**Gesture Recognition - Case Study**

By

Treza Jacob Kannampilly

# Introduction:

In this collaborative project, the task at hand involves constructing a 3D Convolutional Model capable of accurately predicting five distinct gestures. Before delving into the project, it is imperative to import the requisite libraries for initialization.

# Problem Statement:

Picture yourself in the role of a data scientist employed by a leading home electronics company specializing in cutting-edge smart television technology. The aim is to introduce an innovative feature into the smart TV interface: the ability to discern and respond to five distinct gestures performed by users, thereby offering intuitive control without the need for a traditional remote.

The gestures are captured in real-time via the TV-mounted webcam, each corresponding to a specific command:

1. Thumbs up: Increase volume
2. Thumbs down: Decrease volume
3. Left swipe: Rewind 10 seconds
4. Right swipe: Fast-forward 10 seconds
5. Stop: Pause playback

This case study will encompass the development of multiple models, involving varied experiments with model architectures, convolutional neural network (CNN) layers, transfer learning techniques, as well as image normalization and augmentation methods.

## Experiment 1: Generator function image resize

**Decision:**

The decision is to utilize the first 10 image frames from each video without cropping, instead resizing them to a dimension of 180x180.

**Input:**

img\_idx = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Batch size = 32

Dimensions: x = 10, y = 180, z = 180

**Model:**

In this experiment, a 3D Convolutional Neural Network (CNN) model is employed, consisting of 4 layers of Conv3D with filter sizes of 64, 128, 256, and 256 respectively. The architecture includes 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. The activation function used in hidden layers is "relu," while "softmax" is applied in the output layer. The first pooling layer before the 2nd Conv3D layer has a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2), facilitating spatial pooling. The subsequent pooling layer adopts a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2), enabling spatiotemporal pooling. These pooling layers contribute to shrinking the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with filter sizes of 128 and 5 (representing classes).

**Parameters:**

* Optimizer: SGD (lr=0.001)
* Epoch: 10
* Total params: 334,659,845
* Trainable params: 334,658,949
* Non-trainable params: 896

**Result:** Encounter of an error - ResourceExhaustedError: OOM when allocating tensor with GPU.

**Conclusion:**

*The encountered error indicates an "out of memory" (OOM) issue, implying GPU memory depletion. Several strategies can be explored to address this challenge:*

* *Attempt to reduce image sizes in subsequent experiments.*
* *Consider reducing the number of layers in the model architecture.*
* *Introduce MaxPooling2D layers after convolutional layers.*
* *Adjust the batch size (or increase steps\_per\_epoch and validation\_steps).*
* *Decrease the number of filters in Dense and Conv2D layers.*

## Experiment 2: Image Resize and Crop using Generator Function

**Decision:**

* Utilizing the first 10 image frames from each video.
* Incorporating two dimensions of images present in the dataset: 360x360 and 120x160.
* Cropping 120x160 images to 120x120 to ensure symmetry in dimensions.
* Resizing all input images to 80x80 to mitigate potential errors with varying shapes in data, as Conv3D models are sensitive to input dimensions.

**Input :**

img\_idx = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Batch size = 32

Dimensions: x = 10, y = 80, z = 80

**Model:**

This experiment employs a 3D Convolutional Neural Network (CNN) model comprising 4 layers of Conv3D with filter sizes of 64, 128, 256, and 256 respectively. The architecture integrates 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The first pooling layer before the 2nd Conv3D layer utilizes a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2) for spatial pooling. The subsequent pooling layer employs a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2) for spatiotemporal pooling. These pooling layers shrink the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with a filter size of 512 and 5 (representing classes).

**Parameters:**

* Optimizer: SGD (lr=0.001)
* Epoch: 10
* Total params: 265,030,149
* Trainable params: 265,029,253
* Non-trainable params: 896

**Result:** Training Accuracy: **0.78** Validation Accuracy: **0.23**

**Conclusion:**

Resizing images from 180x180 to 80x80 significantly reduced the model parameters while maintaining decent validation (78%) and training (23%) accuracy with 10 epochs.

The noticeable difference in accuracy between training and validation suggests overfitting due to the limited input data (10 images per video). Increasing the dataset size is necessary to enhance model generalization.

Cropping the 120x160 images to 120x120 ensured proper centering on the object, albeit at the expense of discarding pixels from the wider image.

## Experiment 3: Increased Number of Images in Input & Reduced Filters in Model

**Decision:**

* Incorporating the first 18 image frames from each video.
* Dividing the 30 video frames into 3 slots of 10 images each: slot1 [0:9], slot2 [10:19], slot3 [20:29]. Within each slot, selecting 6 images by choosing 2 images from the beginning, 2 from the end, and 2 from the middle.
* Reducing the number of filters in Conv3D layers to alleviate memory usage.

**Input :**

* img\_idx = [0,1,4,5,8,9,10,11,14,15,18,19,20,21,24,25,28,29]
* batch size = 32
* x = 18
* y = 80
* z = 80

**Model:**

This experiment utilizes a 3D Convolutional Neural Network (CNN) model with 4 layers of Conv3D, with filter sizes reduced to 32, 64, 128, and 256 respectively. Additionally, the architecture integrates 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The first pooling layer before the 2nd Conv3D layer applies a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2) for spatial pooling. The subsequent pooling layer utilizes a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2) for spatiotemporal pooling. These pooling layers reduce the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with a filter size of 512 and 5 classes.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 10
* Total params: 473,027,973
* Trainable params: 473,027,269
* Non-trainable params: 704

**Result:** Model did not fit due to memory exhaust error.

**Conclusion:**

* The model's inability to fit due to memory exhaustion suggests an excess of parameters, leading to extended training times.
* A reduction in the size of images while maintaining a larger number of input images could alleviate memory constraints and ensure maximum data fed to the model for training.

## Experiment 4: Increased Number of Images in Input, Reduced Image Size, and Filters in Model

**Decision:**

* Incorporating 18 image frames from each video.
* Dividing the 30 video frames into 3 slots of 10 images each: slot1 [0:9], slot2 [10:19], slot3 [20:29]. Within each slot, selecting 6 images by choosing 2 images from the beginning, 2 from the end, and 2 from the middle, totaling 18 images.
* Reducing the image size from 80x80 to 60x60.
* Further reducing the number of filters in Conv3D layers.

**Input :**

* img\_idx = [0,1,4,5,8,9,10,11,14,15,18,19,20,21,24,25,28,29]
* batch size = 64
* x = 18
* y = 60
* z = 60

**Model:**

This experiment utilizes a 3D Convolutional Neural Network (CNN) model with 4 layers of Conv3D, employing filter sizes of 32, 64, 128, and 256 respectively. The architecture integrates 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The first pooling layer before the 2nd Conv3D layer applies a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2) for spatial pooling. The subsequent pooling layer utilizes a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2) for spatiotemporal pooling. These pooling layers reduce the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with a filter size of 512 and 5 classes.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 268,306,949
* Trainable params: 268,306,053
* Non-trainable params: 896

**Result:** Model did not fit due to memory exhaust error.

**Conclusion:**

* Despite a significant reduction in model parameters achieved by reducing the image size, the model still failed to fit due to the high number of input images the GPU can handle, leading to an out-of-memory error.
* To address this issue, it is necessary to further reduce the number of input images in subsequent experiments.

## Experiment 5(a): Maximizing Batch Size for GPU Utilization

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Maintaining image size at 60x60.
* Attempting to increase batch size to 128, then 96 to leverage GPU capacity.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 128 or 96
* x = 15
* y = 60
* z = 60

**Model:**

In this experiment, a 3D Convolutional Neural Network (CNN) model with 4 layers of Conv3D is utilized, featuring filter sizes of 64, 128, 256, and 256 respectively. The architecture incorporates 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The first pooling layer before the 2nd Conv3D layer employs a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2) for spatial pooling. The subsequent pooling layer adopts a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2) for spatiotemporal pooling. These pooling layers reduce the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with a filter size of 512 and 5 classes.

**Parameters**:

* Model failed to build, thrown OOM error

**Result:** Throws an error.

OOM when allocating tensor with shape[460800,512] and type float on /job:localhost/replica:0/task:0/device:GPU:0 by allocator GPU\_0\_bfc [Op:Add]

**Conclusion:**

* *The OOM error indicates GPU memory depletion, preventing tensor allocation.*
* *To address this issue, considering a smaller batch size or adjusting steps\_per\_epoch and validation\_steps is necessary to avoid memory overload.*

## Experiment 5(b): Utilizing Batch Size Greater than 32 and Less than 96 with Reduced Filters

**Decision:**

* Set the batch size to 64
* Set the Epoch to 30

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 60
* z = 60

**Model:**

In this experiment, a 3D Convolutional Neural Network (CNN) model with 4 layers of Conv3D is employed, featuring reduced filter sizes of 32, 64, 128, and 128 respectively. The architecture integrates 3 BatchNormalization layers, 2 Dropout layers, 2 MaxPooling3D layers, and 2 additional Dropout layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The first pooling layer before the 2nd Conv3D layer utilizes a kernel size of (1 × 2 × 2) and a stride of (1 × 2 × 2) for spatial pooling. The subsequent pooling layer adopts a kernel size of (2 × 2 × 2) and a stride of (2 × 2 × 2) for spatiotemporal pooling. These pooling layers shrink the output size of the 3-D CNN component by ratios of 4 and 2 in spatial size and temporal length respectively, focusing on short-term spatiotemporal features. The model concludes with 2 Dense layers with a filter size of 256 and 5 classes.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 59,706,629
* Trainable params: 59,706,181
* Non-trainable params: 448

**Result:** Training Accuracy: **0.82** Validation Accuracy:  **0.32**

**Conclusion:**

* Attempts with batch sizes of 128 and 96 resulted in out-of-memory errors, prompting the selection of a batch size of 64.
* Utilizing a larger batch size facilitates faster evaluation/prediction processes and potentially better approximations.
* While achieving good accuracy in the training dataset, the model's performance on the validation set suggests overfitting, indicating the need for further data generalization.
* The use of only 15 image frames out of 30 might limit the model's capacity for capturing temporal features effectively. Further exploration with additional frames may be beneficial.

## Experiment 6: CCN + RNN Model with LSTM.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Maintaining image size at 60x60.
* Setting batch size to 64.
* Incorporating LSTM to address vanishing gradient or exploding gradient problems, allowing the model to handle sequential data effectively.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 60
* z = 60

**Model:**

This experiment integrates 4 layers of Conv2D with filter sizes of 16, 32, 64, and 128 respectively. The architecture features 4 BatchNormalization layers, 2 Dropout layers, and 4 MaxPooling2D layers with filter sizes of (3,3) in Conv2D and (2,2) in MaxPooling2D layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The model concludes with 2 Dense layers with a filter size of 64 and 5 classes. A LSTM layer with 64 cells is incorporated to leverage recurrent connections for handling sequential data.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 414,437
* Trainable params: 413,957
* Non-trainable params: 480

**Result:** Training Accuracy: **0.42** Validation Accuracy:  **0.18**

**Conclusion:**

* Despite utilizing LSTM for handling sequential data and maintaining a reasonable number of model parameters, the model's performance on both training and validation datasets is poor after 30 epochs.
* Increasing the image size to 120x120 in future experiments may potentially improve accuracy, as larger images may capture more intricate details and features.

## Experiment 7: CNN + RNN Model(LSTM) with increased image size.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Increasing the image size to 80x80.
* Setting batch size to 64.
* Incorporating LSTM in the model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 80
* z = 80

**Model:**

This experiment integrates 4 layers of Conv2D with filter sizes of 16, 32, 64, and 128 respectively. The architecture features 4 BatchNormalization layers, 2 Dropout layers, and 4 MaxPooling2D layers with filter sizes of (3,3) in Conv2D and (2,2) in MaxPooling2D layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The model concludes with 2 Dense layers with a filter size of 64 and 5 classes. A LSTM layer with 64 cells is incorporated to leverage recurrent connections for handling sequential data.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 938,725
* Trainable params: 938,245
* Non-trainable params: 480

**Result:** Training Accuracy: **0.55** Validation Accuracy:  **0.34**

**Conclusion:**

* Despite increasing the image size to 80x80, the model's accuracy on both training and validation datasets remains relatively low after 30 epochs.
* The increase in model parameters compared to the previous experiment suggests a more complex model, yet it did not lead to significant improvement in accuracy.
* Considering trying GRU in future experiments to explore if it could potentially improve the model's performance.

## Experiment 8: CNN + RNN Model with GRU

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Increasing the image size to 120x120.
* Setting batch size to 64.
* Implementing GRU in the model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 120
* z = 120

**Model:**

This experiment integrates 4 layers of Conv2D with filter sizes of 16, 32, 64, and 128 respectively. The architecture features 4 BatchNormalization layers, 2 Dropout layers, and 4 MaxPooling2D layers with filter sizes of (3,3) in Conv2D and (2,2) in MaxPooling2D layers. Activation functions include "relu" in hidden layers and "softmax" in the output layer. The model concludes with 2 Dense layers with a filter size of 64 and 5 classes. A GRU layer with 64 cells is incorporated to leverage its ability to handle sequential data with fewer parameters than LSTM.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 1,319,781
* Trainable params: 1,319,301
* Non-trainable params: 480

**Result:** Training Accuracy: **0.89** Validation Accuracy:  **0.38**

**Conclusion:**

* Despite achieving a high training accuracy of 0.89, the model's performance on the validation dataset remains relatively low after 30 epochs.
* Overfitting is observed in the model, suggesting that it is fitting too closely to the training data and failing to generalize well to unseen data.
* The increase in model parameters compared to previous experiments might have contributed to the overfitting issue.
* Considering exploring GRU with transfer learning using the VGG16 model in future experiments to potentially improve model performance and mitigate overfitting.

## Experiment 9: Transfer Learning with VGG16 and GRU.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Setting image shape to 120x120.
* Setting batch size to 64.
* Utilizing heavy-duty transfer learning architecture VGG16 with GRU in this model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 120
* z = 120

**Model:**

In this experiment, we utilize the VGG16 architecture for transfer learning. At the end of the model, we include 2 Dense layers with filter size 8 and 5, along with 2 GRU layers. The first GRU layer consists of 32 cells, followed by a second GRU layer with 16 cells.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 15,021,653
* Trainable params: 306,965
* Non-trainable params: 14,714,688

**Result:** Training Accuracy: **0.26** Validation Accuracy:  **0.27**

**Conclusion:**

* After 30 epochs, both the training and validation accuracies remain relatively low, indicating underfitting of the model.
* The number of trainable parameters in the model is lower compared to Experiment 8, potentially contributing to its underfitting.
* Considering exploring MobileNet transfer learning with LSTM in future experiments to further investigate model performance and potentially improve accuracy.

## Experiment 10: Transfer Learning with MobileNet and LSTM.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Setting image shape to 120x120.
* Setting batch size to 64.
* Utilizing lightweight transfer learning architecture MobileNet with LSTM in this model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 120
* z = 120

**Model:**

In this experiment, we employ the lightweight MobileNet architecture. At the end of the model, we include 2 Dense layers with a filter size of 64 and 5, utilizing LSTM layers with 64 LSTM cells.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 3,840,453
* Trainable params: 609,541
* Non-trainable params: 3,230,912

**Result:** Training Accuracy: **0.43** Validation Accuracy:  **0.51**

**Conclusion:**

* After 30 epochs, the model demonstrates improved validation accuracy compared to previous experiments.
* The model does not exhibit overfitting, indicating that it is generalizing well to the validation dataset.
* The number of trainable parameters is slightly lower compared to Experiment 9 with VGG16 and GRU, suggesting that the MobileNet architecture may be more suitable for this task.
* Considering employing image augmentation in future experiments to further enhance model performance and address generalization issues observed in previous experiments.

## Experiment 11: CNN + RNN Model(LSTM) with augmentation image size.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Keeping image size at 120x120.
* Employing image cropping augmentation to address potential overfitting issues observed in Experiment 8.
* Setting batch size to 64.
* Utilizing LSTM in this model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 120
* z = 120

**Model:**

In this experiment, we utilize 4 layers of Conv2D (Filters: 16, 32, 64, 128), along with 4 BatchNormalization, 2 Dropout, 4 MaxPooling2D, and 2 Dropout layers. ReLU is used as the activation function in hidden layers, and softmax in the output layer. We use a filter size of (3,3) in Conv2D and (2,2) in MaxPooling2D layers. At the end of the model, we include 2 Dense layers with a filter size of 64 and 5, utilizing 128 LSTM cells in the LSTM layer

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 30
* Total params: 3,392,869
* Trainable params: 3,392,389
* Non-trainable params: 480

**Result:** Training Accuracy: **0.78** Validation Accuracy:  **0.62**

**Conclusion:**

* After 30 epochs, the model does not demonstrate significant improvement in accuracy on both training and validation datasets.
* Despite employing image cropping augmentation, the model still exhibits overfitting issues.
* The number of parameters in the model is higher compared to Experiment 10, indicating that reducing parameters did not contribute to better performance.
* Considering exploring GRU with augmentation in future experiments to address overfitting and potentially improve model accuracy.

## Final Experiment 12: Transfer Learning with MobileNet and LSTM with more augmentation.

**Decision:**

* Utilizing 15 image frames from each video, selecting alternate images from a total of 30 frames.
* Keeping image size at 120x120.
* Implementing more extensive image augmentation techniques, including rotation, cropping, and shifting, to address previous overfitting issues.
* Setting batch size to 64.
* Utilizing LSTM in this model.

**Input :**

* img\_idx = [0,2,4,6,8,10,12,14,16,18,20,22,24,26,28]
* batch size = 64
* x = 15
* y = 120
* z = 120

**Model:**

In this experiment, we utilize the lightweight MobileNet architecture. At the end of the model, we include 2 Dense layers with a filter size of 64 and 5, utilizing LSTM layers with 64 LSTM cells.

**Parameters**:

* Optimizer: SGD (lr=0.001)
* Epoch: 20
* Total params: 3,516,229
* Trainable params: 285,317
* Non-trainable params: 3,230,912

**Result:** Training Accuracy: **0.98** Validation Accuracy:  **0.78**

**Conclusion:**

* After 20 epochs, the model demonstrates excellent accuracy on both training and validation datasets.
* The number of parameters in the model is significantly reduced compared to previous experiments, contributing to efficient performance.
* The extensive image augmentation techniques, combined with MobileNet and LSTM, have resulted in the best accuracy achieved among all experiments, with minimal parameters.
* This final model appears to be well-optimized, providing a balance between accuracy and efficiency.

**Summary:**

Throughout the experiments, various approaches were explored to address overfitting and improve model accuracy. Despite encountering challenges such as GPU memory issues and overfitting, the final experiment using MobileNet with LSTM and extensive augmentation techniques yielded the best results. This model achieved a remarkable training accuracy of 97% and a validation accuracy of 81%, demonstrating effectiveness in both performance and efficiency.

**Final Model Accuracy: Train 97% and Val 81% .**

