Assignment 3- FIT5221

Group 6

WANG MINGQIU：30988489

SUN WEI：30988500

SUN LI：30988519

Roles

Our group consists of three people, they are SUN LI, SUN WEI and WANG MINGQIU. The jobs we undertake are:

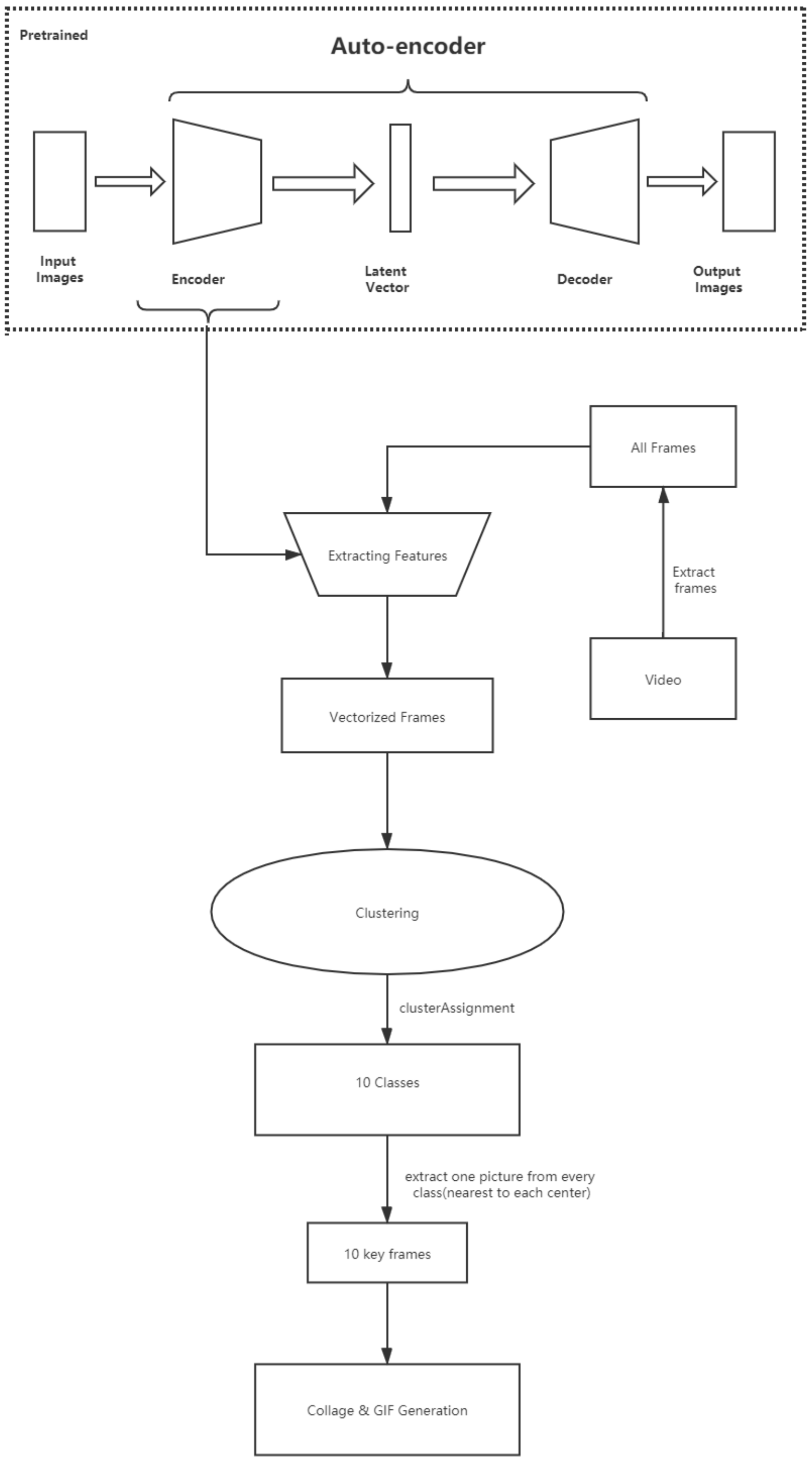
SUN WEI:	Auto-encoder construction, model training
SUN LI:	Clustering algorithm and feature extraction (frame encoding)
WANG MINGQIU:	Collage and gif generation, report writing

Reasons we abandon our original design

Our apology that we did not adhere to our original design to implement the system with a dual stream network architecture and supervised learning paradigm. There are three reasons that we changed our design to the current one. First, the original network architecture is complex and the training process are still not mastered by us. The dual stream network architecture requires two CNN networks to run parallel, one responsible for the extraction of the feature of the frame and the other responsible for the generation of the optical flow across a mini set of frames. The training process would require unbearable time and the fine tuning of this model could be laboring, increasing the infeasibility of this design, given our current situation that we need to construct this system within a few days. The second reason is that we are not able to find the labelled training data in time. The original design adopts a supervised learning paradigm, requiring massive dataset which are different from the usual dataset we see in other learning tasks. It is difficult to collect the necessary amount of dataset within a few days, making the original design highly not possible to be implemented in time. Third, the original design is not ripe enough to guide our actual working. In our delivered PPT, the original design merely describes the framework of our system in a crude way. There would remain lots of points to be discussed until we can work on establishing it. This is because in the original design, the dual stream network is an architecture not that familiarized by us. The lack of details and guidance could result in the failure of our system.
With these reasons, we abandon our original design and deliver a fresh one, more concise, more maintainable, and above all, more feasible considering that the time we can direct to this task is not as adequate as once we thought. Therefore, we formerly ask you, to approve our new design. Thank you for your understanding.
In the following sections, we would present our new design in detail.

System Architecture (SUN WEI)

Our new design system is comprised of a pretrained auto-encoder network, a cluster and a collage generator. The system flowchart lies below.



The pretrained component is an auto-encoder trained by the frames extracted from the video. In an effort to increase the training efficiency, the training frames are sampled from the original video. The video lasts around 3 half minutes, and we extract one frame from every 5 frames. **The purpose of this pretrained model is to compress the representation of these frames so that we can have vectorized features reflecting their similarities. The so-called feature extraction in our system is to encode the**

frames in vectors, and this network is responsible to learn a way to represent it to fulfill the ultimate goal of the system.

So long as the auto-encoder training is finished, the frames in our video is then encoded using this auto-encoder. After the feature vectors are generated and recorded, the system proceeds to the next phase, clustering. In the actual system delivered to your end, we would upload the pretrained model as part of our system. Since this model is trained, the system could use it as a real functioning encoder to encode the image for feature representation. It is pointed out that this pretrained model is part of the system.

The ultimate goal of our system is key frames selection. Therefore, we must divide these frames and analyze their relations. Clustering is here to assist this job. It is aimed at clustering these frames into 10 different clusters. The number of the cluster is the number of the key frames we wish to find. The cluster would group the frames with similar features, for instance, similar figures, scenes or background in the same group. Once the clustering is complete, we would have 10 clusters, each containing the frames distinguished from those in other clusters. The key frames are then selected from these 10 clusters, since each cluster would possess one category of frames similar in a way and different clusters would contain frames separated from a certain point of view.

In the last step of our processing, the key frames are assembled to form the collage.

Auto encoder (SUN WEI)

Purpose of auto encoder

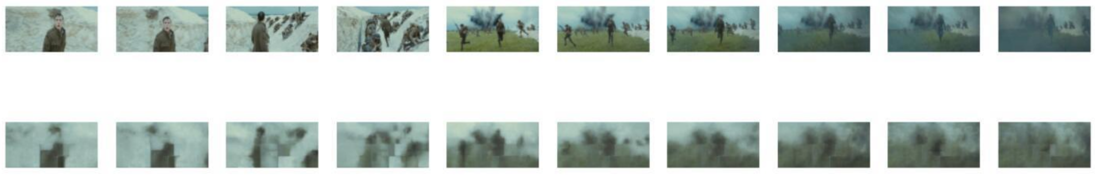
The auto encoder exploits the frames in the video to try to compress the image representation into vector-form, or, to learn the representation of the image in the video. Like Word2Vec, we try to use this network to complete an image2vec task here. We believe that the latent code generated by the encoder, if this encoder is properly trained, can reflect the relations between different frames, thus would serve a good representation of the frames and could be used for further processing. In the system architecture, we specifically marked that the pretrained model is part of this system. We are not going to train the model from scratch because this model is now indigenous to this video. If you wish to apply this system to other videos, then please re-train the model with the frames in that video.

Architecture of the auto-encoder

The encoder part consists of 12 layers, same as the decoder part. The whole network is comprised of 25 layers (12+12+1 extra input layer). The encoder part is basically a CNN network except for the last dense layer outputting a latent code. The decoder part does the reverse of the encoder part. Its job is to reconstruct the image from the latent code. The detailed architecture lies in the appendix.

Perfomance evaluation

To evaluate the performance of this auto-encoder, we list a few frames along with their counterparts of reconstruction.



From the images above, although the details are blurred, the general skeleton of the images are salient, thus this encoder manage to preserve the features of the images and would perform successful feature extraction during the image encoding.

Clustering (SUN LI)

Clustering

Function :clusterAssignment

Idea:

The function uses k-means to cluster.

The step is to pre-divide the data into K groups, then randomly select K objects as the initial cluster centers, then calculate the distance between each object and each seed cluster center, and assign each object to the nearest cluster center. The cluster centers and the objects assigned to them represent a cluster. Each time a sample is allocated, the cluster center of the cluster will be recalculated based on the existing objects in the cluster. This process will be repeated until a certain termination condition is met. The termination condition in our experiment is the times of iterations.

Implementation:

This function takes the features, number of centers and times of iteration as input, and returns centers and assignment result.

clusterAssignment_result is a 1×numSamples array
storing which cluster each sample belongs to.

centers_result is a k× FeatureDimension array storing each center.

Firstly initialize centers and then for each iteration, update the sample's belonged cluster and update centers.

In our test, I set k=10, times of iteration= 40.

Extracting key frames

Function: collect

Implementation:

This function takes the clusterAssignment_result as input and output 10 lists. Each list collects indexes of samples belonged to each cluster. For example: x1 contains the index:i of clusterAssignment_result where *clusterAssignment_result[i]*==0.0

Function: extract

Idea:

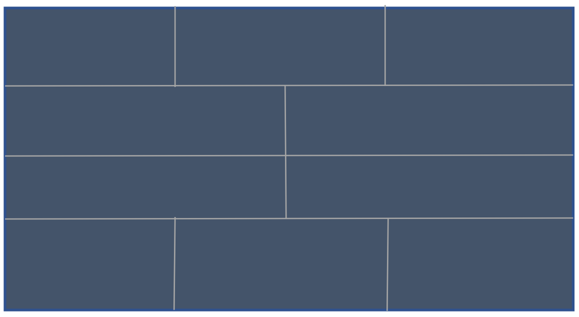
For each cluster extract the frame closest to the center point as the key frame, then collect indexes of 10 key frames into a list. The Euclidean distance is used to calculate the distance.

Implementation:
This function takes the clusterAssignment_result, centers and feature as input , then outputs a list containing 10 indexes of key frames for the following task.
In the function, it uses the results of function collect. For each list from function collect I create an array to store the distance between cluster center and samples assigned to the cluster. Then I use np.argmin to get the key frame which has the minimum distance. After iteration of each list, I create a list:x to collect all the extracted frames.

CollageGenerate method description (WANG MINGQIU)

`generateCoolSummary_10(image_list)`

The *generateCoolSummary_10()* methods takes in 10 frames and assemble them in a way like a post. For this method to function properly, three helper functions are defined. The wrapper function *generateCoolSummary()* is the main function which brings every step of our system together.
generateCoolSummary_10() would create the collage in the following layout.



In the following section, three helper functions would be introduced.

`merge_pixel(imgf1, imgf2, alpha=1, axis='ver')`

This method is used to generate the overlapping part of two frames. According to the formula provided in the assignment instruction, the generated pixel is actually the weighted sum of the corresponding pixels in the two input frames. In this method, the alpha or the weight is decayed along the distance. The closer the pixel is to the frame1(imgf1), the more weight inclined to frame1, as is the same with what could be applied to frame2(imgf2). The alpha in the function signature represents the upper bound of the alpha that would be used in the function.

`concatImg_hor(img1, img2, novlap=100, alpha=1)`

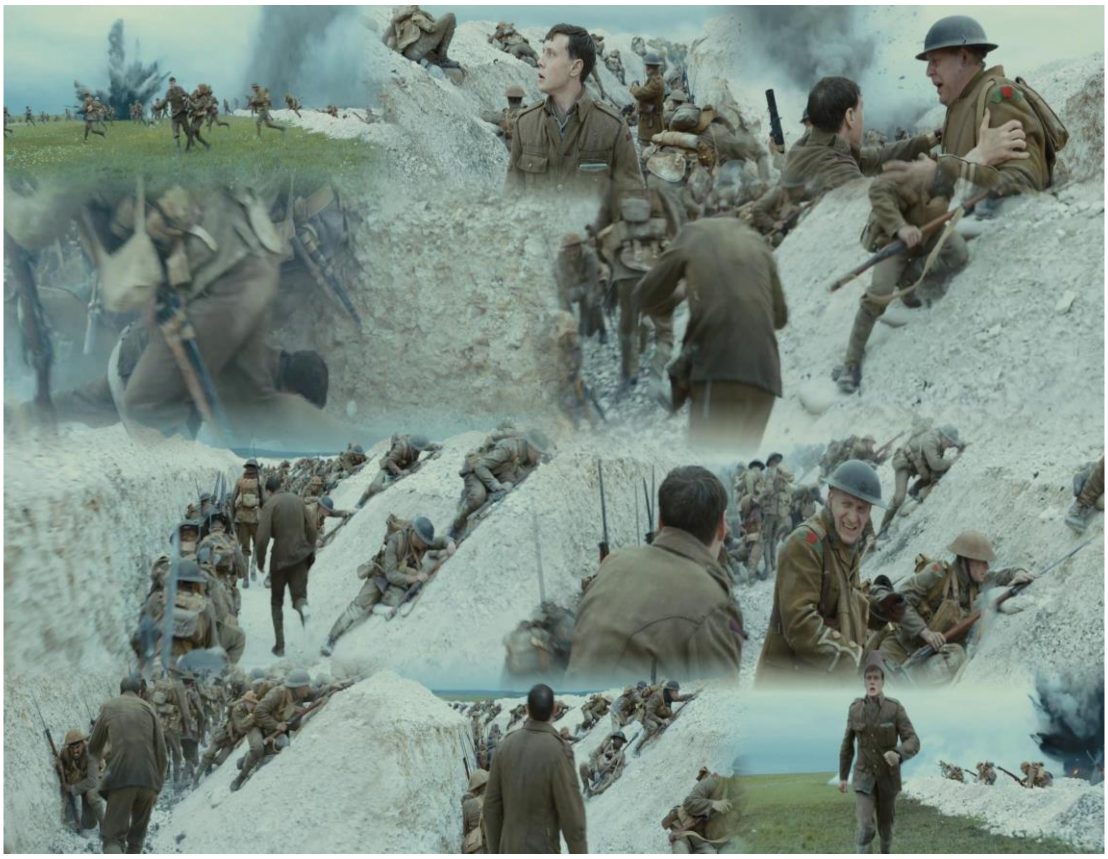
This method is used to concatenate two frames horizontally. The alpha parameter functions as the same in the last method. This novlap parameter is used to control the intersection portion of the two frames. For instance, here we set novlap to be 100, this

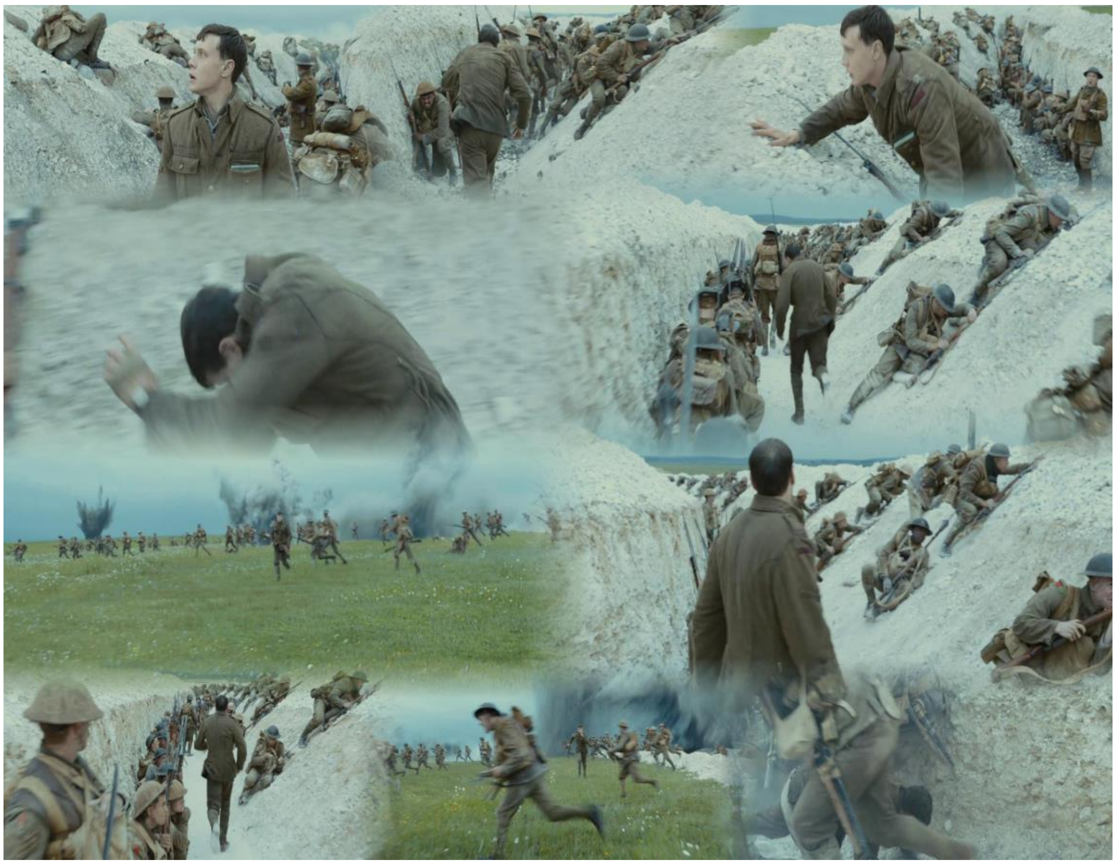
means there would be 100 pixels width area of two frames to be overlapped and the overlapped part would be calculated by the function *merge_pixel*.
`concatImg_ver(img1, img2, novlap=80, alpha=1)`
This method functions similar to the *concatImg_hor* in a vertical way. In other words, it would concatenate two frames vertically with the overlapping part to be integrate by *merge_pixel*. The parameter is in the same way as *concatImg_hor* to control the behavior of the function.

For GIF creation, we call the method
`imageio.mimsave(save_name, imagesets, 'GIF', duration=duration)`
to save our key frames and assemble the GIF. This function is part of the package imageio.

According to our program, this method would save the gif under the root directory of the i python notebook. So does the output summary collage.

Example Output





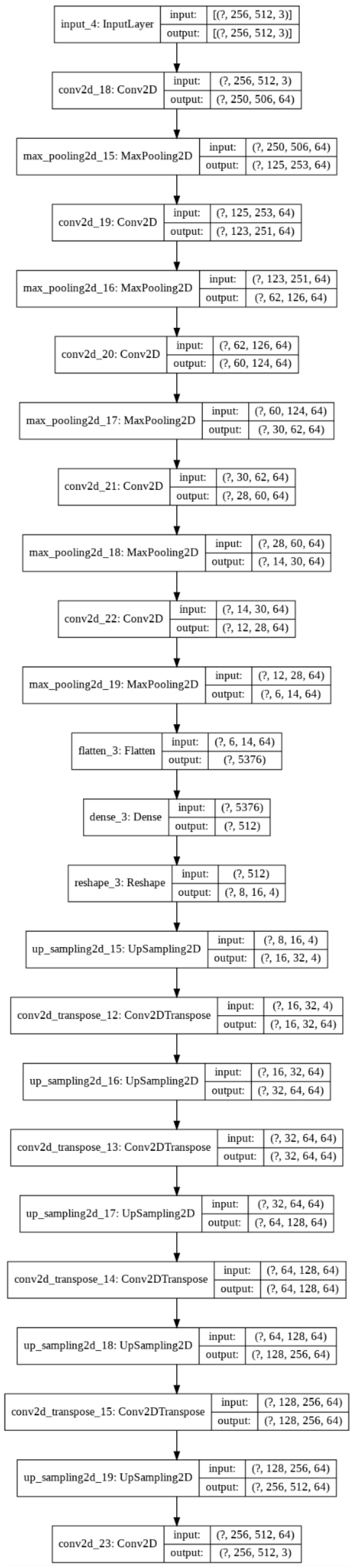
This result can be obtained in 2 minutes.
Generally speaking, this collage does reflect the scene diversity in the video and thus can be regarded as a good summary of the video. This collage can prove that our system can function properly.

How to run

Not necessary to run part1, because we have packed the all necessary data in our work.
generateCoolSummary is the main function and It takes path(all_frames) as the input.
In our folder:
all_frames: storing all frames
model_save: storing training model
Cool_summary.jpg: saving the summary picture
Cool_summary.gif: saving the gif picture
FIT5221_Assignment3.ipynb: the jupyter notebook

Appendix

The following is the detailed information of auto encoder


input_4: InputLayer | input: [(?, 256, 512, 3)] | output: [(?, 256, 512, 3)]
conv2d_18: Conv2D | input: (?, 256, 512, 3) | output: (?, 250, 506, 64)
max_pooling2d_15: MaxPooling2D | input: (?, 250, 506, 64) | output: (?, 125, 253, 64)
conv2d_19: Conv2D | input: (?, 125, 253, 64) | output: (?, 123, 251, 64)
max_pooling2d_16: MaxPooling2D | input: (?, 123, 251, 64) | output: (?, 62, 126, 64)
conv2d_20: Conv2D | input: (?, 62, 126, 64) | output: (?, 60, 124, 64)
max_pooling2d_17: MaxPooling2D | input: (?, 60, 124, 64) | output: (?, 30, 62, 64)
conv2d_21: Conv2D | input: (?, 30, 62, 64) | output: (?, 28, 60, 64)
max_pooling2d_18: MaxPooling2D | input: (?, 28, 60, 64) | output: (?, 14, 30, 64)
conv2d_22: Conv2D | input: (?, 14, 30, 64) | output: (?, 12, 28, 64)
max_pooling2d_19: MaxPooling2D | input: (?, 12, 28, 64) | output: (?, 6, 14, 64)
flatten_3: Flatten | input: (?, 6, 14, 64) | output: (?, 5376)
dense_3: Dense | input: (?, 5376) | output: (?, 512)
reshape_3: Reshape | input: (?, 512) | output: (?, 8, 16, 4)
up_sampling2d_15: UpSampling2D | input: (?, 8, 16, 4) | output: (?, 16, 32, 4)
conv2d_transpose_12: Conv2DTranspose | input: (?, 16, 32, 4) | output: (?, 16, 32, 64)
up_sampling2d_16: UpSampling2D | input: (?, 16, 32, 64) | output: (?, 32, 64, 64)
conv2d_transpose_13: Conv2DTranspose | input: (?, 32, 64, 64) | output: (?, 32, 64, 64)
up_sampling2d_17: UpSampling2D | input: (?, 32, 64, 64) | output: (?, 64, 128, 64)
conv2d_transpose_14: Conv2DTranspose | input: (?, 64, 128, 64) | output: (?, 64, 128, 64)
up_sampling2d_18: UpSampling2D | input: (?, 64, 128, 64) | output: (?, 128, 256, 64)
conv2d_transpose_15: Conv2DTranspose | input: (?, 128, 256, 64) | output: (?, 128, 256, 64)
up_sampling2d_19: UpSampling2D | input: (?, 128, 256, 64) | output: (?, 256, 512, 64)
conv2d_23: Conv2D | input: (?, 256, 512, 64) | output: (?, 256, 512, 3)