Stick Figure Drawings with Human Motion Tracking

Aman Kar, Ethan Cao, Neil Sharma
Data Science Student Society

Abstract

Introducing pose estimation. For those who do not know what pose estimation is, it is a technique used in AI computer vision projects to track a person's or an object's motion in a video. It may be easy for a person to see a person do a dance or an exercise and recreate their movement on their own. However, for computer this task is not so simple. In our project, we use this technique to recognize key nodes in a person's structure to track their motion in a video and return a stick figure that is copying that movement. The nodes we focus on are the ankles, knees, hips, pelvis, thorax, upper neck, top of the head, wrists, elbows, and shoulders. Through just these 16 nodes we are able to accurately represent a person's motion as a stick figure. From these 16 nodes we are able to create a sort of skeletal figure that displays these landmarks as coordinates. Utilizing the python turtle library we were able to connect these nodes in a way that will create a stick figure connecting nodes like a skeleton. With the help of some geometric calculation, we were able to take multiple frames of these skeletal figures and create smooth movement paths for our stick figure.

Introduction

In recent years, computer vision has emerged as a fascinating field with numerous applications across various domains. One particular area of interest is pose estimation, a technique used to track the motion of individuals or objects in videos. Pose estimation involves identifying and tracking specific key points, or nodes, in a person's skeletal structure, allowing for the recreation and analysis of their movements. By employing artificial intelligence and computer vision algorithms, pose estimation has proven to be a valuable tool in fields such as sports analysis, motion capture, human-computer interaction, and surveillance systems.

The ability to accurately track human motion from video data has important implications in diverse areas, including healthcare, sports training, virtual reality, and entertainment. Imagine being able to learn complex dance routines by observing an expert and having an AI system recreate the movements for you to follow. Or envision a virtual reality game that captures and mirrors your body movements in real-time, enhancing the immersion and interaction within the virtual environment. These are just a couple of examples of the exciting possibilities made feasible through pose estimation technology.

In this project, our goal is to implement a pose estimation system that takes a video as input and generates a stick figure that replicates the exact movements performed by the person in the video. By focusing on key nodes such as the ankles, knees, hips, pelvis, thorax, upper neck, top of the head, wrists, elbows, and shoulders, we aim to capture the fundamental elements of human motion.

To create the stick figure, we utilize the Python turtle library, a powerful graphics module that provides a user-friendly interface for drawing shapes and lines on a canvas. By connecting the nodes in the correct order, we can construct a skeletal figure that resembles a stick figure. To achieve smooth and realistic motion, we employ geometric calculations and techniques. By analyzing multiple frames of the stick figure's skeletal representation, we can derive smooth movement paths that accurately mimic the motions observed in the video. These calculations allow us to animate the stick figure and generate a lifelike representation of the person's movements. By combining AI and computer vision methodologies, we aim to demonstrate the potential of pose estimation in capturing and recreating human motion.

Resources

Languages:

- Python: The python language was used in this project for its flexibility and the many libraries within it that helped with machine learning tasks.

Libraries:

- OpenCV2 - The OpenCV2 library was used to allow video capture as well as certain preprocessing tasks before the image was fed into the model
- PyTorch - PyTorch was the machine learning library that we used in order to build out our model architecture
- Turtle - The Turtle library was used to draw the stick figures based off of our model's predictions
- Numpy - Numpy provided various math functions as well as arrays which were useful in handling the data
- Pandas - The model's predictions were stored in pandas dataframes which our model then read and turned into stick figure drawings

Dataset:

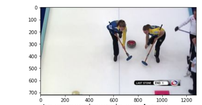
- MPII Human Pose - A dataset of over 25 thousand annotated images of people and the respective joints of their body doing various activities and tasks

Model Architecture:

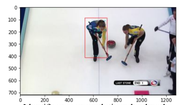
- ResNet 101 - The architecture of choice was ResNet 101 because CNNs are good for computer vision and ResNet 101 has a high-speed architecture for faster predictions

Methodology

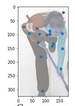
Data Augmentation, Network Architecture and Function Choice



An average image in our dataset



Identify area around given landmarks



Crop on area

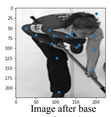
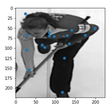
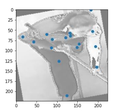


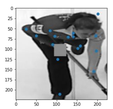
Image after base transformations



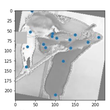
Mirrored



Rotated

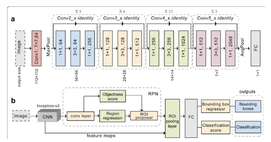


Cutout



Combination of all 3

In order to make the data useable, we had to filter out instances where parts of the body were cutoff, which were denoted with -1 values. We then identified where the body was by drawing a rectangle and cropping on the rectangle. Due to the relatively low amount of useable bodies (11, 231), we had to utilize data augmentation to artificially increase the dataset. Using these methods, we could take our training dataset of 9,000 to a maximum of 63,000.

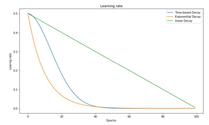


A Diagram of ResNet 101 architecture

We chose to use a ResNet 101 architecture due its ability to capture complex features and its previous successes in computer vision projects. ResNet gives us the complexity we needed to identify the 32 classes required

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

MSE-Loss is a simple loss function that is sensitive to differences and allowing us to simplify optimization



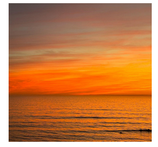
We used an exponential transformation to our learning rate (set at 0.01) which multiplies the learning rate by 0.9 per epoch, combating the problem of overfitting and allow the model to find the small details

Results

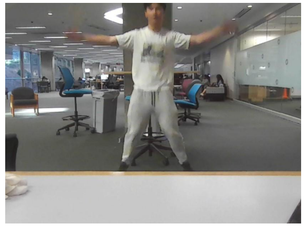
From end to end, the program that we created captured an image, normalized that image with data pre-processing methods, fed the processed image into the ResNet 101 model, and then created a stick figure video from the resulting predictions. While our model was only 74% accurate, it was good enough to get the shape of the person and their movement.

As mentioned in the methodology, we captured the image using OpenCV2 and then pre-processed the image through gray scaling and size normalization. We then fed this into our model and worked with the resulting prediction.

The resulting prediction was not necessarily prepared to be drawn by the Turtle library. Therefore, we needed to normalize each prediction based off of the center of the human body to keep our stick figures from leaving the frame. Turtle allowed us to draw the stick figures by connecting the nodes in a predetermined way. It also allowed us to add additional customization features and images in order to make the stick figure video more entertaining and useful. These features included choosing from various background images, changing the color of the stick figure, and choosing a hat for your stick figure avatar to wear. These were all extensions from our original goal of creating a stick figure video.



Example backgrounds that users can select



Above is an example of an input image captured from our computers that was subsequently fed into the model to get a prediction. This prediction was then transformed and edited to result in a video. One such frame from a resulting video is shown below where the stick figure matches the jumping jacks of the subject above.

Conclusion

In this project, we successfully developed a pose estimation system that takes video input and generates a stick figure that replicates the movements performed by the person in the video. By focusing on key nodes such as the ankles, knees, hips, pelvis, thorax, upper neck, top of the head, wrists, elbows, and shoulders, we were able to accurately represent human motion using just 16 nodes.

Utilizing the Python turtle library, we have effectively connected these nodes to construct a skeletal figure resembling a stick figure. Through the implementation of geometric calculations and techniques, we have analyzed multiple frames of the stick figure's skeletal representation, enabling us to derive smooth movement paths that faithfully mimic the observed motions in the video. This has allowed us to create a dynamic and lifelike animation of the stick figure, showcasing the original movements performed by the person in the video.

Throughout the project, we have leveraged several key resources. The Python programming language, along with libraries such as OpenCV2, PyTorch, Turtle, Numpy, and Pandas, has provided us with powerful tools for data preprocessing, machine learning, visualization, and result interpretation. Furthermore, the MPII Human Pose dataset has served as a valuable source of annotated images, contributing to the accuracy and effectiveness of our pose estimation system.

The implications of our pose estimation system extend across various domains. It holds potential for applications in sports analysis, enabling detailed motion capture for athlete training and performance assessment. Additionally, it can facilitate human-computer interaction by enabling gesture-based control systems and immersive virtual reality experiences.

Looking ahead, further research and development can enhance the capabilities of pose estimation systems. Improving real-time tracking capabilities, refining feature detection algorithms, and expanding the scope and diversity of training datasets could lead to even more accurate and versatile pose estimation models. Furthermore, the integration of advanced deep learning architectures and innovative techniques can potentially yield significant advancements in human motion analysis and synthesis.

Recommendations/Challenges

Since this was a computer vision project done by undergraduate students, there was a huge amount of learning needed to be done before any of the code had to be written. As we approached the project, we had to also ballance our other classes, and one of our members dropping out midway. If we kept a team of four, and didn't have to balance the work of our classes, we would have completed a more complete, and possibly more accurate, project.

As with any deep learning project, a major issue we faced was having to take the time to guess and check why our model wouldn't train in certain instances. We would invest hours slightly tweaking our hyper-parameters hoping to stumble on to the solution. Due to our inexperience in deep learning, and a clear understanding of the specific problem, it would oftentimes become a frustrating trial-and-error process.

A more controllable issue we faced was with our data augmentation, both in the data loader and when measuring accuracy for validation loss. When the images were augmented, the landmarks had to be augmented accordingly in order to be in the same relative position on the image. When measuring the accuracy of the model, we had to reverse engineer the augmentations in order to measure the accuracy on the original images. Since our data augmentation functions were built from scratch, it was a lengthy process to get to our desired result.

A larger, and less noisy dataset, would have been ideal for this project. The data in the MPII dataset weren't great for our use. Since we used a set number classes, images that had parts of the body hidden needed to be taken out of the dataset. This process took away more than a third of the total dataset. We look forward to exploring options that would allow us to work with this issue, but due to the lack of time to finish the project, we opted to utilize more data augmentation to increase our dataset.

In order to actually train the model, we utilized UCSD's JupyterHub, where we requested access to computers that could be accessed remotely. We would highly recommend any current UCSD student to apply for permission to use JupyterHub for their own project, as it provides the high powered hardware required for training purposes.

While the entirety of the project was extremely challenging, it has solidified for the three of us our passion for data science and machine learning. Even though we had to work through many obstacles to reach our result, we got our hands dirty, and learned a lot through the process.

Acknowledgements

Thank you Advaith Ravishankar for leading us to the resources possible for completing this project.

Thank you Gilen, despite dropping, we still appreciate your work ethic and the contributions you made for this project.