GESTURE RECOGNITION – CASE STUDY

GESTURE RECOGNITION CASE STUDY

**Problem Statement:**

To build a model and load it into a webcam mounted on a TV, which reads gestures as a sequence of images and translates it into a specific command.

* 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

**Given Data:**

The data consists of videos in which a gesture is shown across 30 frames or images. There are two folders – Train, Val. Each contains multiple count of gesture sequence recorded, with 30 frames per sequence.

**Aim:**

A Deep Learning model should be trained on the data and then to be loaded on the TV’s webcam. The model should be able to detect the gesture shown by the user of the TV. Both accuracy and time taken must be maintained in optimal range.

**Baseline understanding:**

With Deep Networks, that convolve images to extract useful information called as CNNs, it is possible to read the sequence of images and register the gesture in the sequence.

Since CNN requires images to be loaded as tensors, a video which is a sequence of images, can be translated as a stacked tensor. This means, a normal image which is length \* breadth \* channel depth becomes stack \* length \* breadth \* channel depth.

3

2

1

Folder list

Number of batches \* batch size

1st image with RGB (3) channels

0

0

1st batch containing number of folders equal to batch size

0

4

3

2

1

5

1st folder containing sequence of images

**0**

**1**

**2**

**4**

**3**

**5**

**7**

**6**

0

**0**

Csv file with list of folders

**Batches of 4D Tensors – Sourcing, looping and sending as input**

**Explaining the process:**

1. Import libraries
2. Import and load data into a local directory
3. Read csv files for the folder names and extract images from the folder names
4. Create a generator to load with a random permutation, the data
   1. The generator should yield a batch of sequence, the size determined by batch size
   2. The images within the sequence should be normalized
   3. The images within the sequence should be resized to one size
   4. The yield shall be 4D Tensors count equal to the number of batch size and their corresponding class label
5. Passing data to the model:
   1. If Conv3D Model:
      1. Create a DL CNN model that takes in 4D Tensor data
   2. If CNN + RNN Model:
      1. Import standard model for feature extraction from image (CNN) and then pass it to a Time distributed layer adding then either LSTM or GRU cells to record sequence data.
   3. The final layer shall give 5 values (each value is a probability of each class)
   4. ***The activation function for the final layer shall be a softmax function***
6. Compiling the model will be with:
   1. A suitable optimizer functions
   2. Categorical Cross Entropy loss function
   3. Metric shall be Categorical Accuracy
7. Check the model summary to verify parameters used
8. Instantiate the train and validation generator objects
9. Create a folder for each model structure + hyperparameters to store model stats
10. Use keras callbacks API:
    1. Model Check point - to store stats from each epoch
    2. Reduce LR On Plateau – to reduce learning rate if no change in gradient between epochs (loss does not reduce)
11. Pass a sequence of 4D tensors each epoch using steps per epoch
12. Use keras fit generator to
    1. Forward propagation as many as steps per epoch in one go
    2. Perform backpropagation after steps per epoch 4D tensors have been passed
13. Verify training and validation accuracy and make changes to model structure and/or hyperparameters and repeat till satisfactory results are obtained.

**Recorded Results:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Parameter**  **Count** | **Highest**  **Validation**  **Accuracy** | **Train**  **Accuracy** | **Epoch**  **Count** | **Execution**  **Time** | **Execution Time Per Epoch** |
| model1 | 4001669 | 0.88 | 0.968 | 20 | 812.4169 | 40.62085 |
| model2 | 3586853 | 0.9 | 0.86 | 20 | 706.3572 | 35.31786 |
| model3 | 27807493 | 0.88 | 1 | 20 | 729.2215 | 36.46108 |
| model4 | 4001669 | 0.9 | 1 | 20 | 724.7269 | 36.23634 |
| model5 | 4001669 | 0.91 | 0.994 | 15 | 540.0579 | 36.00386 |
| model6 | 3586853 | 0.91 | 0.988 | 15 | 533.9697 | 35.59798 |
| model7 | 3571717 | 0.89 | 0.997 | 15 | 577.4466 | 38.49644 |
| model8 | 3145989 | 0.92 | 1 | 15 | 545.5273 | 36.36849 |
| model9 | 14857813 | 0.76 | 0.926 | 30 | 1120.26 | 37.34201 |
| model10 | 1002085 | 0.721 | 0.977 | 30 | 705.8129 | 23.5271 |
| model11 | 3840453 | 0.735 | 0.964 | 30 | 722.1983 | 24.07328 |
| model12 | 3693253 | 0.941 | 0.992 | 30 | 526.603 | 17.55343 |

Final Model

**Optimizer Choice:**

**Adam**

* **Punishes over increasing epoch counts.**
* **Default learning rate is high, so converges faster.**
* **Coupled with reduce LR on plateau, can be reduced within lesser number of epochs if loss is not reduced.**

**Detailed analysis of each model in chronological order:**

**Model1:**

* **Type: Conv3D**
* **Parameter Count: 4 million**
* **Activation with ‘elu’ function after each kernel and first dense layer(*Converges cost to zero faster and produces more accurate results)***
* **Epoch count: 20**
* **Batch size: 32**
* **Observation:**
  + **Train Accuracy is high – 0.968**
  + **Validation Accuracy is good – 0.88**
  + **Training time is poor – 812s**
* **Decision:**
  + **Simplify model structure to reduce training time**

**Model2:**

* **Type: Conv3D**
* **Parameter Count: 3.5 million**
* **Epoch count: 20**
* **Batch size: 32**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.9**
  + **Validation Accuracy is good – 0.86**
  + **Training time is better – 706s**
* **Decision:**
  + **Simplifying model structure reduces the validation accuracy marginally but brings down training time by 13%.**
  + **Proceeding to reduce a hidden layer to see if it affects model performance.**

**Model3:**

* **Type: Conv3D**
* **Parameter Count: 27 million**
* **Epoch count: 20**
* **Batch size: 32**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 1**
  + **Validation Accuracy is relatively lowest – 0.88**
  + **Training time is high – 729s**
* **Decision:**
  + **Reducing hidden layer did not help in accuracy or time of training.**
  + **Switching back to model1 structure and reducing batch size to 16**

**Model4:**

* **Type: Conv3D**
* **Parameter Count: 4 million**
* **Epoch count: 20**
* **Batch size: 16**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 1**
  + **Validation Accuracy is relatively low – 0.9**
  + **Training time is worse – 724s**
* **Decision:**
  + **Lowering batch size has brought down training time by 10% without dropping accuracy.**
  + **From the plots, it is seen that the reduction in loss beyond epoch 15 is marginal and therefore 15 epochs should be sufficient.**

**Model5:**

* **Type: Conv3D**
* **Parameter Count: 4 million**
* **Epoch count: 15**
* **Batch size: 16**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.994**
  + **Validation Accuracy is excellent– 0.91**
  + **Training time is worse – 540s**
* **Decision:**
  + **This model is the best so far and shall work on this to see if we can make it better by reducing parameter count.**

**Model6:**

* **Type: Conv3D**
* **Parameter Count: 3.5 million**
* **Epoch count: 15**
* **Batch size: 16**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.988**
  + **Validation Accuracy is relatively low – 0.91**
  + **Training time is worse – 533s**
* **Decision:**
  + **Bringing down the parameter by ½ a million has dropped training time by 7s with a very negligible drop in training accuracy.**

**Model7:**

* **Changes Made:**
  + **Reducing the model structure as 16, 32, 64, 128**
  + **learning rate from 0.001 to 0.0012**
  + **LR reduce on Plateau - factor from 0.5 to 0.7 and patience from 2 to 1**
  + **Increase dropout from 0.25 to 0.4 to reduce overfitting**
* **Type: Conv3D**
* **Parameter Count: 3.5 million**
* **Epoch count: 15**
* **Batch size: 16**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.997**
  + **Validation Accuracy is relatively low – 0.89**
  + **Training time is worse – 577s**
* **Decision:**
  + **Not an improvement over model6.**

**Model8:**

* **Changes Made:**
  + **Increasing the model structure as 16, 32, 64, 128, 256**
  + **Maintaining learning rate as 0.0012**
  + **LR reduce on Plateau - factor as 0.7 and patience as 1**
  + **Increase dropout from 0.4 to 0.5 to reduce overfitting**
* **Type: Conv3D**
* **Parameter Count: 3.1 million**
* **Epoch count: 15**
* **Batch size: 16**
* **Activation with ‘elu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 1**
  + **Validation Accuracy is relatively low – 0.92**
  + **Training time is worse – 545s**
* **Decision:**
  + **Excellent model however not the fastest.**

**Model9:**

* **Type: CNN+RNN**
  + **Structure: VGG16 + GRU**
* **Parameter Count: 14 million**
* **Epoch count: 30**
* **Batch size: 16**
* **Activation with ‘relu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.926**
  + **Validation Accuracy is relatively low– 0.76**
  + **Training time is highest – 1120s**
* **Decision:**
  + **Probably try another architecture**

**Model10:**

* **Type: CNN+RNN**
  + **Structure: VGG16 + LSTM**
* **Parameter Count: 1 million**
* **Epoch count: 20**
* **Batch size: 20**
* **Activation with ‘relu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.977**
  + **Validation Accuracy is relatively lowest – 0.721**
  + **Training time is high – 705s**
* **Decision:**
  + **Probably try another architecture than VGGNet**

**Model11:**

* **Type: CNN+RNN**
  + **Structure: Mobilenet + LSTM**
* **Parameter Count: 3 million**
* **Epoch count: 20**
* **Batch size: 20**
* **Activation with ‘relu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.964**
  + **Validation Accuracy is relatively low – 0.735**
  + **Training time is high – 722s**
* **Decision:**
  + **Opening the frozen layers for training may improve performance.**
  + **In order to reduce number of parameters being trained, switching from LSTM to GRU (25% reduction).**
  + **Also reducing the epochs to 15 to lower time.**

**Model12:**

* **Type: CNN+RNN**
  + **Structure: Mobilenet + GRU**
* **Parameter Count: 3 million**
* **Epoch count: 15**
* **Batch size: 20**
* **Activation with ‘relu’ function after each kernel and first dense layer**
* **Observation:**
  + **Train Accuracy is high – 0.992**
  + **Validation Accuracy is relatively low – 0.941**
  + **Training time is high – 526s**
* **Decision:**
  + **This is the best model so far, with lowest time taken.**
  + **The model also achieved the highest validation accuracy.**

**Conclusion:**

**Model12 with Mobile net and layers opened for training, with GRU is the best model from the list of models which offer 94% accuracy on validation set at best and 99% accuracy on training, in the shortest time of 526s training.**