Object Detection Based on FCN

Chen Shen and Yaochen Liu

Abstract

In this project, a method of detection and segmentation from a given photograph such as means of transportation and animals is designed and improved based on the Fully Convolutional Network. The training and validation models are constructed with the Keras library in-built model named VGG-16 and bilinear upsampling theory. The PASCAL VOC2011 Dataset is utilized both in training and validating the performance of the model.

Keywords: Fully Convolutional Network, Object detection, segmentation

1 Introduction

As is known to all, object detection and segmentation is one of the most important and popular parts in the research of Computer Vision. Due to the rapid development of Deep Learning in recent years, in order to obtain the best performance, profoundly involving Deep Learning, especially convolutional neural network (convnet) becomes a must.

In conventional convnet utilized in object detection and segmentation, the surrounding pixels are always taken into consideration when it comes to classify one specific pixel. In this case, storage spending is relatively high when surrounding pixels are frequently involved. In addition, due to the probability that the attached pixels are identical, futile computations are inevitable. Moreover, the performance of the classification and segmentation are highly affected even limited by the dimension of weights and pixels. Hopefully, with the development of Fully Convolutional Network (FCN), those problems can be effectively resolved.

2 Previous Work

To our knowledge, the ideal of FCN is firstly introduced in topic of extending a convnet to arbitrary-sized inputs by Matan et al. [1]. In their work, the classic LeNet [2] is extended to recognize strings of digits. Due to the limitation of the mentioned convnet which is one-dimensional, Wolf and Platt [3] has extended the dimension to two when handling the problem of detection scores for the four corners of postal address blocks. Since then, FCN has been widely exploited in the research of detection. With the rapid development of utilizing FCN in detection, Ning et al.[4] define a convnet for coarse multiclass segmentation based on FCN and this marks the promising beginning of implementing segmentation with FCN.

By now, FCN has been intensively utilized in object detection and segmentation owing to its advantage in acceptability of images with arbitrary sizes and its efficiency. Numerous recent works have applied FCN on convnets to dense prediction problems, such as semantic segmentation, boundary prediction for

electron microscopy by Ciresan et al. [5] and hybrid convnet model for natural image by Ganin et al. [6].

Whereas in our project, the method of "shift-and-stitch" dense output and in-network upsampling proposed by Shelhamer et al. [7] is implemented to compile our model as the function of autoencoder. The method introduced by Shelhamer et al. is different from other popular approaches such as saturating and nonlinearities. In their idea, deep classification architectures are adapted and extended to use image classification as supervised pre-training. In addition, the extended architectures are exploiting fine-tune Fully convolutional to learn simply and efficiently from whole image inputs and whole image ground truths.

3 Fully Convolutional Network

3.1 Basic Knowledge of FCN

As is discussed in the lecture, in the typical convnets such as VGG-16 (Shown in Fig. 1), the last few layers of a given model are fully connected layers with activation function and an output layer of Softmax function, which transform the pixels in a given image into feature net for classification.

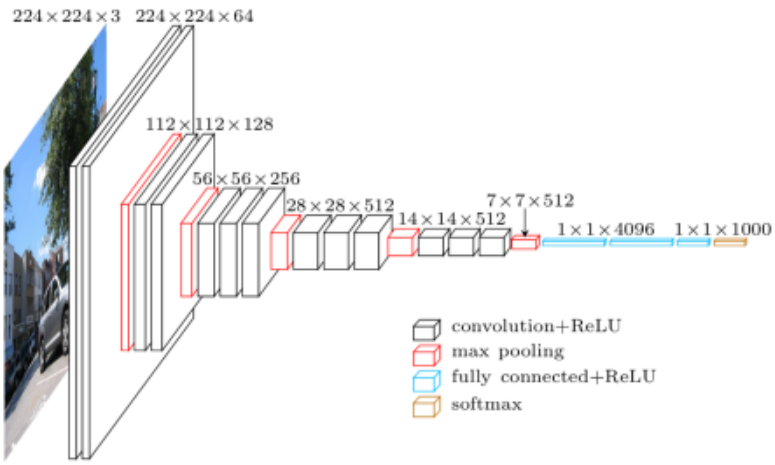

Fig. 1. Model architecture of VGG-16 [8]

While in the FCN method (Shown in Fig. 2), the last few fully connected layers are substituted with convolution layers which in turn enables a transform from a classification net into a spatial map. Moreover, if differentiable interpolation layers and spatial loss exists, end-to-end pixelwise learning will be possible.

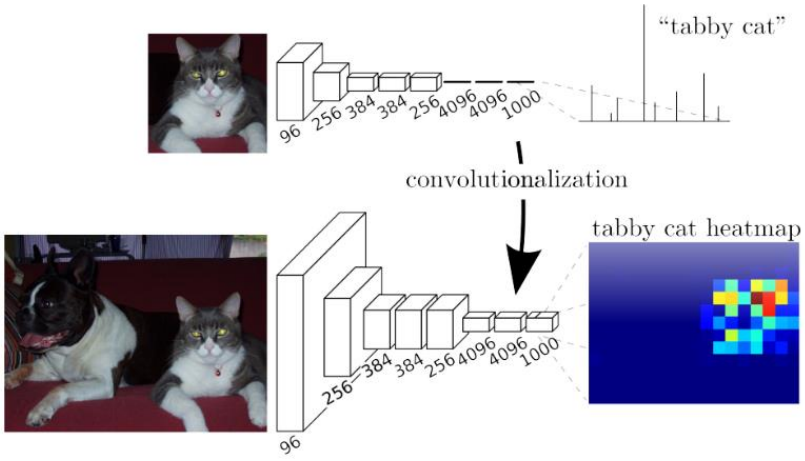

Fig. 2. Transforming fully connected layers into convolution layers [7]

3.2 Autoencoder in FCN

As is discussed in class, a conventional convnet utilized for object detection and segmentation is expected to implement with autoencoder which consists of two parts, namely encoding part and decoding part. The encoding part mainly transforms the image pixels into features and the decoding part functions vice versa. In this project, the VGG-16 model is employed as the encoding part

3.3 Bilinear upsampling and Deconvolution

In this project, the decoding part is implemented by bilinear upsampling, which is the reverse version of VGG-16 model which plays the role of deconvoluting the features generated by encoder to the image pixels again. Due to the limitation of computing power, however, we freeze the encoding part parameters to be untrainable while activate the decoding part parameters to achieve a better performance based on limit resource we can access. In addition, in order to achieve a better performance, the deconvolution layers in this project is set to be trainable.

3.4 Combination and Refining

Since the convnet for segmentation is defined to have feature hierarchy, it is natural to refine the spatial precision of the output at different layer. 3 ways of manipulating upsampling step-size are proposed by Shelhamer et al. [7]. The first one named FCN-32s simply upsamples stride 32 predictions back to pixels in only one step, the second one FCN-16s makes combination of predictions from both the last layer and the 4^{th} pooling layer with upsampling stride 16. While the third one FCN-8s upsamples at a shallower 3^{rd} pooling layer with stride 8, rather than predicting at 4^{th} pooling layer (Shown in Fig. 3). It is intuitive and also proved with experiments that the FCN-8s makes better predictions (shown in Fig. 4). Hence, in our project, the FCN-8s method is implemented.

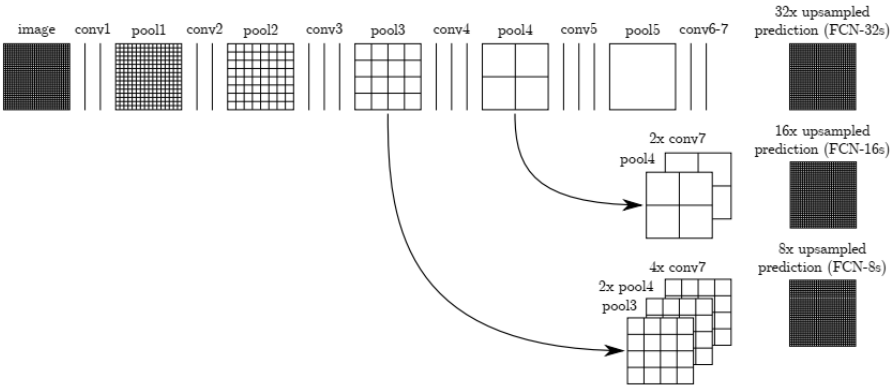

Fig. 3. Combination of predictions at different layer with diverse upsampling stride [7]

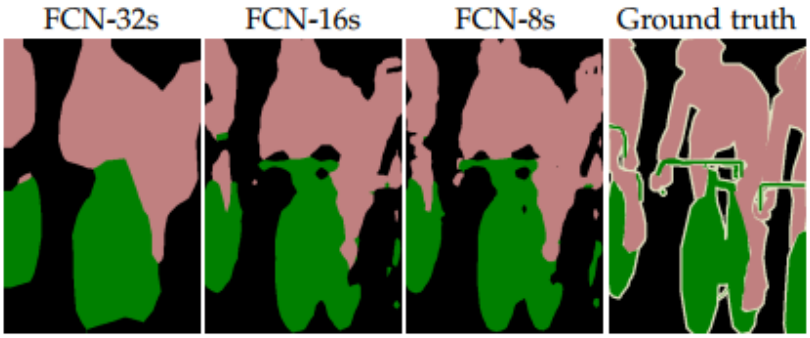

Fig. 4. Comparison among different refining methods and ground truth [7]

4 Project Procedure

4.1 Dataset used for Training and Validation

Since long before the birth of FCN, the topic of object detection and segmentation has already aroused wide attention around the world. Hence, to achieve a better detection and segmentation result, many datasets intended for segmentation are generated and shared in public.

In this project, the famous PASCAL Visual Object Classes Challenge 2011 (PASCAL VOC2011) dataset is implemented for both training and validation. The PASCAL VOC2011 dataset contains 4 major classes of person, animal, means of transportation and furniture. Moreover, these major classes are further divided into 20 specific classes for accuracy. In total, there are 11,530 images containing 27,450 ROI annotated objects and 5,034 segmentations [9] in PASCAL VOC 2011 dataset for both training and validation.

4.2 Data Preprocessing

In this project, before feeding images into the prototype model for training, several preprocessing such as Normalization is conducted to regularize and standardize the image pixels to prevent large coefficients dominate the small ones.

4.3 Model Summary

As is discussed in the previous chapters, the model applied in this project contains two parts, one is the encoder based on VGG-16 with untrainable parameters and the other is the deconvolution decoder by bilinear upsampling. The generated model (shown in Fig. 5) contains 29 layers and several pooling layers. A plot of the model can be seen in the appendix.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
block5_fc6 (Conv2D)	(None, 7, 7, 4096)	102764544	block5_pool[0][0]
dropout_1 (Dropout)	(None, 7, 7, 4096)	0	block5_fc6[0][0]
block5_fc7 (Conv2D)	(None, 7, 7, 4096)	16781312	dropout_1[0][0]
dropout_2 (Dropout)	(None, 7, 7, 4096)	0	block5_fc7[0][0]
score_feat1 (Conv2D)	(None, 7, 7, 21)	86037	dropout_2[0][0]
score_feat2 (Conv2D)	(None, 14, 14, 21)	10773	block4_pool[0][0]
upscore_feat1 (BilinearUpSamplin	(None, 14, 14, 21)	0	score_feat1[0][0]
scale_feat2 (Lambda)	(None, 14, 14, 21)	0	score_feat2[0][0]
add_1 (Add)	(None, 14, 14, 21)	0	upscore_feat1[0][0] scale_feat2[0][0]
score_feat3 (Conv2D)	(None, 28, 28, 21)	5397	block3_pool[0][0]
upscore_feat2 (BilinearUpSamplin	(None, 28, 28, 21)	0	add_1[0][0]
scale_feat3 (Lambda)	(None, 28, 28, 21)	0	score_feat3[0][0]
add_2 (Add)	(None, 28, 28, 21)	0	upscore_feat2[0][0] scale_feat3[0][0]
upscore_feat3 (BilinearUpSamplin	(None, 224, 224, 21)	0	add_2[0][0]
activation_1 (Activation)	(None, 224, 224, 21)	0	upscore_feat3[0][0]

Total params: 134,362,751
Trainable params: 119,648,063
Non-trainable params: 14,714,688

None

Fig. 5. Model Summary

4.4 Training and Validation Parameters

Due to the limitation of computing power, in this project, the `batch_size` is set to 3, the `step_per_epoch` is 1112 and the `epochs` is set 40. Actually, `batch_size` to be 10 and `epochs` to be 100 is recommended.

5 Results

5.1 Training summary

After training at Google Cloud Platform (with 8 v-CPUs, 52GB memory) for about 65 hours, the final accuracy of training and validation data can reach up to 91.29% and 83.09%, respectively.

```
====] - 9209s - loss: 5.0491 - acc: 0.9129 - val_loss: 5.3497 - val_acc: 0.8309
```

Fig. 6. Final accuracy and loss of training and validation data

Based on the log printed by `nohup`, we can review the loss and accuracy during the training process. From the figures, we can see the loss decreasing rapidly at first, then it keeps reducing gently, which is hard to track after several epochs (1112 iterations per epoch). Besides, the accuracy appears to be growing over time except for some singularity. These two curves meet our expectation. They partly imply that the model has captured some features from the data. That is, the parameters have been learnt.

Because keras only shows the validation loss and accuracy after each epoch, the number of validation loss and accuracy are too few to be shown in a figure. Thus, we show the final results of validation summary merely. Actually, they share the same trend with training ones.

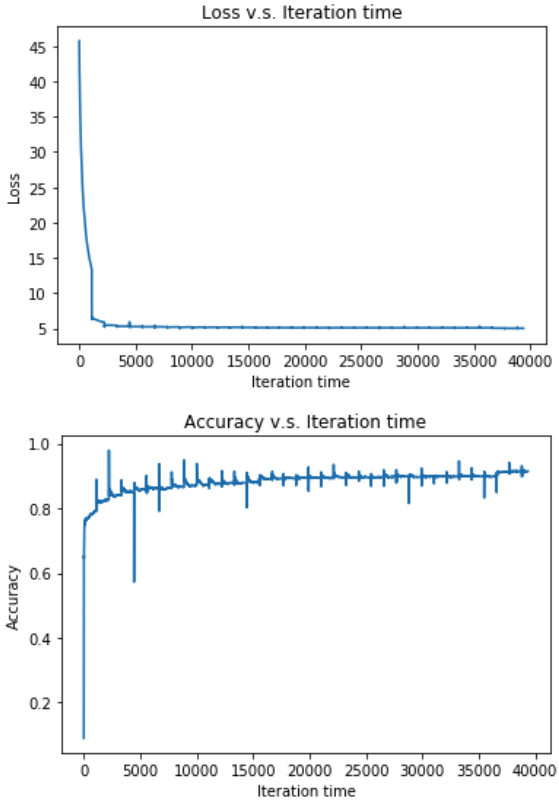

Fig. 7. Figure of loss and accuracy v.s. iteration time

5.2 Visualization the segmentation results

In order to intuitively inspect the performance of detection and segmentation, several images are randomly picked to visualize the effect of segmentation. As is shown in fig. 8, the detection is generally authentic while the segmentation still needs more improvement.

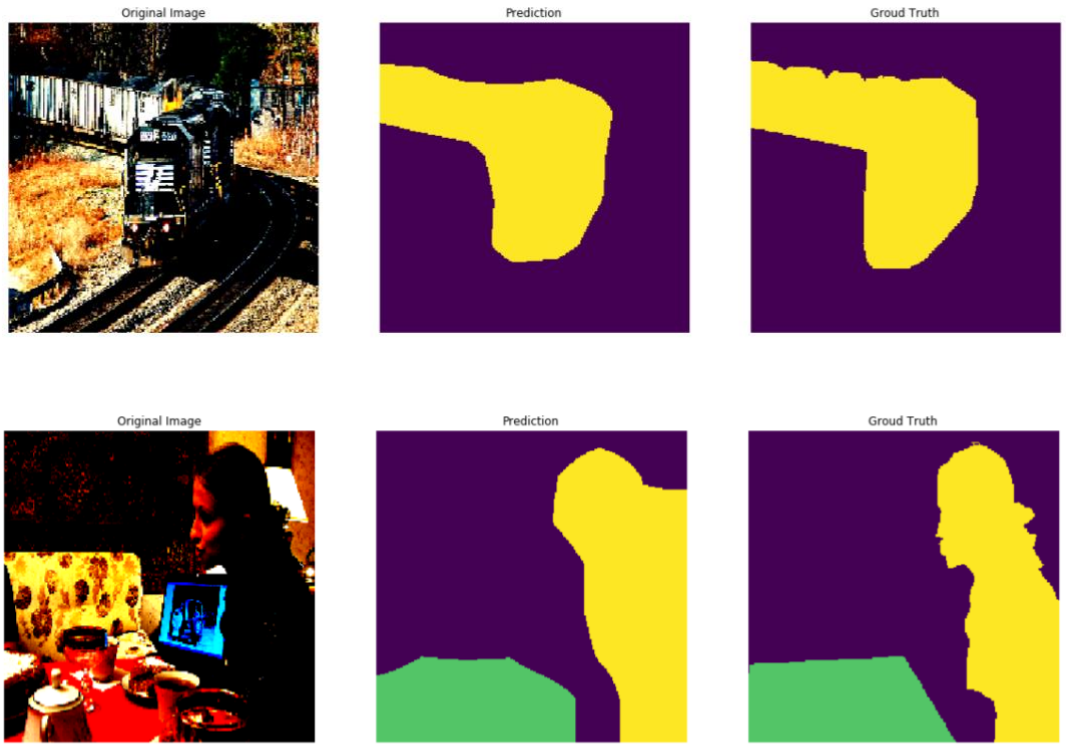

Fig. 8. Some segmentation results

6 Conclusion

As can be vividly seen from the decreasing trend of training and validation loss, the generated model could achieve better performance when given more time. With the current computing power and time limit, however, the number of epochs can hardly reach to what we have expected. Fortunately, as is shown in fig. 7, the model is reasonable and effective.

7 Acknowledgements

Many people have made invaluable contributions both directly and indirectly to our project. We would like to express my warmest gratitude to Prof. Yao Wang, for her instructive suggestions when we are confronted with problems of computing power. At the same time, we are also grateful to Mr. Ish Kumar Jain for providing us with valuable advice and instructions concerning Google Cloud Platform.

8 References

[1] O. Matan, C. J. Burges, Y. LeCun, and J. S. Denker, "*Multi-digit recognition using a space*

displacement neural network" in NIPS,
1991, pp. 488–495. 2
[2] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "*Backpropagation applied to handwritten zip code recognition*" in Neural Computation, 1989.
[3] R. Wolf and J. C. Platt, "Postal address block location using a convolutional locator network," in NIPS, 1994, pp. 745–745.
[4] F. Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P. E. Barbano, "*Toward automatic phenotyping of developing embryos from videos*," Image Processing, IEEE Transactions on, vol. 14, no. 9, pp. 1360–1371, 2005.
[5] D. C. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "*Deep neural networks segment neuronal membranes in electron microscopy images.*" in NIPS, 2012, pp. 2852–2860.
[6] Y. Ganin and V. Lempitsky, "N4-fields: Neural network nearest neighbor fields for image transforms," in ACCV, 2014.
[7] E. Shelhamer, J. Long and T. Darrell. *Fully Convolutional Networks for Semantic Segmentation*. Retrieved May 15, 2018, from https://people.eecs.berkeley.edu/~jonlong/long_shelhamer_fcn.pdf
[8] Retrieved May 15, 2018 from https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/
[9] Retrieved May 15, 2018 from http://host.robots.ox.ac.uk/pascal/VOC/voc2011/index.html
[10] Krizhevsky, Alex, I. Sutskever, and G. E. Hinton. *"Imagenet classification with deep convolutional neural networks."* Advances in neural information processing systems. 2012.
[11] Jinghong Ju, *keras-fcn*, (2017), GitHub repository, Retrieved April 20, 2018 from https://github.com/JihongJu/keras-fcn
[12] Jonathan Long*, Evan Shelhamer*, and Trevor Darrell. CVPR 2015 and PAMI 2016, GitHub repository, Retrieved April 20, 2018 from https://github.com/shelhamer/fcn.berkeleyvision.org

9 Appendix

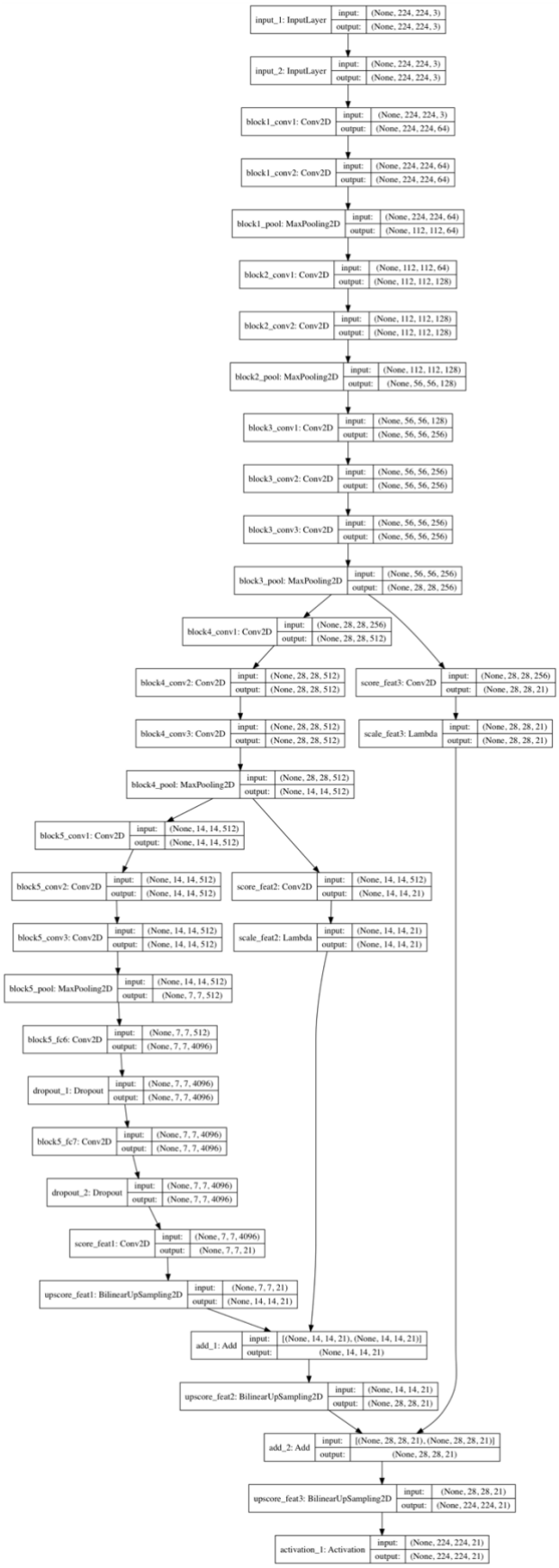

Fig. 9. The plot of the model used in this report