Gesture Recognition Case Study

Problem Statement:

You are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You 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.

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

Solution Design:

- Since the input data is given in form of sequence of images, precisely 30 images for each data point, we needed to find a way to feed the 30 images to the model.

Data Input / Pre-processing -

- We created a customer generator to solve feed the data to the model.
- Using the custom generator served multiple purposes -
    - **Solved Memory Overflow Issue**: Since dataset is too large to be fitted in the memory, the generator only reads the images required for one batch run. This solved the problem to memory overflow.
    - **Pre-processing:** We added some image pre-processing steps to make the model more generalised.
      Without pre-processing, we faced problem of overfitting, which was solved by the pre-processing steps, viz, normalization, image rotation, image resizing/cropping.
    - **Image Resizing:** Since the input data has heterogenous sources, the size of images is not uniform. So, we added the functionality of image resizing to fix share in the generator itself.

Hyper parameter Tuning -

- We tried out multiple combinations of following hyper-parameters –
    - Sampled_frames – No. of images taken out of 30 images for analysis
    - img_height – Height of input image
    - img_width – Width of input image
    - num_epochs – No. of epochs for training
    - batch_size – Batch size used during training

- Based on the time taken during training & the training / validation accuracy, we finalized the following values for the hyper-parameters
    - Sampled_frames – 18
    - img_height – 160
    - img_width – 160
    - num_epochs – 30
    - batch_size – 30

Model Training –

- During model training, we tackled following set of problems with the described solution for each problem, as given below –
    - **Low Training Accuracy** – Initially we made a dry run on complete dataset for a very less epochs (~3 epochs) to check if everything is working fine.
      We then reduced the training data to only a sample of complete dataset & experimented with the hyper-parameters.
      After finding the appropriate hyper-parameters, we trained the model for more number of epochs (~30 epochs), the training accuracy got improved.
    - **Overfitting –** As we took the complete dataset, we got overfitting in the model, as training accuracy increased, but validation accuracy stopped increasing and even started dropping.
      We handled overfitting by following strategies –
        1. **Data Augmentation –** We normalized the data, added rotation in the images with 7 % degree.
        2. **Drop out layers -** We added the drop out layers in the network to handle overfitting.

Model Stats –

Model 1 (Conv3D)

- We used the following set of hyperparameters –
    - Sampled_frames – 18
    - img_height – 160
    - img_width – 160
    - num_epochs – 30
    - batch_size – 30
- Model params – 1,736,389
- Model size on disk – 19.9 mb
- Model performance stats –
    - loss: 0.1832
    - categorical_accuracy: 0.9381
    - val_loss: 0.3895
    - val_categorical_accuracy: 0.8700

Model 2 (CNN + LSTM + GRU)

- We used the following set of hyperparameters –
    - Sampled_frames – 18
    - img_height – 160
    - img_width – 160
    - num_epochs – 30
    - batch_size – 30
- Model params – 2,572,965
- Model size on disk – 29.5 mb
- Model performance stats –
    - loss: 0.1911
    - categorical_accuracy: 0.9686
    - val_loss: 0.5406
    - val_categorical_accuracy: 0.8400

Model 3 (Transfer Learning – Mobilenet + LSTM)

- We used the following set of hyperparameters –
    - Sampled_frames – 18
    - img_height – 160
    - img_width – 160
    - num_epochs – 30
    - batch_size – 30
- Model params – 4,302,661
- Model size on disk – 24.7 mb

- Model performance stats –
    - loss: 0.0528
    - categorical_accuracy: 0.9942
    - val_loss: 0.4628
    - val_categorical_accuracy: 0.8800

We got the best performing model as Model 3 (Transfer Learning + LSTM) with validation accuracy of: 0.88 & validation loss: 0.46

Model 4 (Conv3D) – Memory Efficient Model

- We used the following set of hyperparameters –
    - Sampled_frames – 18
    - img_height – 160
    - img_width – 160
    - num_epochs – 30
    - batch_size – 50
- Model params – 437413
- Model size on disk – 5.1 mb
- Model performance stats –
    - loss: 0.2346
    - categorical_accuracy: 0.9398
    - val_loss: 0.4132
    - val_categorical_accuracy: 0.8600

For TVs with high specs that can support model sized ~26 mb, we'll recommend using Model-3 (Mobilenet + LSTM) as its has best validation accuracy (88%).

However, for low specs TV, we have relatively lighter model – Model 4 (Conv3D) with reduced model size of just 5.1 mb. It also has a decent accuracy of 86% on validation set.