Video Classification

REPORT BY,

Deyu Kong
Aihan Liu

DATS6203
Professor Amir Jafari
April 26, 2022

Table of Contents

INTRODUCTION

With the development of the Internet, more and more videos are shared online. The computer vision community is also developing research on video, such as behavior recognition, abnormal event detection, activity understanding, etc. Considerable progress has been made on these individual problems by employing different specific solutions. However, a broad set of video feature representation learning methods is still needed for solving large-scale video tasks.

Deep convolutional networks pre-trained on ImageNet perform well in transfer to other image domains in the image domain. However, these models cannot be directly transferred to the video domain since these image-based pre-trained models cannot represent motion information in videos. Tran et. (2015) proposes a deep convolutional neural network based on a 3D convolution kernel which is more efficient for spatiotemporal feature learning.

Based on the UCF101 [2] video data set, this project has a preliminary understanding and application of video data classification through the theory and practice of 3D convolution. Due to the large amount of data in this dataset, it is challenging to complete the training of entire datasets in a limited time. This project will focus on the category of playing musical instruments, which has a baseline of 37.42% of accuracy.

In this project, we implemented a modified version of the C3D network, VC3D to better deal with the video classification task. While it achieved 100% accuracy in the Playing Instruments group for a random heldout test set that consisted of 15% of the full data,

robustness issues were exposed when inference on videos collected from external sources.

DATA DESCRIPTION

The UCF101 dataset consists of 101 categories and 13,320 videos recorded in unconstrained environments and uploaded to YouTube, featuring camera motion, various lighting conditions, partial occlusion, low-quality frames, etc.

Actions	101
Clips	13320
Groups per Action	25
Clips per Group	4-7
Mean Clip Length	7.21sec
Total Duration1	1600mins
Min Clip Length	1.06sec
Max Clip Length	71.04sec
Frame Rate	25 fps
Resolution	320 x 240
Audio	Yes

Table 1 UCF101 dataset attributions

It was published in 2012 and solved two problems from the previous datasets: 1. the number of categories is too small; 2. videos usually are not recorded in natural environments. The dataset contains 101 categories, which are divided into the following five groups:

- Human-Object Interaction
- Body-Motion Only
- Human-Human Interaction
- Playing Musical Instruments
- Sports

The baseline of classification task that the author made is:

- Sports 50.54%
- Playing Musical Instrument 37.42%
- Human-Object Interaction 38.52%
- Body-Motion Only 36.26%
- Human-Human Interaction 44.14%
- Overall 44.5%

In this project, we focused on the playing Musical Instruments group, which consists of nine different instruments playing videos, including Playing Guitar, Playing Piano, Playing Tabla, Playing Violin, Playing Cello, Playing Daf, Playing Dhol, Playing Flute, and Playing Sitar.

They have 1268 clips in total, and each of them contains 4-7 videos, and the videos from the same group have some similar characteristics, such as background, characters, etc.

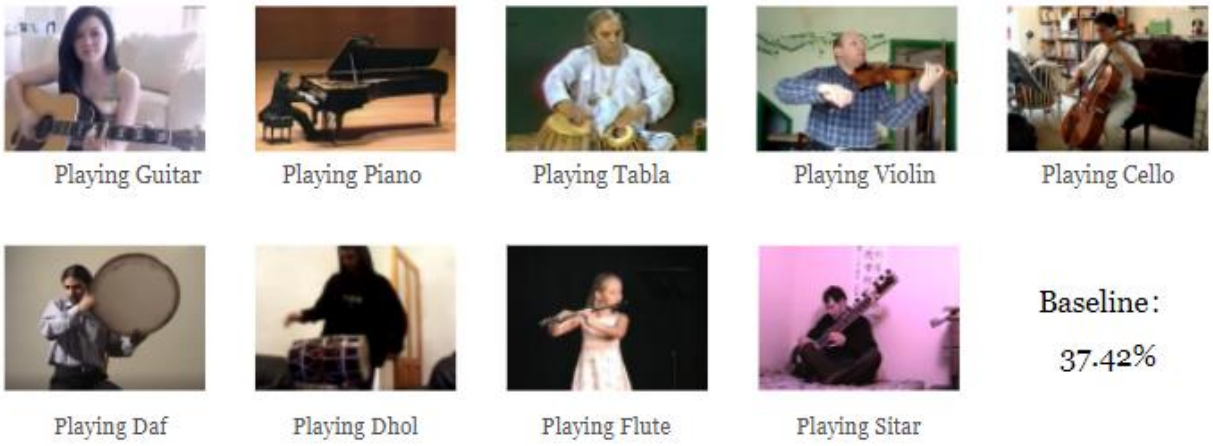

Fig.1 Examples of Playing Instruments Videos in UCF101

These videos were divided into three groups: training (70%), validation (15%), and heldout test (15%).

MODEL ARCHITECTURE

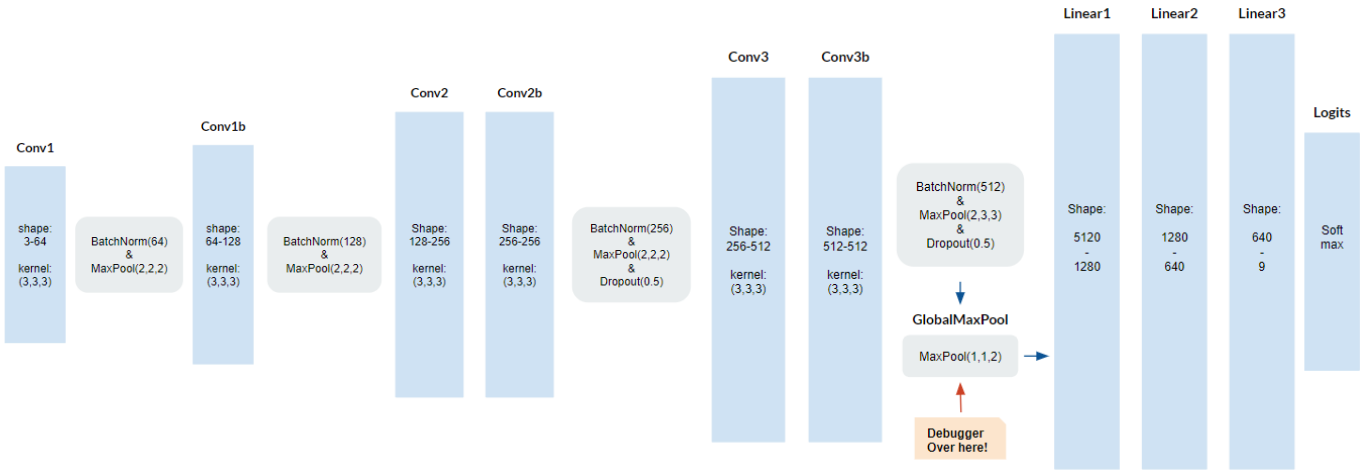

Fig.2 VC3D Model Architecture

We studied the layout of the original C3D build, analyzed its output, and implemented our own version with some structural improvements. We included 6 layers of Conv3d networks in an upscaling manner before attaching a linear classification head. Compared with the original C3D, we reduced two layers of convolution, while inheriting overall grouping (group 2 Conv3d layers together except for the first two). We also included BatchNorm3d layers after each convolution 'group' for better performance. DropOut layer with a factor of 0.5 was applied twice after convolution layer groups, to pursue model robustness.

One of the key challenges lies with ascertaining the correct shape and size of the tensors after convolution layers, before packing them into linear layers for classification. Our solution was to attach a Pycharm breakpoint to the MaxPool3d layer in forward() method, right before the linear layers.

```
42      def forward(self, x):  # x shape = (20, 3, 16, 120, 120)
43          x = self.act(self.conv1(x))
44          x = self.pool1(self.convnorm1(x))
45          x = self.act(self.conv1b(x))
46          x = self.pool1b(self.convnorm1b(x))
47          x = self.act(self.conv2b(self.act(self.conv2(x))))
48          x = self.drop2(self.pool2(self.convnorm2(x)))
49          x = self.act(self.conv3b(self.act(self.conv3(x))))
50          x = self.drop3(self.pool3(self.convnorm3(x)))  # x.shape = (20, 512, 1, 5, 5)
51          # x = self.act(self.conv4b(self.act(self.conv4(x))))
52          # x = self.drop4(self.pool4(self.convnorm4(x)))
53          x = self.global_max_pool(x)  # After pooling x.shape = (20, 512, 1, 5, 2)
54          x = self.linear3(self.linear2(self.linear1(x.view(-1, 5120))))
55          # x = self.softmax()
56          return x
```

Fig.3 Attach Breakpoint to Access the Tensor Shape

After commencing the maxpool in debugging mode, we can access the shape of the tensor after that hidden state, and the total neurons required in the first linear layer will be the last four dimensions (except for the Batch dimension) multiplied. In our case, was $512 \times 1 \times 5 \times 2 = 5120$. We used .view() method to 'flatten' the tensor for input as well.

We used 3 linear layers to downsample the shape from 5120 to 9 classes which we wish to classify. The logits computed by the network shall go through Softmax which is embedded inside CrossEntropyLoss function, to determine the predicted label.

DATALOADER

UCF101 Dataset comes with its built-in API in torchvision.dataset library. However, this API suffers from an unresolved bug (see topic#4112[1]) that prevents us from using it. Upon further research, we decided to deploy a custom dataloader for batch processing, adapted from a base code.

The dataloader reads video clips from the data directory and captures frames to save as .jpg images. To facilitate better GRAM management and reduce training time, the videos are randomly cut to a fixed number of consecutive frames. That number is specified by the 'clip_len' argument and here we used 16. These 16 frames will represent the clip and will be further normalized, resized to 171x128, then center cropped to 120x120 for network input. Train, validation, and heldout test split were performed in this stage as well.

[1] https://github.com/pytorch/vision/issues/4112

TRAINING AND RESULTS

Training loop is performed under these parameters:

Parameter	Value
Epochs	50
Batch Size	20
Learning Rate	1e-03, ReduceLROnPlateau, monitor on test loss
Loss Function	Cross Entropy
Optimizer	Adam
Evaluation Metrics	Accuracy minus hammings metric

Table 2 Training Parameters

The results were outstanding: we achieved 100% accuracy on the test set. On one hand, it proves our model architecture was correct and useful, on the other we were concerned with overfitting and model robustness. So we enlisted several friends with command of various talents in piano, guitar, and flute performance. They were asked to film a short clip of their instrument performance, and these clips will undergo the same preprocessing as were training data, then conduct inference by our trained model. Classification labels with their confidence were overlaid on the frames of the video clip output. This time we received mixed results.

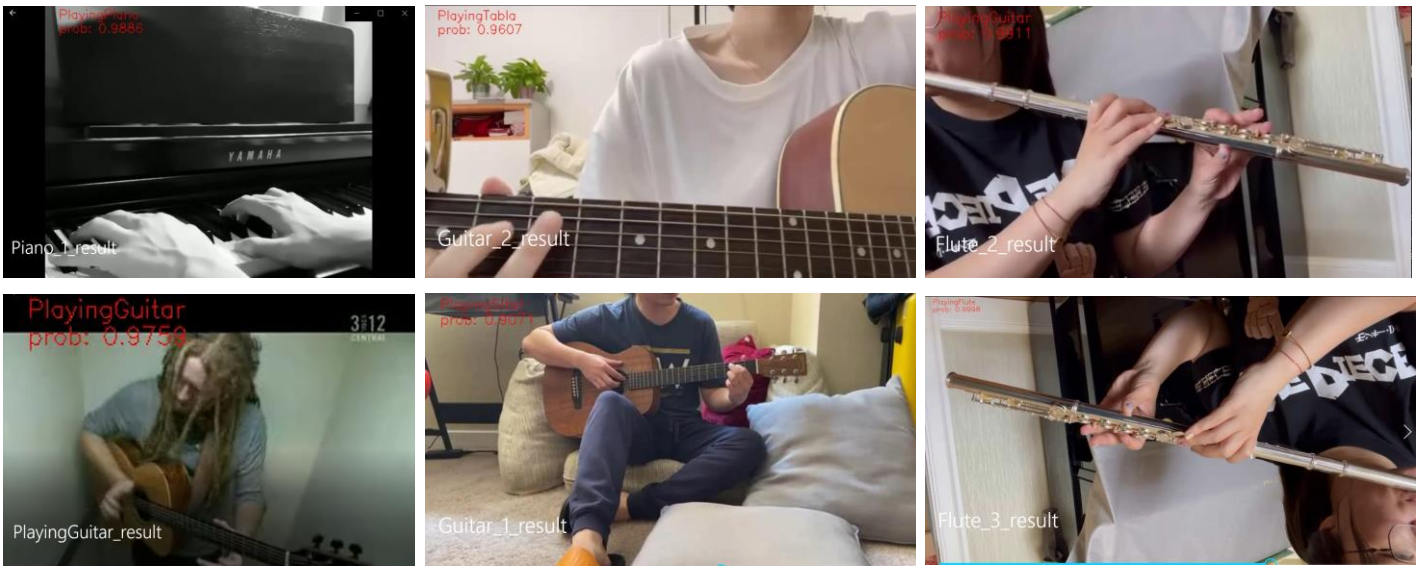

Fig.4 Inference on External Videos

On the lower left, a guitar performance clip from the UCF101 test set was used as control. Our model could classify it properly with high confidence. On the upper left, a piano-playing clip could be classified correctly as well, even if it is a greyscale video. But things went sideways than on. Our model failed to classify two guitar performances on the center, and label them as tabla and sitar. Indeed, these two instruments share similarities in exterior shape and playing methods with the guitar, and these two clips were filmed from different distances and angles with respect to the performer than our training data, but this certainly shows the model is not robust enough. An even more bizarre outcome was revealed, when we find the model could not properly classify a flute performance on the upper right, while when we flip that clip upside down like in the lower right, the model recognizes the action! The reason behind this unexpected behavior is yet to be identified.

LIMITATIONS AND FUTURE WORK

As it showed previously, there are some limitations in the model. It will misclassify some musical instruments when we apply other videos that are not included in the UCF101 dataset as a test set. The model is not robust enough.

One of the reasons is that the resolution of the video in UCF101 is lower than the one that people typically record nowadays. Due to the input quality differences, these high-resolution videos have never been represented and trained in the dataset, and the performance suffered.

There are several methods that could improve the robustness of our model.

First, we could use the transfer learning techniques to train the model with other higher resolution videos.

Second, it will be possible to perform augmentation techniques to expand the training dataset. The most common augmentation techniques include randomly rotating, flipping shifting, and normalizing the color in videos. Other augmentation skills include using an autoencoder (AE) to impose the feature representation with uniform distribution and applying the linear interpolation on latent space that generates a much broader set of augmentations for classification tasks [4].

As for the network architecture, RNN + Conv2d, and two-stream networks[4] were introduced to improve the performance of the video classification tasks. The two-stream convolutional neural network extracts Spatio-Temporal information with two same neural networks with different weights.

Some other techniques could be our potential improvements, such as attention mechanism and loss functions.

CONCLUSION

We proposed a model named VC3D to deal with the video classification task. It achieved 100% accuracy in the Playing Instruments group for the UCF101 dataset.

During this project, we encountered some obstacles and solved them smoothly.

The first obstacle we had was the data loader. The Pytorch build-in API for this dataset was bugged. By querying the error message, we found that this was a problem caused by the incompatibility of the PyTorch Dataloader and the format of video datasets, which we deemed to be beyond our capability to repair. Fortunately, because the dataset UCF101 is open-source data that is generally used in the video classification tasks, we implemented a dataloader building on a base code. After parsing the meaning of each line of the code, we have a general idea of how the video dataloader functions and learned to write and debug one on our own.

The second one is the model architecture introduced in the previous section.

In this project, in addition to learning the video classification methods, we also learned how the number of neural changes through the hidden states. This gives us a deeper understanding of how convolutional neural networks behave.

REFERENCES

[1] Tran, D., Bourdev, L., Fergus, R., Torresani, L., & Paluri, M. (2015). Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 4489-4497).

[2] Soomro, K., Zamir, A. R., & Shah, M. (2012). UCF101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*.

[3] Liu, X., Zou, Y., Kong, L., Diao, Z., Yan, J., Wang, J., ... & You, J. (2018, August). Data augmentation via latent space interpolation for image classification. In *2018 24th International Conference on Pattern Recognition (ICPR)* (pp. 728-733). IEEE.

[4] Simonyan, K., & Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, *27*.

[5] jfzhang95. *Pytorch-video-recognition/dataset.py at master · JFZHANG95/Pytorch-video-recognition*. GitHub. Retrieved April 29, 2022, from https://github.com/jfzhang95/pytorch-video-recognition/blob/master/dataloaders/dataset.py