We have used the Conv3D model to reach an accuracy of 77.6%:

* Layer-1 (Conv3D): We have used 32 kernels of size (3, 3, 3), and each kernel has a single bias, so we have 32 x 3 x 3 x 3(weights) x 3(channels) + 32 (biases) = 2624 parameters (all trainable). Note that the kernels have three channels since the input images are 3D. This convolutional layer uses stride of 1 and no padding. Further, we have performed batch normalization and then the activation function ‘elu’ is used to produce an output to be used as an input to the next layer. The output from this layer is of shape 18 x 84 x 84 x 32 (the first element None is for the batch size).
* Max pooling 3D: The pooling layer gets the (18, 84, 84, 32) input from the previous conv layer and produces a (9, 42, 84, 32) output. Stride of (2, 2, 1) is used. There are no trainable parameters in the pooling layer.
* Layer-2 (Conv3D): We have used 128 kernels of size (3, 3, 3), but this time, each kernel must convolve a tensor of size (9, 42, 84, 32) from the previous layer. Thus, the kernels will also have 128 channels, and so the shape of each kernel is (3, 3, 3, 128). So we have 128 (kernels) x 3 x 3 x 3 (weights) x 32(channels) + 128 (biases) = 1,10,720 parameters (all trainable). Further, we have performed batch normalization and then the activation function ‘elu’ is used to produce an output to be used as an input to the next layer. The output shape is (9, 42, 84, 128).
* Max pooling 3D: The pooling layer gets the (9, 42, 84, 128) input from the previous conv layer and produces a (4, 21, 42, 128) output. Stride of (2, 2, 2) is used. There are no trainable parameters in the pooling layer.
* Layer-3 (Conv3D): We have used 256 kernels of size (3, 3, 3), each kernel must convolve a tensor of size (4, 21, 42, 128) from the previous layer. Thus, the kernels will have 256 channels, and so the shape of each kernel is (3, 3, 3, 256). So we have 256 (kernels) x 3 x 3 x 3 (weights) x 128(channels) + 256 (biases) = 8,84,992 parameters (all trainable). Further, we have performed batch normalization and then the activation function ‘elu’ is used to produce an output to be used as an input to the next layer. The output shape is (4, 21, 42, 256).
* Max pooling 3D: The pooling layer gets the (4, 21, 42, 256) input from the previous conv layer and produces a (2, 10, 21, 256) output. Stride of (2, 2, 2) is used. There are no trainable parameters in the pooling layer.
* The Dropout layer does not alter the output shape and has no trainable parameters.
* Layer-4 (Conv3D): We have used 256 kernels of size (3, 3, 3), each kernel must convolve a tensor of size (2, 10, 21, 256) from the previous layer. Thus, the kernels will have 256 channels, and so the shape of each kernel is (3, 3, 3, 256). So we have 256 (kernels) x 3 x 3 x 3 (weights) x 256(channels) + 256 (biases) = 17,69,728 parameters (all trainable). Further, we have performed batch normalization and then the activation function ‘elu’ is used to produce an output to be used as an input to the next layer. The output shape is (2, 10, 21, 256).
* Max pooling 3D: The pooling layer gets the (2, 10, 21, 256) input from the previous conv layer and produces a (1, 5, 10, 256) output. Stride of (2, 2, 2) is used. There are no trainable parameters in the pooling layer.
* The Flatten layer simply takes in the (1, 5, 10, 256) output from the previous layer and 'flattens' it into a vector of length 1 x 5 x 10 x 256 = 12,800.
* The Dense layer is a plain fully connected layer with 256 neurons. It takes the 12,800-dimensional output vector from the previous layer (layer l-1) as the input and has 256 x 12,800 (weights) + 256 (biases) = 32,77,056 trainable parameters. The output of this layer is a 256-dimensional vector.
* The Dropout layer simply drops a few neurons.
* The Dense layer is a plain fully connected layer with 256 neurons. It takes the 256-dimensional output vector from the previous layer (layer l-1) as the input and has 256 x 256 (weights) + 256 (biases) = 65,792 trainable parameters. The output of this layer is a 256-dimensional vector.
* The Dropout layer simply drops a few neurons.
* Finally, we have a Dense SoftMax layer with 5 neurons which takes the 256-dimensional vector from the previous layer as input. It has 256 x 5 (weights) + 5 (biases) = 1,285 trainable parameters.

Thus, the total number of parameters are 61,13,541 all of which are trainable.

Following are some experiments that we performed to reach a good performing model.

|  |  |  |  |
| --- | --- | --- | --- |
| **Experiment Number** | **Model** | **Result** | **Decision + Explanation** |
| **1** | **Conv3D** | **Throws**  **ResourceExhaustedError**  **error** | **Reduced the number of filters in the first convolutional layer to be adequate for the GPU to be able to process.** |
| **2** | **Conv3D** | **Accuracy: 0.1786** | **We increased the number of kernels.** |
| **3** | **Conv3D** | **Learning was slow.** | **When the validation loss has stopped improving, then we reduced the learning rate. We increased the factor under ReduceLROnPlateau method from 0.1 to 0.5** |
| **4** | **Conv3D** | **Poor** **validation categorical accuracy observed.** | **Validation categorical accuracy of 0.0000e+00 was observed using Adam optimizer. On switching to**  **SGD optimizer validation categorical accuracy improved.** |
| **5** | **Conv3D** | **Accuracy: 0.32** | **Decreased the Image size to improve the performance of the model.** |
| **6** | **Conv3D** | **Accuracy: 0.41** | **Altered the Learning rate.** |
| **7** | **Conv3D** | **Accuracy: 0.65** | **Added another dense layer after 4th Conv3D layer.** |
| **Final model** | **Conv3D** | **Accuracy: 0.776** | **Increased the number of kernels further.** |