Action Recognition Report

March 2020

Action: Shouting

Josiah Coad

With update for submission 2 in blue

With updates for submission 3 in red

1. Background

Action recognition is the identification of actions, normally human actions, in the physical world through visual signals and audio signals. It would seem that action recognition is a natural extension of image recognition which deep learning has excelled at for years. HoIver, there are some difficulties in making the extension. Large computational costs are a major challenge moving forward as videos have a temporal dimension which can greatly increase the computational complexity of learning. Most of the research in action recognition has been in how to effectively take advantage of the temporal aspect of videos.

Significant advances have been made through applying convolutional deep neural networks to action recognition. Much of the literature currently uses pre-trained CNN architectures for faster convergence. Notable advances include the following. In 2014, this foundational work came out that looked at fusing temporal approaches with traditional CNN. Namely, the authors compared various approaches of reading in several frames of the video at once to the CNN. Several feature engineering techniques proved to be useful including spatial flow in this work. Soon, several more works appeared that used 3D convolutions and RNNs to capture temporal information. These techniques have all been combined in various ways, finding improvements in accuracies on benchmark datasets. A more indepth background exposition can be found here.

2. Dataset

In this work, I use the Kinetics 700 dataset, released by DeepMind. Since the Kinetics 700 dataset covers 700 actions, and I are only interested in identifying shouting, I specifically take all the shouting videos and an equal number (651) of randomly sampled videos from the other 699 classes. I then descritize the videos to images with a sampling of 3 frames per second. In the end, this results in nearly even split of 20,094 images labeled as *shouting* and 19,913 images labeled as *other*.

A major difficulty of using the kinetics dataset *false labelling*. That is, in the Kinetics 700 dataset, an entire video has been labeled for an action that may only happen for a short time in the video. I plan to address this in future iterations of this project.

I perform a train and validation split of 2/3 and 1/3 respectively. This results in a training and validation set of 26,805 and 13,202 samples respectively.

In preprocessing, I converted each image to grayscale and reshaped the images to (144, 256).

Update for submission 2: The false labelling problem became too much of an issue. On evaluation by hand, there was simply too much video time in the *shouting* category of the kinetics dataset where no one was shouting. So I made my own dataset of videos where either I was screaming the whole video or *not* the whole video. I took all the frames from the video to feed into my model. I did not grayscale or reshape the images.

Update for submission 3: Personal data was not well enough distributed to generalize well to out-of-sample videos. Thus, I had to take a hybrid approach of using some of my own videos and kinetics samples. The positives from the kinetics samples had to be carefully selected to be nearly all shouting which was hard to find.

3. DNN Model

3.1 Architecture

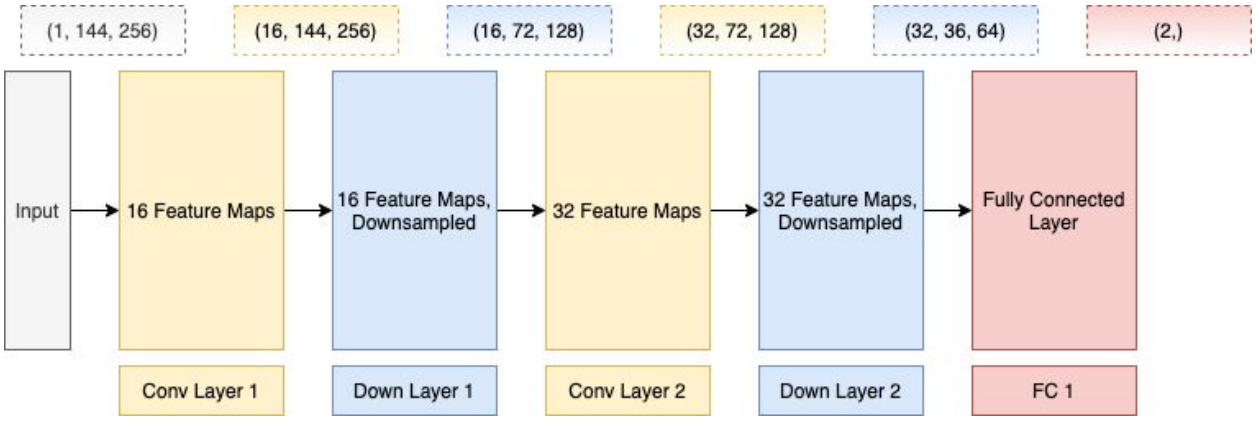


Image 1: Architecture of CNN used (adapted from this)

3.2 Input: Shape of Tensor

X_train shape: 26805 samples, 1 channel, pixel matrix: (144, 256)
X_test shape: 26805 samples, 1 channel, pixel matrix: (144, 256)

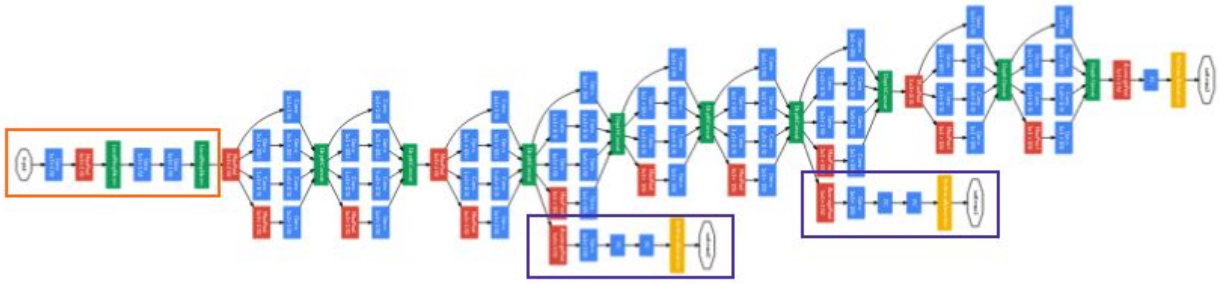
3.3 Output: Shape of Tensor

y_train shape: 26805 samples of binary {0, 1} label
y_test shape: 13202 samples of binary {0, 1} label

3.4 Shape of Output Tensor for Each Layer

Output of Conv Layer 1: (16, 144, 256), since same padding
Output of Maxpooling Layer 1: (16, 72, 128), since kernel size of 2
Output of Conv Layer 2: (32, 144, 256), since same padding
Output of Maxpooling Layer 2: (32, 24, 64), since kernel size of 2
Output of Fully Connected 1: (2,), for shout / not shout

Update for submission 2: I used a completely different model architecture. I first identified and cropped faces out of the images using MTCNN face detection. I then fed the cropped faces into an InceptionResnetV1 pretrained on vggface2 dataset. The architecture is as follows:



Source: Inception V1 Paper

The InceptionResnetV1 yielded 512 features for each image. I fed those features into a PCA to reduce the dimensionality to 3. I then trained a support vector machine with an rbf kernel on the 3 dimensional embedding.

Update for submission 3: I extracted 68 face points from each frame then fed these points into a feed forward fully connected neural network. I used tensorflow.keras for this since it was a relatively simple architecture. The architecture is as follows.

Layer (type)	Output Shape	Param #
dense_30 (Dense)	(None, 128)	17536
dense_31 (Dense)	(None, 50)	6450
dense_32 (Dense)	(None, 50)	2550
dense_33 (Dense)	(None, 50)	2550
dense_34 (Dense)	(None, 2)	102
softmax_6 (Softmax)	(None, 2)	0

```
=================================================================
Total params: 29,188
Trainable params: 29,188
Non-trainable params: 0
_________________________________________________________________
```

4 Hyperparameters

4.1 List of Hyperparameters

In this project, there are three hyperparameters I tuned: batch size; epochs and dropout.

4.2 Range of Value of Hyperparameters Tried

- batch size: {8, 32, 64, 128}
- epochs: {2, 4, 8}

4.3 Optimal Hyperparameters Found

After testing, I chose the following hyperparameters worked best with my architecture:

- batch size = 32
- epochs = 2

Update for submission 2: Since the deep network was already trained and I used a SVM for classification (using the sklearn defaults), I did not tune any hyperparameters.

Update for submission 3: I used the Adam optimizer so I did not tune learning rate. The only hyperparameter I tuned was the epochs. The model didn't learn much past 10 epochs.

5 Training and Testing Performance

```
Epoch: 2 Loss: 0.43146151304244995. Accuracy: 72, Time: 5 min
=> Saving a new best
Epoch: 3 Loss: 0.6871418952941895. Accuracy: 57, Time: 11 min
Epoch: 4 Loss: 0.49511557817459106. Accuracy: 57, Time: 16 min
Epoch: 5 Loss: 0.7898200154304504. Accuracy: 74, Time: 22 min
=> Saving a new best
Epoch: 6, Loss: 0.3489263653755188, Accuracy: 59, Time: 27 min
Epoch: 7, Loss: 0.3444364368915558, Accuracy: 70, Time: 32 min
Epoch: 8, Loss: 0.7275205850601196, Accuracy: 70, Time: 37 min
Epoch: 9, Loss: 0.23404459655284882, Accuracy: 73, Time: 42 min
```

Update for submission 2: Again, there were no epochs. The SVM achieved .9621 validation accuracy on my personal training set with a .67/.33 train/validation split.

Update for submission 3: I realized that I was not previously measuring validation accuracy correctly. This is because I was randomly splitting the frames of the videos into train/validation splits. Because video frames are highly correlated, the model was able to achieve superior

validation accuracy just because the validation set was nearly the same as the training set. I achieved .88 accuracy on the validation set.

6 Instruction on how to test the trained DNN and how to use the demo

6.1 Install Dependencies

av==7.0.1
certifi==2019.11.28
chardet==3.0.4
cycler==0.10.0
facenet-pytorch==2.2.9
idna==2.9
joblib==0.14.1
kiwisolver==1.1.0
matplotlib==3.2.1
numpy==1.18.2
opencv-python==4.2.0.32
pandas==1.0.3
Pillow==7.0.0
pyparsing==2.4.6
python-dateutil==2.8.1
pytz==2019.3
requests==2.23.0
scikit-learn==0.22.2.post1
scipy==1.4.1
six==1.14.0
torch==1.4.0
torchvision==0.5.0
urllib3==1.25.8

6.2 Execution

>>> cd demo
>>> pip install -r requirements
>>> python demo.py
or the following to define run on your own video
>>> python demo.py --vidpath ./videos/myvid.mp4

6.3 Code

See github here with demo and training colab here and inference colab here.

6.4 Video Link

See how to run code here.

7. Future Improvements

There are several advancements I'd like to address in the future iterations.

1. Architecture: I'd like to take advantage of temporal information by feeding in multiple frames at a time. I'd also like to use 3D convolutions.
2. Preprocessing: I'd like to use facial landmarks
3. Audio: Screaming can largely be detected by sound. I'd like to train an audio classifier and experiment with fusing the audio and visual features in the training.

Update for submission 2: I tried using 3D convolutions. Actually I attempted using transfer learning from the 2D+1 model pretrained on Kinetics. However, it resulted in inferior accuracy, around 65%. Additionally, it just learned to identify if there was a face in the frames, and if so, to predict shouting.

Because shouting is nearly completely dependent on facial features, using the MTCNN facial extraction technique proved useful. However, if the MTCNN could not detect a face in the frame, I would predict 0 on a given frame. This is a known limitation. It could be solved by merging the 512 feature output with a simple CNN on the entire frame to arrive at a prediction.

Update for submission 3: In this iteration, I still used facial features, but I used a smoothing post-processing technique on the probabilities so that if a face isn't in a few frames but comes back, the probabilities won't drop right away. In the future, I would like to attempt audio classification on the clips.

Example Results on Demo from Submission 3