 shows us an example that is ‘no transition’.

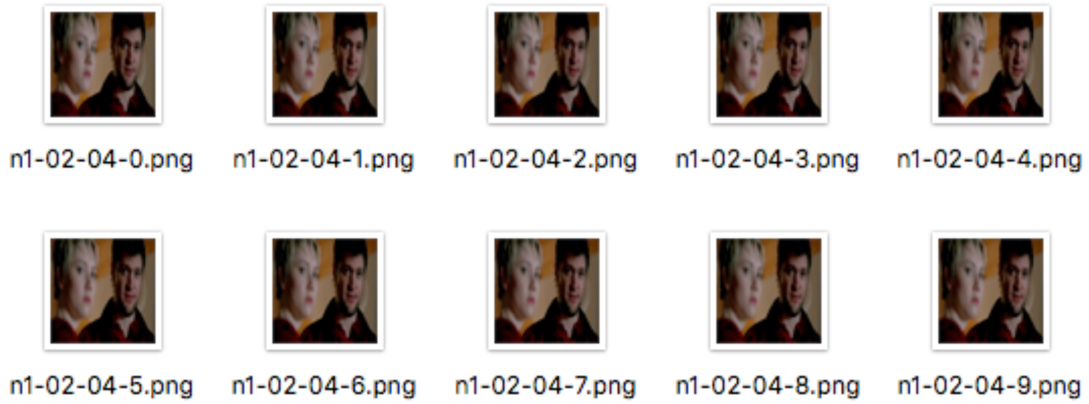
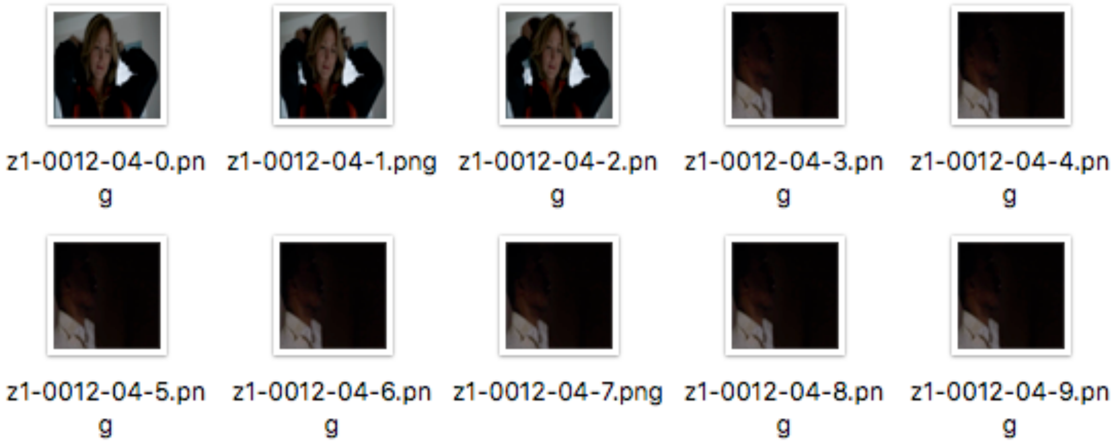


Fig. 3.9 Example of 'no transition'

3.2.2 'Negative' dataset generation

Remember that one important function of our network is to judge if there is a shot boundary in the middle position. Thus, we need to generate a large amount of ‘negative’ datasets, with transition, but not in the middle. The ‘negatives’ includes all kinds of transitions: hard cuts, crops, dissolves, wipes and fade in/outs.

Regarding to the hard cuts and crops, it is obviously a ‘negative’ case if the differ (the distance between the boundary and the middle position) is equal to or more than one frame. Fig. 3.10 and Fig. 3.11 show us some examples of negative hard cuts and crops.



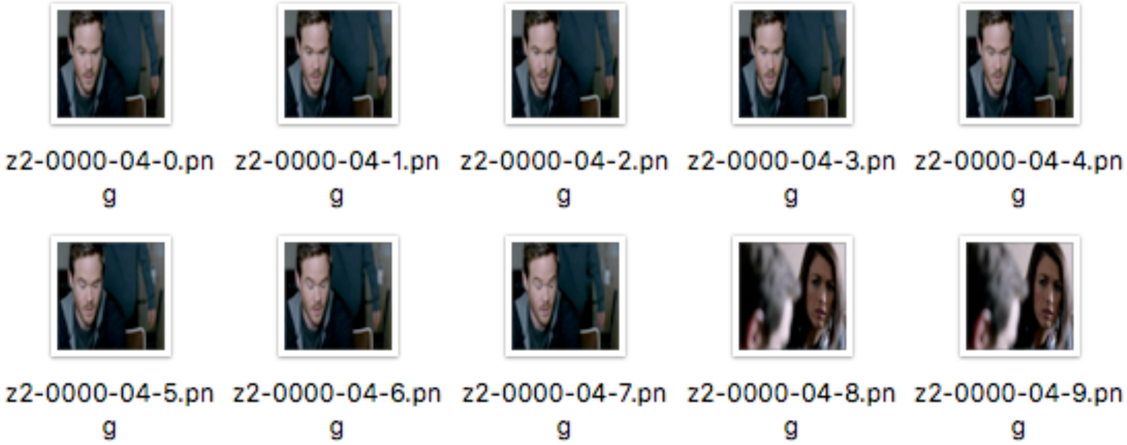


Fig. 3.10 Generated 'negative' hard cuts

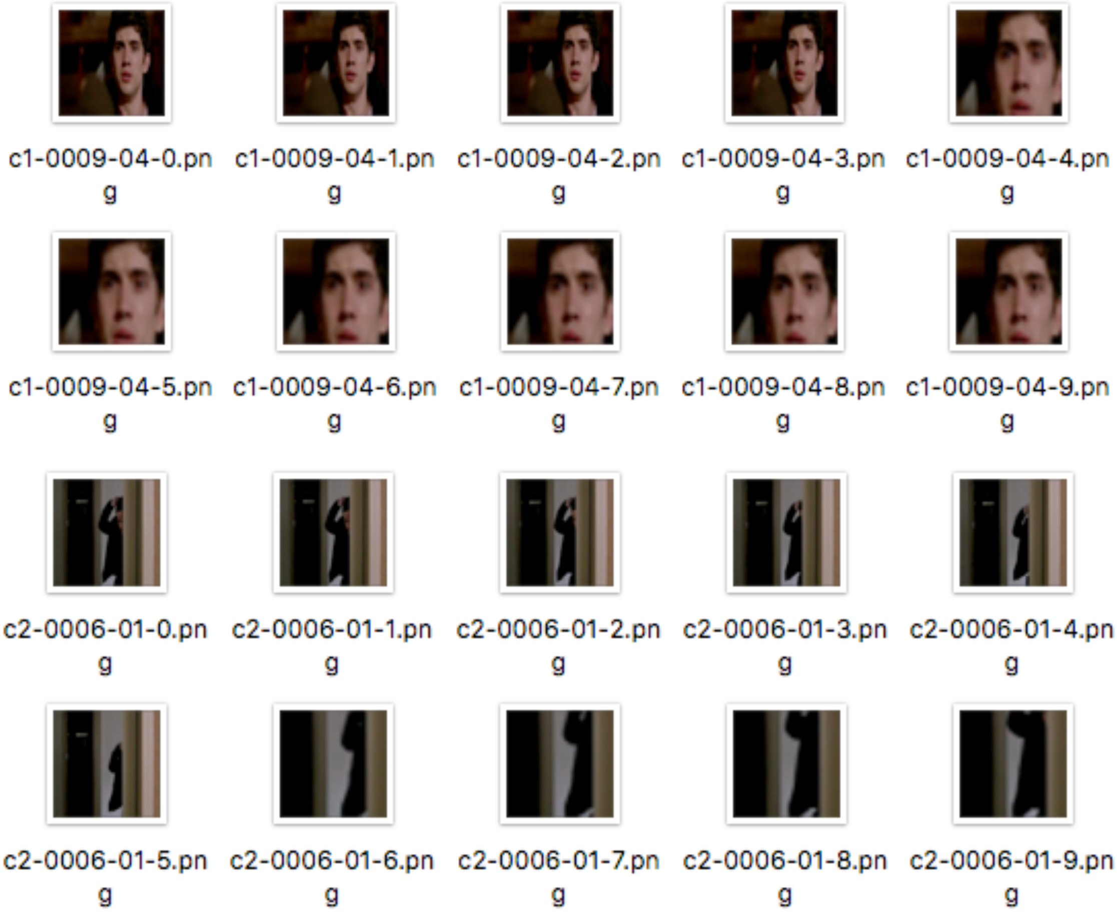


Fig. 3.11 Generated 'negative' crop transitions

And for the gradual transitions, it’s even difficult for human eyes to judge the boundary position, so the threshold for these kinds of transition is larger. In our case, we’ve made ‘negatives’ with the boundary position difference is equal to or more than three frames. Fig. 3.12.a, b, c, d show the negative wipes, dissolves, fade-ins and fade-outs respectively.

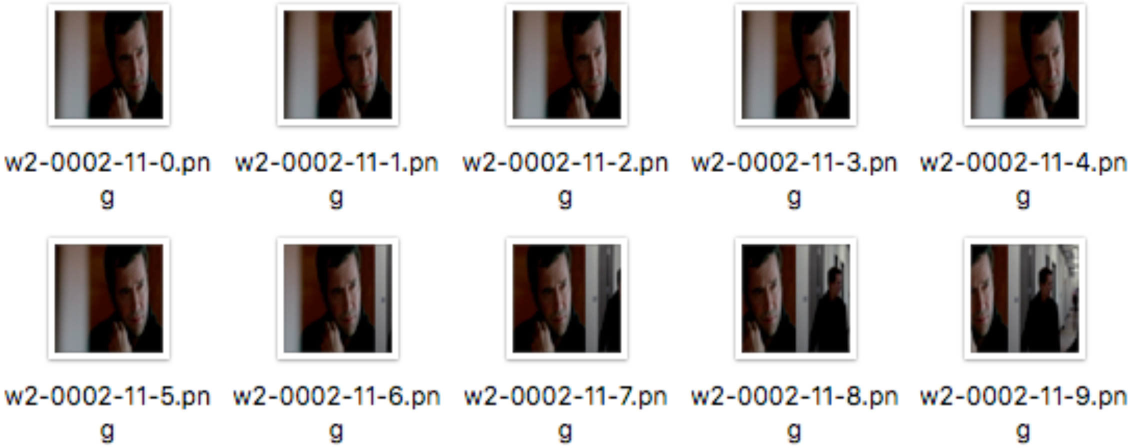


Fig. 3.12.a Generated 'negative' wipe transition

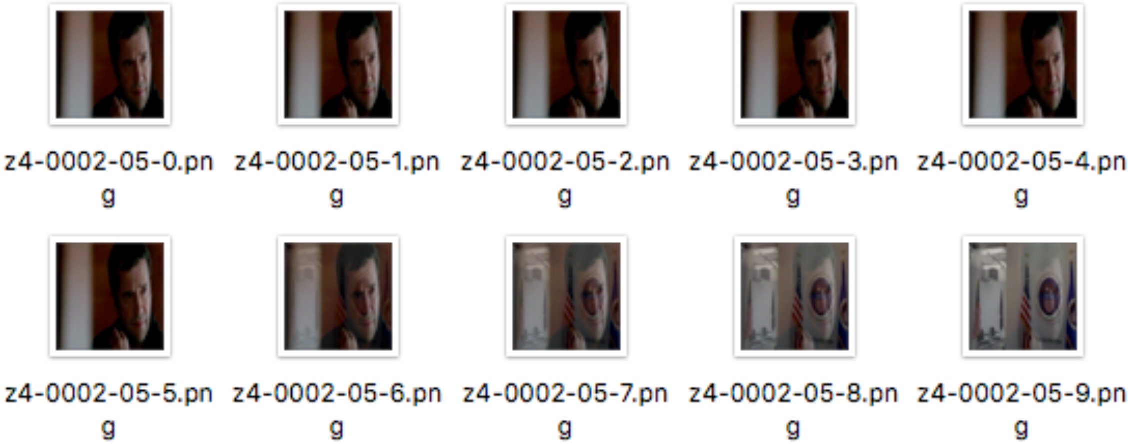


Fig. 3.12.b Generated 'negative' dissolve transition

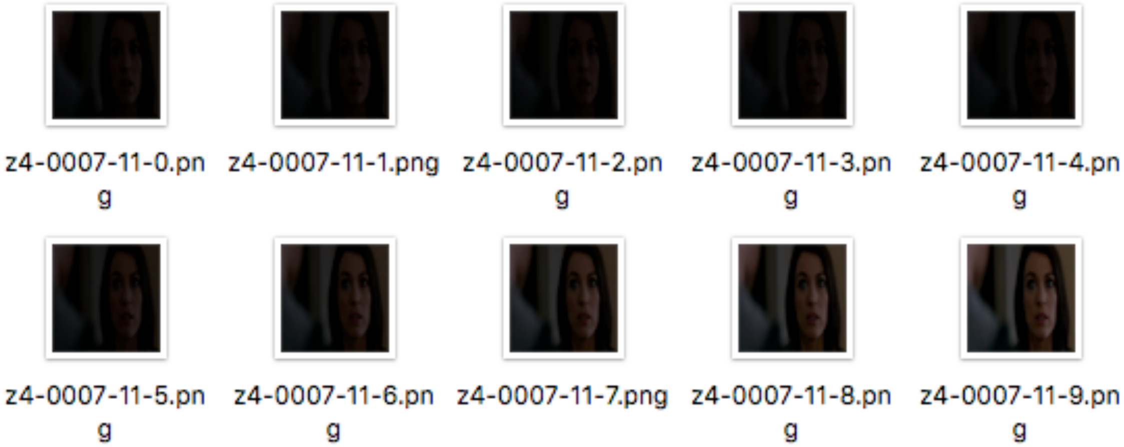


Fig. 3.12.c Generated 'negative' fade-in transition

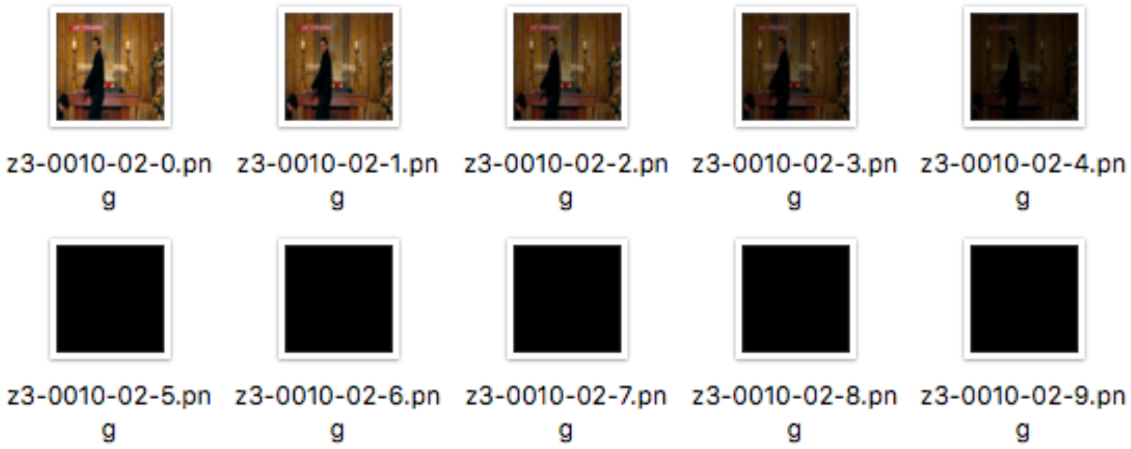


Fig. 3.12.d Generated 'negative' fade-out transition

3.3. Conclusion of dataset generation

In total we have 1,602,349 snippets of augmented data and these snippets have two types: the first is ‘no transition’, which are snippets consisting of frames from a single shot or exist a shot boundary not in the middle; and the second is transition snippets which have a transition from one shot to another, with the boundary in the middle. We made each of our batch with 32 snippets, thus whose batch shape should be (32 * 10 * 64 * 64 * 3).

In order to effectively train our network, each of our batch contains 16 snippets with transition and 16 snippets without transition, that is half of them. We assume that the network will do a better learning by providing a balanced indication. Also we let the labels in one batch is the form: [1, 0, 1, 0…...1, 0], with balanced ‘1’ and ‘0’.

In one batch, we also defined the occupancy of different kinds of transitions, 50% of transitions are hard cuts, which is near to the real case. And dissolves, wipes, fade-ins, fade-outs, crops occupy approximately 10% each. That is to say, in a batch of 32 snippets, 16 of them are ‘0’s and 16 of them are ‘1’s, in which 8 of ‘1’s are hard cuts and the other 8 are dissolves, wipes, fade-ins, fade-outs and crops. In our case, specifically we had 2 snippets of dissolve transition, 2 wipes, 2 crops, 1 fade-in and 1 fade-out in one training batch.

For the test dataset generated, we divided the types of transitions in order to test them individually to measure how good the network in detecting one type or another. In total we have 2127 snippets of data for testing in the generated part.

Thus, in our whole dataset, approximately we have 48500 batches of inputs for training and 2500 batches for testing.

In the next chapter we are going to talk about the results of the network training, with the original real dataset and augmented dataset separately and its analysis.

4. Network training and analysis of results

4.1. Training with extracted dataset

The first training of the network we've used the dataset extracted from American series in real case. We ran it in a laptop, averagely it would take five hours if we run 1000 iterations, which is really slow but we just wanted to test if there is something wrong in it.

To fasten up the speed, we used an Azure Cloud Service Machine with a K80 as GPU and a Xeon E5-2673 v4 as the CPU. After configuring everything inside, we can reach an approximate speed 0.03h/1000 iterations, which is about 150 times faster than a 1.8GHz dual-core Intel Core i5 processor.

In our project, to make it easier to understand, debug, and optimize TensorFlow programs, we used Tensorboard a lot as a tool to visualize our neuron network as well as the accuracy and the loss function. Fig. 4.1 illustrated our network structure, visualized by Tensorboard.

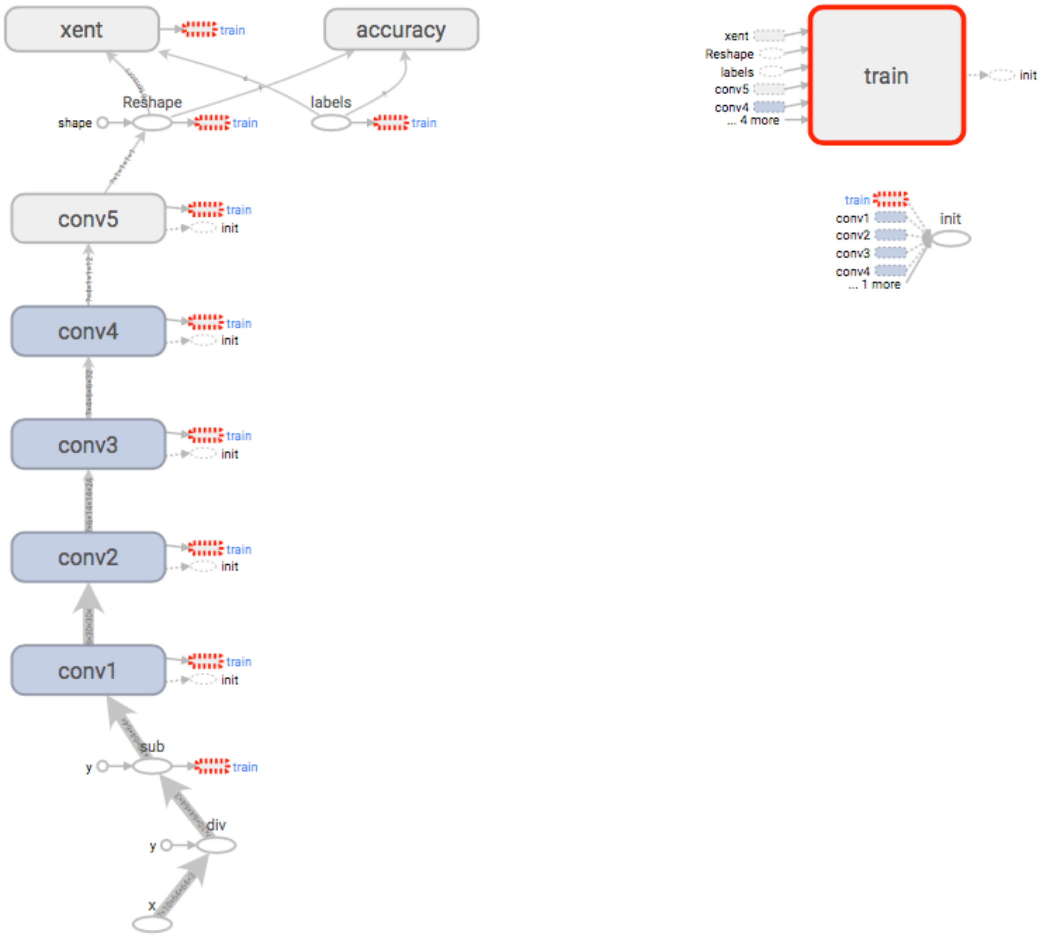


Fig. 4.1 Tensorboard visualization of our network structure

With the input image x, one important step is the normalization. By using the formula (4.1), we kept the image pixel value within [-1,1], which is better for the network to train.

$$y = x / 127 - 1, \text{ where x refers to input images} \quad \textbf{(4.1)}$$

And then we put normalised data into this five-convolutional-layer's network for training. By reshaping the output, we can calculate the loss and accuracy by comparing the predicted label and the real one.

With the dataset taken from the American series, after one epoch, we can get the accuracy 92.47%. With training for more than 50 epochs, we visualized the accuracy and loss using Tensorboard, shown below in Fig. 4.2 & 4.3. They show the accuracy and loss of our network in training progress.

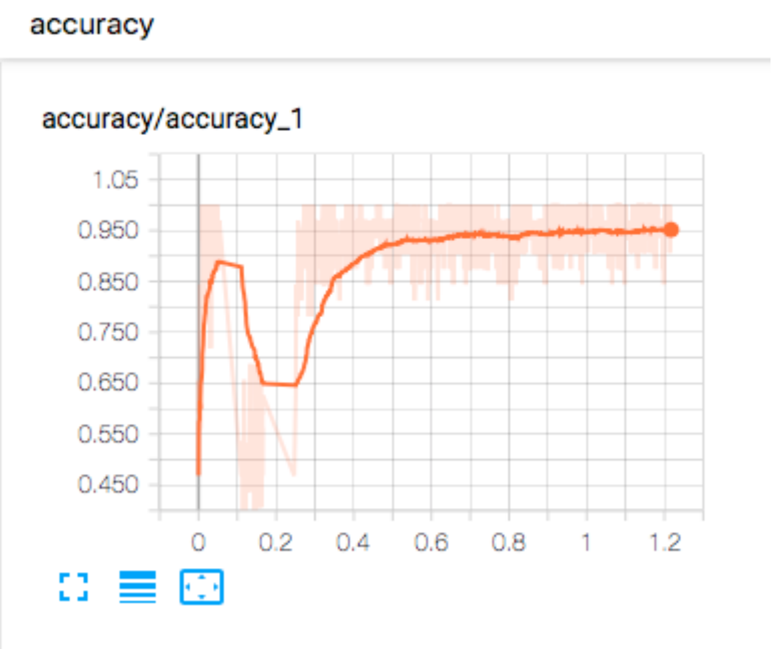


Fig. 4.2 Training accuracy curve

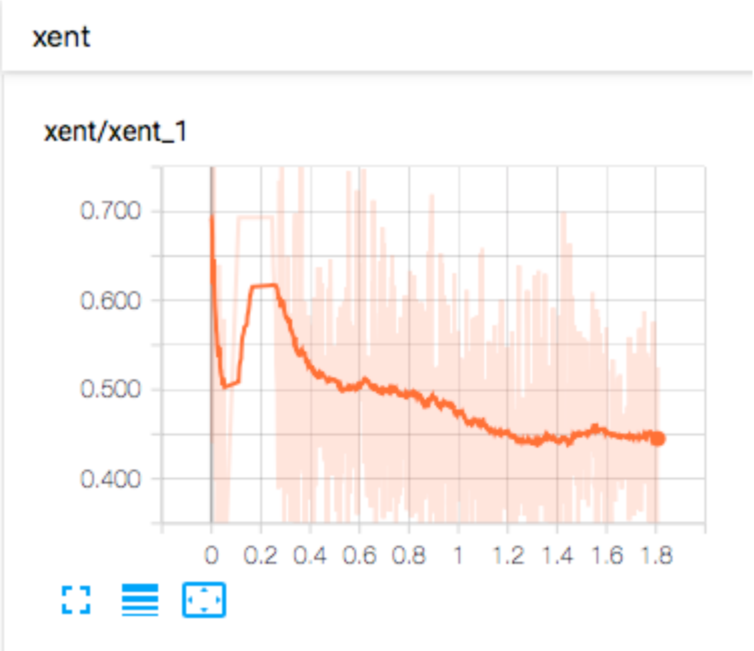


Fig. 4.3 Training cost function curve

We can see in the middle there is an area where the accuracy drops a lot and loss increases. Because we've shuffled the data, so in most cases of training it didn't appear. In this case, we suppose that the reason maybe the influence of unstable dataset, such as introduction in the series, which influence a lot the judgement of the network. In the end, for the training set, the accuracy can reach 95%, also the loss is decreasing continuously, means that we are reaching the optimal situation.

Fig. 4.4 & 4.5 below show the accuracy and the loss of test dataset after each epoch of training. Here the ‘X’ axis refers to the number of steps (epochs) and ‘Y’ refers to accuracy and cost value respectively. So, these figures show mainly the change tendency of accuracy and cost value according to the training steps.

With more epochs are trained, the testing accuracy doesn’t have obvious increase, on the contrary, it goes down a little bit to 92.5%. The reason may be the insufficient dataset and over-fitting. Also, the loss begins to increase after several epochs of dropping, which is normal for existing over-fitting, and this, is one of the reasons to make the accuracy couldn’t be improved.

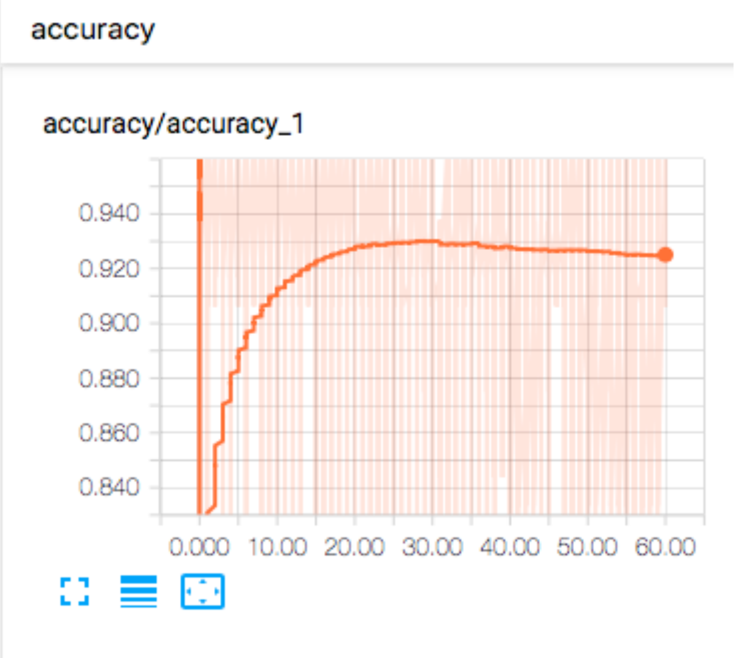


Fig. 4.4 Testing accuracy curve

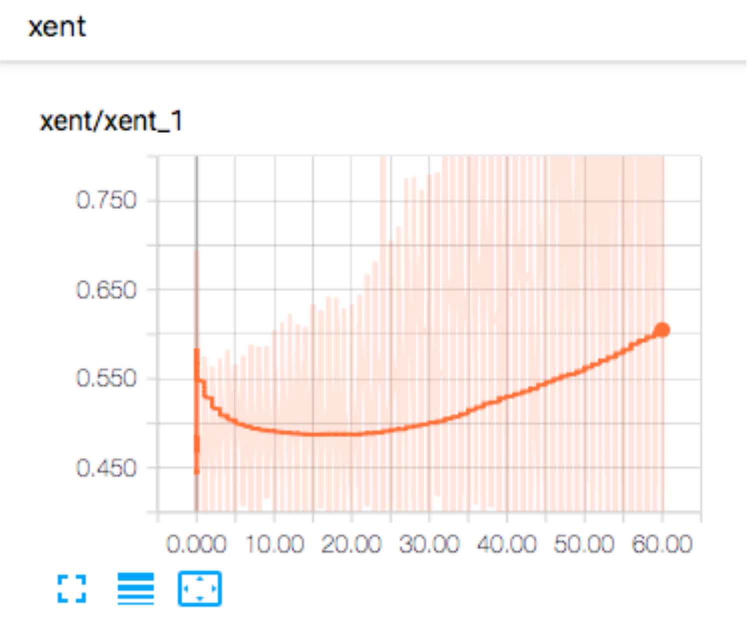


Fig. 4.5 Testing cost function curve

In the original testing set, we didn’t divide the types of transitions, so we didn’t know how good our network facing different transitions. The only thing we are sure about is that the most errors occur in the part of introduction, because the network is really in a dilemma when facing the unstable brought by image flutter. Also, for gradual transition, it’s hard to define if the boundary is exactly in the middle position, which is another dilemma. The last but not least, as we talked about before, we judged them as a cut when there is a part of the image changes, this may influence the judgement of neuron network in the future.

4.2. Training with augmented dataset

To improve the network performance, we trained the network with the augmented dataset. Facing the brand new dataset, the key task is to test the hyperparameters, such as the number of iterations in one epoch, the learning rate, the number of neurons in one layer, etc. After several times of testing, in the end we've decided to train 6000 iterations each epoch with the initial learning rate 0.001, later apply a cosine_decay. For the number of neurons in each layer, we took the same stated in paper [1].

After training for more than 30 epochs, we've seen that the printed accuracy remained the same with the previous, so we stopped it and visualized them using Tensorboard. Fig. 4.6 & 4.7 show us the accuracy and loss function in training progress. What we can see in the figure is that the accuracy in the end remains between 90% to 95% and the cost is decreasing as a whole.

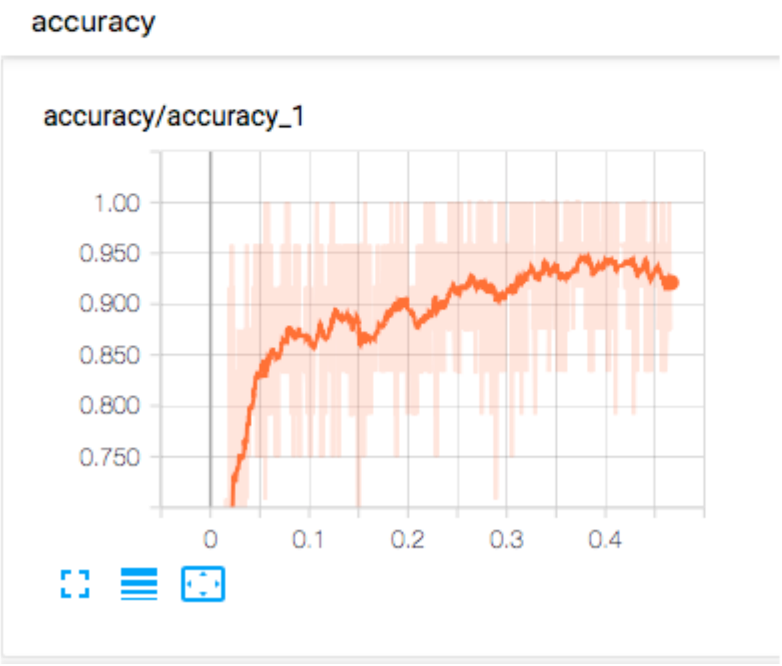


Fig. 4.6 Training accuracy curve

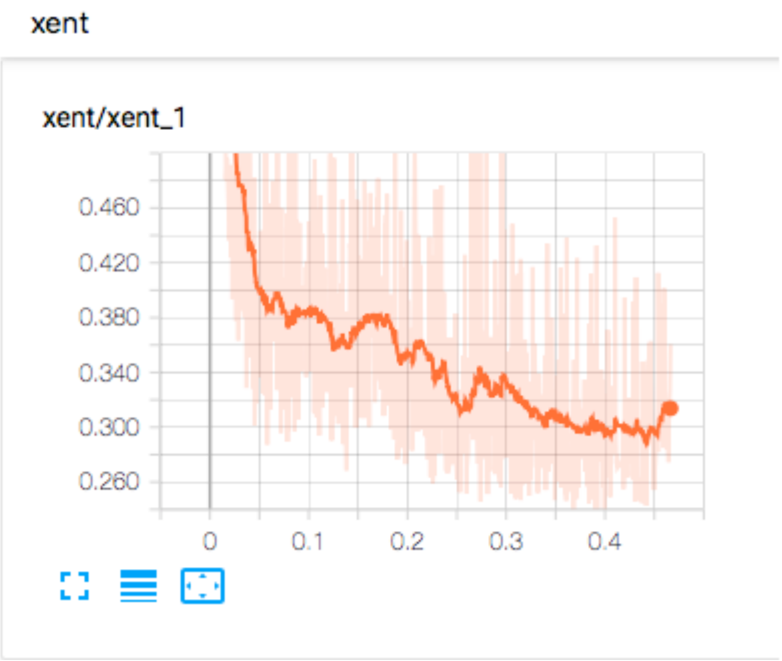


Fig. 4.7 Training cost function curve

4.2.1 Overall testing on the test dataset

For the overall testing, we've tested the evaluate dataset all together without separate the types of transitions. Fig. 4.8 & 4.9 show us the accuracy and the cost function of the whole dataset.

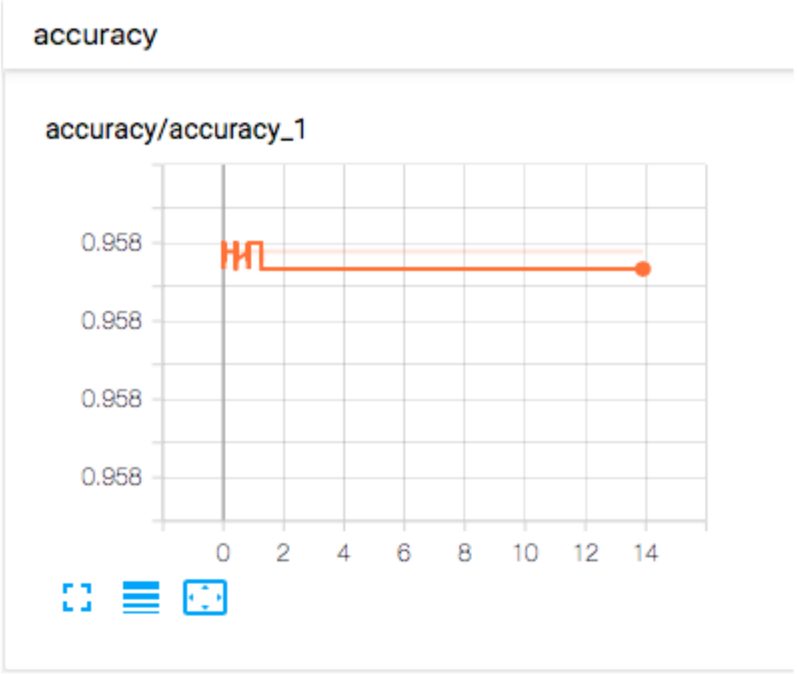


Fig. 4.8 Testing accuracy curve

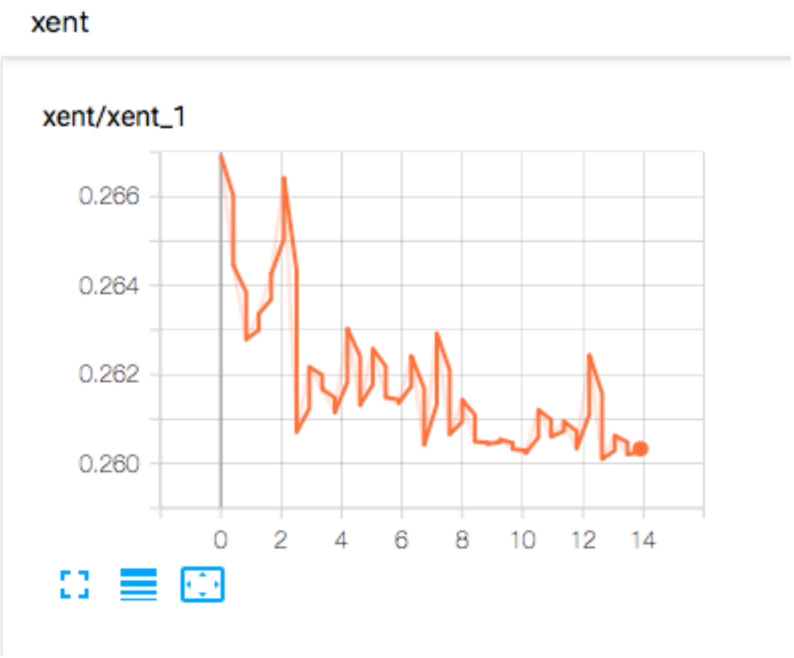


Fig. 4.9 Testing cost function curve

We can see that the accuracy remains the same in after several epochs, which is 95.8%, and the scale value on the axis are all 0.958, maybe for the reason that the accuracy hasn't change. Nevertheless, the cost function keeps going down. From the point of view of the accuracy, it seems that the network hasn't learnt anything new in the end, but actually we are reaching the optimal solution little by little.

4.2.2 Testing separately of different transitions

From here we are going to test different types of transitions individually in order to analyse how good works our network facing a specific transition.

- Hard cuts

Fig. 4.10 & 4.11 are the visualised accuracy and cost function tested with hard cuts. We can say that our network can detect hard cuts accurately with 100%. Here the number of 'X' axis changes, which not refers to the number of epochs any more with the reason that we smooth the figure in Tensorboard to make it shows clearer. But it's necessary to say that for the individual testing, we only shown the value with 10 epochs, with that we can already see the change tendency.

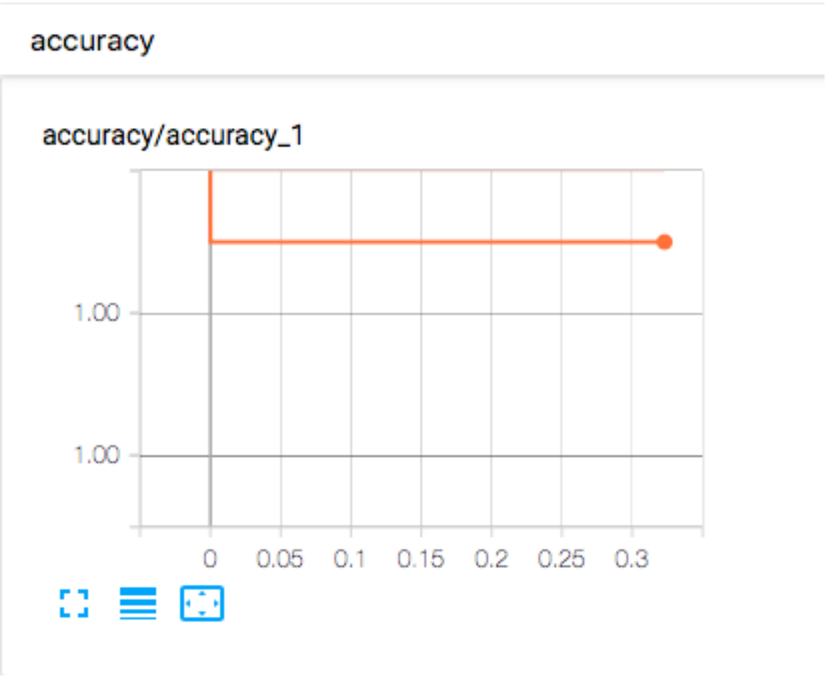


Fig. 4.10 Testing hard cuts accuracy curve

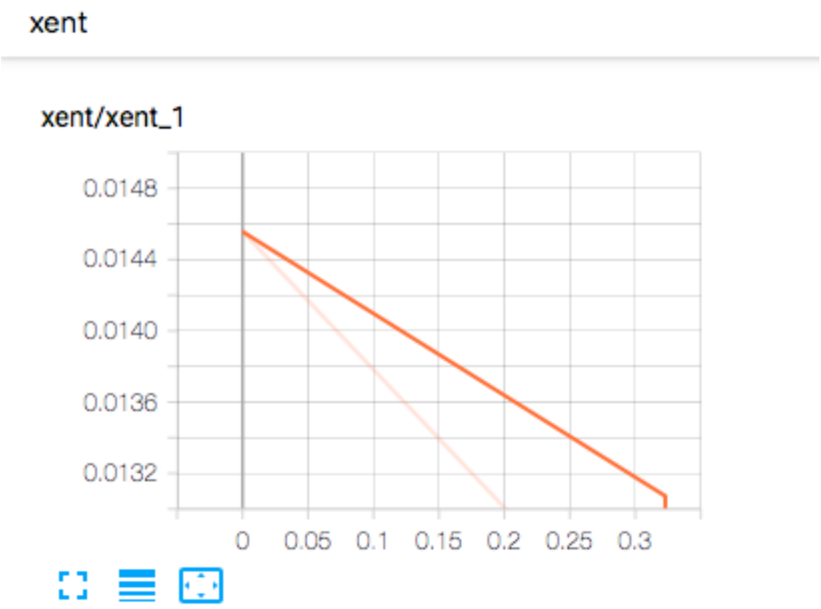


Fig. 4.11 Testing hard cuts cost function curve

The loss function in the end decreases to 0.013, which means that we nearly don't make any error in predicting that if it is a boundary or not.

- Wipe transition

Fig. 4.12 & 4.13 show the accuracy and loss of the wipe transition. Which is similar to detecting hard cuts, our network also works great on wipe transition with the accuracy 100% in almost all the cases. The loss is almost 0.00 in the end.

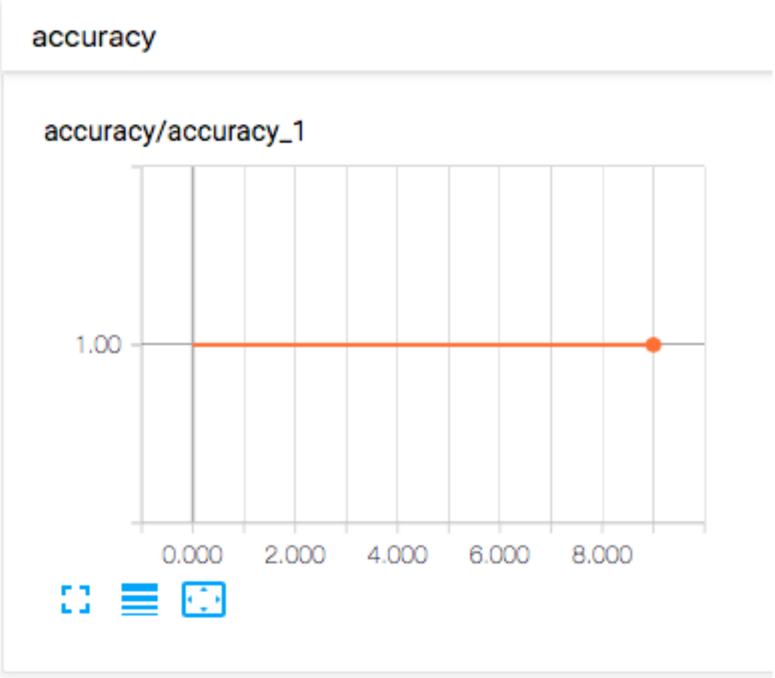


Fig. 4.12 Testing wipes accuracy curve

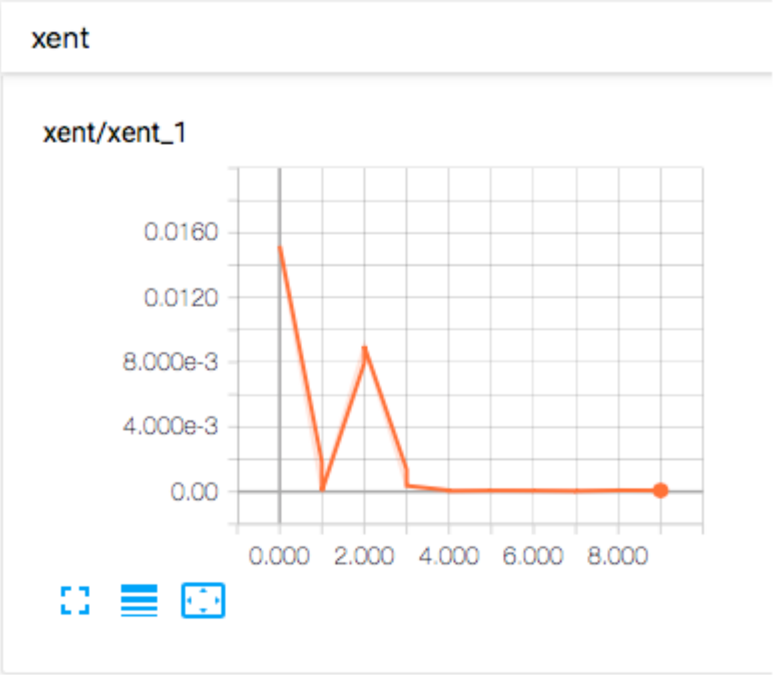


Fig. 4.13 Testing wipes cost function curve

- Dissolve transition

Fig. 4.14 & 4.15 show the testing of dissolve transitions. We can see the accuracy for testing dissolves is increasing up to 90.6% in the 10 epochs and the loss keeps decreasing overtime, however, it's still 10 times larger than the loss of the testing with hard cuts.

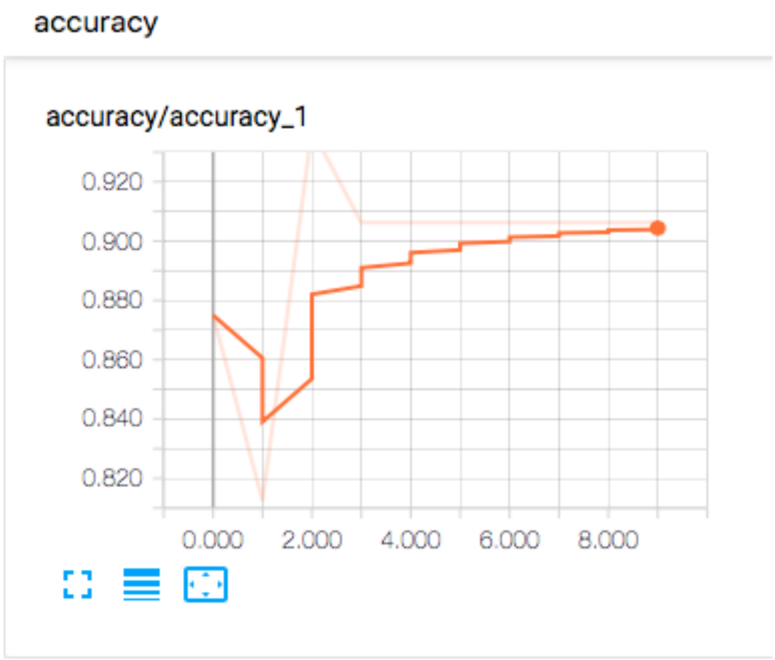


Fig. 4.14 Testing dissolves accuracy curve

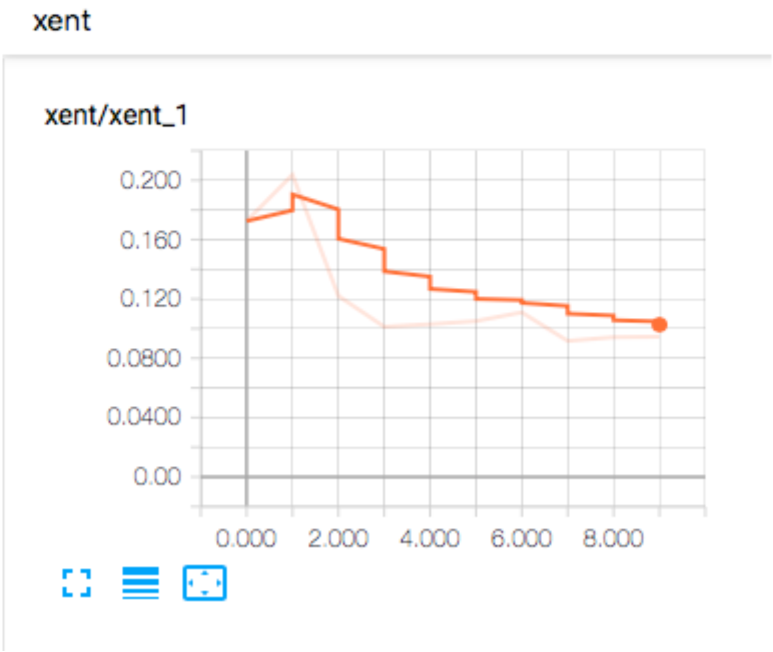


Fig. 4.15 Testing dissolves cost function curve

It makes sense that for dissolves, the accuracy is not so high as testing hard cuts, and the loss is larger. Because for dissolve transition, including fade in and fade out, they are linear interpolation between two different images, it's a gradual progress without an obvious or a sudden change. Thus, after a large amount of training, our network can detect that there exists a transition, but it's confused to judge whether the transition is just in the middle. It may appear in the 5th frame also the 6th frame, or even the 7th. Even our human eyes cannot estimate precisely where should be the boundary. So, for our network, in the end it can get more than 90% accuracy in testing, it's a satisfactory result.

- Crop transition

For the crop transition, we've got a moderate result. Fig. 4.16 & 4.17 show us the accuracy and loss function of crop test dataset. We can see that maybe sometimes it can reach 100% accuracy in some testing, but it maintains 96.9% in overall accuracy. In the generation of crop transitions, we made it crop up to 70% of the whole image, which will contain the main part of the image. If the crop area is just in the middle, or this area have all the contents that the network focuses on, it may be judged as a continuous frame with the previous one.

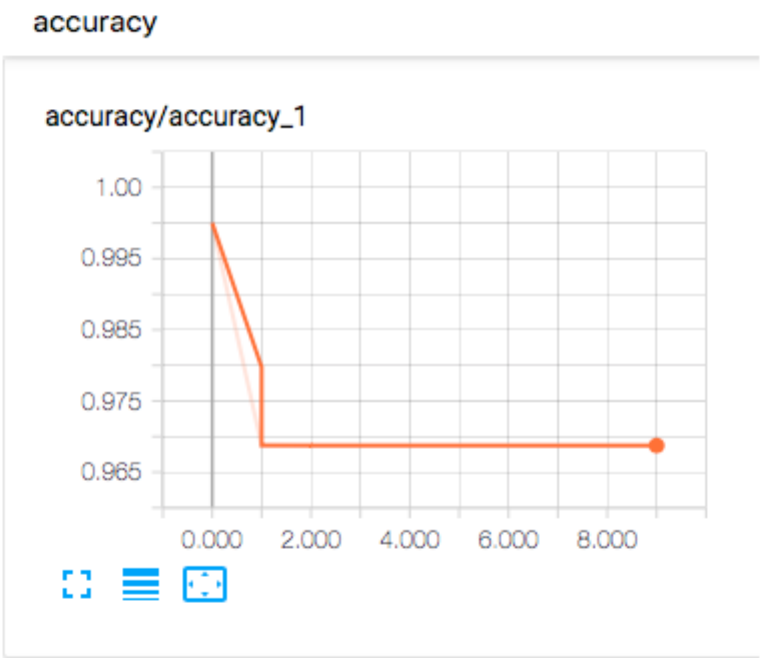


Fig. 4.16 Testing crops accuracy curve

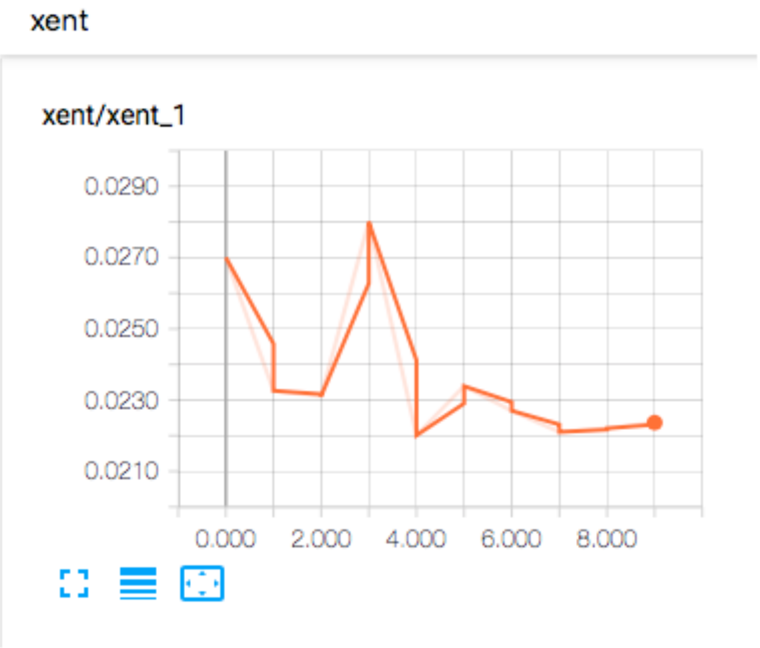


Fig. 4.17 Testing crops cost function curve

And for the loss function, although there is some area that the loss increases, it's still below 0.028. Even though it cannot be that good as detecting hard cuts, our network is quite effective in detecting crops.

- Fade in

Fig. 4.18 & 4.19 show us the testing of fade in transitions. The test accuracy, after a period of decline in the middle, it finally reached 96.8%. In this period, there may be a lot of dark frames in between, and it is not obvious when transitioning from black to real frame, which is easy to be mistaken as 'no boundary'. Another point, as we mentioned in the dissolve transition, is that it is difficult for the neuron network to determine if the boundary is in the middle, even though it has learned many examples. For fade in transition, if the incoming frame is originally dark, it's more confusing for the network to locate the boundary position. So in this area, accuracy has dropped a lot, and the value of loss has risen a lot.

But after more rounds of learning, the state of the neural network has gradually stabilized, the accuracy can reach 96.8%, and the loss is also declining. Overall, it has got very good test results.

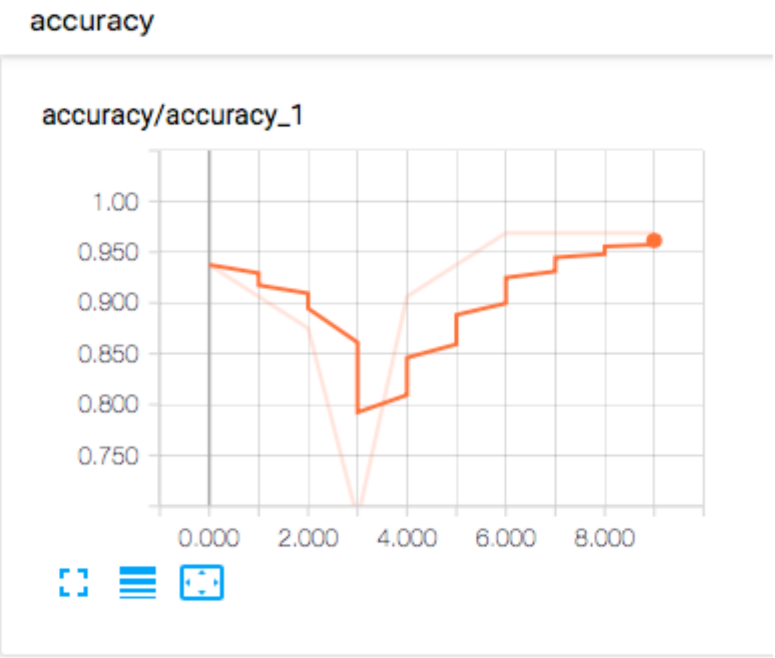


Fig. 4.18 Testing fade-ins accuracy curve

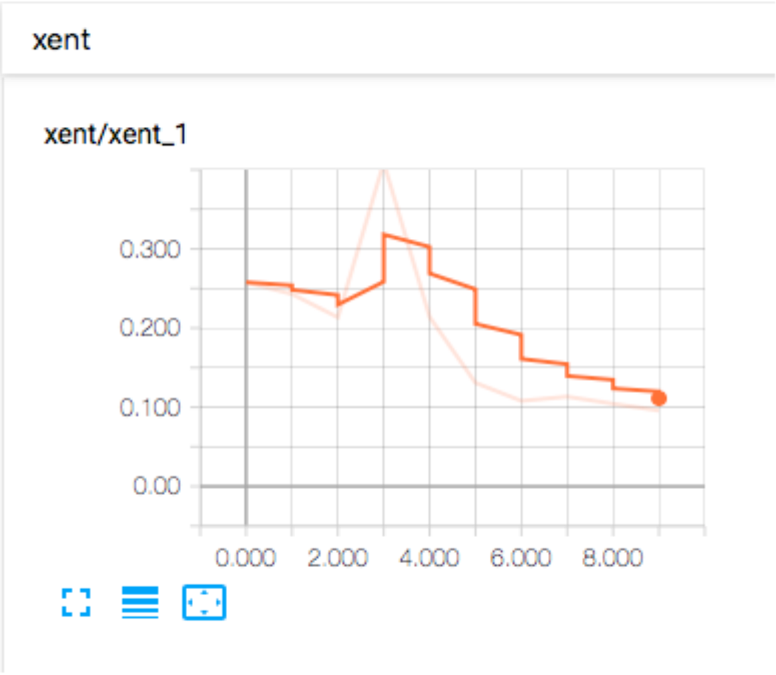


Fig. 4.19 Testing fade-ins cost function curve

- Fade out

The case of fade out transition is much more stable than that of fade in, shown in Fig. 4.20 & 4.21. Basically, it has reached a stable accuracy since epoch 2, accompanied by a gradual decline in the loss value. Although the decline is slower or even stay stable at the end, it is in line with the neuron network learning process, that is, the closer to the optimal value, the slower the parameter update. And the final test accuracy of fade out can reach 96.8%.

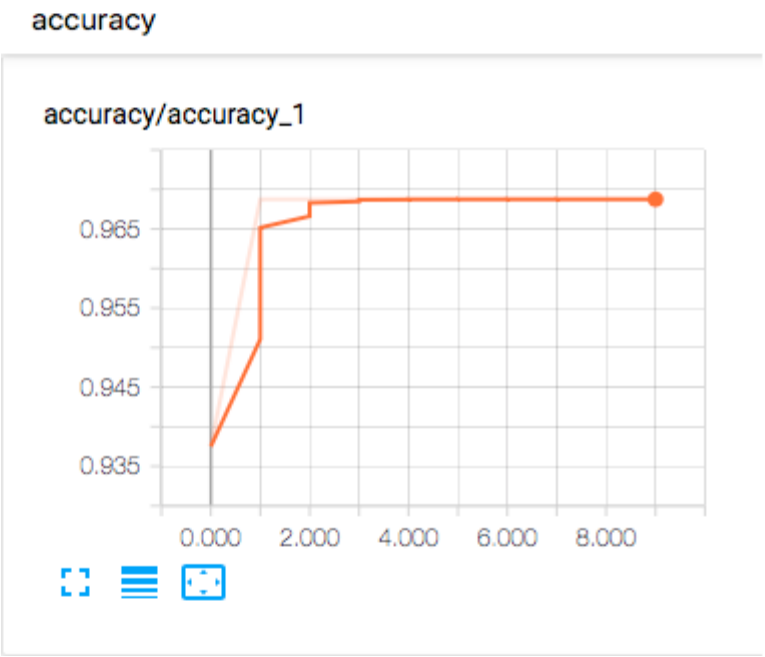


Fig. 4.20 Testing fade-outs accuracy curve

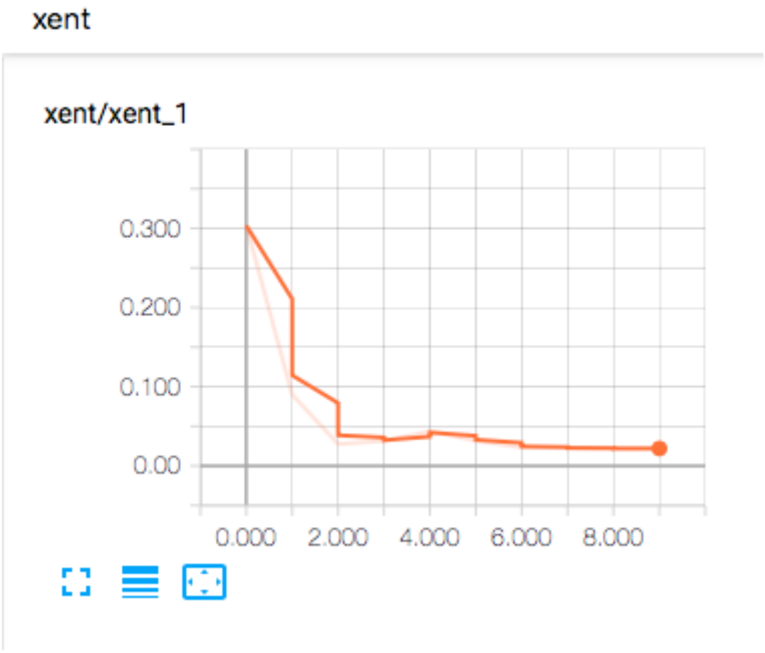


Fig. 4.21 Testing fade-outs cost function curve

- Negative transitions

The dataset of the ‘negatives’ contains all kinds of transitions, but their boundaries are not in the middle, thus they are judged as ‘no transition’. We tested the ‘negative’ dataset separately to determine the neuron network's ability in analysing boundary locations.

Fig. 4.22 & 4.23 show the test of 'negative' dataset. It can be seen that in the 10 epochs that we tested, the state of the network is not very stable. At the beginning, the accuracy was only 84.3%. Later, after a period of training, it finally stabilized at around 93.7%.

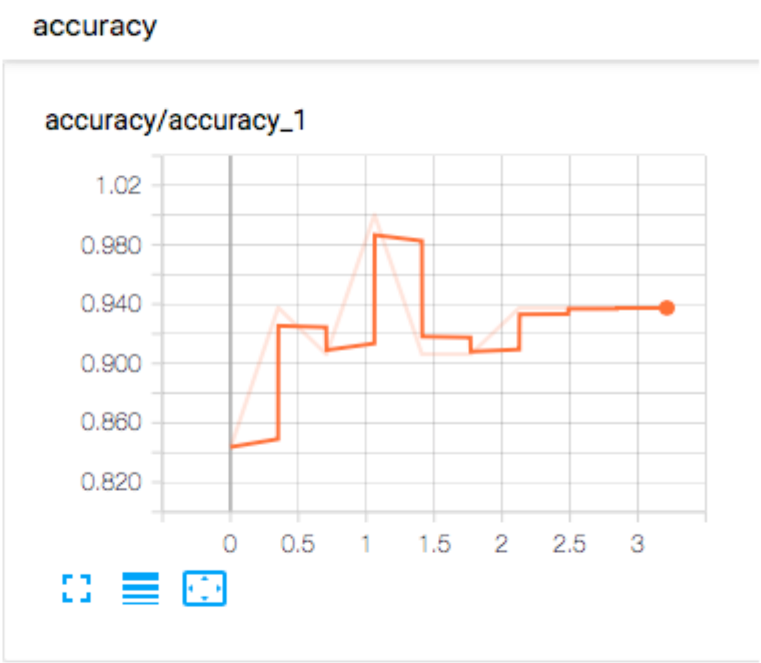


Fig. 4.22 Testing 'negatives' accuracy curve

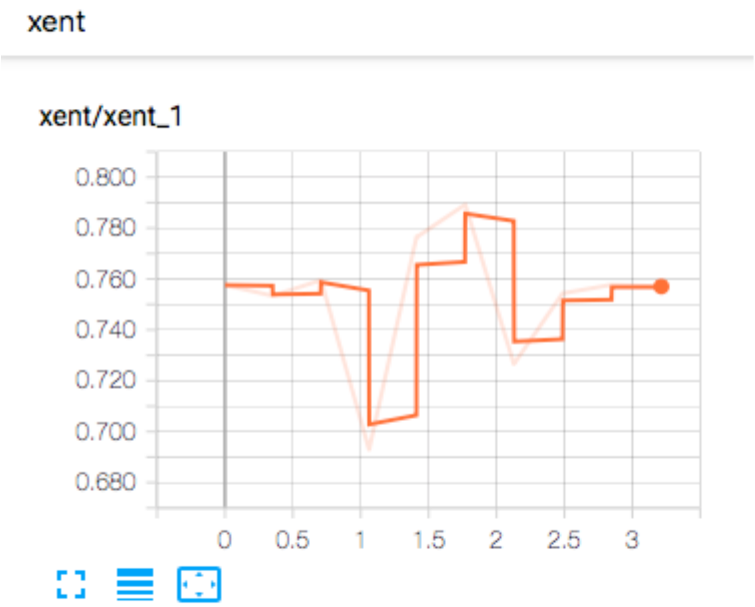


Fig. 4.23 Testing 'negatives' cost function curve

Fig. 4.23 shows that the testing loss of the ‘negative’ dataset is very high, probably because the number of epochs tested is too small and the loss value has not dropped to somewhere near the optimal. However, as it’s shown in Fig. 4.24, the loss during the training is stable at around 0.25 in the end.

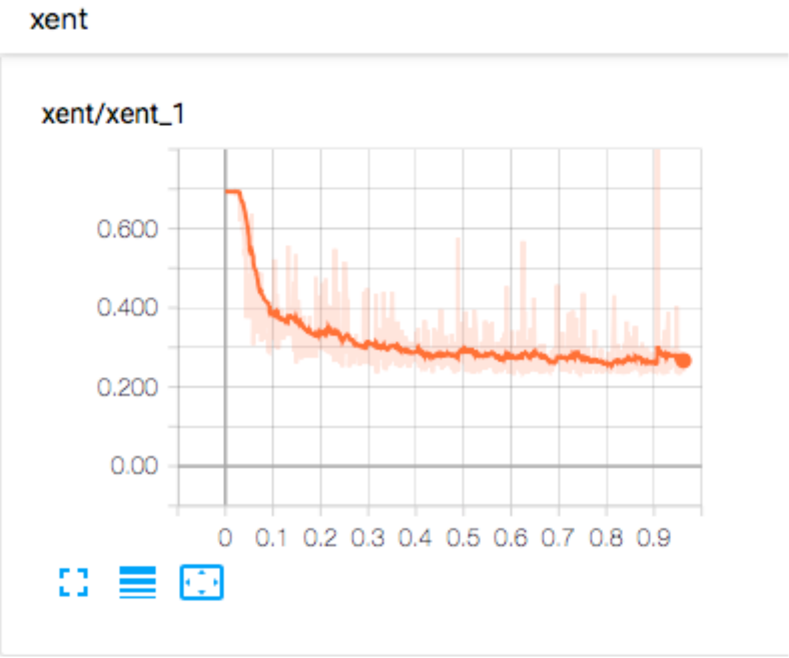


Fig. 4.24 Overall training cost function curve in 10 epochs

In general, for the test of ‘negative’ data, although the test accuracy is above 90%, it is still relatively lower than the test for the whole evaluate dataset.

> Along with the higher loss, it lowers the overall test accuracy, which indicates that the boundary position in the gradual transition is a very important but very problematic task for neuron networks.

In this chapter, we’ve analysed the training and the testing results of our neuron network. We could get a 94% accuracy in testing the extracted real case dataset. Later by training with the augmented dataset, we could get the 95.8% accuracy in testing augmented evaluate dataset.

By testing all types of transitions individually, we’ve seen that our network is very strong and sensitive to hard transitions, such as hard cuts, wipe transition as well as crop transitions. For gradual transitions, it's relatively weaker, especially for dissolve transition. As we’ve analysed before, dissolve is the linear interpolation between two frames, so, all the interpolated frames in the transition duration would be similar to each other, thus influence the network’s judgement.

Meanwhile, the testing of the ‘negative’ dataset also states that for gradual transition, it’s hard to define the boundary position, but we believe that at least our network performs very well in detecting the existence shot boundary. Nevertheless, the network works better in detecting fade-ins and fade-outs comparing to dissolve transitions. The reason may be the interpolation between the colourful frames and all black frames, which is much easier than figuring the interpolation of two colourful frames.

In the next chapter we are going to make a conclusion, about all the works we’ve done from the beginning, including the sustainability considerations and ethical considerations.

5. Conclusions

5.1. Project conclusion

This project is about shot boundary detection based on convolutional neural network, including the material reading in the previous period, the production of the data set and the construction of the overall structure of the neural network. The framework of the specific fully convolutional neural network is based on paper [1]. In paper [1], they tested with the RAI dataset with the final accuracy 88%, it’s not comparable since we didn’t test the same dataset. Overall, we have achieved the final testing accuracy around 95% with our test dataset. This is a relatively satisfactory result.

This is due to our careful handling of the training dataset in the early stage, which ensures the accuracy of the training. By analysing the inadequacies of the investigation in paper [1], we focused on the data types that they did not detect well, that is one of the reasons we achieved good results. At the same time, we have been continuously improving the parameters during the implementation of the project.

Referring to the choice of hyperparameters, such as the number of training iterations in each epoch, we still need a lot of testing to analyse. We've tested the number of iterations per epoch 5000, 6000, 8000, 10000 and 12000, in the end we chosen 6000 iteration/epoch. Also, for the initial learning rate, we chosen 0.001 for the final model after testing with 0.01 and 0.1. However, there are still many aspects for improvement, such as optimization of neural networks and methods to prevent over-fitting.

In the future research, we'd like to make improvement on detecting gradual transitions, as it is a troublesome case for shot boundary detection. We are also continuing finding a more efficient way to make the network learn, such as providing a clear-indicated dataset.

5.2. Knowledge and personal conclusions

This project gives me a chance to learn convolutional neural networks from scratch. Through the previous reading of the materials, I have got a general understanding of the working principle and basic structure of the neural network. Based on the understanding of it and according to the guidance of paper [1], I built the neural network of our project. At the same time, the production of datasets also plays an important role in this project, although it is boring and complicated, it is directly related to the learning quality of neural networks, that’s why I also take this part of the work very carefully and patiently.

It's hard to imagine that in just a few short months, I can learn and manage the convolutional neural network and apply it. I am very grateful to my instructors Francesc and Arnau for guiding me step by step in this project until completing it. Without their help, I will not achieve this satisfactory result.

5.3. Sustainability considerations

Our project is video processing based on convolutional neural network, which has important significance in the field of image processing. Moreover, the research of convolutional neural network is of great significance to the development of the whole society. It is a core part of artificial intelligence technology, its development, together with human understanding of brain cognition, play a mutually reinforcing role. Convolutional neural networks are used in many fields, which greatly promote the advancement of science and technology and the improvement of the efficiency of social mechanisms.

At the same time, in economic terms, it also has great potential. Artificial intelligence is gradually deepening into the normal life of human beings, and it has a huge economic benefit. It changes the value of our goods and the way we pay. We can say that the economic society promotes the development of science and technology.

However, it has some impact on environment and resources. This is a software-based project that runs on a cloud server. In addition to the use of the mobile hard disk in the previous production of the dataset, there is no hardware device used, so there is not much material resource consumption. In the project implementation process, it will occupy certain network resources, and the generated dataset will be rarely reused at the end of the project, so some resource garbage is generated.

5.4. Ethical considerations

Convolutional neural network is the core technology of artificial intelligence implementation. Its research is based on human understanding of brain cognition, with many unexplained aspects. However, due to limitations in data resources and hardware conditions, I believe that its progress will not transcend human wisdom. Due to the strong extensibility and the possibility of creating value of convolutional neural networks, more and more scientists explore it, resulting in many new, socially valuable goods or concepts which inevitably bring huge economic benefits.

ACRONYMS

CNN	Convolutional Neural Network
ConvNets.	Convolutional Networks
FCN	Fully Convolutional Neural Network
SBD	Shot Boundary Detection
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit

REFERENCES

[1] Michael Gygli. Ridiculously Fast Shot Boundary Detection with Fully Convolutional Neural Networks. arXiv:1705.08214v1 [cs.CV] 2017.

[2] A. Hassanien, M. Elgharib, A. Selim, M. Hefeeda, and W. Matusik. Large-scale, fast and accurate shot boundary detection through spatio-temporal convolutional neural networks. arXiv preprint arXiv:1705.03281, 2017.

[3] A. F. Smeaton, P. Over, and A. R. Doherty. Video shot boundary detection: Seven years of trecvid activity. CVIU,2010.

[4] A.Krizhevsky, I.Sutskever, and G.Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

[5] Nikhil Buduma, Nicholas Lacascio. Fundamentals of Deep Learning. June, 2017.

[6] Weiming Shen; Jianming Yong; Yun Yang (18 December 2008). Computer Supported Cooperative Work in Design IV: 11th International Conference, CSCWD 2007, Melbourne, Australia, April 26-28, 2007. Revised Selected Papers. Springer Science & Business Media. pp. 100–. ISBN 978-3-540-92718-1.

[7] Basic Technical Terms
https://en.wikipedia.org/wiki/Shot_transition_detection

[8] Vastness of the problem
https://en.wikipedia.org/wiki/Shot_transition_detection

[9] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In CVPR, 2015.

[10] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.

[11] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatio-temporal features with 3d convolutional networks. In ICCV, 2015.

[12] TensorFlow. https://www.tensorflow.org

[13] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. PAMI, 2008.

[14] 卷积神经网络 CNN（3）—— FCN(Fully Convolutional Networks)要点解释
https://blog.csdn.net/fate_fjh/article/details/53446630

[15] M. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In ECCV, 2014.

[16] Softmax function. https://en.wikipedia.org/wiki/Softmax_function

[17] 杨云, 杜飞. 《深度学习实战》清华大学出版社. ISBN 978-7-302-49102-6

[18] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the best multi-stage architecture for object recognition. 2009a.

[19] 神经网络中的激活函数（activation function）-Sigmoid, ReLu, TanHyperbolic(tanh), softmax, softplus. https://blog.csdn.net/qrlhl/article/details/60883604

[20] Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. Paper presented at the International Conference on Artificial Intelligence and Statistics.

[21] Ilya Loshchilov, Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. arXiv: 1609.03983v5 [cs.LG] 3 May 2017.

[22] Xavier Glorot and Yoshua Bengio: Understanding the difficulty of training deep feedforward neural networks. International conference on artificial intelligence and statistics. 2010.

Annex A

This annex provides some details of the dataset, that are the videos we used to extract the frames.

All the videos we used are American series, with the frame rate 23.976, according to that we extracted frames using ffmpeg through terminal.

In total we took 71 episodes from more than 20 different series. Many themes are included, such as, criminal (Bones), action (Prison Break), horror (America Horror Story) and feature story (Shameless), etc. Some of them have the darker background as keynote, especially criminal and horror type. And such amount of them are used in order to make the network sensible to darker frames.

All the videos used are in the folder (external hard drive):
>> /Volumes/TOSHIBA EXT/shotdataset/DATASET/input/done

All the frames extracted are in the folder of Azure Cloud Service Machine:
>> /mnt/shot_dataset/we

With all the annotations in the folder:
>> /mnt/shot_dataset/annotations

The network read the dataset in the fly, all the .csv files are in the folder of cloud machine:
>> /mnt/shot_dataset

Annex B

This annex gives the main folder to run the proposed algorithm over a different data base. All the codes are uploaded and conserved in the cloud machine.

>> /home/ugiat/src/data_loader.py: load the dataset to the network.

>> /home/ugiat/src/augmenteddata.py: includes several algorithms to generate augmented dataset from the extracted hard cuts.

>> /home/ugiat/src/takename.py: write the augmented data into .csv files.

>> /home/ugiat/src/CNN_arnau.py: together with data_loader.py to load batches into the network to train and to test the extracted dataset.

>> /home/ugiat/src/CNNk.py: together with data_loader.py to train and to test the augmented dataset, with several changes in order to load data in the fly, instead of previously making batches of data.