UDACITY

PROJECT

Follow Me

A part of the Robotics Software Engineer Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

5 SPECIFICATIONS REQUIRE CHANGES

This is a nice resource for semantic segmentation,Take a look at this: http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review

Your project is impressive, nice work so far. I'll outline a solution to your problem here: your model is too big. Remember that your input size is just 160x160. So after 5 encoders, it's down to 5x5. That's pretty small. Take a look at other architectures to see what they do? Instead, use a lower number of encoder/decoder pairs, 2 or 3. To increase the number of trainable parameters, increase the kernel depth! Revision 2&3 should actually suffice, I encourage you to try higher numbers and see what happens. The one thing in your trials that you didn't change, unfortunately was the number of layers!

Another thing to do is increase the number of conv layers per encoder/decoder! Take a look at other architectures and you will notice many do use this. Look closely at the comments in decoder section.

In case you want to continue trying with 5 encoders, I've left further comments below. Maybe far too many, but I couldn't help. Enjoy the project! Do keep experimenting, I learnt everything I've written from experimentation too! Keep it up!

Writeup

The write-up / README should include a statement and supporting figures / images that explain how each rubric item was addressed, and specifically where in the code each step was handled. The write-up should include a discussion of what worked, what didn't and how the project implementation could be improved going forward.

This report should be written with a technical emphasis (i.e. concrete, supporting information and no 'hand-waiving'). Specifications are met if a reader would be able to replicate what you have done based on what was submitted in the report. This means all network architecture should be explained, parameters should be explicitly stated with factual justifications, and plots / graphs are used where possible to further enhance understanding. A discussion on potential improvements to the project submission should also be included for future enhancements to the network / parameters that could be used to increase accuracy, efficiency, etc. It is not required to make such enhancements, but these enhancements should be explicitly stated in its own section titled "Future Enhancements".

An excellent write-up. Tons of details provided, very good job!
However, all the images were missing from the write-up since they are not included in the submission! Please do include those images in the next submission and ensure that the paths are correct in the .md file. Alternately, you can export the markdown file as a pdf using some tools.

To make the report really read well, consider adding in sections of introduction, results, conclusion and future improvements. Many of your discussion points fit well into those segments.

The student clearly explains each layer of the network architecture and the role that it plays in the overall network. The student can demonstrate the benefits and/or drawbacks of different network architectures pertaining to this project and can justify the current network with factual data. Any choice of configurable parameters should also be explained in the network architecture.

The student shall also provide a graph, table, diagram, illustration or figure for the overall network to serve as a reference for the reviewer.

A good explanation of the network architecture.

Some ways here in which you can improve the network, I'll mention them in later points.

The student explains their neural network parameters including the values selected and how these values were obtained (i.e. how was hyper tuning performed? Brute force, etc.) Hyper parameters include, but are not limited to:

- Epoch
- Learning Rate
- Batch Size
- Etc.

All configurable parameters should be explicitly stated and justified.

Needless to say, spectacular job here. Really nice work documenting all the changes.

I'll add in some tips in this section:

- As your batch size changes, the ideal learning rate can change as well. This is because the batch size will determine the amount of change in the gradient descent algo.
- Other params can also affect the learning rate. So it is generally a good idea to keep one constant and change the other. Run experiments greater than just 3-5 epochs. For a network of your size, that's not nearly enough epochs to show results.
- Try a slightly lower value of learning rate than 0.1. A higher learning rate will show faster results, of course. But it may not result in the best final accuracy. Review the section on gradient descent if required. You can verify this statement by observing the differences in row 10 and 11. The hyperparams are the same, but the number of trainable parameters increased by a huge amount! Use the keras Model Summary to view a count of the total number of parameters in your model.
- Try to take the model and params in row 10 and training for a longer time, say 100 epochs. Supplement with your own data.
- Try lower values of batch size. In this project, it's noticed that lower batch sizes have better results. Notice row 13,14. You reduced the number of batches, but at the same time, you also reduced the steps ver epoch! That should have actually increased! What you've effectively done is to reduce the effective number of epochs!
- The val steps should also change depending on the num of images in dataset.
- Num of workers represents the num of cores in your CPU. Do keep it accordingly.

The basic idea of val steps and steps per epoch are to get *all* the images through the network in 1 epoch, based on which the weights will update. Now, you can have a smaller number of images going through, the images and jumbled, so over time it will be exposed to all of them. But for that to work, you need to have enough epochs!

The student demonstrates a clear understanding of 1 by 1 convolutions and where/when/how it should be used.

The student demonstrates a clear understanding of a fully connected layer and where/when/how it should be used.

This rubric point wasn't addressed. Please add it in as per the rubric. This might help: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_segmentation.html

The student is able to identify the use of various reasons for encoding / decoding images, when it should be used, why it is useful, and any problems that may arise.

You've done a good job explaining the structure of encoders/decoders and a brief overview of decoders, but this section needs more details. Please explain each in more depth. What's the encoder doing? What are the disadvantages of this architecture? Try to think of why skip connections are needed.
here's a good resource: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/image_segmentation.html

The student is able to clearly articulate whether this model and data would work well for following another object (dog, cat, car, etc.) instead of a human and if not, what changes would be required.

You're right!

Model

The file is in the correct format (.h5) and runs without errors.

Both the config and weight files are needed! Please submit both next time!

The neural network should obtain an accuracy greater than or equal to 40% (0.40) using the Intersection over Union (IoU) metric.

I'm impressed! It's the first time I'm seeing a network as large as this! You must have a huge, huge PC considering you were able to use batch sizes of 32-128 with a model that large!

So, a lot of your problems arise from the fact that your model is very, very big. Run keras' model summary and you might be surprised to see the total number of trainable parameters in your model. The bigger a model is, normally, the longer it takes to train. You'll need to put in a lot more epochs to start seeing the results. I know the graph may look like it's stagnating, but it's not.

To get better graphs, use the tensorboard callback. Modify your notebook so:

```
#logger_cb = plotting_tools.LoggerPlotter()
logger_cb = keras.callbacks.TensorBoard(log_dir='./logs/model1c', histogram_freq=1, batch_size=32, \
  write_graph=True, write_grads=False, write_images=False, \
  embeddings_freq=0, embeddings_layer_names=None, embeddings_metadata=None)
save_cb = keras.callbacks.ModelCheckpoint(filepath='./saved_chkpts/model1c/weights-e{epoch:02d}-{val_loss:.2f}.hdf5', ve
rbose=1, period=5)
```

```
callbacks = [logger_cb, save_cb]
```

You can then launch tensorboard from a new terminal with: `tensorboard --logdir=/full_path_to_your_logs`.
Notice the `ModelCheckpoint` callback used above. This is used to save your weights at regular intervals.

Use the model with the lowest validation loss! It's normal to have huge spikes in your validation loss in the middle of training, don't worry about it. Again, review the section on gradient descent. The algorithm is making small, jerky movements towards the ideal value. In the process, it will find some areas that are horrible, but with the right params, it will quickly come back towards better values.
Don't worry about overfitting. Look for classical signs to confirm overfitting: your training loss is decreasing, but your val loss is increasing, *consistently*. Take a look at the graphs here: http://cs231n.github.io/neural-networks-3/

It's hard to spot the pattern in the val loss because the number of samples are too few. What you can try is, decrease the number of steps per epoch and increase the total number of epochs significantly. This will give you more frequent updates on the graph, resulting in a better curve. Use tensorboard's smoothing feature to easily see the trend of the curve.

To ensure over-fitting doesn't occur, there are many stategies to use, like dropouts! But only use them if you've confirmed that the network is truly overfitting!

As an answer to your outstanding question - No, it doesn't. Like you said, after a point the network can overfit. In this case, it's becoming **too** good, it's "memorizing" the dataset! Also, in many cases, the network can saturate. Continued training may not improve the results. Note that, in some cases, even though the val_loss is the same, the final score can be very different! This may be a result of the quality of the dataset. Divergent losses do indicate a reduction in effectiveness, it's a classic sign of overfitting. however, the divergence must be continuous and significant. Consult the link I shared earlier.

What makes you think the skip connections are not working effectively? Well, the inputs should not be fed directly. So what can be done is have a conv layer run over the input to produce a layer with the same dimensions! use that for skip connections.

Note that the model is absolutely absymal at detecting the hero from far away: `average intersection over union for the hero is 0.1381095041782656`. To rectify this, collect more images of the hero from far away! Take a look at the testing images to see what they should look like.

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Rate this review

Student FAQ