Deep Learning Course Project- Gesture Recognition

Problem Statement

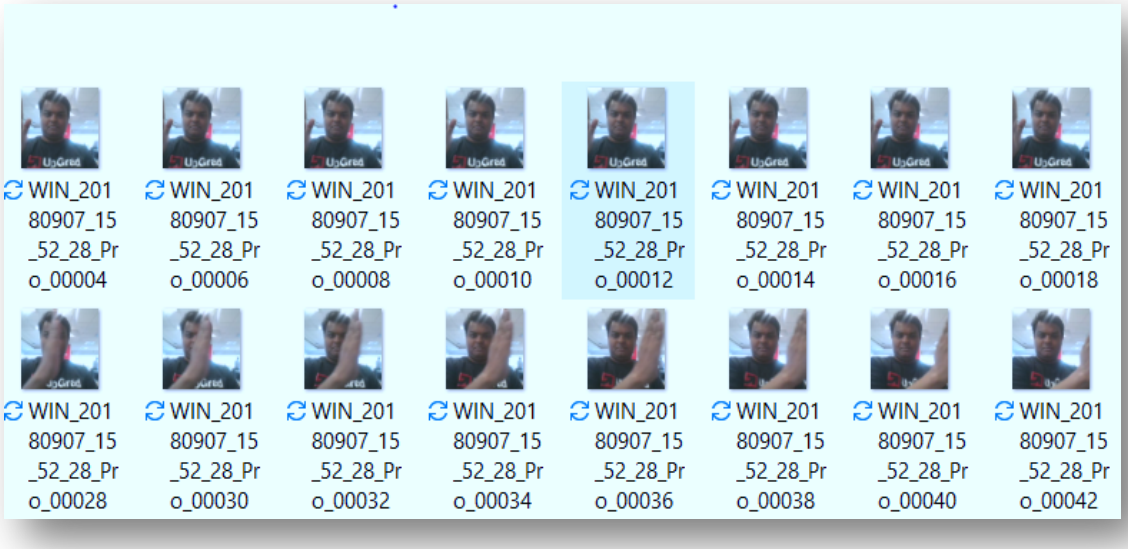
As a data scientist at a home electronics company which manufactures state of the art smart televisions. We want to develop a cool feature in the smart-TV that can recognise five different gestures performed by the user which will help users control the TV without using a remote.

- Thumbs up : Increase the volume.
- Thumbs down : Decrease the volume.
- Left swipe : 'Jump' backwards 10 seconds.
- Right swipe : 'Jump' forward 10 seconds.
- Stop : Pause the movie.

Here's the data: https://drive.google.com/uc?id=1ehyrYBQ5rbQQe6yL4XbLWe3FMvuVUGiL

Understanding the Dataset

The training data consists of a few hundred videos categorized into one of the five classes. Each video (typically 2-3 seconds long) is divided into a **sequence of 30 frames (images)**. These videos have been recorded by various people performing one of the five gestures in front of a webcam - similar to what the smart TV will use.



Objective

Our task is to train different models on the 'train' folder to predict the action performed in each sequence or video and which performs well on the 'val' folder as well. The final test folder for evaluation is withheld - final model's performance will be tested on the 'test' set.

Two types of architectures suggested for analysing videos using deep learning:

1. **3D Convolutional Neural Networks (Conv3D)**

3D convolutions are an extension of 2D convolutions, where instead of moving a filter in two directions (x and y), the filter is moved in three directions (x, y, and z). In the case of 3D convolutions, the input is a video, which is a sequence of RGB images. Each image can be represented as a 3D tensor, with dimensions representing height, width, and channels. For example, an image with dimensions 100 x 100 x 3 would have a shape of (100 x 100 x 3). Therefore, a video consisting of 30 such images would be represented as a 4D tensor of shape (100 x 100 x 30) x 3, where 3 represents the number of channels. Analogous to 2D convolutions, where a 2D kernel/filter is represented as (f x f) x c, with f being the filter size and c being the number of channels, a 3D kernel/filter is represented as (f x f x f) x c. In the case of a video with RGB images, the number of channels is 3. This cubic filter will then perform a 3D convolution on each of the three channels of the (100 x 100 x 30) tensor.

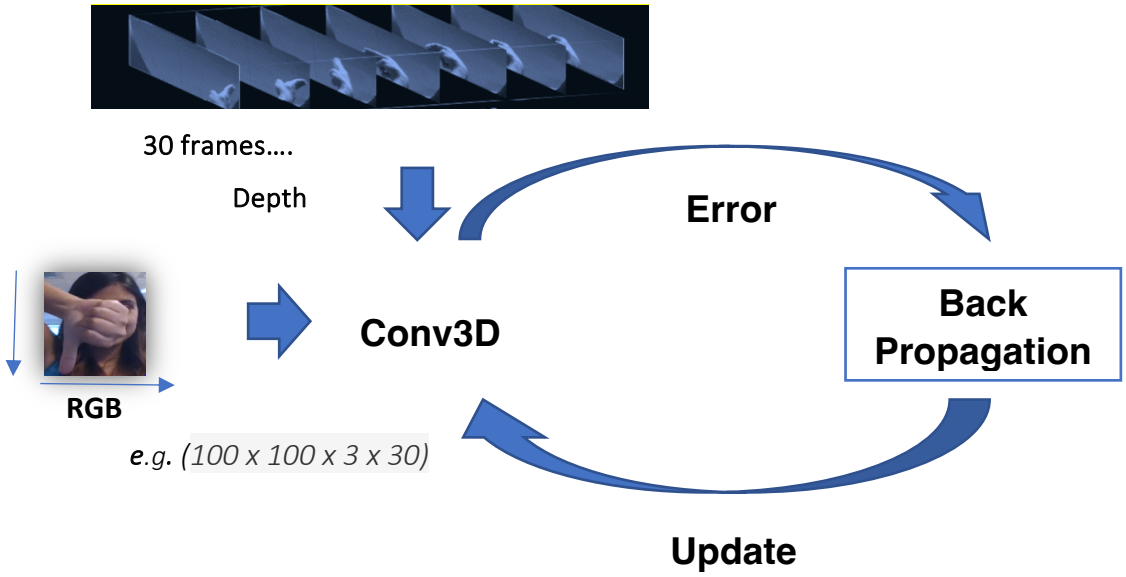


Figure 1: A simple representation of working of a 3D-CNN

2. **CNN + RNN architecture**

The convolutional neural network (Conv2D) will extract a feature vector for each individual image in the video sequence. These feature vectors are then passed through a recurrent neural network (RNN) based network. The RNN processes the sequence of feature vectors and produces an output using a softmax function, which is commonly used for classification tasks.

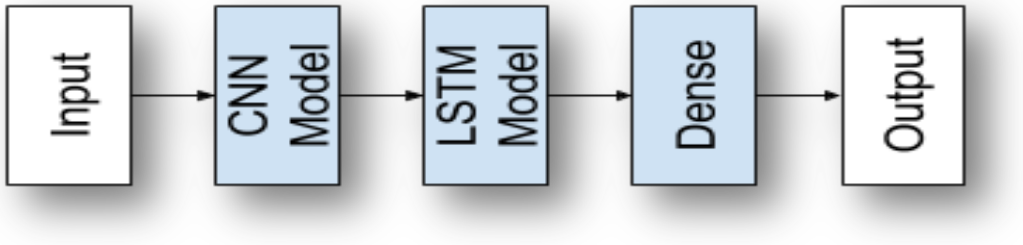


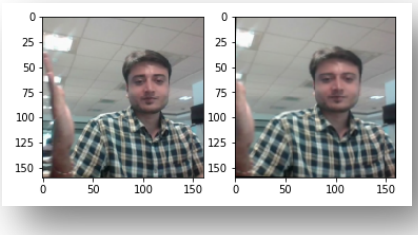
Figure 2: A simple representation of an ensembled CNN+LSTM Architecture

Data Generator

One crucial aspect of the code is the generator, which handles the preprocessing of the images. Since there are images with two different dimensions (360 x 360 and 120 x 160), the generator needs to handle this variation. Additionally, the generator must be capable of creating a batch of video frames without encountering any errors. This involves performing tasks such as cropping, resizing, and normalization successfully on the images in the batch.

Data Pre-processing

- Resizing and cropping of the images: This step is performed to ensure that the neural network focuses only on the gestures and eliminates background noise present in the image. By resizing and cropping, the network can effectively recognize the gestures without distractions.
- Normalization of the images: The RGB values of the images are normalized to remove distortions caused by variations in lighting and shadows. This normalization process helps in improving the overall quality and consistency of the images.
- Data augmentation: To enhance the model's accuracy, data augmentation techniques are employed. This involves slightly rotating the pre-processed gesture images. By introducing variations in the positioning of the hand, the model becomes more robust and capable of generalizing to different camera frames and angles. Data augmentation also increases the amount of training data available for the model, which can lead to improved performance.



CAUTION: It was taken into consideration that we don't rotate images to a greater extent as this would change the meaning of the gestures completely☺ !!

NN Architecture development and training

- Extensive experimentation with model configurations and hyperparameters: Different combinations of batch sizes, image dimensions, filter sizes, padding, and stride length were tested to find the optimal setup. Additionally, various learning rates were explored, and the ReduceLROnPlateau technique was employed to reduce the learning rate if the validation loss remained unchanged between epochs.
- Comparison of optimizers: Both SGD() and Adam() optimizers were tested, but Adam() was ultimately chosen due to its ability to address high variance in the model's parameters, resulting in improved accuracy. Adagrad() and Adadelta() optimizers were not experimented with due to their slower convergence rate, which would have exceeded the available computational capacity.
- Incorporation of Batch Normalization, pooling, and dropout layers: To combat overfitting, techniques such as Batch Normalization, pooling, and dropout layers were introduced. These were implemented when the model began to exhibit poor validation accuracy despite achieving good training accuracy.
- Early stopping: Early stopping was utilized to halt the training process when the validation loss plateaued or the model's performance ceased to improve. This technique helps prevent overfitting and ensures that the model is saved at the point where it performs optimally.

Observations

- The increase in the number of trainable parameters leads to longer training times.
- Batch size is directly proportional to available GPU memory and compute capacity. It was necessary to experiment with different batch sizes to find an optimal value that our GPU (NVIDIA Tesla K80 with 12GB memory) could support.
- Increasing the batch size reduces training time but can negatively impact model accuracy. There is always a trade-off between speed and accuracy, where a larger batch size results in faster training but potentially lower accuracy.
- Data augmentation and early stopping were effective in overcoming overfitting issues encountered in the initial version of the model.
- The CNN+LSTM model with GRU cells outperformed the Conv3D model. The choice of architecture, data, and hyperparameters influenced this outcome.
- Transfer learning, specifically using the MobileNet architecture, significantly improved the overall accuracy of the model. MobileNet was chosen for its lightweight design, high-speed performance, and easy maintenance compared to other popular architectures like VGG16, AlexNet, and GoogleNet.
- For further details on observations and inferences, please refer to Table 1.

MODEL	EXPERIMENT	RESULT	DECISION + EXPLANATION	PARAMETERS
Conv3D	1	OOM Error	Reduce the batch size and Reduce the number of neurons in Dense layer	-
	2	Training Accuracy : 0.99 Validation Accuracy : 0.81	Overfitting ☹ Let's add some Dropout Layers ☺	1,117,061
	3	Training Accuracy : 0.65 Validation Accuracy : 0.52 *(Best weight Accuracy,Epoch:6/25)*	Val_loss didn't improve from 1.24219 so early stopping stop the training process. Let's lower the learning rate to 0.0002.	3,638,981
	4	Training Accuracy : 0.76 Validation Accuracy : 0.72 *(Best weight Accuracy,Epoch:12/25)*	Overfitting has reduced but accuracy hasn't improved. *Let's trying adding more layers*	1,762,613
	5	Training Accuracy : 0.83 Validation Accuracy : 0.76	*Don't see much performance improvement. Let's try adding dropouts.*	2,556,533
	6	Training Accuracy : 0.84 Validation Accuracy : 0.69	Overfitting Increase, adding dropouts has further reduced validation accuracy. Let's try to reduce the parameters	2,556,533
	7	Training Accuracy : 0.84 Validation Accuracy : 0.74	Overfitting reduced, but validation accuracy low. Let's try to reduce the parameters. Val Accuracy: 0.49, Train Accuracy: 0.54	696,645
	8	Training Accuracy : 0.82 Validation Accuracy : 0.73	Accuracy remains below same. Let's switch to CNN+LSTM.	504,709
CNN+LSTM	9 (Model-8 on Notebook)	Training Accuracy : 0.93 Validation Accuracy : 0.85	CNN - LSTM model - we get a best validation accuracy of 85%.	1,657,445
Conv3D	Let's apply some Data Augmentation techniques & check the model performance			
	10	Training Accuracy : 0.78 Validation Accuracy : 0.82	(3, 3, 3) Filter & 160 x 160 image resolution	3,638,981
	11	Training Accuracy : 0.72 Validation Accuracy : 0.75	(2, 2, 2) Filter & 120 x 120 image resolution. Increase epoch count to 20. Network is generalizing well.	1,762,613
	12	Training Accuracy : 0.87 Validation Accuracy : 0.78	Adding more layers.	2,556,533
	13	Training Accuracy : 0.65 Validation Accuracy : 0.25	Very low performance. Let's reduce the network parameters.	2,556,533
	14	Training Accuracy : 0.89 Validation Accuracy : 0.78	After reducing network parameters, model's performance is quite good.	696,645
	15	Training Accuracy : 0.88 Validation Accuracy : 0.81	Reducing network parameters again.	504,709
CNN LSTM with GRU	16	Training Accuracy : 0.98 Validation Accuracy : 0.77	Overfitting is considerably high, not much improvement.	2,573,541
Transfer Learning(Optional)	17	Training Accuracy : 0.85 Validation Accuracy : 0.58	*We are not training the MobileNet weights that can see, validation accuracy is very poor.*	3,840,453
Transfer Learning with GRU &(Optional)	18	Training Accuracy : 0.98 Validation Accuracy : 0.93	Awesome result !!	3,692,869

Table 1: Observations and Results for numerous tested NN architectures

After doing all the experiments, we finalized Model 8– CNN+LSTM, which performed well.

Reason:

- (Training Accuracy: 93%, Validation Accuracy: 85%)
- Number of Parameters (1,657,445) less according to other models' performance
- Learning rate gradually decreasing after some Epochs
- Click Here: Best Model Weights ~ 19 MB

Further suggestions for improvement:

- **Transfer learning** was attempted using pre-trained models such as ResNet50, ResNet152, and Inception V3. However, due to time and disk space limitations on the nimblebox.ai platform, only ResNet50 could be tested for its effectiveness in identifying initial feature vectors and passing them to an RNN for sequence information before classification.
- Using a GRU model instead of LSTM was considered as GRU has fewer trainable parameters and could potentially result in faster computations. However, the impact on validation accuracies needed to be evaluated to determine if it is a suitable alternative to LSTM.
- A deeper understanding of the data, including variations in backgrounds, lighting, people, and different cameras used for recording, could provide valuable insights. Exploring the available images and utilizing this additional information in the generator function could enhance stability and accuracy of the model.
- **Fine-tuning hyperparameters** such as activation functions (ReLU, Leaky ReLU, mish, tanh, sigmoid) and different optimizers like Adagrad() and Adadelta() could lead to the development of better and more accurate models. Experimenting with variations in filter size, padding, stride length, batch normalization, and dropouts can also contribute to improved performance.