# UpGrad Neural Networks Assignment (Gesture Recognition)

Submitted by Vijay Reddy & Viswajit Iyer

### Mon, 23rd Dec, 2019

## Table of Contents

[General Notes](#_heading=h.o4521a8z5b8c)

[LR Monitor Method:](#_heading=h.ifg2989o8d8j)

[Optimiser](#_heading=h.qjuhchoe152h)

[Image Normalisation:](#_heading=h.54uo3tizd2ij)

[Image Re- Sizing](#_heading=h.z8zwqv3afzes)

[Experiments](#_heading=h.fs3pvet72gu7)

[Conv3D 3-layer (8,16,32)](#_heading=h.30j0zll)

[Conv3D 3-layer (16,32,64)](#_heading=h.djgzigv08tfc)

[Conv3D 4-layer (8,16,32,64)](#_heading=h.9bbgq0ptgxwn)

[Conv3D 4-layer (8,16,32,64) with 1x3x3 Kernel](#_heading=h.qui6j5w7uiuj)

[Conv3D 4-layer (16,32,64,128), 1x3x3 Kernel, 1 Dense Layer](#_heading=h.fq7oqdypnpde)

[Conv3D 4-layer (8,16,32,64), 1x3x3 Kernel, 1 Dense Layer - FINAL](#_heading=h.7ffs9kj9tmen)

[CNN + RNN (VGG16 + GRU)](#_heading=h.fc7mlhrv6ydc)

[CNN + RNN (ResNet50 + GRU)](#_heading=h.ixbnwwz21nga)

[Conv3D 4-layer (8,16,32,64), 2x2x2 Kernel](#_heading=h.vp882xbwbqms)

[Errors Encountered](#_heading=h.gjscbuxz1rtk)

[Conclusion](#_heading=h.nmlgnxxamcrf)

[Accuracy & Loss Graphs](#_heading=h.qrlmu985cj0a)

[Fig 1 (FInal Mode) : Conv3D Model (8,16,32,64; 0.5D/o) + 1 Dense (64;0.2 D/o) - Batch Size 32 - 50 Epochs (Nimble Box)](#_heading=h.r671e28ya40g)

[Fig 2: ResNet50+GRU128 Model Accuracy & Loss Graph ( Run @ 2019-12-21 23:38 IST)](#_heading=h.gjdgxs)

We conducted more than 150 different experiments with various combinations of models, primarily of two different kinds ***Conv3D*** and ***VGG16/ResNet50+GRU***

Our initial inspiration for model was based on the recommendation in this paper <https://arxiv.org/pdf/1412.0767.pdf>

## General Notes

### LR Monitor Method

*ReduceLRonPlateau* on Val Loss, Decay Rate of 0.5 & Patience of 2   
Other techniques tried  
- Decay Rate of 0.1   
- Early Stopping based on Val\_Acc with min 1% improvement with patience values of 2 and 5.

None of these had significant effect on the end result

### Optimiser

Mostly Adam (barring a few experiments where we tried SGD Momentum). SGD Momentum fared quite poorly wherever it was tried (low accuracy compared to Adam)

### Image Normalisation

Four different techniques were tried

1. Dividing by 255 (simple)
2. Sklearn.Normalise
3. Subtract min and divide by Min-Max difference
4. Subtract mean and divide by Std Dev

It turned out that the simple method of dividing by 255 turned out to be the best. Other techniques where either the same or slightly worse off in terms of effect on accuracy.

### Image Re- Sizing

* Images of size 360x360 were resized to 120x120
* Images of size 120x 140 were cropped to 120x120

## Experiments

### Conv3D 3-layer (8,16,32)

Our Initial model was as follows (Conv3D)   
- 3-layer (8,16,32) model   
- Batch Normalisation at every layer  
- Kernel 3x3x3 & maxpool 2x2x2   
- 2 Dense layers with a Drop Out rate of 0.5 at the end of Conv layers and 0.5 for last Dense layer.

* By varying the dense layer units, we experimented with models ranging in size from 1.8 million to 5.5 million parameters.
* Conducted **7 experiments**
* Batch sizes of 50 & 100 across 10/50 epochs.
* Varied input frames from 15 to 20 per video
* Validation accuracy ranged to **59% - 74%** with not much overfitting noticed
* Batch sizes of 50 & 100 across 10/50 epochs.
* Used a mix of decay rates of 0.25 & 0.5

### Conv3D 3-layer (16,32,64)

To improve accuracy, we then tweaked the above 3-layer model to use a ***16, 32 & 64 units*** (everything else remaining the same).

* By varying the dense layers we experimented with models ranging in size from 3.7 million to 11. million parameters.
* **3 Experiments** varying input frames from 15-30 per video.
* Batch sizes of 50 & 80 across 50 epochs.
* Validation accuracy hovering around **70%** with slight underfitting noticed (i.e, Test accuracy higher than Train accuracy).
  1. Actually dropped to 54% when we pumped in all 30 frames.
* Batch sizes of 50 & 100 across 10/50 epochs.

### Conv3D 4-layer (8,16,32,64)

We then switched to a four-layer model as follows  
***- 8,16,32,64 units***- Batch Normalisation at every layer (as above)  
- Kernel Size of 3 (same as before) as well as   
- MaxPooling size of 2x2x2 after every layer (same as before) in an attempt to increase accuracy

* We now had around 9 million trainable parameters.
* **10 Experiments** conducted
* By now, we graduated to 30 frames per video since that seemed to provide a better accuracy compartively.
* Batch sizes of 32 to 70 across 50 epochs.
* Improved validation accuracy; ranged from 68% - 83% but overfitting in most cases.
* Varied Dropout factor between 0.25 & 0.5 for the Dense layers but not much improvement in overfit.

### Conv3D 4-layer (8,16,32,64) with 1x3x3 Kernel

Tweaked the above architecture as follows  
- Kernel 3x3x3 for the first two layers (same as earlier)  
- Batch Normalisation at every layer (as above)  
***- Kernel 1x3x3 for the 3rd & 4th layer***- 2 Dense Layers of 256 & 128 units (same as earlier)

* Slightly lower number of parameters at around 864k
* **46 Experiments** conducted (with a variation in the architect of having Max Pooling only for 2nd & 4th layer)
* In addition to Batch sizes of 50, 80 & 100, we also tried with batch sizes of 16, 332 & 64 based on feedback from mentor. These powers of 2 sizes would optimize usage of GPU cycles.
* Usage of drop outs at the end of Conv layers (0.25/0.5) and for each Dense layer (0.25/0.5).
* Highest val accuracy achieved **85.95%**.
* Independently, highest Training accuracy achieved **92%**

### Conv3D 4-layer (16,32,64,128), 1x3x3 Kernel, 1 Dense Layer

We seemed to have reached the best accuracy possible at this point. And wanted to explore models which can perform equally well but with fewer parameters. We settled on the following architecture:  
***- 4 layer Conv 3D with 16,32,64 & 128 units***- Batch Normalisation at every layer (as above)- Kernel 3x3x3 for the first two layers (same as earlier)  
- Kernel 1x3x3 for the 3rd & 4th layer (same as earlier)  
***- 1 Dense Layer of 64 units***

* # of parameters dropped to around 509k
* **Conducted 6 experiments** with Batch sizes of 16 & 32 for Epochs of 5, 10, 20 & 30.
* Drop-Out rates of 0.5 at the end of Conv layers and varied between 0.2 & 0.5 for Dense layer.
* Best Accuracy achieved 85.16% Val (corr Train Acc of 92.56%) for Batch size of 32 on 30 epochs - this was a bit of an overfit.

### Conv3D 4-layer (8,16,32,64), 1x3x3 Kernel, 1 Dense Layer - FINAL

Decided to explore further to see if we can do equally with even lower number of parameters. Used the following architecture  
***- 4 layer Conv 3D with 8,16,32,64 units***- Batch Normalisation at every layer (as above)- Kernel 3x3x3 for the first two layers (same as earlier)  
- Kernel 1x3x3 for the 3rd & 4th layer (same as earlier)  
***- 1 Dense Layer of 64 units***

* # of parameters dropped to around 228k
* **Conducted 39 experiments** with Batch sizes of 16 & 32 for Epochs of 5, 10, 20, 30, 50 & 100.
* Experimented with various drop out values at the Dense layer (0.1, 0.2, 0.25, 0.35, 0.5) and 0.25 & 0.5 at the end of Conv layers.
* **Best accuracy reached was 84.82% Val with 82.7% Training with drop-out values of 0.5 at the end of Conv layers and 0.2 for dense layer.**
* **The above code was run on Nimble Box again and obtained an accuracy of 84.375% Val and 79.167% Train**

### CNN + RNN (VGG16 + GRU)

In order to explore other options, we experimented on CNN+RNN model as follows  
- VGG16 (Top Layer FALSE, Trainable=TRUE)  
- GRU with 64 units  
- Various combinations of 1 & 2 Dense Layers of sizes 128 & 256

* # of parameters ranged from 14 to 15 million
* Conducted **12 experiments** with drop outs varying.
* SIX of these experiments faced an Out Of Memory issue
* Best accuracy achieved for the other 6 successful experiments was 34% Validation.

### CNN + RNN (ResNet50 + GRU)

Modified the above model to the following  
- ResNet50 (Top Layer FALSE, Trainable=TRUE)  
- GRU with 128 units  
- 2 Dense Layers of sizes 256 & 128

* # of parameters ballooned to 26.7 million (not surprising)
* Conducted **26 experiments**
* **Three** of the above 26 were conducted with Trainable=FALSE but they yielded comparatively poor accuracy values.
* **Best accuracy reached, at 97.32 % with no overfitting!!**
* But at a model file size of more than 300 MB, this would be an unwieldy model given the form factor of the device where the model would be loaded (Webcam)

### Conv3D 4-layer (8,16,32,64), 2x2x2 Kernel

Just as an alternative, we also tried a variation of the 4-layer, using a 2x2x2 kernel instead of 3x3x3 with two variants (2 Dense Layers of 256 & 128 and 1-Dense layer of 64)

* 16 Experiments were run with these variants - 3 with the 2-dense layers and rest with the 1-dense layer
* # of parameters for the 2-Dense layer went up to around 868k with pretty good accuracy (79% - 82%) but with overfitting.
* We decided not to pursue this further since the other 1-dense layer had smaller # of parameters.
* # of parameters for the 1-Dense layer came down to around 220k with accuracies comparable to our final 4-layer, 3-kernel, 1-Dense model but overfitting problem.

## Errors Encountered

* Fewer frames but more layers (with all 3x3x3) threw negative dimension error for maxpool
* LSTM error (ValueError: This model has not yet been built. Build the model first by calling build() or calling fit() with some data. Or specify input\_shape or batch\_input\_shape in the first layer for automatic build.

## Conclusion

Even though we got a very high accuracy of 97.32% with the ResNet50 model, we decided to go with a lighter model with an accuracy of ~84%.

The ResNet50 model is more than 300 MB in size whereas the lighter model is just 2.83 MB with a hit on accuracy. Given the consideration of small form-factor of the device where the model would be loaded, we decided to go with the lighter model.

## Accuracy & Loss Graphs

#### *Fig 1 (FInal Mode) : Conv3D Model (8,16,32,64; 0.5D/o) + 1 Dense (64;0.2 D/o) - Batch Size 32 - 50 Epochs (Nimble Box)*

#### Fig 2: ResNet50+GRU128 Model Accuracy & Loss Graph ( Run @ 2019-12-21 23:38 IST)



