CS230

Yog.ai: Deep Learning for Yoga

Anna Lai
alai2@stanford.edu

Bhargav Reddy
brkreddy@stanford.edu

Bruis van Vlijmen
bvlijmen@stanford.edu

Abstract

Many varieties of physical training have benefited immensely from improvements in Pose estimation of humans from images leveraging A.I. However, the field of yoga remains relatively unexplored. The final goal of Yog.ai is to use pose recognition as a tool to allow for a person to practice different yoga poses and receive feedback on their form using their webcams or phone cameras. This would be ideal for beginners to get a yoga practice started with minimal investment. This project covers the pose classification portion of this idea developing upon a baseline yoga-pose classification model from A. Marchenkova. As Pose Estimation is a well explored field within Computer Vision, we can leverage the public knowledge and use it for our application. Our Yog.ai framework uses CMU's open source pose estimator OpenPose as a pre-processing module before our trained CNN. We are able to achieve a test accuracy of 78%, improving upon Marchenkova's initial model. We expect to yield even better results, if we were to use a larger dataset.

Git Repo: https://bitbucket.org/yogai/cs230_yog.ai/

1 Introduction

Pose estimation of humans from images is interesting for its applications in physical training, animation, video games, dance choreography, and VR. Deep learning has been applied to the estimation of body joints from 2D images, even when parts of the body are not visible (3).

The goal of our project is to allow for a person to practice different yoga poses and receive feedback on their correctness while in the comfort of their homes without having to sign up for expensive yoga classes. This would be ideal for beginners to get a yoga practice started with minimal investment of time, money and effort. The input to our algorithm is an image of a person doing yoga which is run through CMU's open source tool for Pose Recognition, OpenPose (1), for initial key-point recognition. The preprocessed skeleton image is run through our convolutional neural network to output a predicted yoga pose.

2 Related work

Our approach is based on a Deep Learning approach to classify yoga images by Georgia Tech researcher Anastasia Marchenkova (2). Her dataset formed the basis of our own dataset, which we will expand upon in the next section. Marchenkova was able to reach a 70% accuracy with her model. Given that her approach is the best of only a few - documented yoga pose classification networks, we consider her approach as the State Of Art in this field. When looking at Pose Estimation in a wider respect, besides yoga, there are many well documented strategies and frameworks that estimate 'pose-keypoints'. In recent years, there has been a surge in research publications by both academic research institutions as well as corporate research teams, such as Google DeepMind, on the topic of

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

pose recognition (3). The sourcecode and networks of some of these approaches have been made open to the public. Most notably, OpenPose, is one of these networks and is developed by CMU (1) and made available to the public. OpenPose uses a 'Convolutional Pose Machine' (CPM) as a key component of their pose estimation framework. A CPM consists of a sequence of predictors which are trained to make dense predictions locally, using multi-part context, on each image location. This makes OpenPose a very strong pose estimation framework with higher 95% precision on the FLIC Dataset for wrist and elbow estimations (4).

3 Dataset and Features

We start with a data set of images of people doing 10 different yoga poses (2) partitioned into train and dev with our addition of a 'random' no pose class with images of random people (Opelt, 2004). This was done to enable the network to learn when people are not striking a yoga pose at all - for example just standing, walking by or sitting. In total approximately 1000 images were used, with 75% train and 25% test.

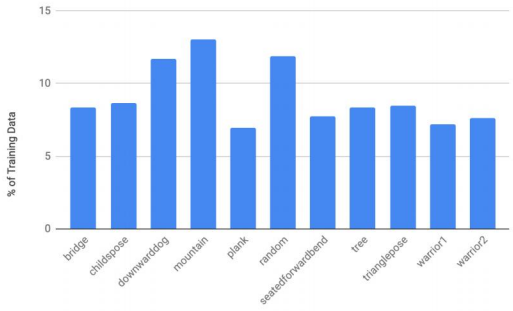

Figure 1: Distribution of training data across classes

Using the OpenPose network, we pre-process our data into images of skeletons against a black background. For our application we used OpenPose to estimate 18 keypoints. Even though OpenPose is able to generate up to 25 keypoints, these extra keypoints would be redundant for our application as they add extra keypoints to the feet and hands.

Figure 2: Sample of pre-processing with OpenPose: Two Stanford grad students playing football.

This not only attenuates biases in the data set (ie. background, gender, attire, etc.), it leaves the yoga pose classification sub-problem for us to further dive into and become creative with different CNN architectures for better classification performance.

3.1 Data Augmentation

In terms of data augmentation, the OpenPose processing can be regarded as a data augmention step. Besides that, the model uses data augmentation functions in Keras to augment the data: re-scale, shear, zoom, rotate slightly, flip horizontally on the OpenPose processed data set to produce an augmented set of 30,000 images between train and test.

Figure 3: Examples of data augmentation: rotate, flip, shear, and crop. In reality these augmentation steps would be executed on the skeleton version of these images.

4 Methods

Having this new dataset, we considered 3 approaches of creating and training a classifier network. One approach takes key-point coordinates for each joint and run through a fully connected Deep NN. The other approach convert the RGB skeleton to grayscale reducing the number of channels and run through the CNN classifier mentioned in the step above. The third approach which is what we decided on for our model is to use the RGB skeleton image dataset run through the CNN classifier. We decide this given that a CNN is not sensitive to the relative location of a figure in an image, compared to the fully connected Deep NN, and we preserved the RGB images as the colors of the skeleton map to the limbs they represent.

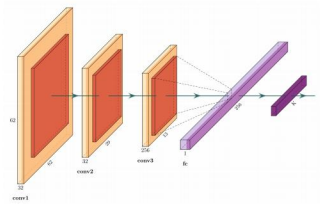

Figure 4: Yog.ai CNN architecture consisting of 3 convolution and max pool layers with a dense layer and softmax activation.

For training the network, we used the Adam optimizer. The loss function we used for training is the categorical cross entropy function. With M the number of classes, $y_{o,c}$ is the binary indicator whether the label c is the correct classification for the observation o.

$$L_{\text{entropy}} = -\sum_{c=1}^{M} y_{o,c} \log p_{o,c} \tag{1}$$

.

5 Experiments/Results/Discussion

5.1 Hyperparameter Tuning

Since we have a relatively small dataset, in the first round of hyperparameter tuning, we varied the learning rates and batch sizes while keeping all the others constant. This way we were able to obtain the best combination of learning rate and batch size. 4 such networks are given below with network 1 being the best one.

Network Name	Learning Rate	Minibatch Size	Dense Nodes	Training Accuracy	Test Accuracy
network 1	**0.0033**	**16**	128	0.947	**0.712**
network 2	0.0001	64	128	0.925	0.668
network 3	0.0087	16	128	0.938	0.702
network 4	0.0063	64	128	0.918	0.645

With the learning rate and Batch size now fixed, in our second round of hyperparameter tuning, we varied the number of dense nodes and the number of convolutional layers to find the optimum. 5 such networks are given below with, network A being the best one in terms of Test Accuracy.

Network Name	Learning Rate	Conv Layers	Dense Nodes	Training Accuracy	Test Accuracy
network A	0.0033	**3**	**256**	0.943	**0.778**
network B	0.0033	3	80	0.943	0.725
network C	0.0033	3	128	0.942	0.726
network D	0.0033	2	128	0.946	0.707
network E	0.0033	4	40	0.935	0.734

5.2 Results

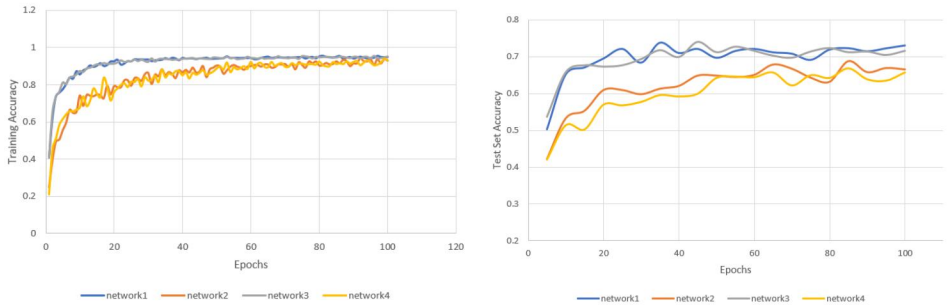

Figure 5: 1st round of Hyperparameter Tuning (Left - training set accuracy; Right - test set accuracy)

From the 1st round of hyperparameter tuning, we learnt that network 1 with learning rate = 0.0033 and minibatch size = 16 would yield the highest test set accuracy. The training set accuracy was very similar for all the networks.

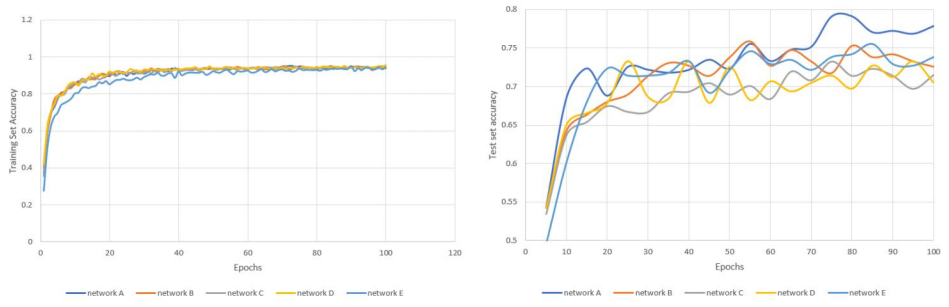

Figure 6: 2nd round of Hyperparameter Tuning (Left - training set accuracy; Right - test set accuracy)

After the 2nd round of hyperparameter tuning, we were able to zero in on the number of convolutional layers and the number of dense nodes. Network A was the best configuration - Learning Rate = 0.0033; Minibatch Size =16; No. of Dense Nodes = 256; No. of Convolutional Layers = 3. Network A yielded a test accuracy of 0.778 when stopped after 100 epochs. However, if we were to stop near iteration 80, the test accuracy would be equal to 0.799.

Another important thing to note is that the graph of test set accuracy vs number of epochs is very noisy. This can be attributed to our small test set.

For our best performing model we find the following:

Class	Recall	Precision	F1
Bridge	0.750	0.750	0.750
Childs Pose	0.600	0.750	0.667
Downward Dog	0.929	0.813	0.867
Mountain	0.735	1.000	0.847
Plank	0.800	0.727	0.762
Random	0.889	0.500	0.640
Forward Bend	0.667	0.800	0.727
Tree	0.800	0.800	0.800
Triangle Pose	1.000	0.800	0.889
Warrior I	0.750	0.600	0.667
Warrior II	0.833	0.625	0.714

A striking result is the high Precision and low Recall score for the Random class. The low Recall score can probably be explained by the fact that the Random class was more represented in the training set, therefore the network might have been overfitting to the Random class. Furthermore, the Random class has images that can be of any type of pose. This can be quite confusing for the network, as some of these images might look like they belong a different class. From our confusion matrix we find images of the Random Class, in which people are just standing, have been classified as the Mountain Pose which causes its Recall Score to be lower than its perfect Precision Score.

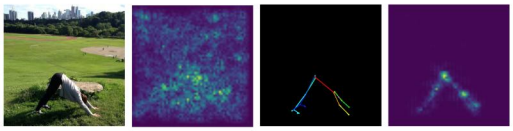

Figure 7: Saliency map of our Yog.ai network (right) on a yoga pose image compared to that of the baseline without OpenPose preprocessing

Starting off, our hypothesis was that using the OpenPose pose predication model as a pre-processor, would remove noise from the input images. In fact, we leverage OpenPose's well trained and complex network to reduce a noisy input image of a person's yoga pose, into a cleaned up skeleton doing a yoga pose. Having just this skeleton image as the input to the model, our CNN can focus on learning the pose classification.

To check whether this hypothesis holds true, we generated saliency maps for a network with without the OpenPose preprocessing module and compared the results. Figure 7 shows these results. Clearly, the image on the right has no incurred losses from any other parts of the image but the pose itself. Whereas the baseline model (no OpenPose module), finds losses in the background of the noisy image.

6 Conclusion/Future Work

Our hypothesis was that incorporating OpenPose as a 'preprocessing' step, would allow the network to train on the noise-free skeleton images. This proved to work well, as can be seen in the set of saliency maps in Figure 7. We were able to reach a test-accuracy of 78%, compared to the SOA model (2) (70%). However, we believe that we are be able to improve our results by expanding our dataset - as the small dataset made our network vulnerable to overfitting.

For next steps we would incorporate more poses and train on a larger data set and experiment with different architectures. We would also like to extend the functionality to providing the user feedback on their form. After the pose that the user is trying to do is classified, it would be compared with an ideal pose calibrated to the user to give advice and avoid injury. Looking further ahead with more computational power, we would also like to analyze dynamic poses with RNNs.

7 Contributions

Equal split of work.
Expertises
Reddy: Virtual Machine training
Anna: Error Analysis
Bruis: OpenPose

References

[1] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. 2018.

[2] Anastasia Marchenkova. Convolutional neural network for classifying yoga poses, 2018. `https://github.com/amarchenkova/yoga-pose-CNN/`.

[3] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. 2014.

[4] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. 2016.