Human Action Recognition Using CNN and BoW Methods

Stanford University
CS231A Computer Vision Spring 2016

Max Wang
mwang07@stanford.edu

Ting-Chun Yeh
chun618@stanford.edu

Abstract

We set out to validate, examine, and improve on methods in recognizing human actions in still images. Human action recognition is still a challenging task, despite recent advancements in object recognition, due to the variabilities in real-world images containing human subjects and surrounding objects and environments. We experimented with traditional Convolutional Neural Network and Bag of Word methods, while also using various classifiers in conjunction. After examining outputs and weights in both methods, we saw that noise is often considered important features. This lead to further examination and experimentation of our data set and its properties. We will discuss our many findings in this paper, with a surprising takeaway that specific background and contextual information are essential for classification.

I. Introduction

Recognizing human actions is a popular area of interest due to its many potential applications, but it is still in its infancy. Successful human action recognition would directly benefit data analysis for large-scale image indexing, scene analysis for human computer interactions and robotics, and object recognition and detection. This is more difficult than object recognition due to variability in real-world environments, human poses, and interactions with objects. Since researches on human action recognition in still images are relatively new, we rely on methods for object recognition as basis of our approaches. In particular, we were interested in seeing how convolutional neural networks (CNN)[1] perform in comparison with past feature selection methods such as Bag of Words (BoW). Also, we experimented with various supervised and unsupervised classifiers, examined our methods' properties and effects on our action data set, and also pre-processed our data set in order to better our results.

II. Related Work

In past decades, many ideas proposed to solve the human action recognition problem. Some people put interest on understanding human-object reaction. Bourdev et .al [9] proposed Poselet to recognize human body part and further research on human pose relation. Although those methods have very impressive result, hand-crafted feature method still can't be very generalized to all purpose. They all are used for specific goal.

To conquer that, Krizhevsky et al. [5] first used Convolutional Neural Network(CNN) for image classification in 2013. Convolutional Neural Network is a powerful method because, unlike handcrafted feature methods, it learns features from whole image through forward and backward processes in deep layer structure. In 2014, Ji, Shuiwang et al. [6] first apply Convolutional Neural Network to recognize human action in video and popularized CNN methods. However, this is unlike our project goal. There's a necessity to using still images over videos, which is less challenging due to inclusion of temporal information and foreground subject segmentation, because some actions such as "holding an object" and "drinking" are rather inactive and analysis would only be based on static features. To reduce the cost of CNN, it is common to use a pre-trained model, as demonstrated by Chi Geng et al. [8] use pre-trained CNN model to learn features from images and classify images by SVM. To reduce the over-fitting problem of CNN, Srivastava et al. gave "dropout" which prevent neural units from co-adapting too much to address over-fitting problem.

We examined Convolutional Neural Network (CNN) and Bag of Word (BoW) methods that have proven to be successful in object recognition, and applied them to the human action recognition problem. Based on our initial experimental results and analysis of our data, we attempted to further improve our results using supervised and unsupervised methods. To fully understand CNN, we looked into feature extracted by CNN. We then drew insightful observations that we think would help future work in this area.

III. Methods

Data and Setup

We utilized Caffe, Python (and pycaffe), and Matlab to create and run our CNN and BoW models. We rented a server with 30GB of harddrive space and 4GB of Nvidia GPU memory, costing roughly $400 including usage time. Due to hardware limitations, we had to reduce our data set size, so we chose to classify 8 actions out of the Stanford40 data set (B. Yao November 6-13, 2011), using 1839 images for training (and validation, for CNN), and 456 images for testing. With such small data set, we allocated more images

[1] Motivation came from neither partner having prior CNN experiences

for training, which only had 100 images per action for training. Instead, we used a train-val-test ratio of 7-1.5-1.5. We were at risks of overfitting, but we took precautions to prevent overfitting.

As a default, we used the images as given in the data set. Then, we applied cropping to our images in two ways: one with a tight bound to isolate our subject and nearby objects, and one that is 50% larger than our tight bound to capture some background information. Lastly, we pre-processed images to a color segmentation process using k-means.

Bag of Words

In general, objects in an image can be described by feature descriptors, forming a histogram for the image. A collection of histograms from different images form the Bag of Words (BoW) model, which can be learned by a classifier. During training, we used a Scale-Invariant Feature Transform (SIFT) method to extract features, then we utilized Spatial Pyramid Matching (SPM) (S. Lazebnik 2006) to obtain histograms from increasingly small sub-regions. Stacking these histograms together helps us maintain spatial information. We then used K-means method to cluster the final histograms into K code words. During testing, match the histogram of the input image with our BoW model. BoW is unaffected by position and orientation of objects in the image, and the SPM method gives us more spatial information to help us localize objects.

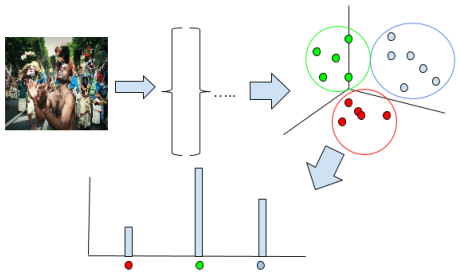

Figure 1 BoW Histogram

Convolutional Neural Network

CNN is a different method of obtaining image features and training on feature representations in high dimensional space. It has been quite successful in recent years, since its introduction in 2012 (Alex Krizhevsky 2012).

We used Caffenet (Jia 2014) architecture as the basis to our experiments. It is similar to AlexNet, but pooling is done before normalization in Caffenet. In brief, Caffenet has 5 convolution layers followed by 2 fully connected layers and a softmax layer. We trained using pre-trained weights, which have ran for 350,000 iterations, to give better generalization and to prevent overfitting our data. This is our control case.

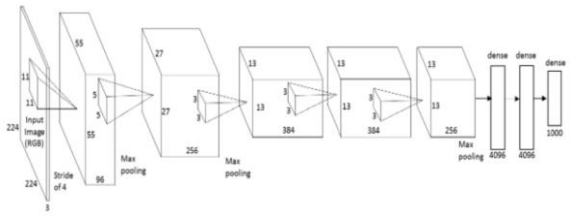

Figure 2 Caffenet architecture

Then, we experimented with changing learning rates and hyperparamters for each layer, which are: kernel size, padding amount, stride, and number of outputs. Hyperparameter tuning involves changing the sizes of the CNN layers, creating a very different CNN, despite having the same number of layers. To study the effect of locality sizes on our results, we conducted two tests with the first layer's kernel size being 15 and 7, respectively, and different amounts of paddings were used to keep other layers the same. In a third test, we also changed the first layer's kernel size from 11 to 27, then decreased our kernel sizes in the following layers until the 5th layer matches the original 256x13x13 dimension.

We also created CNN's from scratch, using our custom-defined layers and hyperparameters. Below is a summary of our three custom models (we only show kernel size, k, since we only adjusted other parameters to suit our new k):

Custom 1: Conv(k=11) → RelU → pool → norm → Conv(k=3) → RelU → Conv(k=3) → RelU → FC → Softmax

Custom 2: Conv(k=13) → RelU → pool → Conv(k=7) →RelU →pool→ Conv(k=3) →RelU →pool→FC→ FC→Softmax

Custom 3: Conv(k=13) → RelU → pool → Conv(k=7) → RelU → pool→ Conv(k=3) → RelU → pool → FC → Dropout → FC → Softmax

Our custom CNN 1 is a small CNN with 3 layers. The other two are larger. The difference between our custom CNN 2 and 3 is that custom CNN 3 has a dropout layer. This is to prevent our network from overfitting by giving each neuron a 0.5 probability that its activation will become zero in each iteration. In other words, a dropout of data. This avoids co-adaption of units.

We also ran Googlenet for comparison, which uses an "atypical" architecture embedded with inception layers that contain multiple convolutions. In terms of recognition, Googlenet is known to yield better results than Caffenet, but it is more difficult to fine-tune so we kept Caffenet as our basis.

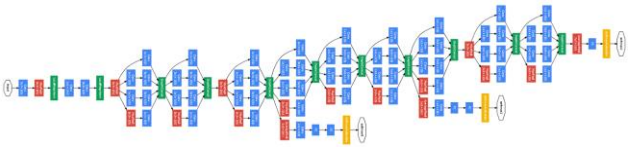

Figure 3Googlenet architecture

t-Distributed Stochastic Neighbor Embedding

We used the t-SNE algorithm to help us visualize the features obtained from the last FC layer of the Caffenet in relation to our actual data. Features from this layer is a high dimensional histogram for each image, and t-SNE allows us to cluster these images together in 2D space. With t-SNE, we set similarities of high dimensional points (distribution Q) and low dimensional points (distribution P) as two different joint probabilities, where a higher probability indicate similarity. The cost function is then a Kullback-Leibler divergence of distribution Q from P.

$$KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}}$$

This leads to the minimization problem.

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

Coincidentally, since t-SNE is an unsupervised method to cluster our data, we also tested to see how well it classifies our data by applying a K-means algorithm on top of t-SNE.

CNN + Classifier

Similar to using the t-SNE algorithm, we extracted activations from the last fully connected layer of our CNN's as features and put them through various classifiers. We are interested in using features from CNN for image classification problem, but skip the Softmax layer that Caffe uses.

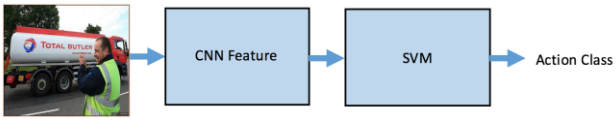

Figure 4 Our pipeline: applying SVM on extracted features

Support Vector Machine

SVM is to find a hyperplane that give the largest minimum distance to training data. It is to optimize

$$min_{\gamma,w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m} \xi_i$$

$$s.t. \qquad y^i(w^T x^i + b) \geq 1 - \xi_i, i = 1, \dots\dots, m$$

$$\xi_i \geq 0, i = 1, \dots\dots, m$$

The second term $C\sum_{i=1}^{m} \xi_i$ let us can have margin less than 1. C control the two goal want to achieve: Keep $\|w\|^2$ small and make margin less than 1. To use this linear SVM on our multiclassifier data set, we used "one vs one" comparison. We had experimented with "one vs all" method but decided "one vs one" yields better results.

Multi-Class Support Vector Machine

We used one versus one for our dataset. For one versus one method, if we have N class, there will be N(N-1)/2 classifier. Each classifier is for two classes from our dataset. We are going to solve the following optimization problem.

$$\min P(w^{ij}, b^{ij}, \xi^{ij}) = \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_n \xi_n^{i,j}$$

$subject\ to$:

$$(w^{ij})^T \emptyset(x_n) + b^{i,j} \geq 1 - \xi_n^{i,j}, y_n = i$$

$$(w^{ij})^T \emptyset(x_n) + b^{i,j} \leq -1 + \xi_n^{i,j}, y_n \neq i$$

$$\xi_n^{i,j} \geq 0,\ n = 1,2, \dots\dots, N(N-1)/2$$

Each classifier will vote to one class, and the most voted class will be final result

Additive Chi Square Kernel

Additive Chi-square kernel does normalization to the feature histograms, so that spikes in the histograms will not be heavily affect the result. We used the "one vs one" comparison.

$$k(x, y) = \sum_i \frac{2x[i]y[i]}{x[i] + y[i]}$$

K-nearest neighbor algorithm

Choose an integer K. KNN classifier will find the nearest K neighbors of x0 from training data. According to the class of nearest k point, it give conditional probability for each class.

$$P(Y = j|X = x_0) = \frac{1}{K}\sum I(y_i = j)$$

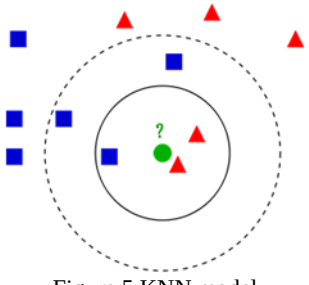

Figure 5 KNN model

Random Forest

A random Forest method is an ensemble method. It build a series of simple trees which are used to vote for final class. For classification, RF predict the class that

predicted by most trees. The predictions of the RF will be the average of the predictions by simple trees

$$P = \frac{1}{K}\sum_{I=1}^{K} Ith\ tree\ prediction$$

IV. Experiments and Results

Default CNN

We first obtained data from running our data with pre-trained weights of Caffenet and Googlenet. We obtained these accuracies:

Model	Top1 accuracy
Caffenet	0.8223
Googlenet	0.8552

We then examined some properties of Caffenet. We verified that our model has converged by looking at the 1st layer weights to verify there's no noise.

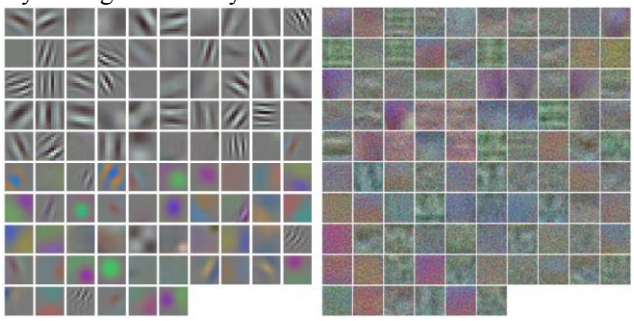

Figure 6 Nicely converged 1st layer weights (left) vs noisy weights (right)

Testing on training data yielded an accuracy of 0.9833. This may indicate some overfitting, but we believe it is mostly due to the original models doing well. This is because using pre-trained weights and giving 0 learning rates to some of the weights should provide enough generalization.

We examined Caffenet's first layer's outputs and noticed that while Caffeent can capture large features correctly, it sometimes recognizes background noise and irrelevant information as key features.

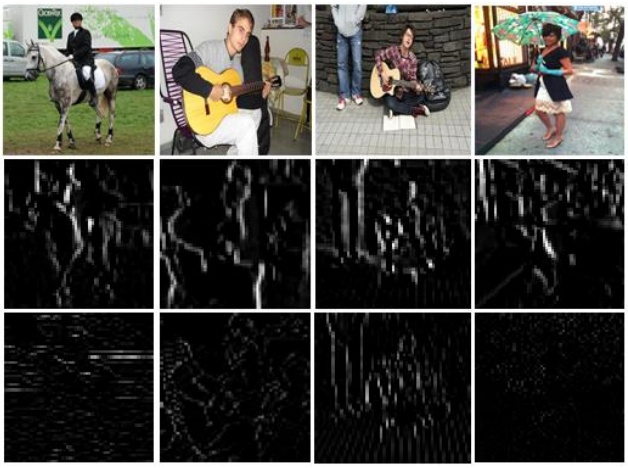

Figure 7 First layer outputs. 2nd row shows main features and local objects are captured. 3rd row shows some noise is captured.

For improvement, we believe it would be beneficial to filter out noise and have larger locality of features.

It becomes difficult gauging the activations in later layers, due to the locality of each neuron, so it was not used.

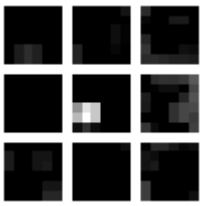

Figure 8 Activations from 5th layer of CNN

Custom CNN

Based on preliminary results, we wanted our CNN to capture larger features and ignore smaller objects or noise. Hence, we created our custom CNNs, as described in the Methods section.

CNN	Top 1 accuracy
Kernel size 7	0.5679
Kernel size 15	0.5021
Kernel size 27	0.4208
Custom CNN 1	0.1251
Custom CNN 2	0.5501
Custom CNN 3	0.5317

None of our custom CNN's matched the default model's accuracy. This could be we did not have the time to train our models for long enough because we could only run for 20,000 iterations, which takes half a day. But we noticed that the 1st layer's weights appear to converge nicely, so it's also possible that the default Caffenet was designed to be the best CNN of its kind. Hyperparameter tuning is, we realized, an optimization problem of its own.

We noticed that, unexpectedly, a larger kernel size at the first convolution layer yielded lower accuracies. We compared the 1st layer's outputs and noticed that, while a larger kernel size does give us larger locality and capture bigger features in the images, as intended, it is perhaps too broad for our CNN. The smaller kernel size, on the other hand, captures too much detail.

Figure 9 Layer 1 outputs, same column is from the same model. From left to right: K=15 K=11, K=7

Bag of Words

From looking at BoW code words, we also thought it would be beneficial to filter out background noise.

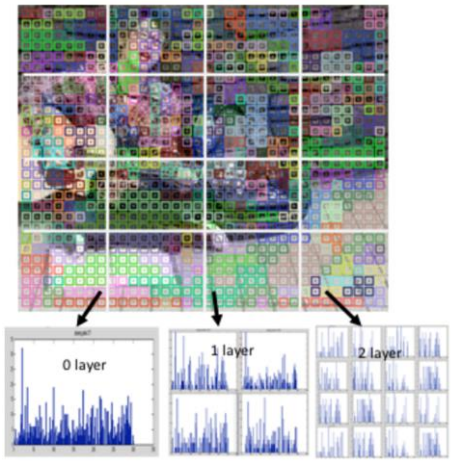

Figure 10 BoW features

We tried to filter out the background by changing our K size for the k-mean cluster, but it's not inherently obvious how many codewords to use. We tried K=200 and K=300.

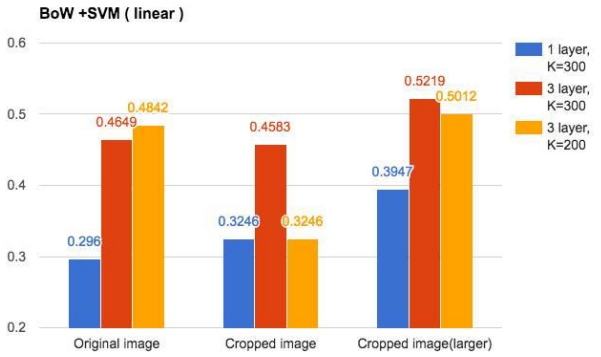

Figure 11 Linear SVM performances

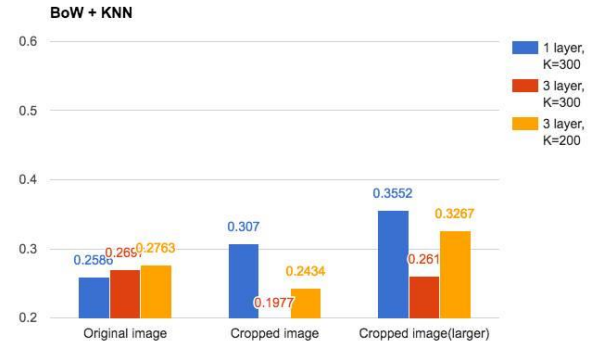

Figure 12 KNN performances

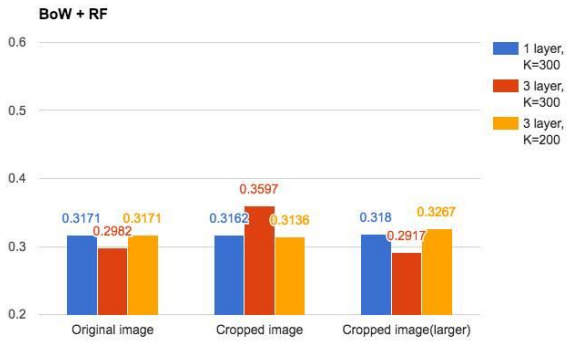

Figure 13 Random forest performances

We saw that for the most part, K=200 performed better. But this may not be optimized, since number of code words is heavily related to the properties of images, so there is no best way to find K but trial and error, like finding CNN hyper parameters. According to our result, K=200 is better, so we can deduce that SIFT doesn't use as many distinct features from our images, so that we don't need too many words.

A more useful takeaway is looking at our results of our cropping. After we cropped the image based on tight bounding box, we saw that accuracy actually dropped. This is contradictory to our expectation. We thought that removing

background noise would reduce error and improve our result. However, we realized that contextual information is actually important for classification.

We then expanded our bounding box by 1.5 times to include local background information. As predicted, we saw an improvement in our result.

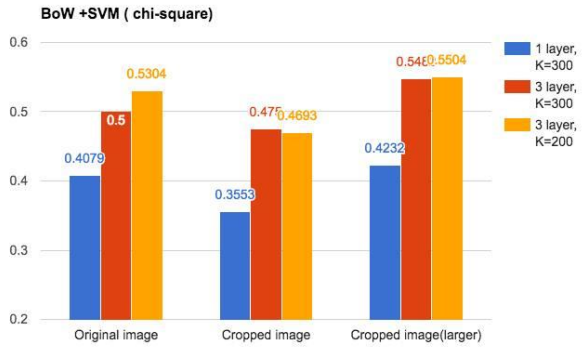

Figure 14 Plot of our results

CNN + CLASSIFIER

Fine-tuned CaffeNet

Top 1 accuracy			
	Original	Cropped	Cropped (1.5x)
CaffeNet	0.8223	0.7785	0.8377
CaffeNet+SVM (linear	0.8469	0.7938	0.8728
CaffeNet+SVM (chi-square)	0.8487	---	---
CaffeNet+KNN	0.8443	0.7982	0.8618
CaffeNet+RF	0.8333	0.7872	0.8465

As shown above, if we train SVM and other classifiers on top of features extracted by CNN, we achieve better results than using CNN alone. This was surprising, since CNN's own accuracy was already high.

We again thought it may be due to the overfitting issue described in previous section. So, when we use SVM for classification, we made SVM resistant to overfitting by tuning the parameter C.

Although kernel trick perform better than linear SVM in BoW model, we didn't use it on CNN feature because CNN feature is very high dimension. Using kernel on CNN will be time consuming with not a better result. So, we simply use linear SVM here for CNN feature.

We observe that SVM, KNN, RF all perform well on our dataset when using CNN features, even though in our BoW model KNN and RF both did badly. Even though CNN is not perfect at extracting features, it is much better than BoW model, which takes in too much noise from the image.

We saw CNN+KNN have even higher accuracy after we cropped the image. Table below show some predictions using CNN+KNN on different cropped images. We can see that the background is a contributing factor. Image 3 was classified as climbing because of the wall background, so does the image 4. After we cropped the image and put tight bounding box on action, image 3 and image 4 became right but image1 was missed. Without rock in image1, it was classified as jumping. In our expanded bounding Box, predictions for images 1,3, and 4 became correct..

We can see that the background is necessary when the action relies on the environment. Some action is highly related to the background, like climbing, where as some do not, like jumping. If we could recognize the relationship between the background and the action, we can achieve better results.

number	1	2	3	4
action	Climbing 090	Jumping 078	Drinking 171	Jumping 021
				
prediction(original)	climbing	climbing	climbing	climbing
prediction(cropped)	jumping	climbing	drinking	jumping
Prediction (larger cropped)	climbing	climbing	drinking	jumping

Un-Fine-tuned CaffeNet

Top 1 accuracy			
	Original	Cropped	Cropped (1.5x)
CaffeNet	0.7084	0.6012	0.7114
CaffeNet+SVM (linear	0.8421	0.7807	0.8421
CaffeNet+KNN	0.7390	0.6798	0.7675
CaffeNet+RF	0.7215	0.6008	0.7324

We also tested the classifiers on non-trained CaffeNet. Surprisingly, we found SVM give a pretty good accuracy. It is only a little lower than fine-tuned feature. KNN and RF are not like SVM, their accuracy is much lower than fine-tuned caffeNet feature. This confirms that Caffenet's pretrained model does a very good job at recognizing objects, such that when we insert our data set we do not need to train much.

Feature Examination: t-SNE

After applying the t-SNE method, we noted that the accuracy was only 0.5203, much like our other classifiers. What's more interesting, though, is visualizing our data. We see that images with clearly distinct objects (holding an umbrella, riding a horse, playing guitar, etc) are more distinguishable. On the other hand, actions that require environmental interactions (jumping, climbing) are not as obvious. Also, images taken from afar or from

unconventional angles would be harder to cluster. This could be due to the introduction of background noise or occlusion. It becomes obvious that pre filtering our data set would be an important step prior to training.

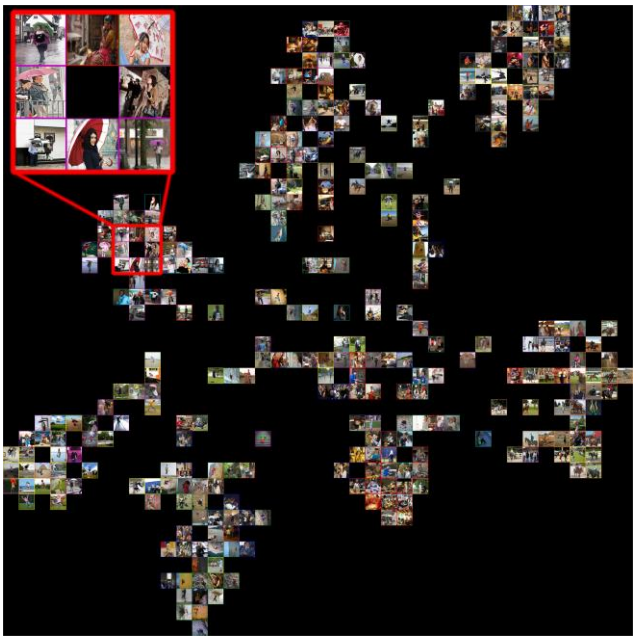

Figure 15 t-SNE visualization

Data Processing

We attempted to further pre-process our images using color segmentation with k-means algorithm. However, the results were not desirable most of the time. We see that this method works great on images where the human subject is clearly separated from the background, but it fails to do the same when there are other objects near the human subject, especially when the colors are similar. While the idea of segmentation is novel and may provide more accurate results, the color segmentation method is not ideal for such wide-range data.

Figure 16 Results of sgmentations on different images.

Conclusion

We experimented with and validated many methods and techniques in our project. The most useful takeaways for future work is that, for either supervised or unsupervised learning, it is important to include sufficient but not excess background and contextual information prior to training for human action recognition. The key point is how to select the region from image. We saw that cropping is a strong tool to use, but we cannot crop too much or too little background.

Then, we found that KNN performs well with fine-tuned CaffeNet model on our dataset. KNN is a very fast calculating model. For future work, we will test and evaluate KNN using the whole 40 action dataset. This is because CNN does a good job extracting features, so when used in conjunction with KNN, we achieve a much better result, as oppose to using low level features from BoW.

In general, CNN is a great tool at extracting features from images, even though it lacks the ability to distinguish subject, object, and background, similar to BoW. Even so, it significantly outperforms BoW model, as we expected from literature. For small size dataset, using SVM, KNN on CNN feature gives even higher accuracy than CNN itself. We thought that CNN could overfit such small size dataset, but it is possible to prevent overfit with CNN settings. In small size dataset, it may be more accurate to combine SVM,KNN with CNN feature.

Code link:
https://drive.google.com/a/stanford.edu/folderview?id=0B0GAxnZFVLhBTmVuRWVxeUdTcGs&usp=sharing

V. References

[1] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional. NIPS Proceedings.

[2] B. Yao, X. Jiang, A. Khosla, A.L. Lin, L.J. Guibas, and L. Fei-Fei. November 6-13, 2011. Human Action Recognition by Learning Bases of Action Attributes and Parts. Barcelona, Spain.: Internation Conference on Computer Vision (ICCV).

[3] Chi Geng, JianXin Song. 2015. Human Action Recognition based on Convolutional Neural Networks with a Convolutional Auto-Encoder. 5th International Conference on Computer Sciences and Automation Engineering.

[4] Girshick, Ross, et al. Proceedings of the IEEE conference on computer vision and pattern recognition. "Rich feature hierarchies for accurate object detection and semantic segmentation." 2014.

[5] Ji, Shuiwang, et al. n.d. 3D convolutional neural networks for human action recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions on35.1 (2013): 221-231.

[6] Jia, Yangqing and Shelhamer, Evan and Donahue, Jeff and Karayev, Sergey and Long, Jonathan and Girshick, Ross and Guadarrama, Sergio and Darrell, Trevor. 2014. "Caffe: Convolutional Architecture for Fast Feature Embedding." arXiv preprint arXiv:1408.5093.

[7] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deepconvolutional neural networks. Advances in neural information processing systems.

[8] S. Lazebnik, C. Schmid and J. Ponce. 2006. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06),.

[9] Lubomir Bourdev, Jitendra Malik. Poselets: Body Part Detectors Trained Using 3D Human Pose Annotations