Review of "RadNet 1.0: Exploring deep learning architectures for longwave radiative transfer"

I thank the authors for their replies. However, I still have some comments and open questions which are listed below. Quotes "..." are from your reply. In particular, I still believe that the current architecture choices are not necessarily the most informative. I strongly urge the authors to actually run some new experiments as suggested below, particularly since the comparison of network architectures is probably the most interesting feature of the paper.

Network design: *I only realized from looking at your code that you are using a max-Pooling layer between every convolutional layer. Searching for "pooling" in the text only gives one result: "CNNs usually consist of three types of layers, convolutional layers, pooling layers and fully-connected layers." I think the max pooling layers should be explicitly mentioned in the text, preferably in table 1. Also, there are many CNNs without max pooling, so I think your statement is not quite accurate. See also comment below.*

"There is nothing in our setup which limits our simulations to a single column. This is because radiative transfer is assumed to have no contribution from adjacent columns (as we have described in Lines 91–92.). Our motivation to use a single column model for validation has been described in the manuscript in line 350: it allows us to study the evolution and possible errors in greater detail.
"

Theoretically you could use your RadNet in a 3D CLiMT simulation. But as far as I can tell, it would be very difficult to create a CliMT simulation that was realistic enough so that the profiles are close to the ERA profiles. If you just plugged your ERA-trained RadNet into a simple CliMT simulation, I would expect your inputs to be very much outside of your training range. For this reason, I think training on CliMT data and then running an online 3D CliMT simulation would be the easier approach to test a 3D online run.

"The channel number is correct. The increase is caused by the zero-padding. [...] Model E: 3∗3∗128+3∗3∗128∗256+58∗2∗256∗512+512∗60 = 15531136"

Why is the number of channels increased for zero padding? Usually, only the non-channel dimensions are padded.

*Also, I am not sure I understand how you compute the number of parameters for the first layer. As far as I understand your input shape is (batch_size, 62, 6). As you do in the following layers, the input channels should also be included in the computation of the parameter number, so 3*3*128*6, right?*

Then I am confused why you have 58 in 58∗2∗256∗512. I thought you used max-pooling, shouldn't the size have been reduced by a factor of 4?

“We do not think that adding padding to all conv layers will act the same as fully connected layers.”

That is not what I intended to say. In fact it wouldn’t act as a fully connected layer but I still think it would be an interesting experiment because you would have a lot fewer parameters. As mentioned above, I did not realize that you used max pooling in-between every convolutional layer. I was thinking of a network like this (potentially with many more layers):

```
model = Sequential()
model.add(Conv2D(c0_size, kernel_size=(6, 3), activation=’relu’, input_shape=(60,3,1), padding='same'))
model.add(Conv2D(c1_size, kernel_size=3, activation=’relu’, padding='same'))
model.add(Conv2D(c2_size, kernel_size=3, activation=’relu’, padding='same'))
model.add(Conv2D(1, kernel_size=3, activation=linear, padding='same'))
```

That way you would avoid that huge fully connected layer after the Flatten() which I assume is the main reason the CNNs are so slow. Other ways to avoid this would be to have a much smaller number of filters in the last layer, or have more conv-max-pool layers to further decrease the size of the signal. (For an input size of 60, using 5 conv layers would decrease the signal size to around 2. Flattening then would result in a much smaller vector.) As it is currently your experiments do not support your conclusions that CNNs cause a performance loss because I assume that the performance loss comes from the big fully-connected layer rather than the conv layers.

“We would expect that Resnet will give more accuracy
in this use case. However, we expect that such a model will be even
slower than the model E given that Resnet-50 has over 25 million
Parameters.”

Resnet does not specifically refer to the Resnet-50 used for image classification. Rather it just refers to the idea of using skip connections which come at basically no extra cost except for one addition. I would suggest trying out a network that looks something like this (here I am using the Keras functional API):

```
x = inp = Input(shape=(60, 3,))
x = Conv2d(size, kernel_size, activation='relu')(x)
x = Conv2d(size, kernel_size, activation='relu')(x)
x = Conv2d(size, kernel_size, activation='relu')(x)
x = Add()([inp, x])
outp = Conv2d(1, kernel_size, activation='relu')(x)
```

You could also add several resblocks (something like 5 not 50). You could either do this in a fully convolutional way as in my code, or add pooling layers between the resblocks

(bottlenecks) and use a fully connected layer in the end. In general, I would strongly suggest that you at least test the experiments I suggested since it shouldn't be much extra work.

Also, you did not directly reply to my comment about the scores of model A and B: " I am surprised a bigger network (B vs A) did not give you a better score. If you are not yet overfitting, bigger networks should always result in better losses, or am I missing something? You mention overfitting when talking about the generalization experiments but that still doesn't explain why the skill for Dataset 1 wouldn't be better for model B." Do you have any explanation why model B isn't better than model A? Is the training loss lower for B than for A?