Full Model Architecture

Learn about the full model architecture of ResNet.

Chapter Goals:

- Learn about the full model architecture of ResNet

A. Regularization

For extremely deep models like ResNet, it is vital that we *regularize* the model, i.e. apply techniques to prevent overfitting. In the **CNN** and **SqueezeNet** sections, we used *dropout* for regularization. However, in ResNet we don't use dropout because it is normally not necessary to use dropout together with batch normalization.

The creators of batch normalization found that, along with reducing internal covariate shift, batch normalization also regularizes the model. This is because we take into account the entire batch of inputs when performing batch normalization, which reduces the chances that the model overfits on a few outliers. Since we apply batch normalization before nearly every weight layer, it alone is sufficient for regularization.

B. Increased filters

As always, we increase the number of filters used for deeper layers in the model. In ResNet, the number of filters is doubled as we go from one block layer to the next.

We increase the number of filters at block layers, rather than at individual blocks, due to the large number of blocks in ResNet. Increasing the number of filters at each block would lead to an incredibly large number of parameters, slowing down training and greatly increasing the model's memory usage.

C. Logits

The final part of the ResNet model architecture is the layer used to obtain logits.

For the SqueezeNet model, we used a convolution layer to make the number of channels equal to the number of image classes. Then we applied global average pooling across the channels to obtain the logits.

However, while the CIFAR-10 dataset had only 10 image classes, the ImageNet dataset has 1000. Using a convolution layer with 1000 filters would require many weight parameters.

So to save weight parameters in our ResNet model, we first apply global average pooling across the channels, rather than using a convolution layer. Then we flatten the data and use a fully-connected layer to obtain the logits.

Below, we show the full code for the ResNet model architecture (using the functions from previous chapters as helpers):

```python
import tensorflow as tf

class ResNetModel(object):
    # __init__ and other functions omitted

    # Model Layers
    # inputs (channels_last): [batch_size, resize_dim, resize_dim, 3]
    # inputs (channels_first): [batch_size, 3, resize_dim, resize_dim]
    def model_layers(self, inputs, is_training):
        # initial convolution layer
        conv_initial = self.custom_conv2d(
            inputs, self.filters_initial, 7, 2, name='conv_initial')
        # pooling layer
        curr_layer = tf.layers.max_pooling2d(
            conv_initial, 3, 2, padding='same',
            data_format=self.data_format,
            name='pool_initial')
        # stack the block layers
        for i, num_blocks in enumerate(self.block_layer_sizes):
            filters = self.filters_initial * 2**i
            strides = self.block_strides[i]
            # stack this block layer on the previous one
            curr_layer = self.block_layer(
                curr_layer, filters, strides,
                num_blocks, is_training, i)
        # pre-activation
        pre_activated_final = self.pre_activation(curr_layer, is_training)
        filter_size = int(pre_activated_final.shape[2])
        # final pooling layer
        avg_pool = tf.layers.average_pooling2d(
            pre_activated_final,
```

D. Example image classifier

To play around with a powerful image classification tool, click this link. This model allows you to upload an image, and it will generate sets of tags relevant to that image.