Neural Networks Project - Gesture Recognition

- Parinita Dwivedi
- Ankit Sharma
- Pardeep Kumar

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 recognize five different gestures performed by the user which will help users control the TV without using a remote.

The gestures are continuously monitored by the webcam mounted on the TV. Each gesture corresponds to 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.

Dataset

The training data consists of a few hundred videos categorised 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 - like what the smart TV will use.

The dataset folder contains a 'train' and a 'val' folder with two CSV files for the two folders. Each row of the CSV file represents one video and contains three main pieces of information - the name of the subfolder containing the 30 images of the video, the name of the gesture and the numeric label (between 0-4) of the video.

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.

Architectures

For analyzing videos using neural networks, two types of architectures are used. One is standard CNN + RNN architecture in which images of a video is passed through a CNN which extracts a feature vector for each image, and then pass the sequence of these feature vectors through an RNN. The other popular architecture used to process videos is a natural extension of CNNs - a 3D convolutional network.

Data Generator

Creating data generators is probably the most important part of building a training pipeline. Although libraries such as Keras provide built-in generator functionalities, they are often restricted in scope, and you must write your own generators from scratch. In this project we will implement our own custom generator, our generator will feed batches of videos, not images.

Data Pre-processing

- **Resizing**: We converted the images of the train and test set into matrix of size 120*120.
- **Cropping**: We cropped the non-symmetric images into 120*120, because conv3D will throw error if we don't resize the data. Cropping is different from resize as in resize changes the aspect ratio of rectangular image. In cropping we will center crop the image to retain the middle of the frame.
- **Normalization**: We will use mean normalization for each of the channel in the image.

Observations

- It is observed that as the number of trainable parameters increase, the model takes more time for training.
- The model was overfitting in the start (fluctuating validation accuracy), hence had to increase the dropout to get a stable model.
- A large batch size can throw GPU Out of memory error, and thus here we had to play around with the batch size till we were able to arrive at an optimal value of the batch size which our GPU could support.
- Increasing the batch size greatly reduces the training time but this also has a negative impact on the model accuracy. So, there is a tradeoff between the two and we can decide based on the resources available.
- CNN based model with GRU cells had better performance than Conv3D for this dataset but it seemed to overfit for our dataset.
- Transfer learning boosted the overall accuracy of the model. We used the MobileNet Architecture due to its lightweight design and high-speed performance coupled with low maintenance as compared to other well-known architectures like VGG16, AlexNet, GoogleNet etc.

Results

Experiment Number	**Model**	**Result**	**Decision + Explanation**
1	Conv3D	Training Accuracy:96.6% Validation Accuracy: 100% Batch Size: 39, No. of Epochs: 30	Model seems to be overfitting and there are a lot of fluctuations in the validation accuracy. For some sample of the validation data, it gives high accuracy but for some very low.
2	Conv3D	Training Accuracy: 98% Validation Accuracy: 85%	Model performs well on the training

		Batch Size: 17, No. of Epochs:30	data but not on the validation data. Hence overfits.
3	Conv3D	Training Accuracy: 85.3% Validation Accuracy: 83.3% Batch Size: 17, No. of Epochs: 25	Here we see that the model performs well both on the training and validation data. Also, fluctuations in the validation accuracy have reduced.
4	CNN +GRU	Training Accuracy:95% Validation Accuracy: 76% Batch Size: 39, No. of Epochs: 30	This model does not give a good accuracy on the validation data
5	Transfer Learning using MobileNet	Training Accuracy: 99.5% Validation Accuracy: 96% Batch Size: 39, No. of Epochs: 25	This can be used but might overfit and the training accuracy is almost 100%.
Final Model	Conv3D	Training Accuracy: 85.3% Validation Accuracy: 83.3%	This model performs well both on the training and validation data. Also, fluctuations in the validation accuracy is less, hence is a more stable model. Thus, we chose this as our final model.