DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

Problem Statement

The problem statement is for the data scientist who is working at a home electronics company. He has been asked 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:

1. Thumbs up: Increase the volume
2. Thumbs down: Decrease the volume
3. Left swipe: 'Jump' backwards 10 seconds
4. Right swipe: 'Jump' forward 10 seconds
5. Stop: Pause the movie

The task for the data scientist is to train a model on the 'train' folder (i.e. Training Data) which performs well on the 'val' folder (i.e. Validation Data) as well.

Dataset Information

The training and validation 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)**.** All images in a particular video subfolder have the same dimensions but different videos may have different dimensions.

The videos have two types of dimensions - either 360x360 or 120x160

The data is provided in a zip file. The zip file contains a 'train' and a 'val' folder with two CSV files for the two folders. These folders are in turn divided into subfolders where each subfolder represents a video of a particular gesture.

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.

DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

Deep Learning Models

Approach

In this case study we have tried different experiments with different models that we tried to generate. The experimentation is based on tuning different hyperparameters like batch size,number of epochs,image size etc. Given below is a summary table of all models generated and their metrics along with the explanation/remarks.

Model Type	Model Specification	Model #	Hyperparameters and Image resolution	Total Number of Parameters	Training Accuracy (in %)	Validation Accuracy (in %)	Explanation/Remarks
Conv 3D / 3D CNN	With Dropout in FC only & Without Batch Normalization	1	Batch Size: 40 Epoch: 10 Image Size: (120,120) filterSize : (3,3,3)	2,139,589	48.1	66.67	This was the basic model without applying any Normalization. We tried several combinations/ experiments with changing the batch size and number or epochs etc With all the above manipulations/ experimentation the accuracy of the train set as well as validation set did not go above 67% in this type of model. Hence moved to other models
		Model 1A	Batch Size : 40 Epoch : 20 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	59.52	66.67	
		Model 1B	Batch Size : 30 Epoch : 20 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	26.09	55	
		Model 1C	Batch Size : 50 Epoch : 20 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	54.39	58	
		Model 1D	Batch Size : 70 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	26.67	55	
		Model 2					

DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

Conv 3D / 3D CNN	With Dropout in Dense Layer & With Batch Normalization	Model 2A	Batch Size : 40 Epoch : 10 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	79.24	61.67	In this model we have 4 convolution layers and 2 dense layers followed by the softmax layer. We also have applied Batch Normalization. In this model we made several experiments with batch size,number of epochs,choosing different image sizes etc. Apart from this, we experimented with different filter sizes from (2,2,2) & (3,3,3). With such different experimentations and trials, we got better results and this type of model gave us max training accuracy upto 96% (approx.) and training accuracy upto 87%(approx.) Also we observed that the number of trainable parameters if few of models were high.
		Model 2B	Batch Size : 50 Epoch : 20 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	71.43	65	
		Model 2C	Batch Size : 50 Epoch : 40 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	90.65	85	
		Model 2D	Batch Size : 60 Epoch : 40 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	50	55	
		Model 2E	Batch Size : 30 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	46.67	65	
		Model 2F	Batch Size : 40 Epoch : 10 Image Size : (160,160) filterSize : (3,3,3)	3,574,405	95.16	83.33	
		Model 2G	Batch Size : 40 Epoch : 20 Image Size : (160,160) filterSize : (2,2,2)	3,574,405	86.16	80	
		Model 2H	Batch Size : 40 Epoch : 20 Image Size : (120,120) filterSize : (2,2,2)	2,140,805	96.54	86.67	

DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

		Model 3					
Conv 3D/ 3D CNN	Without Dropout with Batch Normalization	Model 3A	Batch Size : 40 Epoch : 10 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	56.06	46.67	In this type of model, we did not apply dropout but applied batch normalization. We experimented with changing values for batch size and number of epochs However we did not get good results / metrics and hence we moved to other models.
		Model 3B	Batch Size : 40 Epoch : 50 Image Size : (120,120) filterSize : (3,3,3)	2,139,589	75.79	71.67	
		Model 4					
Conv 3D/ 3DCNN	With Dropout in FC Layer & With Batch Normalization	Model 4A	Batch Size : 30 Epoch : 50 Image Size : (120,120) filterSize : (3,3,3)	2,140,805	97.57	88.34	In this type of model we applied dropouts in the FC layer and with batch normalization. We also tried to experiment with different batch size , number of epochs,dense neurons ,different image size,different dense neurons. With all the above experiments we got good results with max training accuracy around upto 98% and validation accuracy around or upto 95%. We also observed that in this model the loss values are also extremely low.
		Model 4B	Batch Size : 40 Epoch : 60 Image Size : (100,100) filterSize : (3,3,3)	2,140,805	75.08	86.67	
		Model 4C	Batch Size : 40 Epoch : 50 Image Size : (120,120) filterSize : (3,3,3) dense neurons : 128 dropout : 0.4	2,669,893	90.31	91.67	
		Model 4D	Batch Size : 40 Epoch : 30 Image Size : (160,160) filterSize : (3,3,3) Dense neurons : 256 dropout : 0.5	13,468,357	90.31	95	

		Model 4E	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) Dense neurons : 128 dropout : 0.5	3,996,997	82.01	81.67	
		Model 4F	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) Dense neurons : 128	3,996,997	98.96	91.67	
		Model 4G	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) dropout : 0.5	2,140,805	78.54	76.67	
		Model 5					
Conv 3D/ 3DCNN	With dropout in FC layer and with Batch Normalization + Adding one more Convolutional layer	Model 5A	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) dropout : 0.5	2,143,909	76.82	81.67	In this model we used an extra convolution layer in addition to the one that we had in the previous model. In this model as well, we experimented with different dropout rates etc and we got good results with this experiment with the max accuracy for training upto 95% and for validation upto 87%.
		Model 5B	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) dropout : 0.25	2,143,909	95.55	86.67	
		Model 5C	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3) dropout : 0.4	2,143,909	79.23	81.67	
		Model 6					

Conv 3D/ 3D CNN	With BN in Convolution & Drop out in both FC layer	Model 6A	Batch Size : 40 Epoch : 30 Image Size : (120,120) filterSize : (3,3,3)	1,476,997	32.52	43.33	In this model we experimented with normalization in convolution and dropouts in both the FC layer. However looking at the metrics the model generated , we did not get good results and hence we moved to other models.
		Model 7					
CNN-RNN	CNN-LSTM	Model 7A	Batch Size : 20 Epoch : 20 Image Size : (120,120) LSTM Cells-128	3,384,293	53.99	58	In this model we used four convolution layers, followed by flatten layer, LSTM layer, dense layer and softmax layer. In this model as well we experimented with several permutations like different numbers of epochs, batch sizes, dropout rates etc. This model gave us max training accuracy of upto 87%(approx.) and validation accuracy upto 82% (approx.) The number of hyperparameters are also a bit higher. Based upon the outcome and results generated from this model we can say that 3D CNN gives us better results and accuracy.
		Model 7B	Batch Size : 40 Epoch : 40 Image Size : (120,120) LSTM Cells-128	3,384,293	86.85	81.67	
		Model 7C	Batch Size : 20 Epoch : 20 Image Size : (120,120) LSTM Cells : 128 Drop out-0.45	2,532,325	67.74	65	

DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

CNN-RNN	CNN-LSTM with GRU	Model 8	Batch Size : 40 Epoch : 20 Image Size : (120,120) LSTM Cells : 128 Dense Neurons : 128 Drop out-0.25	2,573,541	99	72	In this model we used four convolution layers, followed by flatten layer, GRU layer, dense layer and softmax layer. In this model as well we have tried with different numbers of epochs, batch sizes, dropout rates etc. This model gave us max training accuracy of upto 99% and validation accuracy upto 72% (approx.) The number of hyperparameters are also a bit higher. Based upon the outcome and results generated from this model we can say that the model is considerably overfitted.

Author: Solon Kumar Das

Best Deep Learning Models (both 3D CNN and CNN-RNN explained)

Based on the above experimentations carried out we have now the 2 best models and they are explained below -

3D CNN

```
class Model4_Conv3D_FC_BN(ModelGenerator):

    def define_model(self,filtersize=(3,3,3),dense_neurons=64,dropout=0.25):

        model = Sequential()
        model.add(Conv3D(16, filtersize, padding='same', input_shape=(15,self.image_height,self.image_width,self.num_Of_char
        model.add(Activation('relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling3D(pool_size=(2, 2, 2)))

        model.add(Conv3D(32, filtersize, padding='same'))
        model.add(Activation('relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling3D(pool_size=(2, 2, 2)))

        model.add(Conv3D(64, filtersize, padding='same'))
        model.add(Activation('relu'))
        model.add(BatchNormalization())
        model.add(MaxPooling3D(pool_size=(2, 2, 2)))

        model.add(Conv3D(128, filtersize, padding='same'))
        model.add(Activation('relu'))
        model.add(BatchNormalization())
        #model.add(MaxPooling3D(pool_size=(2, 2, 2)))

        #Flatten Layer
        model.add(Flatten())
        model.add(Dense(dense_neurons,activation='relu'))
        model.add(BatchNormalization())
        model.add(Dropout(dropout))

        model.add(Dense(dense_neurons,activation='relu'))
        model.add(Dropout(dropout))

        # Softmax Layer
        model.add(Dense(self.num_Of_classes,activation='softmax'))

        optimiser = optimizers.Adam()
        model.compile(optimizer=optimiser, loss='categorical_crossentropy', metrics=['categorical_accuracy'])
        return model
```

This model gave us training accuracy as 98% and validation accuracy 95% with different values and tune up of hyper parameters as explained in the table above. (see Model 4 results).
In this model we have -

- four convolutional layers followed by
- two dense layers followed by
- softmax layer

In the first three convolutional layers we have used Max pooling layer with pool size of (2,2,2)
Batch normalization is used with every layer in this model.
Dropouts are used with the dense layers only.
We have used Adam optimizer in this model. Adam is an optimization algorithm for stochastic gradient descent for training deep learning models. It combines the best properties of the **AdaGrad** and **RMSProp** algorithms to provide an optimization algorithm that can handle sparse gradients.

The .h5 file of this model is enclosed in the zip file.

DEEP LEARNING COURSE PROJECT - GESTURE RECOGNITION

Author: Solon Kumar Das

CNN RNN

```
class Model7_CNN_RNN_1(ModelGenerator):

    def define_model(self,lstm_cells=64,dense_neurons=64,dropout=0.25):

        model = Sequential()

        model.add(TimeDistributed(Conv2D(16, (3, 3) , padding='same', activation='relu'), input_shape=(15,self.image_height,
        model.add(TimeDistributed(BatchNormalization()))
        model.add(TimeDistributed(MaxPooling2D((2, 2))))

        model.add(TimeDistributed(Conv2D(32, (3, 3) , padding='same', activation='relu')))
        model.add(TimeDistributed(BatchNormalization()))
        model.add(TimeDistributed(MaxPooling2D((2, 2))))

        model.add(TimeDistributed(Conv2D(64, (3, 3) , padding='same', activation='relu')))
        model.add(TimeDistributed(BatchNormalization()))
        model.add(TimeDistributed(MaxPooling2D((2, 2))))

        model.add(TimeDistributed(Conv2D(128, (3, 3) , padding='same', activation='relu')))
        model.add(TimeDistributed(BatchNormalization()))
        model.add(TimeDistributed(MaxPooling2D((2, 2))))

        #Flatten Layer
        model.add(TimeDistributed(Flatten()))


        model.add(LSTM(lstm_cells))
        model.add(Dropout(dropout))

        model.add(Dense(dense_neurons,activation='relu'))
        model.add(Dropout(dropout))

        #Softmax Layer
        model.add(Dense(self.num_Of_classes, activation='softmax'))
        optimiser = optimizers.Adam()
        model.compile(optimizer=optimiser, loss='categorical_crossentropy', metrics=['categorical_accuracy'])
        return model
```

This model gave us training accuracy as 87% and validation accuracy 82% with different values and tune up of hyper parameters as explained in the table above. (see Model 7 results).
In this model we have -

- four convolutional layers followed by
- flatten layers followed by
- LSTM followed by
- dense layer followed by
- softmax layer

In the first three convolutional layers we have used Max pooling layer with pool size of (2,2,2)
Batch normalization is used with every convolution layer in this model.
Dropouts are used with the LSTM & dense layers only.
We have used Adam optimizer in this model. Adam is an optimization algorithm for stochastic gradient descent for training deep learning models. It combines the best properties of the **AdaGrad** and **RMSProp** algorithms to provide an optimization algorithm that can handle sparse gradients.

Both the .h5 files are also available on google drive link below:

1. model-00023-0.06841-0.98962-0.32275-0.91667.h5 https://drive.google.com/open?id=1N5nmzW6bzbVtClFT2S5azw1ucOUQpwTq
2. model-00039-0.43407-0.86851-0.64222-0.81667.h5 : https://drive.google.com/open?id=1VIIWEaz063OEk_xQrHWhr_G-HAPEcUY