Convolutional Neural Networks for Classification of Noisy Sports Videos

Joey Asperger
Stanford University
joey2017@stanford.edu

Austin Poore
Stanford University
hapoore@stanford.edu

Abstract

Effectively classifying amateur videos of activities is an important problem to solve, since it is much more difficult than classifying stable, professionally filmed videos. In this paper, we apply a variety of convolutional neural network architectures on the challenging Sports Videos in the Wild dataset, which contains non-professional, noisy videos of sports filmed on mobile devices. Using our own relatively simple architectures, we were able to achieve up to 47.7% validation accuracy, while our most successful architecture based on a pre-trained ImageNet model achieved 84.3% validation accuracy and 75.9% test accuracy for the 30-class classification problem.

1. Introduction

Our work is related to the task of activity classification in noisy sports videos. In particular, our dataset consists of thousands of amateur videos of a number of different sports and activities, shot on mobile device cameras in a variety of locations and from a variety of non-standard angles. We seek to automatically identify which sport is present in each of these videos. This problem is widely applicable beyond the sports world; classifiers and architectures that perform well at this task on our sports videos should also be capable of good performance on activity classification tasks in a variety of other areas, such as surveillance or automatic content tagging for online videos. Video processing can be computationally challenging, as videos are composed of many individual image-sized frames which need to be processed. One approach to video classification, then, is to simply examine frames as individual images and attempt to classify them, and then combine the results into a single output classification for the video as a whole. While this approach makes some intuitive sense, because a human can likely distinguish between hitting a baseball and shooting a basketball with even a single frame, it also seems to discard much of the temporal information encoded in the source videos. Thus, we will also attempt to capture this temporal information in some way, in addition to simply processing the spatial dimensions of input frames.

1.1. Previous work

Recent work has been done on the video classification problem with convolutional neural networks, much of which has focused on ways to preserve the temporal information encoded in video frames. For instance, Ji et al. in [1] propose what they refer to as 3D convolutions across spatial and temporal dimensions, with the goal of extracting information about movement that occurs between frames. Meanwhile, Ng et al. [2] propose a pair of distinct approaches: the first relies on temporal pooling methods, while the second uses LSTM cells to process the videos, which have in this case been treated as ordered sequences of frames fed to an RNN. Meanwhile, Wang et al. [3] propose slightly more widely-applicable techniques for action recognition that go beyond layer types and model architectures, including the idea of breaking each video up into chunks, classifying each chunk as its own miniature video, and then aggregating the predictions across all of the chunks to produce a final prediction for each full video. Karpathy et al. [4] also investigated different strategies for video classification that relied on fusing information from different frames. In particular, they invested 3 different strategies: early fusion, where the frames are fused in the first convolution of the network, late fusion, where the frames are fused just before the softmax classifier, and slow fusion, where pairs of frames are fused gradually throughout the network. They found that slow fusion performed the best out of those models. Some previous work has also incorporated additional information such as audio spectrograms and optical flow information. Wu et al. used these sources in addition to the frame pixel values as features in a model that used convolutional nets and LSTMs to classify videos [5]. Our work is somewhat unique in particular due to our chosen dataset: *Sports Videos in the Wild,* which we will discuss in the next section.

1.2. Dataset

For our project, we used the *Sports Videos in the Wild* (SVW) dataset from Michigan State [6]. The dataset

consists of roughly 4200 videos (average length 11.6 seconds) encompassing 44 different activities across 30 different sports. Baseball, for instance, includes both hitting and pitching, which makes classification more challenging. All of the videos are labeled with their corresponding sport, and about half are labeled with the specific activity. The most interesting characteristic is that the videos are user-generated, which means the activities take place in a variety of non-standard locations and are filmed on smartphones or tablets. The dataset is also relatively small, which suggests that a transfer learning approach, where we begin with a pre-trained model and then tune it on our specific dataset, might be most effective. In our experiments, we split the dataset up into 3400 training examples, 300 validation examples, and approximately 500 test examples. To demonstrate how noisy our dataset is, figure 1 below shows screenshots of 7 different videos that are all labeled as football. Notice that the videos in this figure include an a fairly normal game, and indoor game, single players kicking a ball or doing drills, a kid in a football helmet pushing a tire, and several people in the ocean miming football throws.

Figure 1: Videos labeled as football

In the paper that introduces the SVW dataset, the authors provide three baseline models, none of which involve neural networks, evaluated on the sport classification task. Two of these achieve accuracies nearing 40%: their context-based approach and their motion-assisted context approach achieve accuracies of 37.08% and 39.13%, respectively. Their more complex motion-based approach, which applied video stabilization and other techniques, achieved an accuracy of 61.53%. The authors also note that this approach yielded accuracies above 90% on a nicer dataset, which hints at the complexity of classifying these noisy, real-world sports videos. Rachmadi et al. [7] are able to achieve accuracies just above 80% using CNNs and classifying on a frame-by-frame basis.

2. Technical approach

We tested a variety of features and model architectures in search of the best possible performance we could find. Initially, we tried a variety of relatively simple models, which we trained from scratch. For most of these models, we extracted a fixed number of frames from each video (most often 10 or 16, evenly spaced), which we then cropped to either 270x270, 224x224, or 200x200 depending on the current experiment. We sampled video frames at a rate of 10 frames per second, so when we refer to taking consecutive frames of a video they are spaced 0.1 seconds apart temporally.

2.1. Simple baseline model

To begin, we implemented a simple baseline model that uses two conv-relu layers on the individual frames with a stride of 2, followed by averaging the output across all frames and finishing with a fully connected layer to compute scores for each class. We also updated it to include batch normalization following each of the two relu steps. Finally, we incorporated dropout in addition to batch normalization, which ultimately yielded the best performance. We also experimented with adding one or two additional convolutional layers and a second affine prediction layer, but surprisingly these more complex architectures yielded lower validation scores.

2.2. Chunking

One of our attempts to capture some additional temporal information from our videos was the idea from [3] mentioned earlier, where we segment each video into some number of chunks. Then, we treat each chunk as a video to be classified. Finally, we combine the chunk-level predictions into a single prediction for the whole video. This is somewhat similar to our naive approach with the simple model where we began with 10 frames from each video, except that rather than averaging the network's outputs for each frame and then classifying, we actually output a class prediction for each chunk. Then, we simply treat each chunk's classification as a single vote, and our video-level prediction is the class with the most votes.

We put this technique in practice with the simple model as follows: First, we looped all videos shorter than three seconds so that each video would be at least three seconds long. Then, we broke each video into ten chunks, and

extracted the first three frames from each chunk. We classified each chunk based on its three frames using the simple model with dropout and batch normalization, averaging the three frames and outputting a prediction for the chunk, then aggregated the 10 chunk scores and took the most popular prediction as our output for the whole video.

2.3 Three-Dimensional Convolutions

Our next attempt to capture temporal information from our videos was the three-dimensional, or temporal, convolution layer described in [1]. At a high level, the idea follows quite naturally from two-dimensional, or spatial, convolutions with which we have become intimately familiar this quarter. We expand the idea to three dimensions by stacking contiguous frames on top of each other to form a three-dimensional tensor with dimensions of width, height, and time, and then use three-dimensional convolution filters, which slide along the spatial dimensions of our input tensors. Dot products are computed over the three-dimensional intersection between the filter and the input tensor, rather than the two-dimensional intersection that we're used to for image recognition.

We tried a simple architecture to see if it would be worthwhile to pursue further. Our initial model featured two layers of temporal convolutions. The first layer featured 16 filters with spatial dimensions of 7x7 and a temporal width of three, and the second layer featured an additional 16 filters of spatial size 5x5 and again a temporal width of three. After each layer, we used a ReLU activation and then 2x2 maximum pooling along the spatial dimensions. We then reshaped the output into a vector and used an affine layer to make the final classification prediction. For each video, we sampled thirty frames (looping for the few videos under three seconds) at even intervals. We also tried the model with batch normalization, but it did not perform as well. This model did not perform as well as the simple model featuring two-dimensional convolutions, so we chose not to pursue it much further.

2.4 Simple Recurrent Neural Network

We also attempted a simple version of a recurrent neural network (LSTM) to see if preliminary results looked fruitful. Recall that a recurrent neural network maintains an internal hidden state and gets a new input at each time step. It then computes a function of the hidden state and the new input to produce an output vector and a new hidden state. We thought a recurrent neural network might be a logical choice for our problem because it can take a variable number of frames as input, and thus we could sample frames from each video at the same interval, rather than evenly spaced, which we thought might be more informative. We built our LSTM layer on top of our simple two-layer architecture, passing each frame's encoding to the LSTM step by step rather than averaging all of the frames together for a specific video. Unfortunately, the results from our simple model were substantially worse than for our simple model, so we decided not to spend additional time fine-tuning it.

2.5 Pre-Trained Models

Next, we tried using pre-trained ImageNet models fine-tuned on our video dataset. For our pre-trained model, we chose to use Inception-Resnet-v2, described by Szegedy et al. in 2016 [8], because it was the highest performing ImageNet classification model we could find and had a pretrained implementation readily available in TensorFlow-Slim. Inception-Resnet-v2 combines the Inception modules created 2015 by Szegedy et al. [9] that utilize a Network in Network structure and increase the expressivity of a network while constraining computing requirements and the residual connections proposed by He et al. [10] in Resnet that dramatically increase the depth with which networks can be efficiently trained. This model was used on individual frames, and the outputs were either averaged across all frames or fed into an LSTM before classification. We experimented with allowing different amounts of layers to be fine-tuned in this model because we expected that the lower levels of the model detected features like edges and corners, which would be the same in our dataset as in ImageNet. We tried 3 variations of this: one where the entire model could be fine-tuned, one where only the top half could be fine-tuned, and one where only the final 2 layers could be fine-tuned.

We also experimented with different methods of incorporating the temporal information after using Inception-Resnet-v2 on the individual frames. First, we tried just using the first frame of the video to see if we could obtain a high classification accuracy from the single frame. Next, we tried a simple model that averaged the output from 10 evenly spaced frames in the video before feeding into a softmax classifier. Lastly, we tried feeding the frame outputs into an LSTM. For this model, we looked at the first 8 seconds of the video and fed a frame from every half-second into the LSTM. Any videos shorter than 8 seconds were zero-padded. We then used the output of the final LSTM cell to feed into a softmax classifier.

3. Results

Table 1: Validation Results of Different Models

Model	Validation Accuracy
Simple Baseline	43.4%
Chunking	47.7%
3D Convolutions	41.7%
Inception-Resnet-v2, single frame	72.3%
Inception-Resnet-v2, averaged across frames	84.3%
Inception-Resnet-v2, LSTM	74.7%

As is shown in Table 1 above, the pretrained model dramatically outperformed the models trained from scratch. Among our models trained from scratch, the chunking model performed the best, and the 3D convolution. Overall, our best model by far was the Inception-Resnet-v2 model averaged across frames, which surprisingly outperformed the LSTM version.
Within the pretrained model, we experimented with allowing different amounts of backpropagation into the pretrained layers. As shown in Table 2 below, it performed the best when allowing backpropagation through all layers.

Table 2: Effect of allowing different amounts of backpropagation in Inception-Resnet-v2 (trained only on single frame of video)

Model	Validation Results
Backpropagating through entire model	72.3%
Backpropagating through top half	71.0%
Backpropagating only through 2 top layers	61.7%

Finally we tested our best model, the Inception-Resnet-v2 averaged across frames, on the test set and achieved a test accuracy of 75.9% While this was a significant drop from our validation accuracy of 84.3%, this drop was not particularly surprising to us because the validation accuracy was fluctuating significantly between epochs. Furthermore, this is still a very impressive result considering this is a 30-class problem with very noisy data. Table 3 below shows our accuracies on each class.

Table 3: Accuracies by class

Gymnastics	72%	Golf	67%
Diving	75%	Hurdling	71%
Tennis	71%	Discus	78%
Longjump	100%	BMX	86%
Pole Vault	91%	Javelin	50%
Rowing	100%	Hammer	64%
Skiing	71%	Football	83%
Volleybal	98%	Running	12%
Cheerleading	90%	Highjump	85%
Baseball	88%	Basketball	66%
Shot Put	77%	Wrestling	82%
Swimming	100%	Boxing	71%
Hockey	86%	Socker	83%
Bowling	90%	Skating	100%
Archery	90%	Weightlifting	100%

4. Analysis

We break up our analysis into several topics of interest.

4.1 Transfer Learning

It is not particularly surprising that the models that were by far the most successful were those based on the pre-trained Inception-Resnet network. Pre-trained models like that one have already been trained on a huge dataset like ImageNet to effectively detect edges, corners, and other important attributes, and are thus able to output very useful features. This model is also far more complex than any of those we trained from scratch, so it is not an entirely fair comparison. Nevertheless, the performance difference was substantial.

However, we were surprised that our best results came when allowing full backpropagation. Since our dataset is so small compared to the ImageNet dataset used for pretraining, we expected that allowing full backpropagation would lead to substantial overfitting. Furthermore, the lower levels of the network theoretically are mostly just detecting things like edges and corners, which are the same in any dataset. However, full backpropagation still performed the best. This suggests that detecting sports requires different low-level features than the categories in ImageNet. One possible explanation is that the ImageNet classes are concerned mostly with the foreground object, whereas with sports, the background is much more important (i.e. a person running could mean virtually any sport depending on what kind of field or court they are on.)

Another surprise to us was that the LSTM performed significantly worse than the averaging model. This model didn't overfit as much on the training data, so we think it's possible that more hyperparameter tuning could have improved the model. Since this model was extremely slow to train (~1.5 hours per epoch), we weren't able to experiment as much as we would have liked with different hyperparameters.

4.2 Overfitting

Our dataset is relatively small, and one thing we struggled with throughout our experiments was the gap between training and validation accuracy. All of the models we trained from scratch tended to overfit our training data after a few epochs. Introducing techniques like batch normalization and dropout did tend to improve validation accuracy, but the models incorporating those ideas continued to overfit. We did not find a good solution to mitigate this problem on our trained-from-scratch models, though we did not try things like data augmentation or training on additional labeled data to try to narrow the gap in performance between training and validation.

4.3 Temporal Convolutions

We were a bit disappointed by the performance of our simple architectures featuring temporal convolutions, since these models did not even perform as well as our simple architectures featuring standard convolutional layers. One possible issue we identified was that there was a reasonably wide variety in source video length, which meant that the thirty frames we sampled from each video featured different-length gaps in between them. This seems like it could certainly harm performance of three-dimensional convolutional layers, because the stacks of frames along temporal dimensions now span different lengths of time, which might make it difficult for our three-dimensional features to learn good, generalizable weights. Perhaps a more effective approach would have been to zero-pad shorter videos, or even to simple look all of the short videos to a longer length, and then simply attempt to classify a fixed-size chunk from each video, such that the gaps between frames were equivalent for all videos.

4.4 Feature Selection

We did a bit of experimentation with different features, but our tests were certainly not comprehensive. In general, our approach was to take some number of evenly spaced frames from each video, most often between 10 and 30. The biggest decision we made related to feature selection was at the final prediction layer for our models. Initially, we started by averaging the features derived from each frame and then outputting a prediction for each video's averaged features, but we also tried other methods like feeding each frame's output features to an RNN or even just concatenating them all into a single long vector. Results were far from conclusive as to which method was best, but in general the averaging method outperformed the RNN-based methods we tried. We also had to crop our frames, because various videos had different sizes. We initially cropped each frame to 270x270, and later reduced to 224x224 and even 200x200 for some models, due to computational restraints. This cropping was likely unfair to some videos, because larger videos had more information discarded than smaller videos. We would have liked to try zero-padding as a way to standardize video size without this asymmetric discarding of information, but at times it would have made our training quite slow.

5. Conclusions and Future Work

Given additional time and computational resources, there is much more we could try in order to achieve optimal performance on our dataset.

5.1 Conclusions

Our biggest takeaway from this project, without a doubt, is that using a pre-trained model is definitely the way to go. Not only did we not have to make complicated architectural decisions ourselves, but we also reaped the benefits of hours and hours of GPU training time where the network already learned how to recognize informative features. After a relatively quick fine-tuning on our actual data set, we were able to nearly double the performance of the simpler models we developed and trained from scratch. There is a reason these models achieve state-of-the-art performance: they're really good. Our work shows that they are not just good at their original ImageNet task, either. They can easily be tweaked to classify different,

messy datasets as well. This lesson is broadly applicable to any smaller dataset; when in doubt, try starting with a good ImageNet model and see if transfer learning yields good results. In our experience, it is likely to.

Another observation we made was that in general, we had more success with treating frames as still images to be classified than by attempting to capture complex temporal features. Intuitively, this makes some sense, because with a few exceptions, a person can likely distinguish among all of these sports pretty easily given a handful of frames. That said, simple techniques like breaking videos into chunks did appear to yield some performance gains, suggesting that temporal information is still useful.

Much like the results from [5], our models tended to have much more trouble with certain classes. This suggests that something about these classes is confusing or complicated. In the case of running and long jump, for instance, the sports look quite similar, especially when the system only gets to see one frame at a time. This leads us to believe that an approach like one-versus-all models for these problematic classes combined in some sort of ensemble might be a worthwhile addition to a real system where our goal was to classify this dataset as accurately as possible. Baseball, long jump, and running, for instance, tended to have relatively low accuracies across models. If we could find a model that did well on these three classes, it would likely be a great addition to our system and increase our overall accuracy by a few percentage points. Sadly, we did not have a chance to experiment as much as we would have liked in this regard.

5.2 Future Work

Many of our suggestions for future work are based on issues raised in the conclusion section or earlier in the paper. If we had more time, we would like to conduct more experiments with different model architectures and features. Our best results by far came from the pre-trained ImageNet model architectures we tried, so it seems to make the most sense to continue work on those models initially, perhaps by exploring different features which could be fed into the model. We would also be interested in running additional experiments related to the layers we built on top of the pre-trained model. So far, our most successful prediction layer averaged the results of ten input frames before making a prediction with an affine layer, but this approach does not seem to fully capture the temporal information available to us in videos. In fact, it effectively treats the task as one of image classification, rather than video classification. We would like to run more experiments to try to find new and innovative ways to capture additional temporal information, because unfortunately the methods we did attempt did not work particularly well.

Our experiment with chunking the videos yielded gains of a few percentage points with the simple model. This suggests that perhaps a chunking approach paired with one of the pre-trained models might yield small performance gains as well, so we would be interested in trying that and other techniques in conjunction with our best-performing models to try to obtain peak performance.

References

[1] Ji, Shuiwang, et al. "3D convolutional neural networks for human action recognition." *IEEE transactions on pattern analysis and machine intelligence* 35.1 (2013): 221-231.

[2] Yue-Hei Ng, Joe, et al. "Beyond short snippets: Deep networks for video classification." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

[3] Wang, Limin, et al. "Temporal segment networks: towards good practices for deep action recognition." *European Conference on Computer Vision*. Springer International Publishing, 2016.

[4] Karpathy, Andrej, et al. "Large-scale video classification with convolutional neural networks." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2014.

[5] Wu, Zuxuan et al. "Multi-Stream Multi-Class Fusion of Deep Networks for Video Classification." 2015

[6] Safdarnejad, Seyed Morteza, et al. "Sports videos in the wild (SVW): A video dataset for sports analysis." *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*. Vol. 1. IEEE, 2015.

[7] Rachmadi, Reza Fuad, Keiichi Uchimura, and Gou Koutaki. "Combined Convolutional Neural Network for Event Recognition." *Korea-Japan Joint Workshop on Frontiers of Computer Vision*. 2016.

[8] Szegedy, Christian et al. "Inception-v4, Inception-Resnet and the Impact of Residual Connections on Learning." 2016

[9] Szegedy, Christian et al. "Going Deeper with Convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2015.

[10] He, Kaiming et al. "Deep Residual Learning for Image Recognition." 2015.