Invisible to Machine Perception: Attacking Pose Estimators with Attribution Methods

Alexander Lelidis

Master Thesis
May 2020

Prof. Dr. Markus Gross
Dr. Cengiz Öztireli
Marco Ancona

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

UNIVERSITY OF CAMBRIDGE

computer graphics laboratory

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

INVISIBLE TO MACHINE PERCEPTION:
ATTACKING POSE ESTIMATORS WITH ATTRIBUTION METHODS

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
LELIDIS	ALEXANDER

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
ZURICH, 31.05.2020	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Acknowledgements

I would first like to thank my thesis advisor Marco Ancona and Dr Cengiz Öztireli. Marco was always very helpful when I ran into technical problems as well as when I was struggling with the experiment design. Additionally, I want to thank him for being very responsive to messages. I also want to thanks Cengiz for his professional insights and for always proposing creative new methods and applications for our work.

I would also like to thank the volunteers who were involved in creating imagery data for this research project:
Drilon Shabani and Lea Sutter
Without their passionate participation creating these data would have been a very tedious process.

Finally, I would like to thank Michael Rabinovich and Radek Daněček for proofreading this thesis.

Abstract

Neural networks are currently the most accurate techniques to tackle computer vision problems. However, with the increase of accuracy, an increase in complexity has emerged leading to black-box systems. This is especially problematic in safety-critical situations. In this thesis, we are using various attribution methods in 2D or 3D to understand a human pose estimation model. During this process, we develop a new method for the 3D attribution case, called 3D Saliency map.

To test the robustness of this model we identify adversarial examples in 2D and propose a new method in the 3D domain for computing adversarial texture for clothing. We show that our approach works by rendering a video of simulated meshes with and without the adversarial texture and feeding into the human pose estimator. To quantify the results we develop three different metrics, measuring various aspects of the attacks.

Zusammenfassung

Neurale Netze gehören heutzutage zu den genausten Techniken um Computer Vision Probleme zu lösen. Jedoch, mit den Anstieg von Genauigkeit kam ein Anstieg von Komplexität, was zu sogenannten schwarzen Kasten System geführt hat. Dieses Verhalten ist besonders problematisch in sicherheitskritischen Situationen. In dieser Arbeit benutzen wir verschiedene Zuschreibungsmethoden in 2D oder 3D um ein besseres Verständnis für menschliche Pose schätzungsmodelle zu bekommen. Während dieses Prozesses haben wir eine neue 3D Zuschreibungsmethode, genannt 3D Ausstrahlungs Karte, entwickelt.
Um die Robustheit von diesem Modell zu testen, identifizieren wir verschiedene feindliche Beispiele in 2D und schlagen eine neue Methode vor wie man in dem 3D Bereich feindliche Drucke für Kleidung berechnen kann. Wir zeigen, dass unsere Methode funktioniert, indem wir ein Video rendern mit simulierten Netzen mit und ohne feindliche Texturen, welches wir in das menschliche Pose Schätzungsmodell weiterleiten. Um unsere Ergebnisse besser zu quantifizieren entwickeln wir drei verschiedene Metriken, welches verschiedene Aspekte der Attacke messen.

Contents

List of Figures

List of Tables

1 Introduction

In recent years, many sectors of the industry have realized the potential of Machine Learning (ML) and started incorporating it in their core technology stack and workflows. They are ranging from medical applications like detecting tumors in MRI data [MEDM12] or predicting lung cancer from CT scans [AKB$^+$19] over to classification of content in a photograph [SZ14, HZRS15, SVI$^+$15] to self-driving cars [BTD$^+$16]. The state-of-the-art techniques are even outperforming humans in various tasks right now and will in many more in the future [GSD$^+$17]. For example, in the image classification task on the Imagenet dataset [DDS$^+$09] with 1000 classes, where it is required to assign a certain class to an input image, Deep Neural Networks (DNN) have an error rate around 3%, while humans have around 5% [DK17].

Due to this success, ML is being deployed on a large scale. Applications where it improves the quality of the results, like Amazon Echo where the understanding of your particular voice gets better with its use [Kel19], to systems which were not even possible without ML. A great example is the Social Credit Score in China, which assigns every citizen a score representing their trustworthiness. The big difference to the European credit system is that is not only influenced by financial actions, but also by social ones. In combination with a large scale network of surveillance cameras, face recognition and activity recognition, this system runs nearly autonomously. For instance, if someone jaywalks and gets filmed by a traffic camera, the system can detect the face. It gets person profile, the current score and the "illegal" activity, which leads to a reduction of n points. A possible consequence would be that it is no longer allowed to take certain jobs or to travel abroad [Gei19].

Besides being a very controversial topic in the western world, the Chinese Social Credit system serves as a perfect example of how ML is used to make decisions affecting the lives of people every day. Most of the time these outputs are rather accurate, but most people are not very comfortable with the fact that a ML system makes life-changing decisions for them [Peg19]. This led the European Union to publish a law that requires every algorithm which makes significant user-level predictions to provide an explanation as well [Cou16]. This grants the European citizens the right to explanations.

This law, in combination with the very high success rates of ML predictors, shifted the research from solely optimizing task performance and produce some kind of black-box systems to taking other aspects into account as well. In particular, producing a non-racist system [ZS18] in terms of skin color or gender, a more safe system in the medical field [CDP$^+$19] or for autonomous cars, and generally a more understandable and interpretable system as applied in insurance rate computations. As mentioned in [RHDV17] the system "needs to be right for the right reasons".

To deploy ML in safety-critical environments, it is essential to quantify how the system is handling these situations. In general, the problem is that these situations are usually not enumerable. For example, it is not possible to count and even come up with all the safety-critical traffic situation in terms of autonomous driving. Therefore most research is trying to understand the behavior of these systems, leading to the newly emerging field of interpretability.

A ML model is interpretable if its decisions can be explained in an understandable term to a human with a non-technical background [DVK17]. However, the research community is missing a clear metric or definition of interpretability in terms of ML. A common way to approach this problem is to compute an importance value, called attribution, per input feature. This value should approximate the importance of that particular feature to the final output. However, the missing metric makes it hard to benchmark various explainability algorithms. Some might argue that interpretability is not even needed in all cases, which is true. Some systems just provide additional feedback to a human, which makes the final call or the system not life-threatening like an ad-person matching system. Additionally, it is desirable that the DNN finds new ways to tackle a problem, which were unknown to humankind before, which is conflicting the paradigm of "being right for the right reasons". A great example is the Multi-Agent Hide and Seek experiment from OpenAI [BKM$^+$19] where the agents create unforeseen scenarios, like using the 3D environment to building forts or in case of an unrestricted play area endless running away, and breaking the game.

What are good reasons for interpretability, besides providing explanations for users? One reason is to gain a deeper understanding of the system created. If they are well understood, it is easier to fix issues and improve them, as well as integrate domain knowledge. Furthermore, interpretability can also be used to approximate uncertainty and act as an estimate of how and why certain results appear. Finally, interpretability can help to determine the model's robustness to small changes of the input, leading to big changes in the output. By finding these small perturbation, a new field has developed in recent years, which is known as Adversarial Machine Learning.

The goal of Adversarial Machine Learning is to find small changes in the input to create a different output. The motivation of finding these is two-fold, first to show security issues and second to test the robustness of a system. These changes are usually unnoticeable to the user, comparing a corrupted image to an original. Most works are currently targeting digital inputs, like pictures or videos, by slightly changing pixel values ([SZS$^+$13], [SVS19] or 2.3). Some recent research started bridging the gap to the real-world and finding patterns or prints to get a different output. An example is the adversarial glasses, which fool a face recognition system by having a particular print on the frame [SBBR19].

In this thesis, we start in chapter four with evaluating and computing attribution of a 2D hand pose classification model. Moving on in chapter four to analyzing decisions and outputs of a state-of-the-art pose estimation network and gaining enough insights to set up an adversarial attack in the 2D image space. Finally in chapter five, bringing the attack from the 2D image plane into the 3D world, by leveraging differentiable rendering and computing adversarial prints for clothing. Additionally, using the differentiable pipeline to compute attributions in the 3D space. The main contributions of this work can be summarized as follows:

- We present a detailed analysis of the openPose network, including robustness testing and attribution of the openPose network in the 2D image space
- We introduce a novel way to bridge the gap between 2D and 3D by connecting openPose with a state-of-the-art differentiable render. We use the aforementioned setup to compute attributions and perform robustness testing.
- A new direction for fooling pose detection methods in 3D by directly optimize for the texture of clothing.

Additionally, we highlight all the technical contributions we made to the research community. Theses contributions should help others in their future work.

1. DeepExplain TensorFlow 2.X support The DeepExplain framework [Anc20] provides a unified implementation of the most state-of-the-art attribution methods, including gradient-based methods as well as perturbation-based methods. The tool is supposed to help researchers understand their models better as well as serving as a framework to compare new attribution methods to existing ones.
The current implementation relies on TensorFlow 1.X [AAB+15]. In June 2019, Google released TensorFlow 2.0 which enables eager execution and integrates Keras directly into the framework. This version bump introduced some breaking changes. To have a working DeepExplain framework, we refactored the current implementation to support both versions of TensorFlow enabling researchers to analyze new models as well as old ones.

2. Jupyter-browser connection JupyterLab [KRKP+16] is a web-based tool for interactive python development, data exploration and visualization. The biggest benefit compared to running plain python scripts is that the notebook keeps its state, and if the researchers need a different visualization, they can produce one in the next cell. In cases of a computationally heavy preprocessing step, this is extremely powerful. However, even with already many build-in visualizations and plugins, it is a common issue in advanced data analysis, that custom and interactive visualizations with a minimal user interface are required. To tackle that problem, we provide a simple way to share information between a web browser and a Jupyter notebook. This allows us to use the whole world of web tools quickly to visualize our preprocessed data. The IFrame plugin in the Jupyter allows us to see that website directly in the notebook, leading to a very natural visualization. The communication between the two entities is done via base64 encoded strings and a web socket.

3. Dataset creation A good dataset is paramount to many ML algorithms. However, getting real labelled data is a tedious and expensive manual process. Therefore, many researchers rely on digitally mocked datasets, using physically-based simulations or renderings. We provide a tool to generate a human pose image dataset. The user can control how many images they want as well as what poses the character should have. Additionally, a range of possible translations can be specified allowing to create datasets of different difficulty levels and classes. The final output consists of an XML scene description file (see A.6) as well as a rendered image (an example can be seen in Figure 4.24) using the Mitsuba renderer [NDVZJ19]. Additionally, the translations information is saved and used to render a mask covering the area of the poster. In Figure 1.1 a general overview of the creation pipeline is shown.

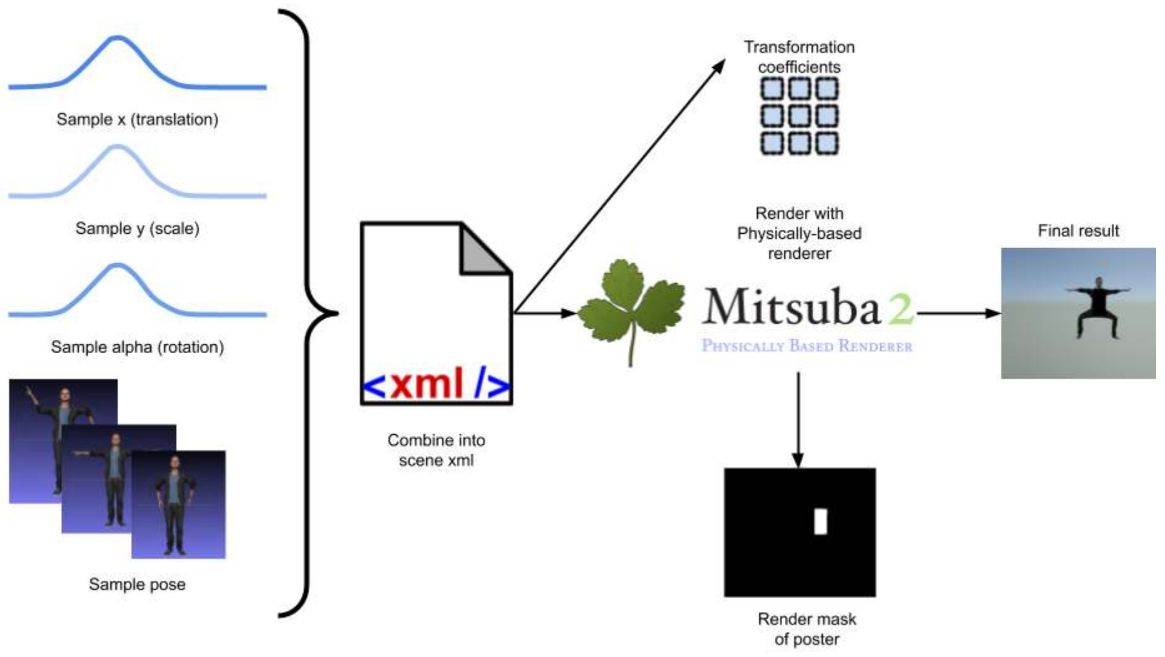


Figure 1.1.: *Dataset creation pipeline*

2

Related Work

In recent years the well-known field of Machine Learning started to grow significantly. Among various subfields, ML methods is commonly split into two main branches, supervised and unsupervised learning. Supervised learning is defined as the task of finding a function that maps an input to an output, based on labeled training data pairs [RN09]. Unsupervised learning, in contrast, does not require labeled data and tries to find previously undetected patterns in the data [HSSP99]. In this thesis, we are only going to examine supervised learning models.

Many different approaches exist to tackle supervised learning problems like classification, where the goal is to assign a label or class to an input, or regression, where the goal is to approximate a continuous function.

In recent years, deep neural networks have outperformed the more traditional well-studied methods [BCV13] like decision trees [Bre01] or support vector machines [MLH03]. A neural network can be seen as a loose approximation of a simple animal brain, where each node of the network model represents a neuron and the connections between them are the synapses. A signal is represented as a real number, a vector or sequence. Mathematically speaking, every neuron can be seen as computing a linear combination of the input components followed by a nonlinear transformation of the result.

$$y = f(\mathbf{w}^t \mathbf{x} + b) \tag{2.1}$$

Where f is the nonlinear function (e.g. tanh or sigmoid), $\mathbf{w}$ represents the connection strength and $\mathbf{x}$ is the current signal, which is being the input sample or, in case of a hidden layer, the output of the neurons in the previous layer. b represents the bias term. Many of these single neurons are getting combined into a large scale network, where an input feature vector is mapped to an application-specific output.

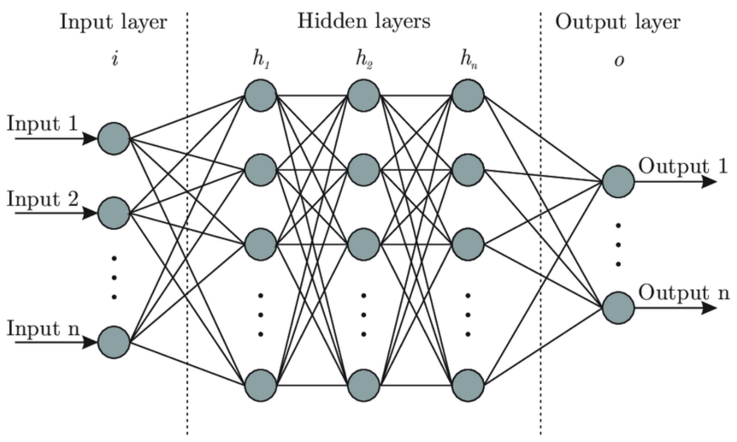


Figure 2.1.: *Example neural network architecture (image courtesy of [BGF17])*

The concept of neural networks has been around for decades, mentioned for the first time in 1946 by Warren McCulloch and Walter Pitts in [SM56] as a computational model. Quickly after in 1958 followed the invention of the Perceptron, an algorithm for binary classification by Frank Rosenblatt [Ros57]. Subsequently followed the first AI winter, until the discovery of the backpropagation rule based on the chain rule [Dre62] to efficiently train neural networks. However, being theoretical ahead of the time and limited by the hardware performance, the neural networks did not take off until late 2000. During that time, researchers discovered the potential of GPUs for large scale training leading to the ability to train very big and deep neural networks. From now on nearly everybody is able to train their networks leading to a wide variety of neural network types. The increasing computational power led unavoidably to increasing complexity, where models have millions of parameters, and humans are lacking the ability to understand the prediction internals anymore. Motivated by this fact, a new research field emerged, called interpretability or explainability of neural networks. This thesis is going to examine various applications in the visual field of explainability of existing neural network architectures.

2.1. Explainable artificial intelligence (XAI)

XAI deals with the problem of explaining decisions of artificial intelligence. One commonly used approach to tackle this challenge is assigning an importance value to each input feature for the neural network for a particular output. Due to the broad definition XAI can be applied to nearly every field in machine learning. For example, in natural language processing the authors in [Mar19] used XAI to visualize the importance of words in tweets used for a tweet classification problem. However, in this thesis we are going to tackle visual learning. In that particular field, two types of approaches are widely used to compute attribution.

The first type can be classified as *perturbation-based methods*, which directly compute attribution by occluding, removing or changing input features and computing the outcome with running a forward pass. This technique was successfully applied in the context of images [ZF13] where occluding single pixels or whole patches reveal the importance of that region.

On one hand, this allows a direct computation to the marginal influence of single attributes but tends to be very slow and most of the time a decision is not dependent on a single feature rather on a combination of them. In the context of images, it is usually feasible to choose groups due to the local coherence of the features, but the definition of removing a feature is not straight forward. Most of the time this is approximated by setting the pixel to black, which is not always a good choice due to the fact that black is a valid value for a pixel.

To overcome this limitation the research community started exploring *gradient-based methods*. The key difference is that the attribution for a sample is computed in a forward and backward pass. This assigns attribution to all features in a single step resulting in a much faster runtime. In the span of three years, many new approaches were proposed.

Taking a step back and looking at the basics we can define a neural network as a function taking a particular input x_i from a data set where $x_i = [x_1, \cdots, x_m] \in \mathbb{R}^m$ and producing an output $S(x) = [S_1(x), \cdots, S_k(x)]$, where k is the number of output neurons, which is equivalent to the number of classes. To compute the attribution R_i^c we need to pick an output neuron c, where we want to know which parts of the input feature x_i contributed positive to that decision and which ones negative.

One of the first techniques was a Saliceny map [SVZ13], where the attribution of a parameter is defined as the magnitude of its gradient. Resulting in heatmaps, which can be overlayed on top of the input. This technique visually displays the importance of every feature but is not able to differentiate between positive and negative contribution. This issue was tackled by Hechtlinger et al. in [Hec16] and computes the gradient of a given input for a given class and multiplies it with the input.

$$R_i^c(x) = x_i * \frac{\delta S_c(x)}{\delta x_i} \tag{2.2}$$

Resulting in a fast attribution including positive and negative attribution. However, in the image domain, it can happen that if an image has dark regions (black pixels) it tends to cancel out the gradient resulting in a misleading attribution.

A different approach was proposed by [STY17] and works by computing an average gradient starting from a user-defined baseline $\bar{x}$ and linearly varying the gradient towards x_i.

$$R_i^c(x) = (x_i - \bar{x}_i) * \int_{\alpha=0}^{1} \frac{\delta S_c(\tilde{x})}{\delta \tilde{x}_i}\bigg|_{\tilde{x}=\bar{x}+\alpha(x-\bar{x})} \tag{2.3}$$

while satisfies the property completeness ($\sum_{i=1}^{N} R_i^c(x) = S_c(x) - S_c(\bar{x})$) it requires the user to set a baseline, which is usually the black image and in certain cases not easy to determine. Technically this method takes more time to evaluate since the integral needs to be discretized.
These two methods have in common that they are able to output positive and negative attribu-

tion values. The current state-of-the-art methods are getting more advanced like the approach proposed in [AÖG19], where the authors relay on the concept of Shapley values, a well-known concept in cooperative game theory to assign importance to players in a cooperative game settings. By introducing a polynomial-time algorithm they are able to approximate them and make this approach feasible for real data. As mentioned before a considerable issue with all attribution methods is a missing metric to compare them and decide which one is better. Big a step forward was done in [ACÖG17b], where the authors formally prove the equivalence of two methods and describe four different gradient-based methods in a unified framework, which enables a better comparison between them.

These methods have been used for various problems and datasets. A very prominent case is where the authors in [LWB$^+$19] use Saliceny maps to find out what an image classifier had really learned on the Pascal VOC dataset [EVGW$^+$]. They found that different features can be used to detect a certain class. They looked into images containing various types of boats and found out what made the network predict that this sample belongs to the "boat" class. The compelling result is that these features vary from sample to sample. For example, the sails are the most prominent feature for a yacht, while for a motorboat the bow matters most. Additionally, the authors looked at the most important features for the images getting classified as members of the "horse" class and surprisingly the most important feature was the presence of a copyright watermark on the bottom of every image belong to the horse class. This shows us that neural networks can pick up correlations in the data, which humans did not see, leading to not learning the right features. The biggest problem about this issue is that it is not visible in the accuracy score of the test set. Most likely it will even improve the accuracy. State-of-the-art-datasets are becoming bigger and bigger and no one is looking at the samples anymore. Therefore these mistakes can easily get unnoticed showing the importance of XAI. To the best of our knowledge, these methods have not been applied to the 3D objects, specifically 3D meshes, which are a common representation of 3D shapes.

2.2. Adversarial Machine Learning

Most modern neural networks are not easily explainable to humans. Not only their prediction reasons are unknown but also their robustness to small perturbations. In general, a neural network can be seen as a statistical tool trying to capture a distribution of data, which is presented in samples from that distribution. The key assumption here is that these samples are representing this distribution well, and if the model is deployed in a real-world scenario, it will perform well. However, due to the high dimensional nature of the majority of the data, it is a challenging task to cover that distribution satisfactorily. This fact leads to the existence of slightly altered input data, which changes the output of the network significantly. Finding these perturbations is not always easy and therefore it turned into its own research field called Adversarial Machine Learning (AML).

These attacks started on vision-based systems because altering the input images is usually unnoticeable to the viewer. On the other hand, humans are very sensitive to small changes in text for NLP-based applications. Formally speaking, an adversarial example of a computer vision system f and a given input x is a vector p, which leads to a different output than x.

$$y = f(\mathbf{x}) \neq f(\mathbf{x} + \mathbf{p}) = \bar{y} \;\; \text{st.}\mathbf{x} \approx \mathbf{x} + \mathbf{p} \tag{2.4}$$

The key point is that $\mathbf{x}$ and $\mathbf{x}+\mathbf{p}$ should have a minimal distance in terms of visual similarity for a human observer, while still producing a different output. These attacks can be distinguished into two main categories, targeted and untargeted attacks. An untargeted attack tries to fool the system into producing any type of output, which is not the same then by just looking at $\mathbf{x}$. A targeted attack specifies a target output which should be produced by supplying $\mathbf{x} + \mathbf{p}$. The second type is usually considered a more complex problem. Another way of categorizing adversarial attacks is by having access to the model parameters. If the attacker has access to the weights of the target network architecture, the attack is considered a *white-box attack*. Conversely, when the attacker is allowed to feed input samples to the network without access to its internal weights, we normally talk about *black-box attacks*.

In this thesis, we are dealing with white-box attacks. Fooling neural networks started in the digital world but got recently extended to the real-world as well. In chapter 2.2.2 we are going to list a few example of the state-of-the-art in that field.

2.2.1. Digital

In [SZS$^+$13] the authors discussed for the first time the existence of adversarial examples in the domain of image classification. They formalized the task of discovering them as a constrained minimization problem, which they tackled using a box-constrained L-BFGS approach. The authors were able to create sample-specific noise to fool AlexNet [KSH12], which is invisible to a human observer as seen in Figure 2.2. The first image (left) is one of the sample images, the last (right) is the original image with the adversarial noise added to the image. The image in the middle displays the difference between them. After adding the noise, the bus gets classified as an ostrich. This attack changes every pixel value slightly and is able to find permutations, which are basically invisible to the human eye, achieving a change of output. The reader might ask how many pixels does one need to alter to change the outcome of an image classifier. In [SVS19] the authors succeed in fooling an image classifier by changing a single pixel. Notably, they achieved this result on a black-box attack and used differential evolution to find these perturbations. This attack is more compelling since it requires only the change of a single value. Additionally, it serves well as an indicator of how robust a neural work is. Both previous methods were looking into finding sample-specific perturbations, which is a great limitation since the input might be unknown to the attacker in advance. In [MKBF17] the authors tried to overcome this limitation by computing a universal noise pattern, which can be added to any input image and creates the required output. This pattern is computed in an iterative fashion using many training samples. Additionally, they move from the image classification domain to the image segmentation domain, which is commonly used by self-driving cars to assess their surroundings. The authors propose two different types of attacks, static or dynamic ones. In a static attack, the target output or the segmentation is predefined and will always produce the same output independently of the input image. For a dynamic attack, the goal is to suppress only some parts of the segmentation, for example pedestrians crossing a sidewalk resulting in a

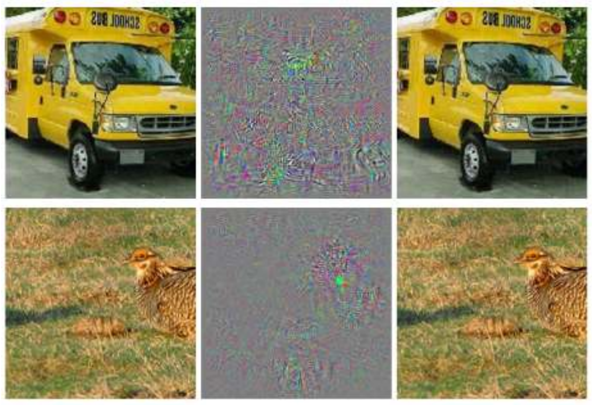


Figure 2.2.: *Adversarial examples generated using the L-BFGS approach. Left: input sample $\mathbf{x}$, middle: noise pattern $\mathbf{p}$ and right: resulting input. Courtesy of [SZS+13]*

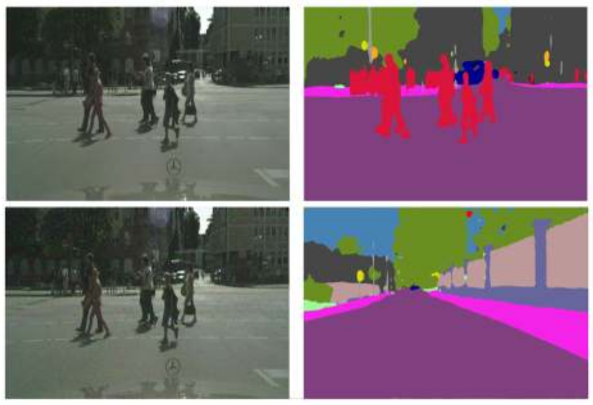


Figure 2.3.: *Universal dynamic noise example in the domain of image segmentation. Upper row normal input, bottom row input with universal noise applied. Courtesy of [MKBF17]*

much higher risk potential as seen in Figure 2.3.

This algorithm was adapted in various fields. In [SjSJ19] the authors used the same method to check the robustness of various human pose estimation (HPE) algorithms. While they were successful in fooling some HPE algorithms, they concluded that accomplishing this is a way harder problem then fooling image classification. Furthermore, they found out that altering the position of the faces is harder than the positions of the wrists or feet.
All previously mentioned methods are working in the digital world, where we assume that the attacker has access to the input image of the system, which already indicates a full compromise of the system.

2.2.2. Analog

In [EEF+17] the authors pointed out that DNN are already used in safety-critical environments such as self-driving cars. For that reason, these methods need to be resilient to adversarial examples in the physical world. They proposed an attack algorithm (RP_2) which is targeting the real-world scenario of road sign classification. Using their methods they can compute the location of stickers on the stop sign leading to a misclassification of it being a speed limit sign with a very high confidence rate. For the first time, the attacks move away from altering the digital input on a pixel level to modifying the reality to fool a machine learning algorithm. The clear benefit is that the attacker never needs to have access to the in- and outputs of the algorithm.
This approach got pushed further in [BMR+17] where the authors computed a universal "Adversarial Patch" to fool the VGG16 image classification network [SZ14]. This patch can be added into any scene with different illumination conditions and translations, which always leads to the class label "toaster". Notably, the paper shows that the patch is much more effective than adding a real-world toaster to the scene. In [TRG19] Simen Thys et al. use a similar approach, but with a different objective. Instead of always returning the same output class, they use a patch to suppress a class. In their set up they are targeting the object detector YOLOV2 [RF16].

This is considerably harder since the class of a human pose and body shapes contains a lot of intra-class variety in contrast to a stop sign. The authors demonstrate that they completely hide a human by holding the patch in front of their body. This would allow an intruder to hide from AI surveillance cameras.

Change of clothing prints

The previous works inspired research to compute patterns or prints for clothing to hide from an object detector. This task is harder than computing just a patch/poster, since the effects of ripples in the materials, as well as the changes of shading, lighting and general translations, need to be taken into account. In [XZL+19] the authors create a T-shirt fooling the same YOLOV2 architecture. To account for the appearing ripples in the fabric, they leveraged the thin-plate spline (TPS) to model the deformation of the non-rigid material. To compute the required coefficients for the TPS, the authors track points on a checkerboard printed on a T-shirt from frame to frame. This information combined with random noise and brightness changes is used to minimize the expectation over transformation to finally obtain a print for a T-shirt fooling an object detector. This technique is approximating the 3D properties, but is still operating in the 2D space.

2.2.3. Differentiable rendering

To overcome this limitation, a new research direction called differentiable rasterization or rendering has emerged, motivated by the fact that images are 2D projection of 3D geometry interacting with light. Giving machine learning algorithms the option to understand this process could help to close the gap between 2D vision and 3D scenes.
Many currently successful ML techniques are gradient-based, which requires computing gradients of pixels with respect to the scene parameters, like vertex positions or texture. The well-known conventional graphics pipeline is for most parts naturally differentiable. However, the step of rasterization requires a discrete sampling operation, which is not differentiable, blocking the gradient from flowing into the mesh vertices.

This issue was tackled by Kato et al. [KUH17] by separating the forward rendering and backward gradient computation pass, additionally circumventing the rasterization problem with a hand-crafted function. This achieved the property of differentiable rendering, however, the split led to an uncontrolled optimization behavior in some cases. Liu et al. [LLCL19] took a different approach to the problem and connected the forward and backward pass again. To overcome the rasterization issue, they treat this process as a probabilistic aggregation of triangles, which enables them to compute gradients even to occluded vertices. They demonstrate the quality of their approach by significantly improving the 3D unsupervised single-view reconstruction problem. However, this approach still approximates reality with a very simple model in terms of materials and lighting.

In [NDVZJ19] the authors make a big step forward by implementing a fully end-to-end physically-based differentiable render. In contrast to the previous methods, it does not try to make the rendering pipeline differentiable. Instead, it uses physically correct light transport simulation methods, which converge to the correct solution. By making these methods differentiable, machine learning techniques have the opportunity to incorporate them into existing deep learning pipelines. These systems can be then trained end-to-end to have a better understanding of 3D data.

2.2.4. Human pose estimation

This thesis uses for most parts a human pose estimation network, therefore, we will give an overview of the state-of-the-art techniques in multiple human pose estimation (HPE). Human pose estimation refers to estimating the joint locations of one or multiple humans on a single image or video. The output can be 2D or 3D. Pose estimation is used in many fields like activity recognition, gaming, or animation and became a critical component in many ML pipelines.
Traditionally there are two approaches to tackle this problem. First, the top-down strategy, which first runs a person detector and afterward estimates the pose for each detection [FXTL16] or [HGDG17]. This approach has the benefit of being able to directly apply the research of the single pose estimation to the multi-pose domain. However, coming with the downside of having correlated runtime and people on the image.

Second, the bottom-up strategy proposed by [PIT+15] targets the problem from a different angle. The general idea is to find all possible positions of a certain joint in the image and afterward pairwise connect them to get the human poses. To get the best result a spatial score for each possible connection is computed and the minimum value is chosen. This approach circumvents the direct human detection and is not depend on number people, but suffers of required integer optimization on a fully connected graph. Therefore this approach is not able to run in realtime.

3

Hand pose classification

In this chapter, we will explore the use of attribution techniques in the context of hand gesture classification. Over recent years this problem became more and more important since the trend is moving away from conventional human-computer interfaces like keyboards and touchscreens and towards communicating just with our plain hands. Many applications require to know the position of the hands.

In Augmented Reality or Virtual Reality interacting with the application using hand-gestures becomes more and more the state-of-the-art. The Microsoft HoloLens 2 [Mic20] drops all dedicated controllers and can be only controlled by hand gestures.
The car industry as well moves towards gesture control e.g. for volume or fan changes. BMW equipped all it's 850i models with a small set of gesture controllable commands [Bur20].

To obtain the gesture two main branches exist. The first one uses sensors, like the stretch-sensing soft glove [GWP$^+$19] with 44 stretch sensors. This glove is able to estimate the 3D pose in realtime, which can be inputted into a classifier. The main benefit of that approach is that it can deal out of the box with occlusions. However, a drawback is the natural requirement to wear a silicon glove.

The second approach uses raw video input and is usually camera-based. This is considered a challenging task due to the missing start and endpoint of the pose, as well as occlusions and self occlusions. Additionally, all general problems of using raw image data, like varying lighting conditions and complex background scenes, apply as well. These additional challenges urge the use of a learning-based approach using convolutional neural networks and the leverage of depth sensors, like [GLYT17]. In [SSPH18] the authors propose an objective function to compute a Cross-modal latent space where 2D keypoints can be obtained from only a single RGB image and used to compute a 3D hand configuration. Various other approaches have been proposed like the use of binocular vision [JZL$^+$19] or the use of a time of flight sensor [ZKH18].

However, in our case, we reduce the complexity by just looking into single gray-scale images and skip the pose estimation and directly predict the shown pose.

3.1. Leap Motion data challenge

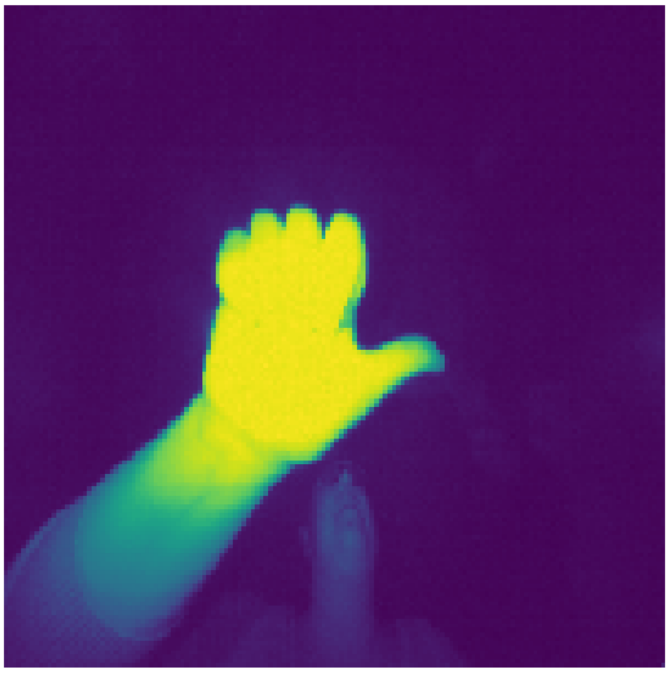


Figure 3.1.: *Example image representing the thumb class*

To compute the attribution of a model we need to train it first. Therefore we used the Hand Gesture Recognition Database acquired by Leap Motion [MDBJG16]. The dataset consists of 20000 infrared images of size 150x150 captured using the Leap Motion sensor. The database contains 10 different classes (hand gestures) performed by five men and five women. An example of the class thumb can be seen in Figure 3.1.
This dataset was posted a challenge on the platform Kaggle and is publicly available [GTI20].

3.1.1. Architecture and training

The model used for this multiclass classification task consists of four convolutional layers, each followed by a max-pooling layer. Followed by a 512 densely connected layer leading to the dense output layer of size 10 and with a softmax activation. Full details of the network architecture can be found in the section A.1.

To train the network the data is split into a training and a validation set with a nine to one ratio. The training is done until the validation accuracy reach above 99 percent, which is roughly after 5 epochs.

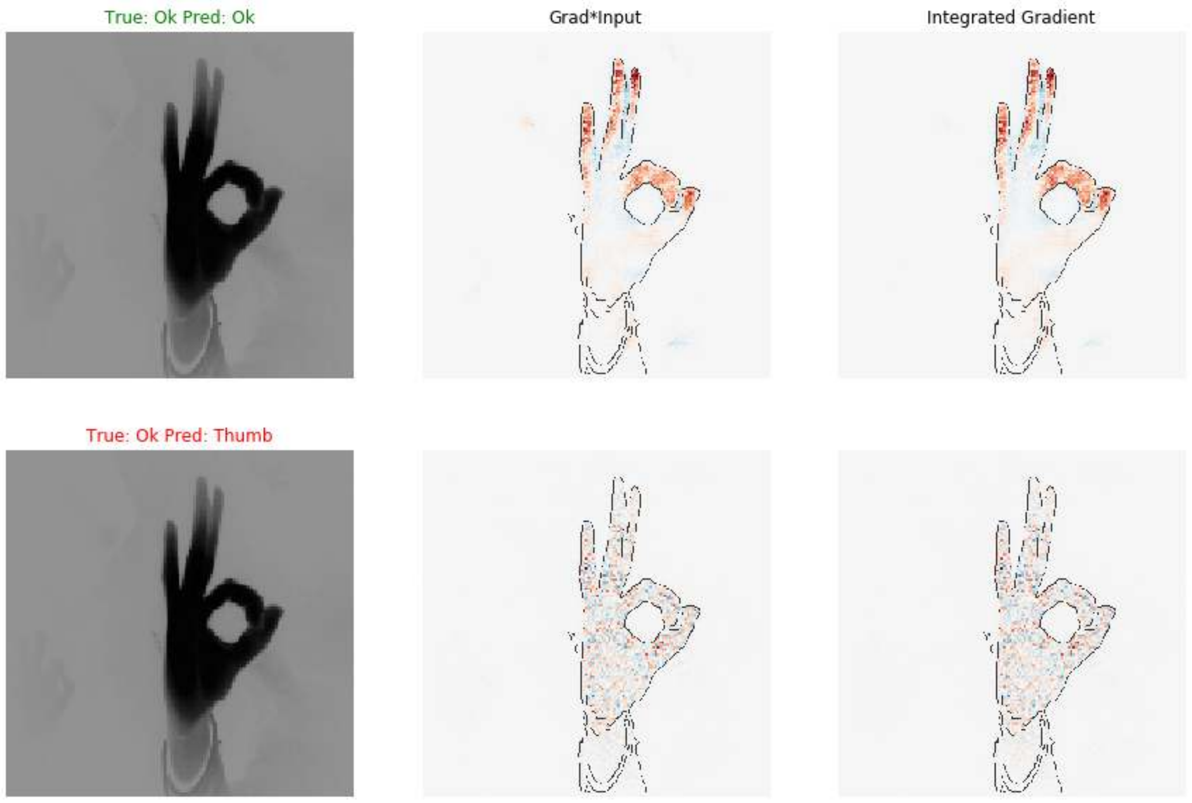


Figure 3.2.: *Attribution for OK class using gradient times input and integrated gradients. The first row shows the attribution of the fully trained network, the second one at the beginning of training.*

3.1.2. Results

To understand the important parts of the infrared image for a decision we are going to compute the attribution using gradient times input and integrated gradient methods. Finally, we examine what our neural network learned and which parts of the images are used to decide which gesture is shown. To demonstrate we use an image showing the OK gesture seen in the first column of Figure 3.2 and compute the attribution at the beginning of training and after the training is finished. In the bottom row, we can see the results of the untrained model and it is clearly visible that the network does not know what to patterns to look for. In the gradient times input column the red and blue heatmap (red positive and blue negative contribution) looks very random. The same pattern is seen for the Integrated Gradients.

These two methods are generally considered to produce very similar results as shown in [ACÖG17a]. However, in the top row, we can see that the attribution heatmap changed and became much more structured. We can see that the three fingers pointing up as well as the circle formed by the thumb and pointing finger contribute most to the decision that this image belongs to the OK class. In conclusion, we can say that the network has learned to correctly classify gestures and is looking at the right parts of the image.

4

Human pose estimation

After getting promising results in the field of classification, we want to move towards a more sophisticated problem with a state-of-the-art network. The problem of human pose estimation is fundamental in computer vision. The task is to derive a 2D or 3D pose of multiple humans from a single RGB image or a short video sequence. For the 2D methods the output consists usually of a keyframe drawn on the input image. The 3D methods take that a bit further and estimate the 3D position of each key point as seen in Figure 4.1.

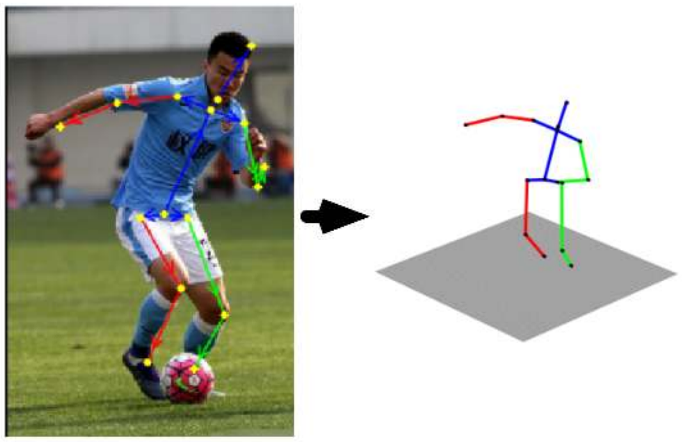


Figure 4.1.: *Example for 2D and 3D pose estimation. Courtesy of [SHJ+ 18]*

In this chapter, we will focus on 2D human pose detection, which is already a challenging problem by itself, due to humans appearing in a different scale, a picture containing an unknown number of individuals as well as occlusions and human-environment interactions. The goal of human pose estimation is to output a keyframe for each human on the input picture. The state-of-the-art keyframe is defined by COCO dataset [LMB+14] and consists of some facial key points as well as hands, elbows, knees, shoulders, feet, hips and spine, resulting in a total of 19 points (see Figure A.1).

In the following section, we are going to reveal the details of openPose [CHS+18], which overcomes the problem of integer optimization by introducing Part Affinity Fields and which we are going to use for further investigation.

4.1. Analysing OpenPose

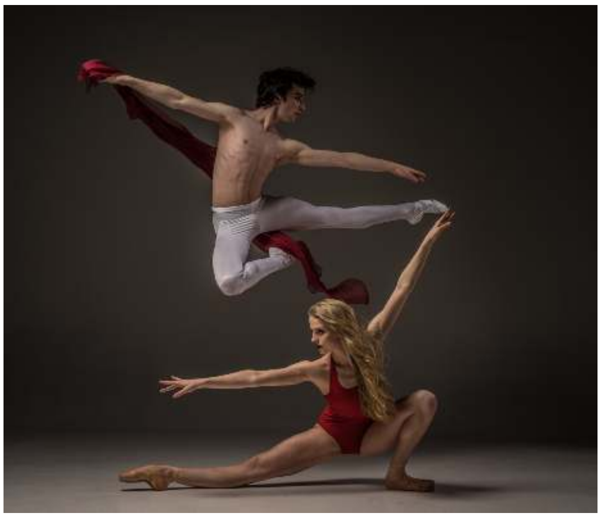


Figure 4.2.: *Input photograph*

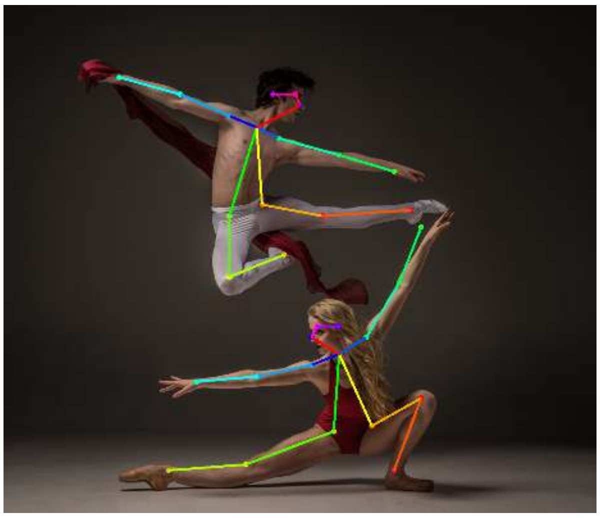


Figure 4.3.: *OpenPose final Output*

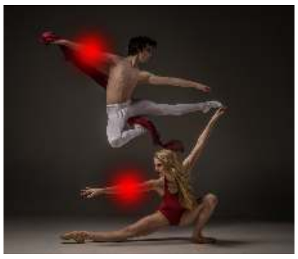


Figure 4.4.: *C-map right elbow*

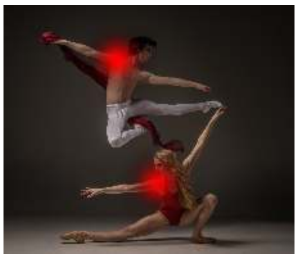


Figure 4.5.: *C-map right shoulder*

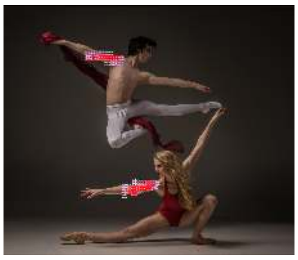


Figure 4.6.: *Part Affinity Fields*

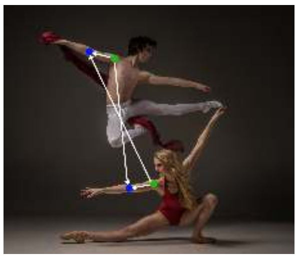


Figure 4.7.: *Bipartite Matching*

The openPose network uses the bottom-up approach and takes a single RGB image of size $w \times h$ as an input and computes the COCO key points for every person on the image. The main advantage over previously proposed methods is the use of Part Affinity Fields (PAF), which the authors proposed first in the version of their method [CSWS16]. A PAF is a 2D vector field describing the relationship between two key points (e.g. wrist and elbow) and preserving location and orientation information of the limb.

The method can be split into two parts and the overall pipeline can be seen in Figure 4.2. In the first part the algorithm takes a single image as input (4.2) for a convolutional neural network

(CNN). The network consists of feature extractor, which are the first ten layers of the VGG-19 [SZ14] network followed by a stage-wise prediction of PAF and Confidence maps (C-maps or heatmaps). Each stage gets the predictions of the previous stage as well as the image features from the feature extractor and optimizes these. There are seven stages in total. For more details about the network we refer the reader to the original openPose paper [CHS+18]. The output C-map of the network can be seen in Figure 4.4 and 4.5, as well as the returned PAF in Figure 4.6.

In the second part of the algorithm the perform non-maximum suppression to get a single peak location from each peak in the confidence map. This results in multiple candidates for a part (e.g. right hand). Every possible connection needs to be scored to pick the best match. Due to the NP-Hard nature of the K-dimensional graph matching, the authors propose a greedy relaxation, which uses line integral computation on the PAF for the score computation.

$$\text{Score} = \int_{u=0}^{u=1} L_c(p(u, \mathbf{d_{j1}}, \mathbf{d_{j2}})) \cdot \frac{\mathbf{d_{j2}} - \mathbf{d_{j1}}}{||\mathbf{d_{j2}} - \mathbf{d_{j1}}||_2} \mathrm{du} \tag{4.1}$$

where $\mathbf{d_{j2}}$ and $\mathbf{d_{j1}}$ are possible peak locations (e.g. elbow and shoulder) and L_c is the PAF for the upper arm. The function p interpolates between two peak positions.

$$p(u, \mathbf{x}, \mathbf{y}) = (1 - u)\mathbf{x} + u\mathbf{y} \tag{4.2}$$

This results in a graph, where the joints are the nodes and the limbs are the edges scored by (4.1). Finding a matching with the maximal score is equal to the maximal weighted bipartied graph matching problem. To solve this problem the authors use the Hungarian algorithm [KY55] with two relaxations. The first one is to only consider matching, which make sense in terms of human pose estimations. For example connection between the head and the feet are neglected. The second one is to look at each matching problem individually and not care about adjacent nodes. The authors show that with these simplifications can get a good approximation for a fraction of the cost.

The final result of the algorithm consists of a list of detected joints with relative location $(x, y) \in [0, 1] \times [0, 1]$ and the confidence score. If these are multiplied with the image width and height they can be drawn on the input image as shown in Figure 4.3.

The authors released the source code of openPose as well as the pre-trained models. However, the implementation is done in Caffe [JSD+14] and support in total 135 key points, which include the hands and some detailed facial keypoints. However, it is not compatible with the DeepExplain [ACÖG17b] framework. Therefore we are going to use the TensorFlow 1 implementation by Ildoo Kin [Kim18], which only outputs the 18 pose key points.

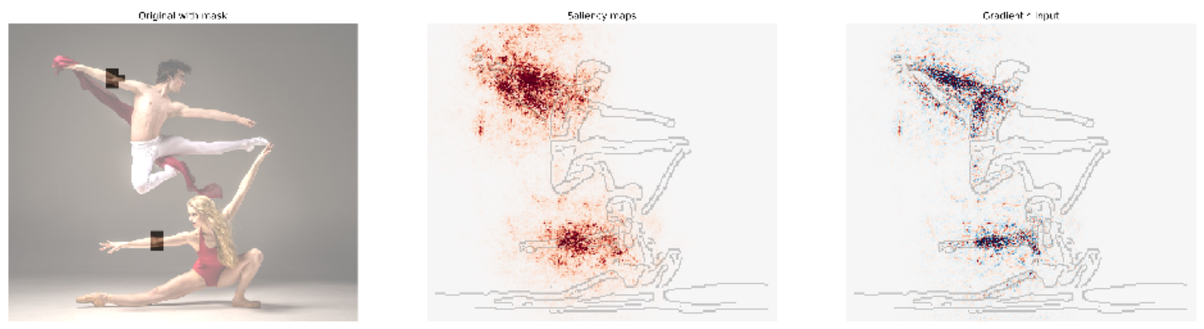


Figure 4.8.: *Attribution for right elbow with saliency maps and input times gradient. The background of the heatmaps is the detected edges of the original image.*

4.1.1. Attribution of openPose

In this section, we are going to tackle the computation of the attribution of openPose outputs. The main difference between the gesture classification and the pose estimation is that the output is more of a continuous nature vs a discrete class probability. Additionally, the second part of the algorithm, which outputs the final pose, is not differentiable due to the incorporation of non-differentiable functions, such as selecting the maximum. This means we are going to focus on computing the attribution of the output of the CNN.

The output of the CNN is of shape $46 \times 54 \times 57$ for each image with an input width of 432 and a height of 368. These particular image dimensions are used as in input, due to the pretraining was done on the same ones. The output tensor has a height of 46 pixels and a width of 54. The first 18 channels are the C-maps, channel 19 is all C-maps combined and the remaining 38 channels are x and y component of the PAFs.

We started with computing the attribution for single channels of the C-maps, in particular the right elbow. The question we answer is how much does each pixel contribute to the whole C-map. The reader should keep in mind that there is a positive and negative contribution, as well as contribution to small values in the C-map. However, the real question we are trying to answer is why did openPose place the elbow at that particular position. Since the algorithm uses non-maximum suppression to extract the position from a given C-map, we compute a mask around the peak and only attribute pixels inside that mask.

The mask is computed on the fly by calculating the 0.99 quantile of the C-map and thresholding afterwards. An example of a mask can be seen in Figure 4.8. To understand the use of global and local context we use saliency maps to display the attribution. Saliency maps were first introduced in [SVZ13]. The core idea is to compute importance by the magnitude of the gradient.

$$R_i^c(x) = \left\| \frac{\delta S_c(x)}{\delta x_i} \right\| \tag{4.3}$$

This method does not distort by multiply with the input value. In Figure 4.8 we first compute a mask for the C-map of the right elbow and afterward we use the input multiplied by the gradient and the saliency method to compute attribution. The saliency map reveals that the CNN used in the openPose algorithm is rather local and the most important pixels are all placed near the right elbow. However, some global context is used as well, since most of the arm and some of the background around contributed to the decision. The reader might have noticed that the gradient times input method does not put any attribution onto the background and it seems that only the arm is used for the decision. The feedback is misleading and results from the multiplication with input. In our case the background is nearly black, so the pixel values are close to zero. This reduces or cancels out the final attribution.

How are occlusions handled?

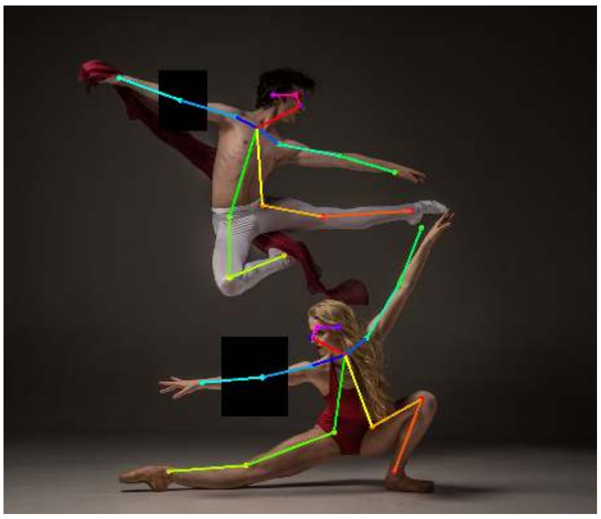


Figure 4.9.: *Key Pose detection with occlusions over the elbow*

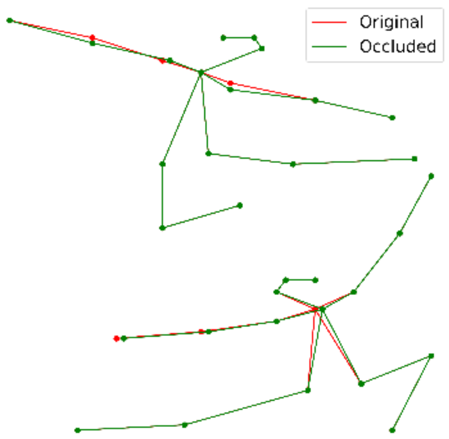


Figure 4.10.: *Poses before and after occlusion*

The openPose network is very powerful and can handle correctly a pose even with occluded joints, as demonstrated by the authors in Figure A.2. It is thought-provoking what type of information or which parts of the image the network uses to compute the correct position of the joint. We formulate the hypothesis that due to the lack of local context the most important parts for the placement will be the global context at the border of the occlusion. To test the hypothesis we added an artificial occlusion to our image. The occlusion is just a black rectangle. For the upper person we use a small rectangle and for the lower one a bigger version (4.9).
In Figure 4.10 we can see that the original pose and the pose after adding the occlusion are quite close together. OpenPose was able to place both elbows correctly even with a big part of the image masked out. Another unexpected fact is that due to the masking the position of other joints moved as well. This implies a correlation between neighboring joints as well as the second level neighbors.

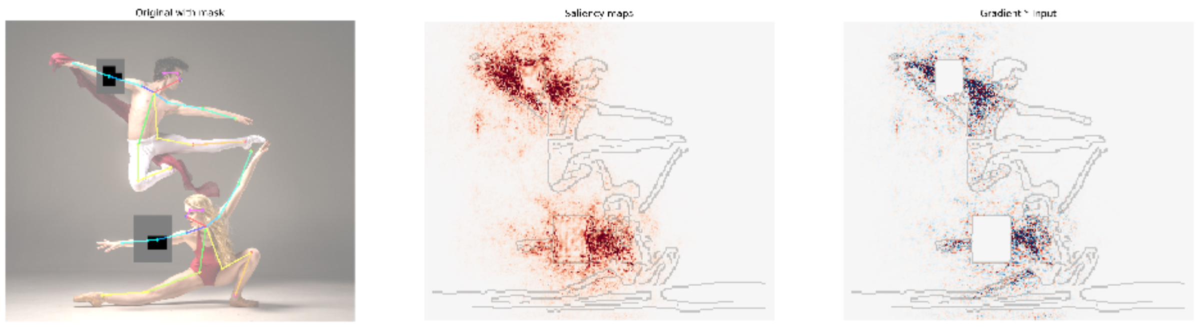


Figure 4.11.: *Attribution for right elbow with saliency maps and input times gradient with occluded input*

In Figure 4.11 we can see the resulting attribution. Both methods put the most value on the shoulder and wrist. This shows that our hypothesis is true and valid. For the lower person most attribution is put on the shoulder and very little on the wrist itself, which leads to the assumption that the network as learned roughly how long an upper arm is.

Resulting insights

In the previous sections, we looked into details of how openPose works and how it makes predictions. We saw that from a high-level perspective is consists of two parts. One neural network creating C-maps and PAF's and a greedy algorithm combining that output into human poses. For the predictions of certain joints we saw that some local, as well as some global, context was used. Additionally, we demonstrate that occlusions are handled very well. In that case, the global context is used to place the joint correctly. This observation led us to the idea that openPose might be vulnerable to an adversarial attack.

4.1.2. Attacking openPose

In this section, we explore different attack vectors for the openPose system. We started by looking at the details of the implementation, which is possible due to the open-source nature of the project. Afterward, we started with attacks, which compute adversarial noise for single images to achieve a required output. Moving to more sophisticated attacks such as changing only certain parts of the image. Finally, we computed a universally applicable noise pattern, which can output a given pose or output no pose at all.

Understanding the prediction internals

To understand how a final prediction is made we took a closer look at the algorithmic part of openPose. For this part, we used a realtime C++ implementation to improve the run time. The algorithm consists of 3 main parts:

1. Computing the C-Maps and PAF's with the CNN

2. Preprocessing the C-Maps into Peaks.
3. Using the Peaks and PAF's to compute human pose skeletons

1. Computing the C-Maps and PAF's with the CNN is done by resizing the image to the required width and height and passed into the CNN. There's no need in normalizing the image pixels, as this is done by the network.

2. Preprocessing the C-Maps into Peaks is done by smoothing the C-maps with a gaussian kernel with a filter size of 25 and a sigma of 3. Afterwards, the smoothed data gets passed into a max-pooling operation with a window size of 3×3. This output is compared to the input and where the matches the C-map value is returned. All steps can be seen in Figure 4.12.

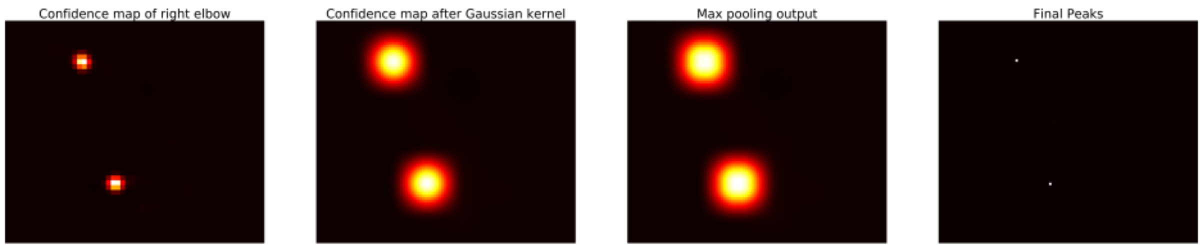
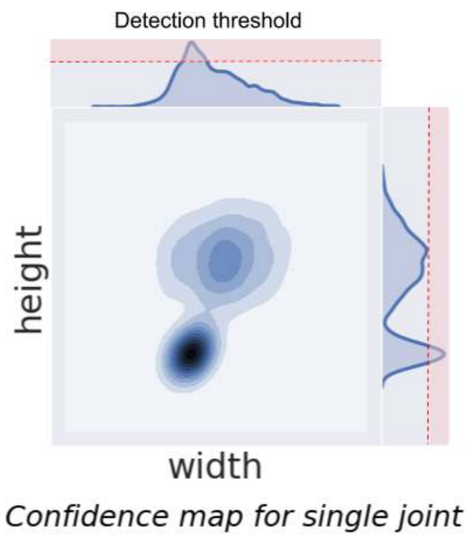


Figure 4.12.: *Pipeline for computing the peak locations. (left to right) raw confidence map of the right elbow, confidence map after smoothing, confidence map after the max pooling operation and final peak locations*

3. Using the Peaks and PAF's to compute human skeletons starts with parsing every pixel from every peak map to check if it is above 0.05. If the peak is below it is not consider confident enough to be further processed, all the others are valid candidates. In the next step for every limb, the algorithm checks if the start and the endpoint exist in the candidates. For example, if the limb is the lower right arm, the right wrist and right elbow need to be present. For every limb three conditions need to be satisfied.

1. The distance between the two peaks needs to be bigger than $1e - 12$, this is due to numerical reasons.
2. On the line between the two peaks there needs to exist a point at which dot product of the normalized directional vector from peak A to peak B and the PAF map of that limp is bigger than 0.05. In practice, this is done by discretizing the line into 100 steps and checking the dot product at each step. This process is visualized in Figure 4.14.
3. Finally, the mean dot product of the line samples and PAF plus $min(0.0, \frac{0.5*h1}{norm} - 1.0)$, which is called the limb score, needs to be bigger than zero. In this case, $h1$ is the height of the heatmap and the norm is the length of the line between the two peaks.



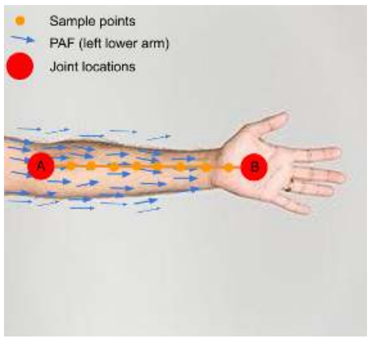


Figure 4.13.: *Confidence map detection condition threshold*

Figure 4.14.: *PAF joint conditions*

If limb satisfies all three properties is becomes a limb candidate. All limb candidates are sorted by their limb score and past to the greedy bipartite matching part. As we can see there are two main attack points. Either stop right at the start with enforcing all the value of the confidence map to be below 0.05 (shown in Figure 4.13) or make the PAF small enough that there is no value where the dot product is above 0.05.

Single image attacks

As a first attempt to alter the output of the openPose network to a target pose we started with a single image adversarial attack. Theoretically speaking, this means finding a noise pattern n which we add to the input x_i for the neural network $S(x)$ to produce a required output y. We started with the iterative gradient sign method target run for N times **(IGSM-T-N)**. This method was proposed by [KGB16] can be described by the following equation.

$$x_i^{n+1} = x_i^n - \epsilon \cdot \nabla(\mathcal{L}(S(x_i^n), y)) \tag{4.4}$$

where x_i^0 equals x_i and $\mathcal{L}$ is an application-specific loss function as well as the scaling factor ϵ. In our case, we added clipping from 0 to 255 after each iteration to stay a valid image value range and defined the loss function as the L2 norm between the given C-maps and PAF and the computed ones.

$$\mathcal{L}(S(x), y) = \sum_{i=0}^{57} \|S(x)_i - y_i\|_2^2 \tag{4.5}$$

The next question is how to specify the PAF and the C-maps for a wanted target pose. We found up that it is possible to artificially assemble them to match the requirements from the algorithm explained in 4.1.2. However, this requires a lot of manual work so we decided to use the output of the openPose network of a target image as a target state. This produces also a more "realistic" target state. Our goal for this experiment is to alter the output pose as much as possible.

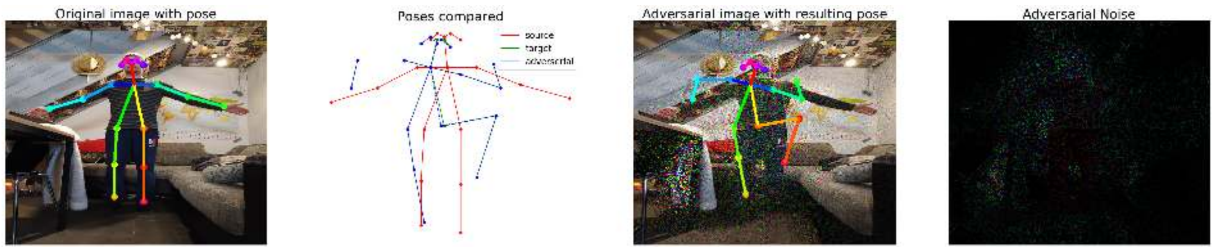


Figure 4.15.: *Overview of a single image adversarial attacks. From left to right: original image with detected pose, source target and adversarial poses compared, image with adversarial noise and resulting pose, adversarial noise*

In Figure 4.15 we can see the initial image and the target pose. After 100 iterations we completely match the target. The third image represents the input plus the noise and the final image is the computed noise. We conclude that single image attacks are possible. However, these changes are quite visible and this attack works only on that particular image, reducing the significance of that attack.

Single image attacks with restrictions

In the following paragraph, we restrict the problem. We aim to answer the question if it is possible to restrict the noise to a given area of an image and still achieve the target pose. In particular, we care about clothes. We manually create a binary mask $M \in \{0, 1\}$ and change the equation.

$$x_i^{n+1} = x_i^n - \epsilon \cdot M \odot \nabla(\mathcal{L}(S(x_i^n), y)) \tag{4.6}$$

where $\odot$ is an element-wise multiplication. An example of a cloth mask can be seen in Figure 4.16. We can see that the result is not a full match as before since the left knee score was not above the heat map threshold after 100 iterations. This results in not having a left leg. However, with the restriction, we are still able to match the target pose quite well. In the last image of Figure 4.16 we can see that the noise is only inside the mask.

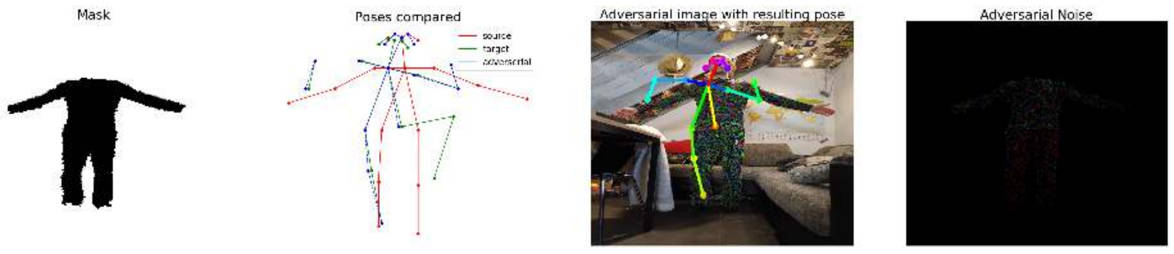


Figure 4.16.: *Overview of single image adversarial attacks with a clothing mask. From left to right: original image with detected pose, source target and adversarial poses compared, image with adversarial noise and resulting pose, adversarial noise*

This attack is more powerful than the previous one since it only requires to change the color of the clothing. Additionally, this change does not need to be invisible to the user, since cloth can have a wide variety of prints. However, this attack is still only valid for a single given input image.

Universal attacks

All the previous attacks were image specific. In this part, we overcome this problem by introducing universal noise. This method was proposed by [MKBF17] in the context of image segmentation. The core idea is to compute a single noise pattern u, which can be added to every image and still creates the required output. This pattern is computed iteratively with the following equation.

$$u^{n+1} = u^n - \delta \cdot \nabla(\mathcal{L}(S(x_i^n), y)) \tag{4.7}$$

To compute the noise pattern u for our case we used the first 4000 images of the COCO human validation dataset as a training set and the rest as a test set. With a single epoch and a batch-size of 16 we compute the pattern in Figure 4.17 to yield the pose of the image next to it. The universal noise is visually similar to the input image. However, it is in a representation which the openPose network learned to view images. In Figure 4.18 we can see an example of the universal noise applied to an image from the test set. In the final image, it is visible how the noise suppresses all other signals and generates the required output. More examples can be found in the appendix at A.5. The takeaway message from this section is that we were able to compute a universal pattern, which was able to fool the network and give us the required output.

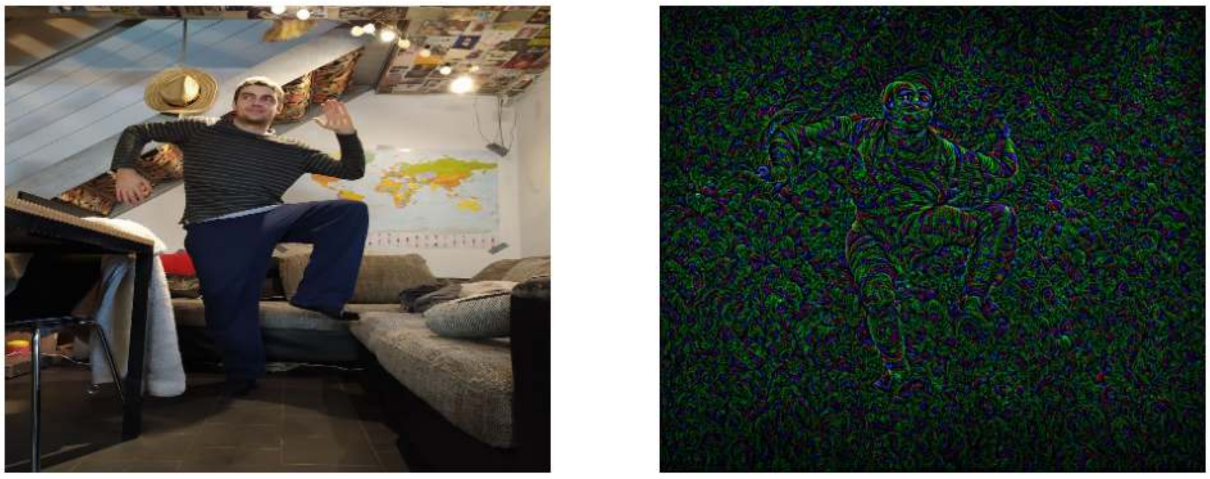


Figure 4.17.: *Target pose and resulting universal noise pattern*

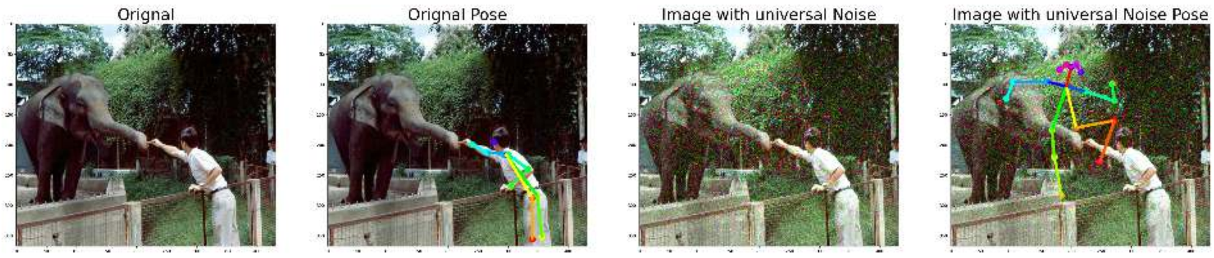


Figure 4.18.: *Left to right: Input image, detections on input image, image with universal noise and final detections*

Transform universal noise and attribution

Due to the success of the universal noise, we investigate how robust it is to transformations and understand which parts of the noise are important for the output. We examine translations, scale changes and rotations. Figure A.4 shows the result for translations, from which we conclude that the pattern is translationally invariant. To test to what degree is scale affecting the results, we scale up the image with a factor ranging from 0.5 to 2.0 and verify that the required pose is still produced. This particular noise pattern is resistant to scale changes ranging from 0.75 to 1.7. In terms of rotation, we rotate the noise pattern by 15 degrees and verify if the pose is still detected. In this case, we measure that the pose is still detected if the rotation is between 330 and 30 degrees.

In conclusion, we observed that the noise is robust to some transformations. We have also observed cases where larger transformations change the required output. The next property we inspect is the importance of the background. Therefore, we use the manually drawn mask to cut out the figure from the universal noise pattern and add it to various images. In Figure 4.19 an example of only the "pose" and the full noise added to an image. It is evident that only the pose figure is enough to get the required output. However, the pose only is not sufficient to suppress all the other signals, that is why the shoulder and head of the referee are still detected.

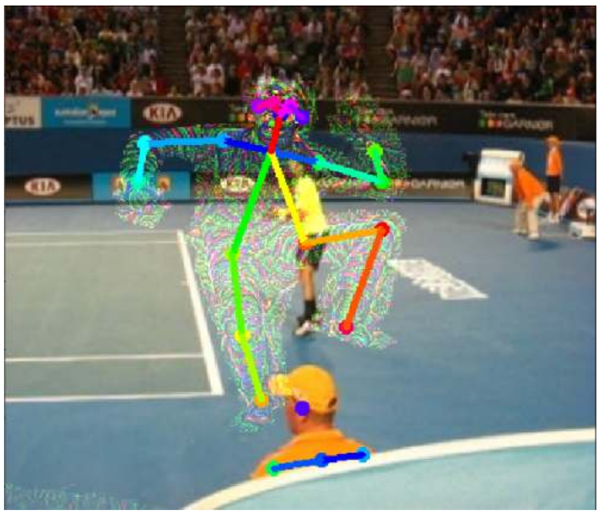


Figure 4.19.: *Universal noise pose after mask is applied*

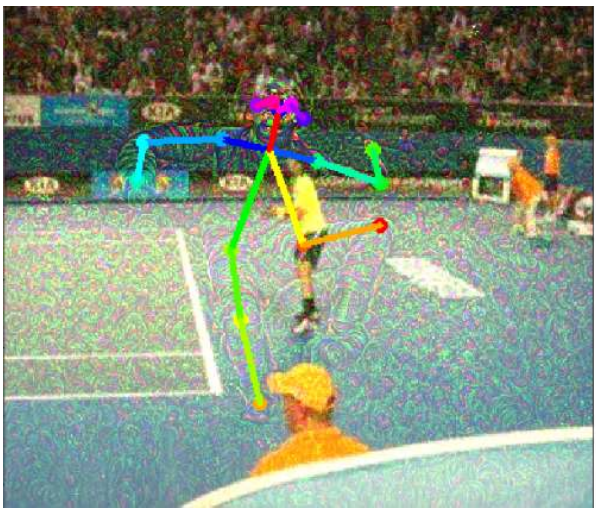


Figure 4.20.: *Universal noise result with full background*

Another aspect we analyse is how does the universal noise change the attribution computed on the test images. Our hypothesis is, that most attribution should be given to the pose in the noise and nearly nothing to the original joints. We tested this hypothesis by taking an image from the test set and displaying the C-maps for a certain joint (e.g. the right wrist). Afterwards, we compute a mask of the C-map to cover only the highest peaks. For the resulting mask, we compute the attribution leading to the high confidence values in the masked region. This gives us a notion of which parts of the image contribute to producing a peak at that particular location. In the next step, we repeat the same procedure, with the slight difference that we add the universal noise to the input. In Figure 4.21 we can see that our hypothesis was correct and all attribution is given to the noise pose wrist and the original input is completely ignored.

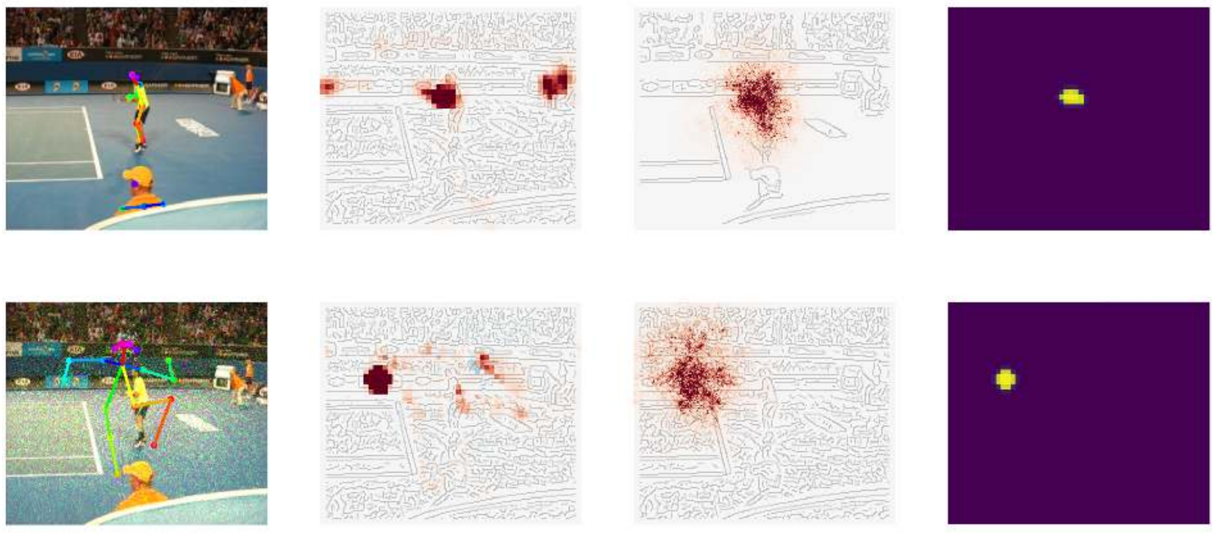


Figure 4.21.: *Columns from left to right. Input image, C-maps for right wrist, attribution (saliency map), mask of the C-map. The first row is using unchanged data and the bottom row is using data with added noise*

Adding universal noise to make a target invisible

Following the successful attempt to create universal noise, which can output a required target pose. We aim to create a universal noise pattern, which can make a target person invisible. This noise pattern should mislead the network such that no pose is being outputted. Therefore, we are required to redefine the objective function, since we do not have a target state anymore. As mentioned in 4.1.2 there are two ways to make a pose being undetectable. First forcing the confidence map below the detection threshold or pushing the average PAF value between two joints below the threshold. Therefore, we decided to use the following objective function, which penalizes the magnitude of all values.

$$\mathcal{L}(S(x), y) = \sum_{i=0}^{57} \|S(x)_i\|_2^2 \tag{4.8}$$

Using the same optimization procedure as in the previous section yields the following universal noise pattern 4.22. This pattern gives the impression of being a blended mix of joints, like knees or elbows. Adding this noise to an image from the test suppressed all signals and no pose gets detected. For more examples please refer to the appendix A.5.

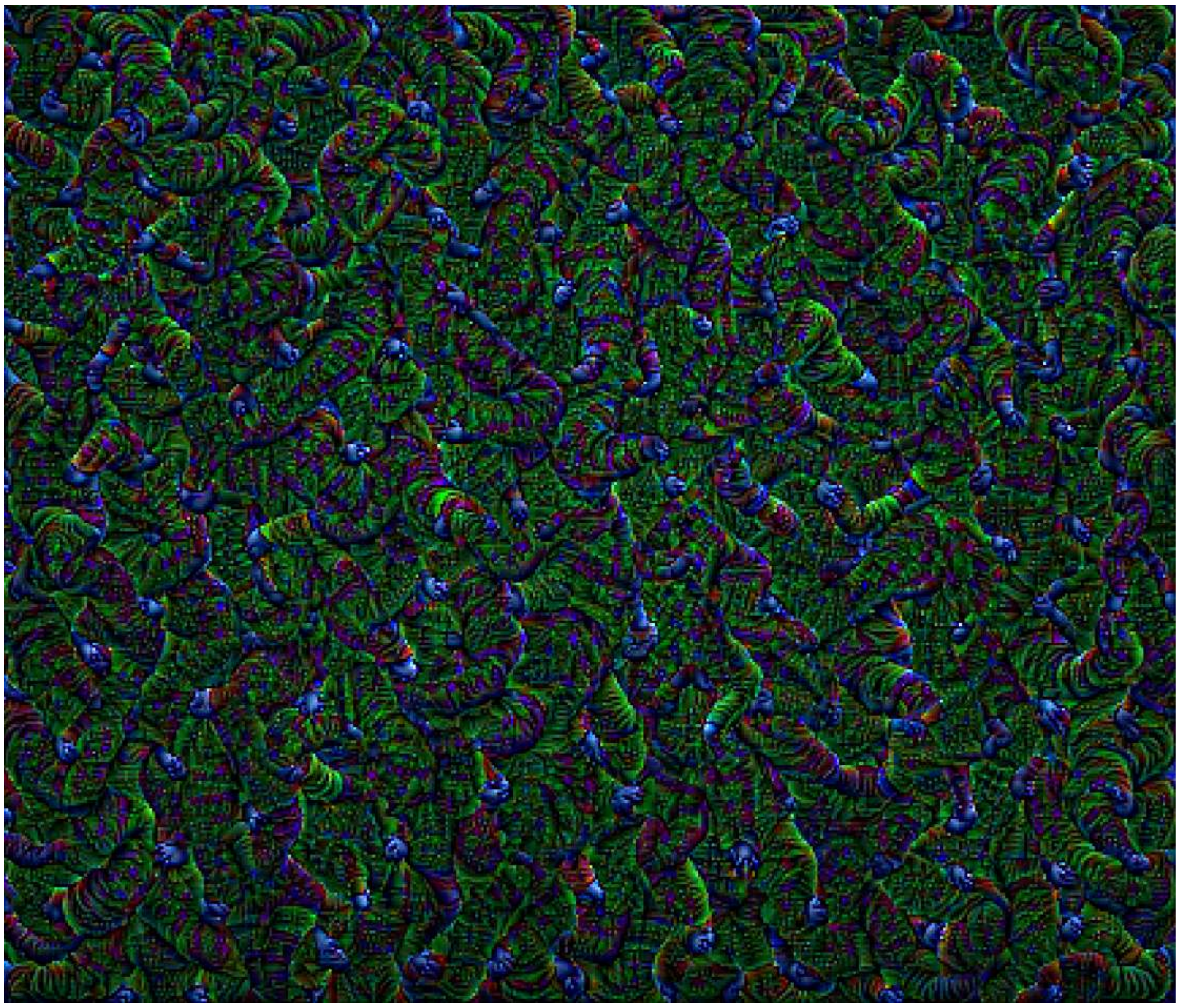


Figure 4.22.: *Resulting universal noise pattern to suppress detections*

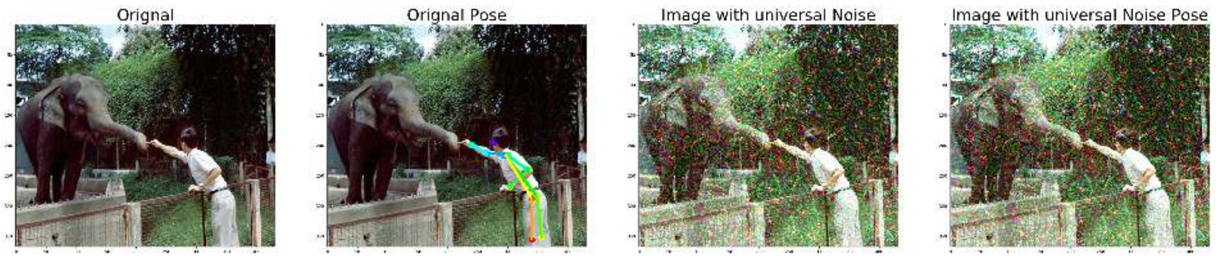


Figure 4.23.: *Left to right: Input image, detections on input image, image with universal noise and final detections*

4.2. Attacking openPose in the real world

Our previous adversarial model required that the attacker has access to the input of the openPose algorithm. In practice, this means usually that she has already compromised the system. A much higher threat would be if just by changing the appearance of real-world objects in the physical world a given output could be achieved. For example, creating a poster or a T-shirt pattern that

Name	#Poses	Scaling (z-axis)	Translation (x-axis)	Rotations (y-axis)
all	10	0 to -300	-100 to 100	300 to 60
same distance	10	0	-100 to 100	300 to 60
same rotation	10	0 to -300	-100 to 100	0
same distance and rotation	10	0	-100 to 100	0

Table 4.1.: *Different parameters for the dataset creation*

would be sufficient to fool the openPose system. In the following sections, we are going to investigate digital ways of optimizing for such a poster texture or T-shirt print.

4.2.1. Creating a dataset for a poster

To train the noise pattern of a poster, we created a method to quickly scaffold a dataset from a given parameter vector. This dataset contains ten different poses. Each image consists of a random pose placed at a random position with a random rotation. All these values are picked from a given range. This way it is easy to create datasets with only rotations or only scaling (moving only in the z-axis). Additionally, to placing the Figure we add a rectangle in front of the chest of the pose. This rectangle should represent the poster. The total pipeline consists of five steps

1. Create a scene XML file for the Mitsuba render [NDVZJ19] with the random transform.
2. Use the render to create a sample image
3. Use transforms from the XML file to create a mask of the poster
4. Use the transforms to compute the 2D warp coefficients

In the next section, we are going to explain the details of every step.

Creating the scene

Since we are going to render our image using Mitsuba we need to create a scene XML file, this file defines the whole scene, including light, objects and textures. An example can be found in the appendix A.6. In the following table, we show the different parameters used to create four different datasets. Each dataset has a different challenge and helps us identify the difficult attributes. Every dataset contains 2000 images and a test/train split of 9:1.

Render image

Every scene file is rendered with Mitsuba with 64 samples per pixel, resulting an openEXR file. This file is tonemapped using the linux utility tool image magick [The]. In Figure 4.24 you can see 4 examples with 4 different poses. This process takes roughly 2 hours to finish.

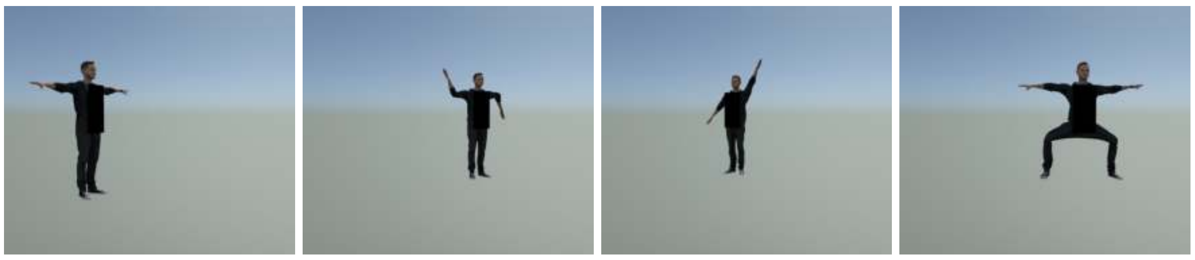


Figure 4.24.: *Examples from the generated dataset*

Transforms to compute mask

In this step, we just use the rectangle vertices and openGL to render 2 white triangles on a black image. The pixel position of the vertices are calculated using the openGL pipeline. This includes transforming the Mitsuba scene parameters into model, view and projection matrices and passing them to openGL. The result can be seen in Figure 4.25.

Compute 2D warp coefficents

Additionally to the mask, we need to compute the perspective transform M from the texture onto the poster. This perspective transform operates in 2D and warps one image onto another one. The transformation is calculated using 4 source points $(p_i = (px_i, py_i), i \in \{0, 1, 2, 3\})$ and 4 target points $(t_i = (u_i, v_i), i \in 0, 1, 2, 3)$. In our case, the four source points are the corners of the texture so $(p_0 = (0, 0), p_1 = (w, 0), p_2 = (w, h), p_3 = (0, h))$ and the target points are the corners of the poster in the rendering. All coordinates are in pixel space. The target point on the image are computed in the following way:
As a first step, we read out the current transformation parameters (Translation (x, y, z) and rotation angle α) out of the scene file. From these parameters we create the model matrix M.

$$M = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.9}$$

The view matrix V is create from the camera parameters, these include the up $v_u = (x_u, y_u, z_u)$, target $v_t = (x_t, y_t, z_t)$ and origin $v_v = (x_v, y_v, z_v)$ vectors. First, compute the normalized forward vector from the target position v_t to the origin position.

$$f = \frac{v_v - v_t}{||v_v - v_t||} \tag{4.10}$$

In the next step, we recalculate the normalized left vector using the up vector and re-calculated the normalized up vector u.

$$l = \frac{v_u \times f}{||v_u \times f||} \tag{4.11}$$

$$u = f \times u \tag{4.12}$$

The final view matrix is defined as follows.

$$V = \begin{bmatrix} l_x & l_y & l_z & 0 \\ u_x & u_y & u_z & 0 \\ f_x & f_y & f_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & x_v \\ 0 & 1 & 0 & y_v \\ 0 & 0 & 1 & z_v \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

The last matrix we need is the perspective projection matrix

$$P = \begin{bmatrix} \frac{\cot(\frac{fov}{2})}{aspectRatio} & 0 & 0 & 0 \\ 0 & \cot(\frac{fov}{2}) & 0 & 0 \\ 0 & 0 & \frac{f}{f-n} & 1 \\ 0 & 0 & -\frac{f \times n}{f-n} & 0 \end{bmatrix} \tag{4.14}$$

where $aspectRatio$ is the aspect ratio of the renderings width and height, fov is the field of view in radians, n is the near and f is the far clipping plane. This parameters do not vary and are fixed for every scene.

Now we project all homogeneous vertices of the rectangle into clip space.

$$\text{vertex}_{clipSpace} = MVP \cdot v \ \ \forall v \in \{(1,1,0,1),(1,-1,0,1),(-1,1,0,1),(-1,-1,0,1)\} \tag{4.15}$$

The final screen coordinates t_i are obtained, by applying a perspective division by the last coordinate and stretching the output to the required screen width and height.

To get the final warping transform $T \in \mathcal{R}^{3\times3}$ we solve the following linear system.

$$\begin{bmatrix} px_0 & py_0 & 1 & 0 & 0 & 0 & -px_0 \cdot u_0 & -py_0 \cdot u_0 \\ px_1 & py_1 & 1 & 0 & 0 & 0 & -px_1 \cdot u_1 & -py_1 \cdot u_1 \\ px_2 & py_2 & 1 & 0 & 0 & 0 & -px_2 \cdot u_2 & -py_2 \cdot u_2 \\ px_3 & py_3 & 1 & 0 & 0 & 0 & -px_3 \cdot u_3 & -py_3 \cdot u_3 \\ 0 & 0 & 0 & px_0 & py_0 & 1 & -px_0 \cdot v_0 & -py_0 \cdot v_0 \\ 0 & 0 & 0 & px_1 & py_1 & 1 & -px_1 \cdot v_1 & -py_1 \cdot v_1 \\ 0 & 0 & 0 & px_2 & py_2 & 1 & -px_2 \cdot v_2 & -py_2 \cdot v_2 \\ 0 & 0 & 0 & px_3 & py_3 & 1 & -px_3 \cdot v_3 & -py_3 \cdot v_3 \end{bmatrix} \cdot \begin{bmatrix} t_{0,0} \\ t_{0,1} \\ t_{0,2} \\ t_{1,0} \\ t_{1,1} \\ t_{1,2} \\ t_{2,0} \\ t_{2,1} \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \tag{4.16}$$

The matrix coefficient $t_{2,2}$ is set to one. All nine coefficient are saved to a npy file for later use. The transform T can now be used to warp texture to the poster. In Figure 4.25 the red points show the resulting corner points of the poster in image space.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{4.17}$$

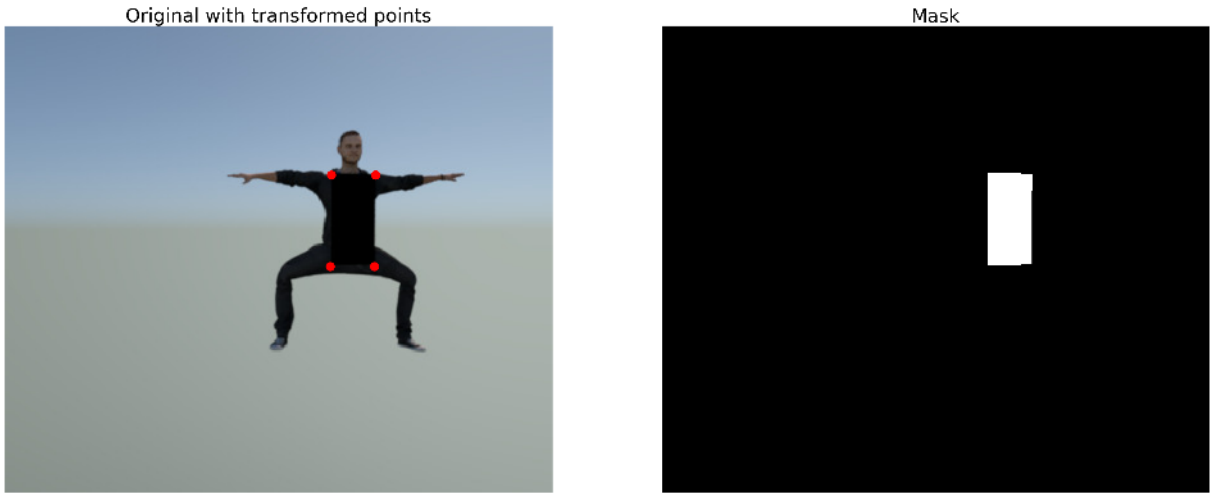


Figure 4.25.: *Left: Original image with transformed corner points. Right: Resulting mask from openGL pipeline*

4.2.2. Universal attacks with 2D warping transforms

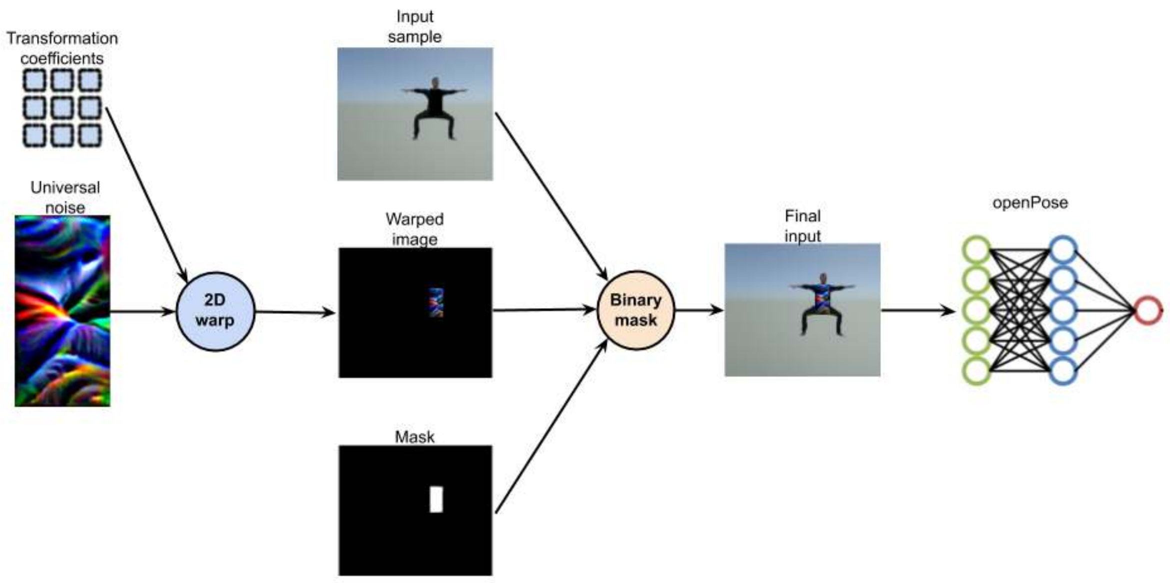


Figure 4.26.: *Overview of new pipeline for poster texture optimization*

Using this dataset we are able to create a new pipeline and directly optimize for the poster texture. Instead of using an image as an input we extend the computational graph from the top and leverage the Tensorflow function *tf.contrib.image.transform*. This function uses the 2D warp transform coefficients we computed in the previous section and transforms the poster onto the right position in the rendering. The mask is used to combine these two images. An important factor is the interpolation method used in the Tensorflow transform function. We will discuss this further in the next chapter. The main reason for this approach is that we are now able to directly optimize for the poster texture. See Figure 4.26 for an overview of the new pipeline.

To be able to compare and rank the different attack vectors we developed a metric. The metric is defined as the fraction between the number of joints being detected after the adversarial noise is applied divided by the number of detected joints in the original image. Resulting in a number ranging zero to roughly one, were zero means everything was suppressed and one means nothing was suppressed, saying that the noise was ineffective. We want to mention two points about this metric.

- The metric does not account for movements of joints if a joint gets detected it contributes to the metric, it does not matter in which location.
- In rare cases it can happen that the resulting metric is above one. This happens, when more joints get detected when the noise is added.

Analysis of the poster attack

For a better understanding of the strength and weaknesses of our poster, we trained the texture on the datasets mentioned in table 4.1. The goal is to find out which changes (translation, rotation or scale) are hard to tackle by the noise pattern. In Figure 4.27 we can see an untampered input sample with the detected pose onto of it. In the next image, we can see the detected peaks as well as their confidence values, which are all roughly around 0.1. The next image showcases the effect of the poster, which is able to suppress the signal. The final image checks which of the detected peaks is above the detection threshold of 0.05. We can see that for some peaks the confidence was below the threshold and therefore the detection failed. However, the knee of feet peaks got still detected, resulting in the PAF thresholds being the reason for failure.

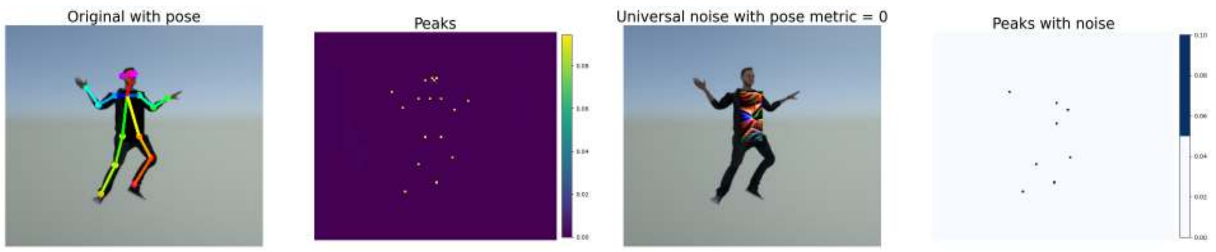


Figure 4.27.: *From left to right sample image with detected pose, detected peaks, image with poster applied and resulting peaks*

Additionally, we wanted to find out the impact of the interpolation function attribute in the Tensorflow image transform function. Therefore, we made two runs on each dataset, ones with the texture interpolation set to nearest neighbors and the other time with bilinear interpolation.

Finally, we wanted to see the impact of the PAF and the C-maps in the loss function. Consequently, we tested for each run optimizing for both or just optimizing for the PAF only. This setup resulted in a total of 16 runs.

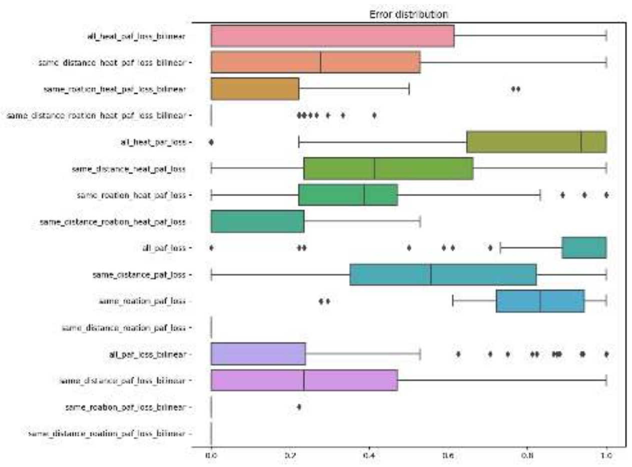


Figure 4.28.: *Metric distribution on different datasets and different loss functions*

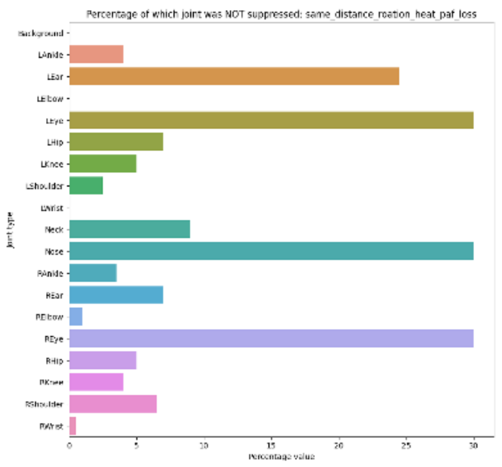


Figure 4.29.: *Which joint is hard to suppress with a poster*

In Figure 4.28 we compare the metric for the different setups and from these graphs we can directly see the following properties.

1. Posters using the bilinear interpolation method have generally a lower score, meaning they are performing better. This could be explained due to the fact that resulting noise patterns are smoother and slight changes in pose result in slight changes in the texture.
2. Optimizing for the PAF only tends to create better results, we assume that suppressing the peaks resulting from the heatmaps is a too challenging task for the noise, so most of the time their confidence value just gets lower, but not low enough to make the detection fail.
3. We can see that translation is not a problem in any setting. However, in both setup rotation is more challenging and than scaling but in the case of using linear interpolation and using only the PAF as a loss function argument the scaling becomes more difficult.
4. We want to point out that for four setups it was possible to find a poster texture, which completely suppressed every joint for every test sample. It was most of the time achieved on the simplest dataset containing only translations. However, in the setup called same_rotation_paf_loss_bilinear the poster became also invariant to scaling.
5. In Figure 4.29 we looked into details of the same_distance_rotation_heat_paf_loss setup and inspected the composition of the metric. In the graph, we can see four distinct peaks, which are all facial features. This means, suppressing the face is the hardest challenge for the poster and wrists are the easiest. We picked this setup as an example, however, this is true for all.

In Figure 4.30 we can see two resulting poster textures for two different setups. It is clearly visible that the first one using the bilinear interpolation method is way smoother than the other using the nearest neighbor interpolation. Fascinating as well is the content of these two images, where the first one looks like a person from behind and the second one reminds of two crossed legs.

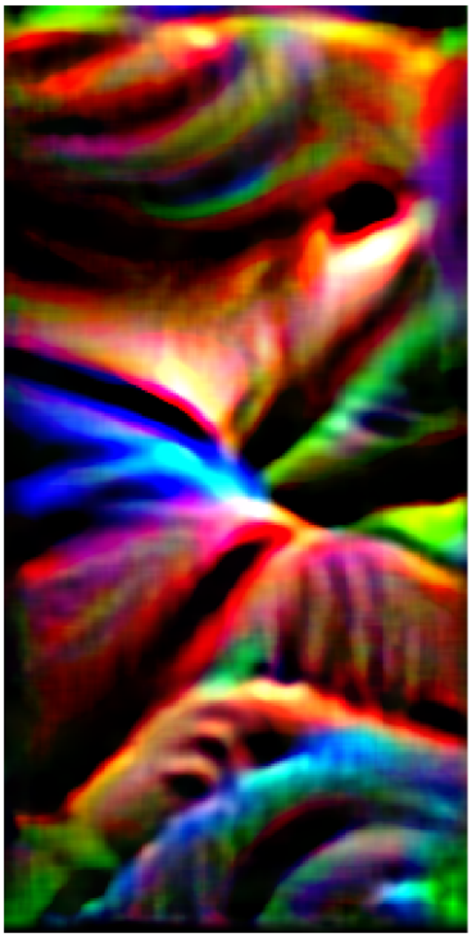


Figure 4.30.: *Resulting texture for the setup with using all transforms, the combined loss (C-maps and PAF) and the bilinear interpolation*

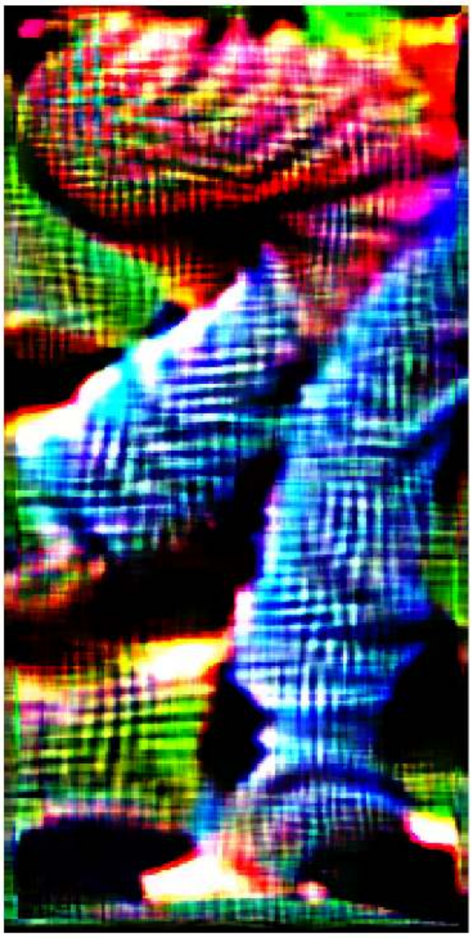


Figure 4.31.: *Resulting texture for the setup with using all transforms, only the PAF loss and the nearest neighbor interpolation*

5 Three-Dimensional Attribution

This chapter is about advanced three-dimensional attribution and robustness testing. By leveraging a differentiable renderer we chain openPose with a renderer and compute attribution down to the geometry level. Using this setup we conduct various experiments to gain insights about openPose. Additionally, we use adversarial methods to test the robustness of the network.

5.1. Open Pose T-shirt texture

Due to the successful attack on openPose using a poster with an adversarial print, we explore how well this attack can be applied using a T-Shirt as a noise canvas. The main difference is that a poster has only affine transforms and can be very well approximated in 2D during training, as done in section 4.2.1. A T-Shirt produces wrinkles and ripples which are very complex and simulating these is a research field on its own. Previous methods [XZL+19] proposed to approximate those using a thin-plate spline model, a technique relying on splines for 2D space deformation, but due to the missing dimensionality, they are not able to capture the full behavior of non-rigid material deformation.

We aim to tackle this issue from a new angle and directly optimize for the texture of the T-Shirt.

5.1.1. Experiment setup

To directly optimize for the T-Shirt texture including the ripple effect we need a special 3D dataset containing humans in different poses wearing the same simulated T-Shirt. In [DDO+17] the authors construct such a dataset in five steps.

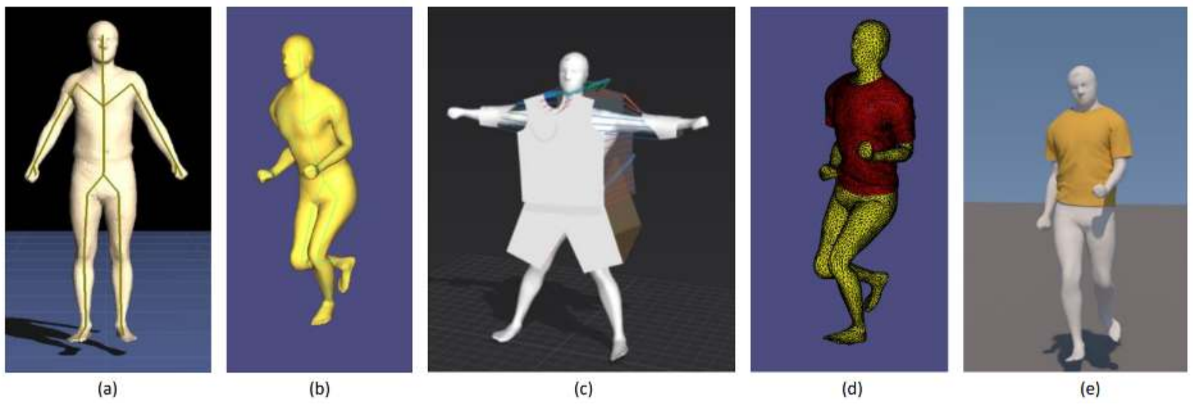


Figure 5.1.: *Data generation pipeline. A human model is automatically rigged (a) and animated (b). Then a desired garment is designed (c), simulated under some motion (d), and realistically rendered (e). Courtesy of [DDO+17]*

1. Ten human meshes are created using a data-driven method [ASK+05]
2. Every human mesh is automatically rigged using [BP07] to compute the required weights.
3. The rig is used in combination with data from the CMU MoCap database [Lab] to create realistic motions.
4. A manually designed T-shirt is fitted to the mesh and physically correct simulated using ARCSim [NSO12, NPO13] keeping the topology of the mesh.
5. Finally, the scene is rendered using Mitsuba.

The biggest benefit of this dataset is that, while each T-shirt has unique vertex positions, the topology is the same for all examples. That allows us to compute a UV map and directly optimize for a texture. In Figure 5.1 we can see the single steps of the pipeline.

To combine this dataset with openPose we use the differentiable renderer named pytorch3D [RRN+20]. This render allows us to specify a perspective camera, as well as lightning and geometry information and computes an image represented as a tensor. This tensor can be now passed on to openPose. Since the renderer is written in Pytorch we rely on the Pytorch implementation of openPose [Hzz20] using the same weights as the original version.
The render splits the rendering pass into two separable components, shading and rasterization. For the shading, we use the HardPhongShader, a technique using surface normals of the triangle to interpolate the final color. The color is a sum of an ambient, diffuse and specular components of the material. This method was proposed in 1973 by Bui Tuong Phong [Pho75].
The rasterization is done in a probabilistic way as proposed in [LLCL19]. The core idea is to give every triangle a soft contribution to the final pixel value, making this process differentiable.
To get a more realistic model we introduce a random background sampled of a training background set of 20 images. All images are indoor scenes and represent typical CCTV scenes. To combine the background with rendering we leverage we SoftSilhoutteShader, a shader which segments the person from the background. Using this segmentation we can combine rendering and background.
The final optimization pipeline can be described in the following steps and an overview is shown in Figure 5.2.

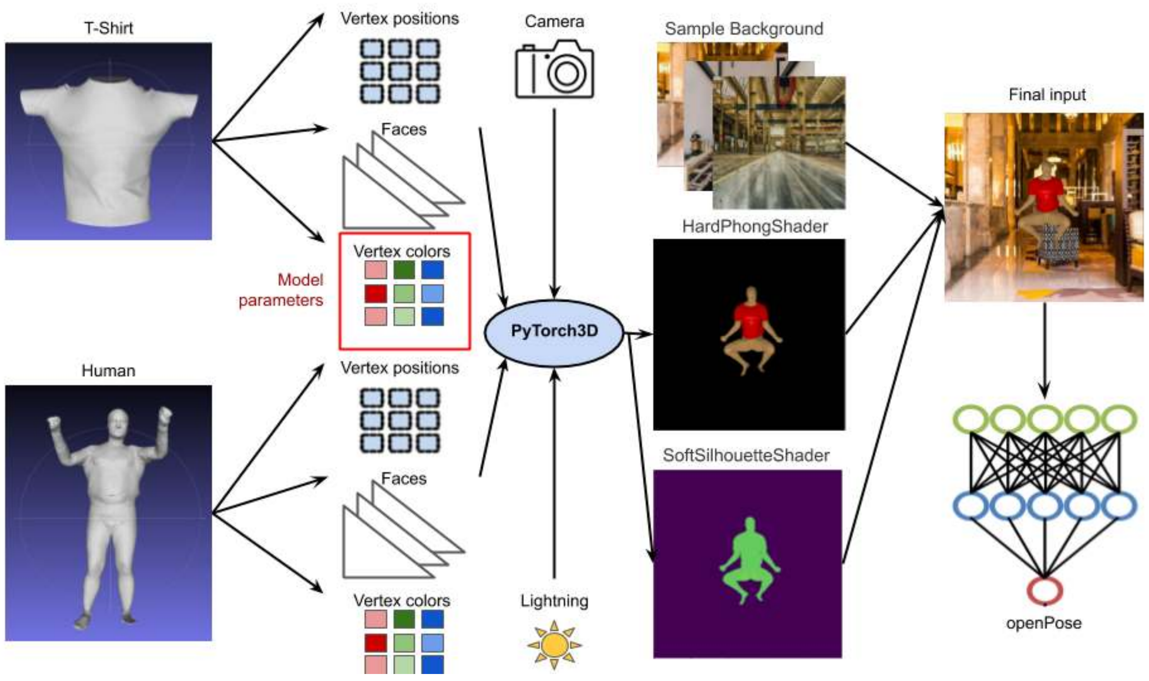


Figure 5.2.: *Model defintion and and computational graph*

1. The simulated T-shirt vertices, faces and texture are combined with the ones from the character. This step is needed, due to the limitation of the pytorch3D render only allowing to render a single mesh. The T-Shirt texture is set as the model parameter.
2. The set up the camera to render an image of size 512×512 with the position defined in the dataset.
3. We define a directional light coming from the camera towards the mesh.
4. The mesh is rendered using the HardPhongShader
5. The mesh is rendered a second time using the SoftSilhoutteShader
6. A background image is sampled and the final input for openPose is created using the rendering and the silhouette
7. OpenPose creates 19 confidence maps as well as 38 part affinity fields, which are combined into a loss function (more details in 5.1.1).

The loss function is optimized using the ADAM optimizer [KB14] with the standard parameters (betas=(0.9, 0.999)) and a learning rate of 0.05. As a training set, we sample 2000 meshes from the dataset from [DDO$^+$17]. To verify results we use a test/train split with a ratio of 1:9 and run the optimization for three epochs.

Loss functions

Formulating a loss to suppress detection is not an easy task. Therefore, we define three different loss functions and compare their results.

The first loss function is the L2 norm of the confidence maps and the PAF as described in equation 4.5. This loss is only penalizing the magnitude of the openPose output.

In the next loss function, we introduce a regularizer on the vertex colors. The motivation behind this approach is to create a smoother texture, which could be less influenced by small camera changes. The regularizer is defined as the "color laplacian" on the vertex colors. More formally defined as:

$$CL_i = \sum_{j=0}^{m} w_{ij}(v_i - v_j) \;\; st.\, j \in S \tag{5.1}$$

where v_* is defined as the RGB value of the vertex and $w_{ij} = \frac{1}{|S_i|}$ with S being the set of neighbors of the vertex i. We compute the color laplacian for every vertex and use the mean value as a regularization term. This term is added to the loss function with a regularization coefficient of 0.01.

Finally, we use a more probabilistic approach by leveraging the Kullback-Leibler (KL) divergence on the confidence maps. The KL divergence is used to measure the difference between probability distributions P and Q and is defined as:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) \log \left(\frac{P(x)}{Q(x)} \right) dx \tag{5.2}$$

To be able to apply this function we divide each of the 19 joint channels by the sum of their values turning them into a probability distribution and use a uniform distribution as target distribution. Since the confidence map is discrete the reformulated the KL divergence as a sum.

$$D_{KL}(P||Q) = \sum_{w=0}^{width} \sum_{h=0}^{height} P(h,w) \log \left(\frac{P(h,w)}{Q(h,w)} \right) dx \tag{5.3}$$

where $P(h, w)$ represents the entry in the confidence map. For the PAF we are still using the L2 norm since it is not possible to turn this into a probability distribution.

5.1.2. Results and Conclusion

To be able to better compare the strength and weaknesses of different methods, we introduce two new metrics additionally to the basic metric defined in 4.2.2. The first one should measure the ability to totally suppress the signal and is defined as a binary metric, where it returns 0 if no joint is detected and 1 otherwise.
The second one is related to the metric defined in 4.2.2, but it takes the distance into account. This means in detail, that a joint is only considered correctly detected if it is placed within a radius r of the correct position. Our test dataset does not provide ground truth pose annotations, we assume that the output of openPose in the original rendering is close enough to the ground

truth. Therefore, we use these poses to compute the metric. The fact that all our meshes are roughly at the same distance makes it possible to approximate the radius with a constant.

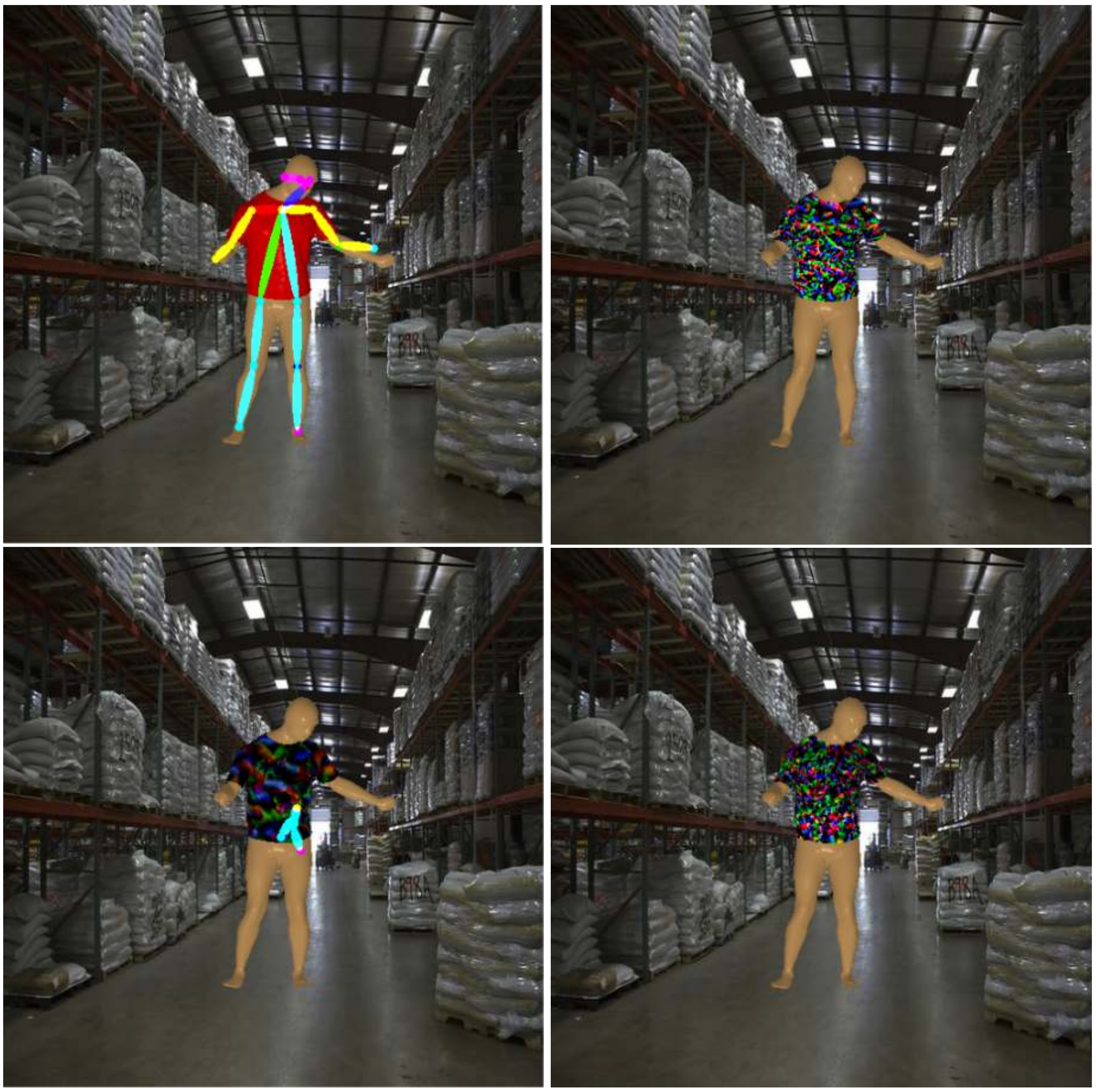


Figure 5.3.: *First row (Left to right): Orignal image, result of L2 loss. Second row (Left to right): L2 loss with color laplacian regularizer and KL divergence loss*

In Figure 5.3 we can see the results of the different approaches. The texture for the L2 norm and KL divergence look quite similar while the variant with the color laplacian creates a smoother version. However, it is not able to fully suppress the signal.

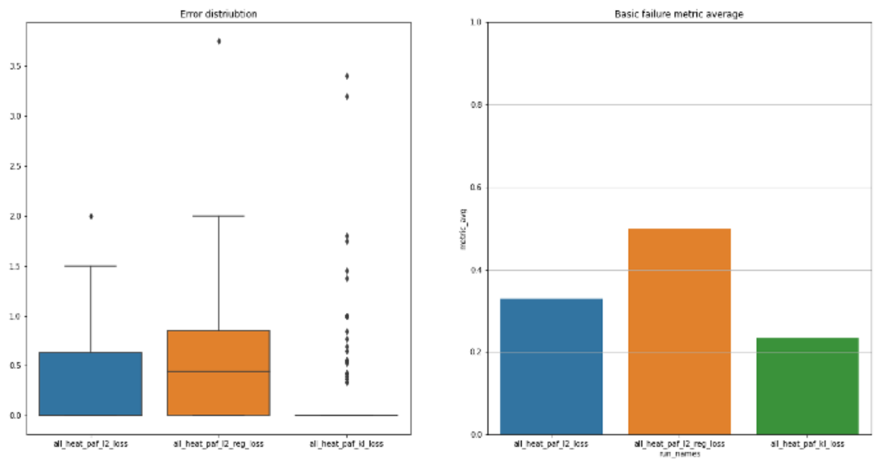


Figure 5.4.: *Basic metric distribution and average*

In Figure 5.4 we compare the basic metric for the three approaches. The first plot shows the error distribution as a box plot where it is visible that the KL divergence has by far the lowest score, which is also visible in the average metric plot. The average of the KL divergence method is so high, while the median is roughly at zero as well as the first quantile, due to some outlier in the data. The outliers are having metric values up to 3.5, which can be explained with the detection of new joints that were not part of the original detection. These joints are dreamed up by the network on the T-shirt and do not form a valid pose most of the time. An example can be seen in the third image in Figure 5.3. This issue increases the basic metric, however, one might argue that random joints on the T-Shirt should still be considered a failure case.

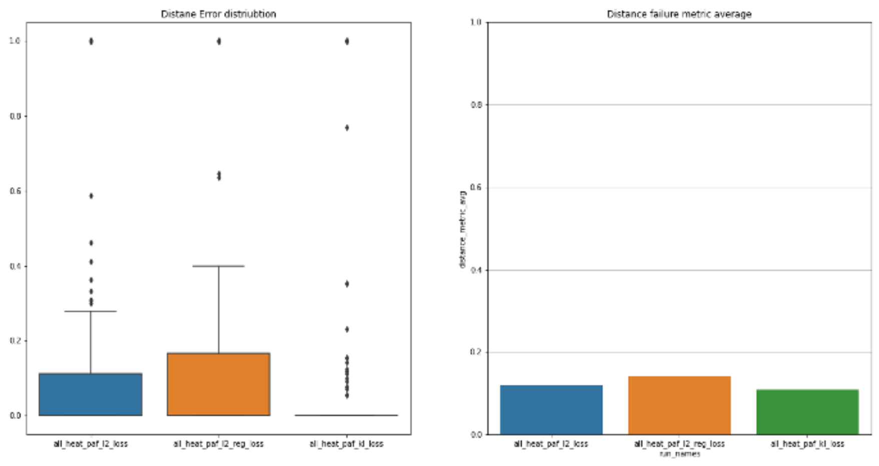


Figure 5.5.: *Distance metric distribution and average*

To account for this effect we introduce the distance metric, which takes the detected joint position into account. Since our dataset is mostly in the same scale we set the valid distance radius to generous 30 pixels which correspond to the size of a lower arm. In Figure 5.5 we see that most methods are around 0.1 with KL divergence having the lowest score still.

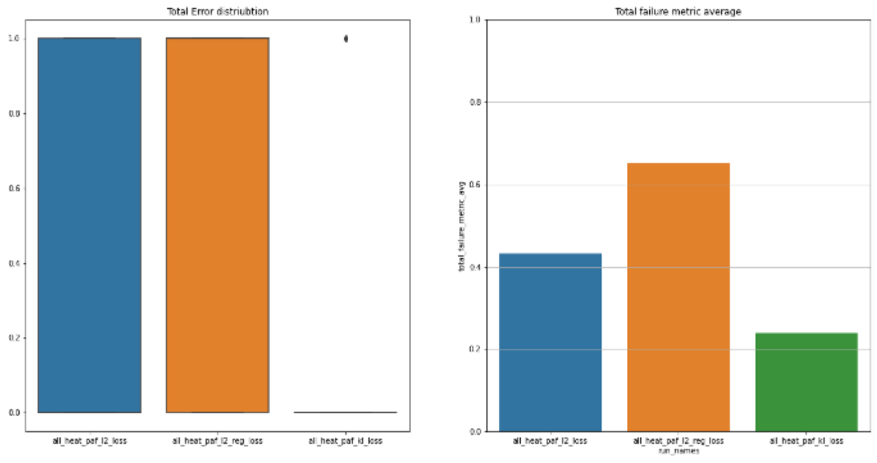


Figure 5.6.: *Total fail metric distribution and average*

Finally, in Figure 5.6 we explore the total fail score distribution and average. Again the KL-divergence method is much better than the other two with an average score of roughly 0.2. It is evident that introducing a smoothness regularizer did not improve the overall performance in any metric. To get a better feeling for the total suppression we use 30 meshes of a MoCap running sequence and render a short video. On each frame, we paint the detected pose. In Figure A.7 we can see that every frame is perfectly detected, which would allow an activity recognition tool to classify the motion as running. However, in Figure A.8 we can see that in only three frames some parts of the legs got detected, which would be not enough for valid activity classification.

To see the robustness of different camera positions, we render the same scene from multiple viewpoints and compute the average scores. The setup is similar to Figure 5.14, but this time we use six elevation angles (-45, -30, -15, 0, 15, 30). The results for the basic metric and the distance metric are shown in the radial heatmap in Figure 5.7. It is important to mention, that viewpoints, where no pose was recognized, are set to NaN since there is no signal to suppress. We can see that the adversarial noise T-shirt is more successful from the front, then from the back. Additionally, we can see that the openPose network generally has more difficulty with side views, since there are angles where nothing gets detected even without adversarial noise (displayed in white). The red spikes in the white sections (300 and 60 degrees) are outliers, which can be explained with a single frame where the pose got detected, but not suppressed. Furthermore, the noise seems not to be affected by the elevation angle.

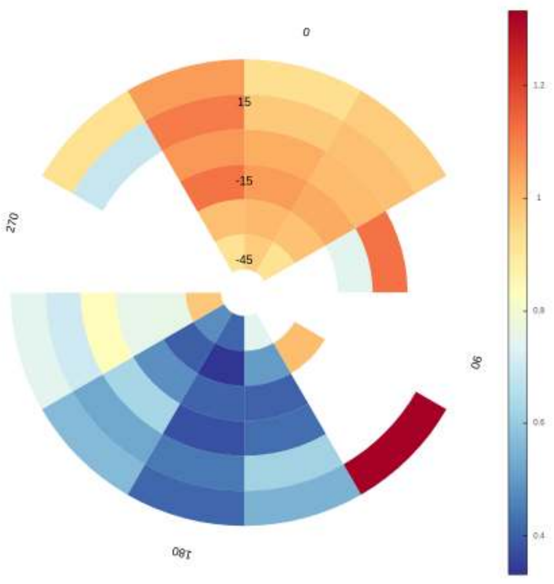


Figure 5.7.: *Average basic metric score for radial analysis*

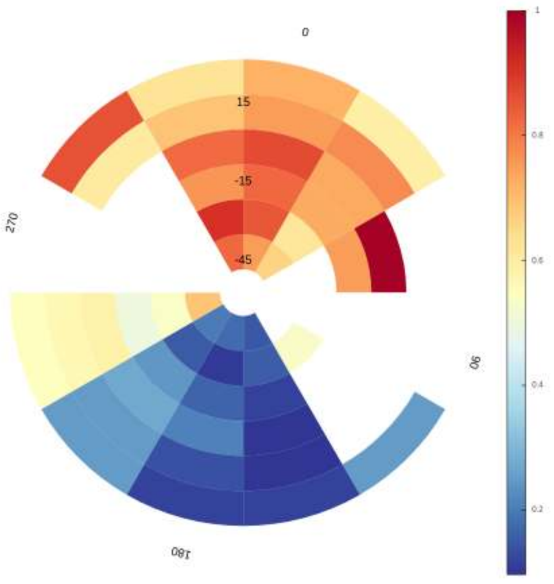


Figure 5.8.: *Average distance metric score for radial analysis*

Finally, to approximate the behavior in the real world we manually fit the T-shirt with the adversarial noise onto a real photograph. In Figure 5.9 we show that the noise can fool the openPose network and lead to none or misdetection.
In conclusion, we can say that openPose is not robust to color changes in the T-shirt and can be fooled into not detecting the correct position or a position at all. These results can be improved by optimizing for the print of some simulated pants as well.

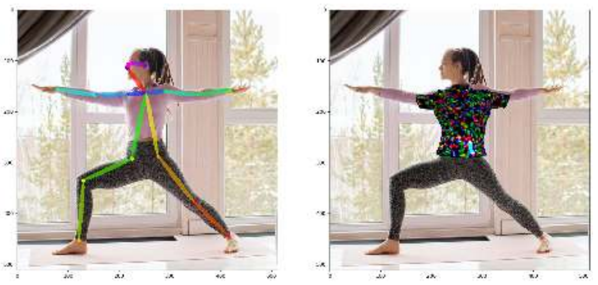


Figure 5.9.: *Female frontside photograph with and without T-shirt*

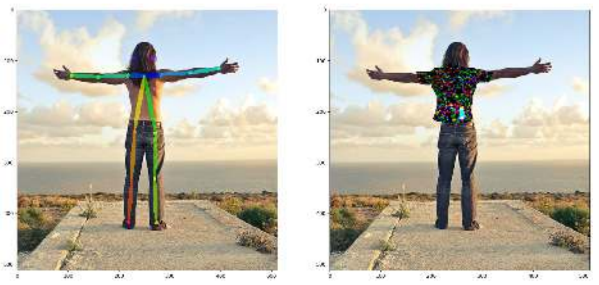


Figure 5.10.: *Male backside photograph with and without T-shirt*

5.2. OpenPose T-Shirt geometry

In this experiment, we test the robustness of openPose to slight clothing geometry changes. OpenPose should be resistant to most slight vertex position changes since clothing is not part of the human pose. These changes can appear in real life in windy conditions or when the subject starts to move.

5.2.1. Experiment setup

In this experiment, we have the same setup as shown in Figure 5.2 with the only difference that instead of using the vertex colors we use of the vertex position as model parameters. We first run an unregularized version using the L2 loss and afterwards we introduce a L1 regularization term on the displacement vectors.

5.2.2. Results and Conclusion

In Figure 5.11 we can see the results of the unregularized experiment. The T-Shirt geometry exploded and started to cover the whole figure, succeeding in optimizing the loss, but not being very helpful. In Figure 5.12 we introduce the L1 loss term with a regularization coefficient of 0.00001. The resulting T-shirt is slightly less deformed, however, still looking more like a modern fashion piece than an average T-shirt. The resulting openPose output is still affected and only the legs get detected. Increasing the regularization coefficient to 0.0001 results in having a much more realistic T-shirt deformation. However, the model is not able to fool the openPose network anymore. Finally, we can say that the openPose network is quite robust to realistic T-shirt geometry deformations.

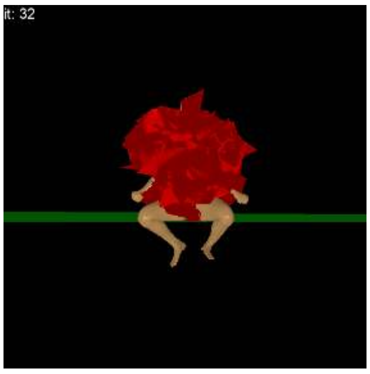


Figure 5.11.: *Result of unregularized optimization (explodes)*

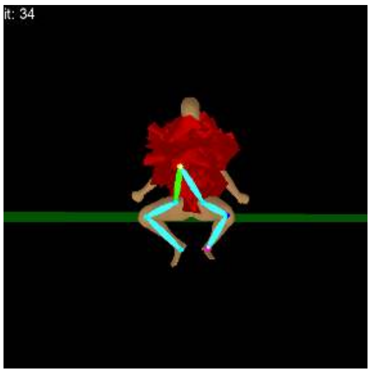


Figure 5.12.: *Result of regularized optimization with coefficient of 0.00001*

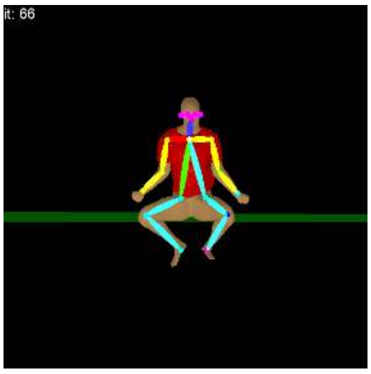


Figure 5.13.: *Result of regularized optimization with coefficient of 0.0001*

5.3. Vertex attribution

Since we compute gradients down to the vertex level, we can apply gradient-based attribution techniques in 3D. This should show direct attribution on the mesh and give us a clear understanding which parts of the geometry are important for the decision.

5.3.1. Experiment setup

We extend the definition of Saliency maps to the 3D domain and propose various ways of approaching this. We define that the importance of a vertex is given by the magnitude of its gradient. This is an absolute value, so we don't have positive or negative attribution in that model. Additionally, we introduce a viewpoint averaging to reduce the noise. This is done by computing the attribution from n different viewpoints around the model and averaging the results. There are two possible ways to average the results. First, by adding up the different gradients, averaging and then computing the magnitude.

$$R_i^c(x) = \left\| \frac{\sum_{i=1}^{n} \nabla S_c(x, v_i)}{n} \right\|_2^2 \tag{5.4}$$

Second by adding up the viewpoint gradient magnitude and average afterwards.

$$R_i^c(x) = \frac{1}{n} \sum_{i=1}^{n} \|\nabla S_c(x, v_i)\|_2^2 \tag{5.5}$$

where v_i defines the viewpoint parameters and x is the vertex we compute the attribution for. We are using three different elevation angles (0, 10, 20) and sample each incrementally by 30 degrees azimuth, resulting in 36 different viewpoints. In Figure 5.14 we can see the setup for a sample pose. S_c defines the output neurons in our case we apply a mask to C-map of a particular joint and average the values contributing to the peak.

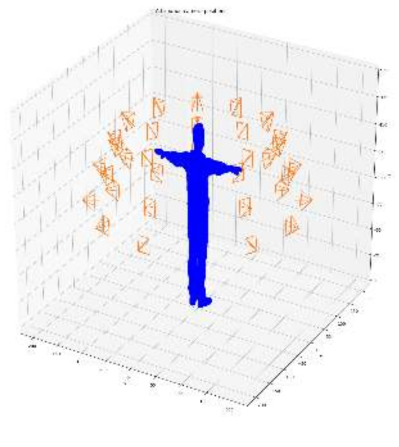


Figure 5.14.: Camera positions for 3D attribution

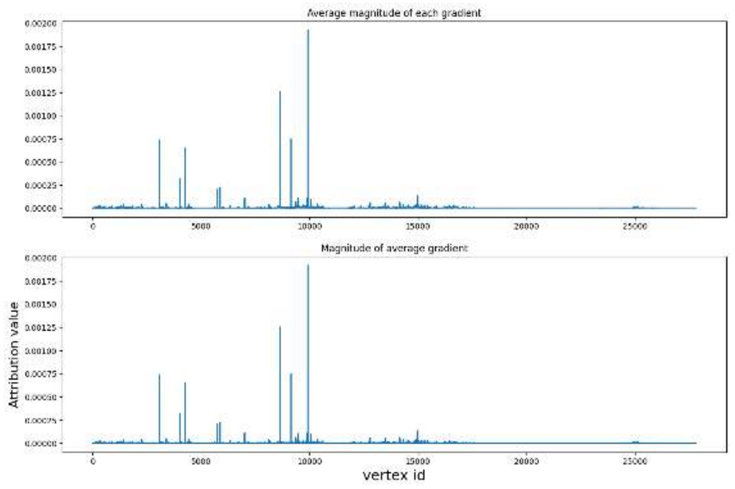


Figure 5.15.: Magnitude per vertex

5.3.2. Results and Conclusion

In Figure 5.15 we can see the attribution per vertex for the right wrist. The graph shows that some areas receive higher attribution values than others as well as a few outliers. These outliers indicate the possibility of an adversarial attack, by moving the single vertex. In Figure 5.16 we map each attribution value to a color value from the plasma heatmap and render it again. The third image shows a clear localized attribution around the wrist and was computed by the method 5.4. The last image shows a less clear attribution and was computed by method 5.5. This phenomenon can be explained by the inconsistency of the single viewpoint gradients. Adding these up cancels out the attribution for some parts of the vertices. In Figure 5.17 we are computing the attribution for the left knee, which does not result in a localized heatmap. Using this information we can conclude that the openPose directly inferences position of leaf joint (eg. wrist or feet) and indirectly inferences intermediate joint positions (eg. knee or elbow), by using the connected joint positions.

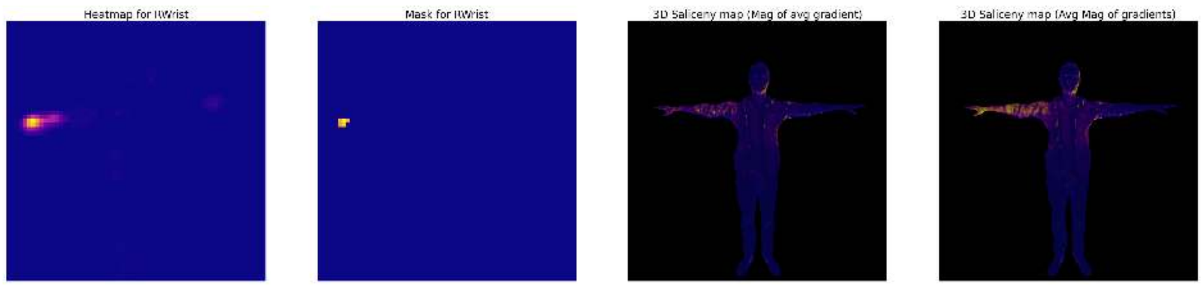

Heatmap for RWrist
Mask for RWrist
3D Saliceny map (Mag of avg gradient)
3D Saliceny map (Avg Mag of gradients)

Figure 5.16.: *3D Attribution for right wrist*

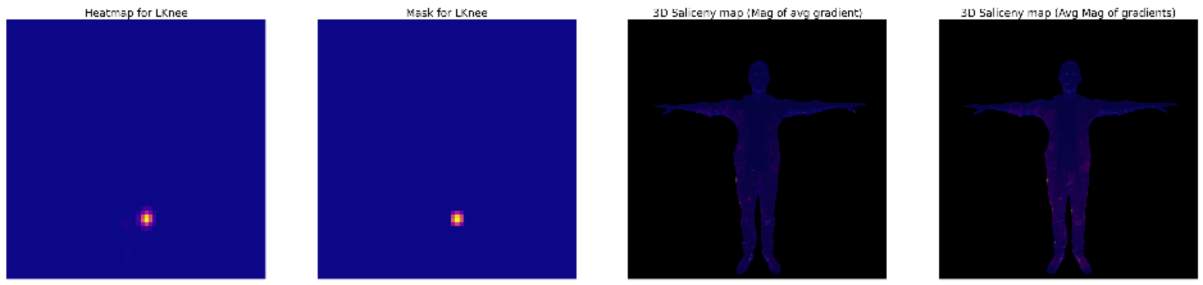

Heatmap for LKnee
Mask for LKnee
3D Saliceny map (Mag of avg gradient)
3D Saliceny map (Avg Mag of gradients)

Figure 5.17.: *Attribution for left knee*

Figure 5.18.: *Left to right: Heatmap, mask, magnitude of average gradient, average magnitude*

5.4. SMPL attribution

In this experiment, we try to push the boundaries of gradient-based attribution even further and compute attribution not on the geometry level but on the parameters of the human geometry generating model. As a generating model we use SMPL [LMR$^+$15], which is based on skinning and uses blend shapes learned from the 3D data-scans. The generator is controlled by 72 pose parameters and 10 body shape parameters. The method was ported to Pytorch in [Zho20], allowing us to extend the computational graph by the SMPL model.

5.4.1. Experiment setup

In Figure 5.19 the reader can see the new pipeline. The model has 72 pose parameters and 10 body shape parameters. In our case, we are fixing the body shape parameters to a constant and only examine the attribution of the pose parameters. To get good attributions, we set all parameters to very small values to prevent cancellation and one parameter to 0.2. The SMPL model generates two tensors. The first one contains all vertex positions and the second one all face indices. These tensors can be directly transformed into a Pytorch3D mesh and rendered to an image tensor. This image tensor can then be forwarded to openPose. Now we can compute joint position attributions with respect to the SMPL pose parameters. As mentioned before we apply a mask to the peak of the right wrist confidence map and use Saliency Values and the gradient multiplied by the input methods to understand the contribution of the input parameters.

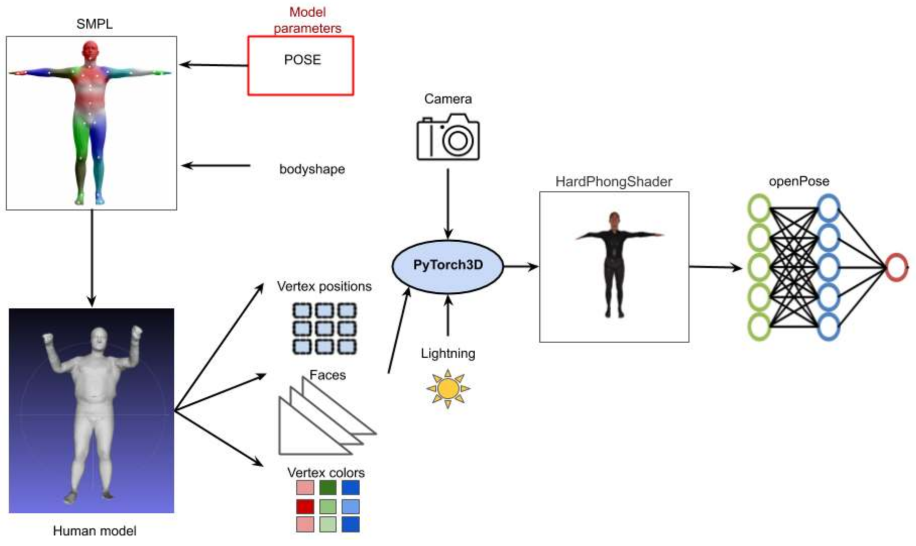


Figure 5.19.: *Pipeline for attribution computation*

5.4.2. Results and Conclusion

Before diving into the attribution of the 72 pose parameters we have to take a closer look at what the parameters represent. In SMPL the pose is defined as a hierarchical tree structure made out of 24 joints see Figure 5.20, important to notice the pose joints are different than the joints openPose uses. The positions of these joints are encoded as axis-aligned rotations relative to their parent joints.
In Figure 5.20 we can see the results of the two attribution methods. In the first plot, there are seven distinct clusters of three parameters influencing the position of the wrist. First, note that the last parameter gets always the most attribution. This can be explained by looking at the position of the camera, which is moved back on the z-axis and since the rotations parameters are axis-aligned the most influence on the position is given by the rotation in the xy-plane.
It is also visible in Figure 5.20 how the cluster corresponds to the pose joints and the most important parameters for the wrist position are the root rotation parameters since they influence all other positions. However, the gradient times input method produces only a single peak at the

index where we set out parameter. This shows us that this particular input parameter is the most important for the placement of the right wrist, which is correct since it represents the rotation of the right elbow.

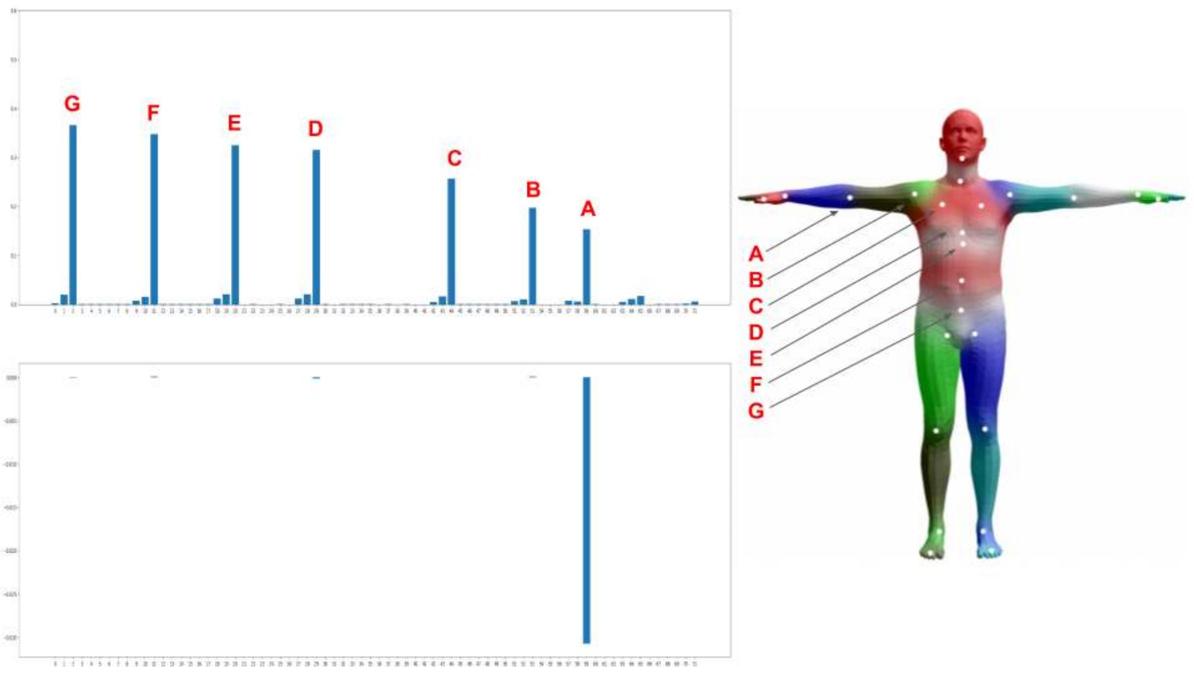


Figure 5.20.: *Top plot Saliency Values, bottom plot gradient times input method. Courtesy of [LMR+15]*

In conclusion, we see how chaining different types of networks and leveraging gradient-based attribution techniques is beneficial for getting a better intuition for the model parameters. Additionally, it also allows one to find how different parameters correlate to each other.

6 Conclusion and Outlook

In this work, we have applied various attribution methods to demonstrate how they can be used to deepen our knowledge of visual learning techniques. We focused mostly on human pose estimation using the openPose network. We started in the 2D domain and analysed interesting cases, such as occlusion or robustness. We have shown that robustness strongly correlates with finding adversarial examples. Therefore, we attacked the network starting with single image noise, move to universal noise and finally universal noise restricted by area.

Due to the success of the previous attacks, we extended the openPose network with a differentiable renderer. This opened up the door to various robustness experiments, like geometry changes or texture changes, as well as attribution computation down to the geometry level. Using this setup we introduced 3D Saliency maps, a technique for computing attribution per vertex, which can be displayed as a 3D heatmap.

Additionally, we proposed a new approach on computing adversarial T-Shirt texture, by leveraging a three-dimensional dataset of various human models wearing the same simulated T-Shirt in different poses. We have shown that our generated texture is universal, robust and fools the openPose network in many cases, which include all networks relying on the pose output.

Finally, we connected the differentiable renderer with the SMPL layer, a human geometry generating model, which allowed us to compute attribution from the openPose network down to the SMPL parameters.

Though we presented a novel way of computing texture for an adversarial T-Shirt, to fully bridge the gap between the digital and physical world a few more steps are required. To get a better understanding for the approach it would make sense to swap out the differentiable rasterizer with a differentiable renderer and create physically-based images for the openPose network. This would allow to model effects like soft shadows and global illumination, which are generally a closer approximation of reality.

As the last step, we would need to ensure printability, which could be achieved by adding a regularizer to the loss function. To be able to iterate fast one could use projection mapping and a projector to display the texture on a T-shirt allowing to change the color faster without the need to reprint the T-shirt.

These types of attacks showed that ML methods are still not very robust and have not learned the essential features in the human perspective. A possible way to tackle this problem would be to incorporate adversarial examples already during the training process and achieve resilience against them. Finally, developing new attribution methods in the 3D domain would also help to identify strength and possible weaknesses of visual learning techniques.

A
Appendix

A.1. Neural network architecture

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_4 (Conv2D)            (None, 150, 150, 32)      832       
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 75, 75, 32)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 75, 75, 64)        18496     
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 37, 37, 64)        0         
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 37, 37, 96)        55392     
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 18, 18, 96)        0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 18, 18, 96)        83040     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 9, 9, 96)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 7776)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 512)               3981824   
_________________________________________________________________
activation_2 (Activation)    (None, 512)               0         
```

```
________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
________________________________________________________________
activation_3 (Activation)    (None, 10)                0
================================================================
Total params: 4,144,714
Trainable params: 4,144,714
Non-trainable params: 0
```

A.2. COCO dataset keypoints

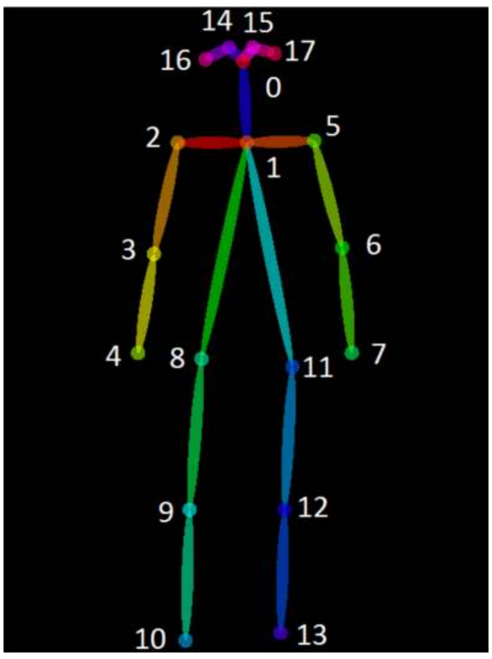


Figure A.1.: *Skeleton keypoints from the COCO dataset*

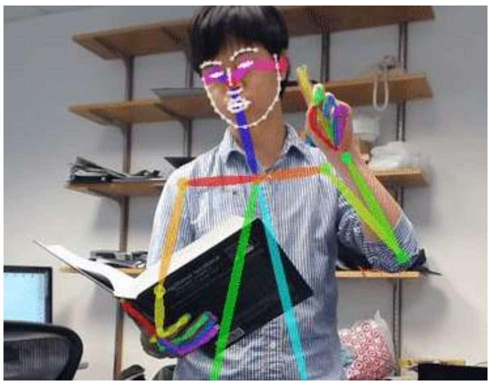


Figure A.2.: *Correct detections even with occluded joints*

A.3. OpenPose universal noise results

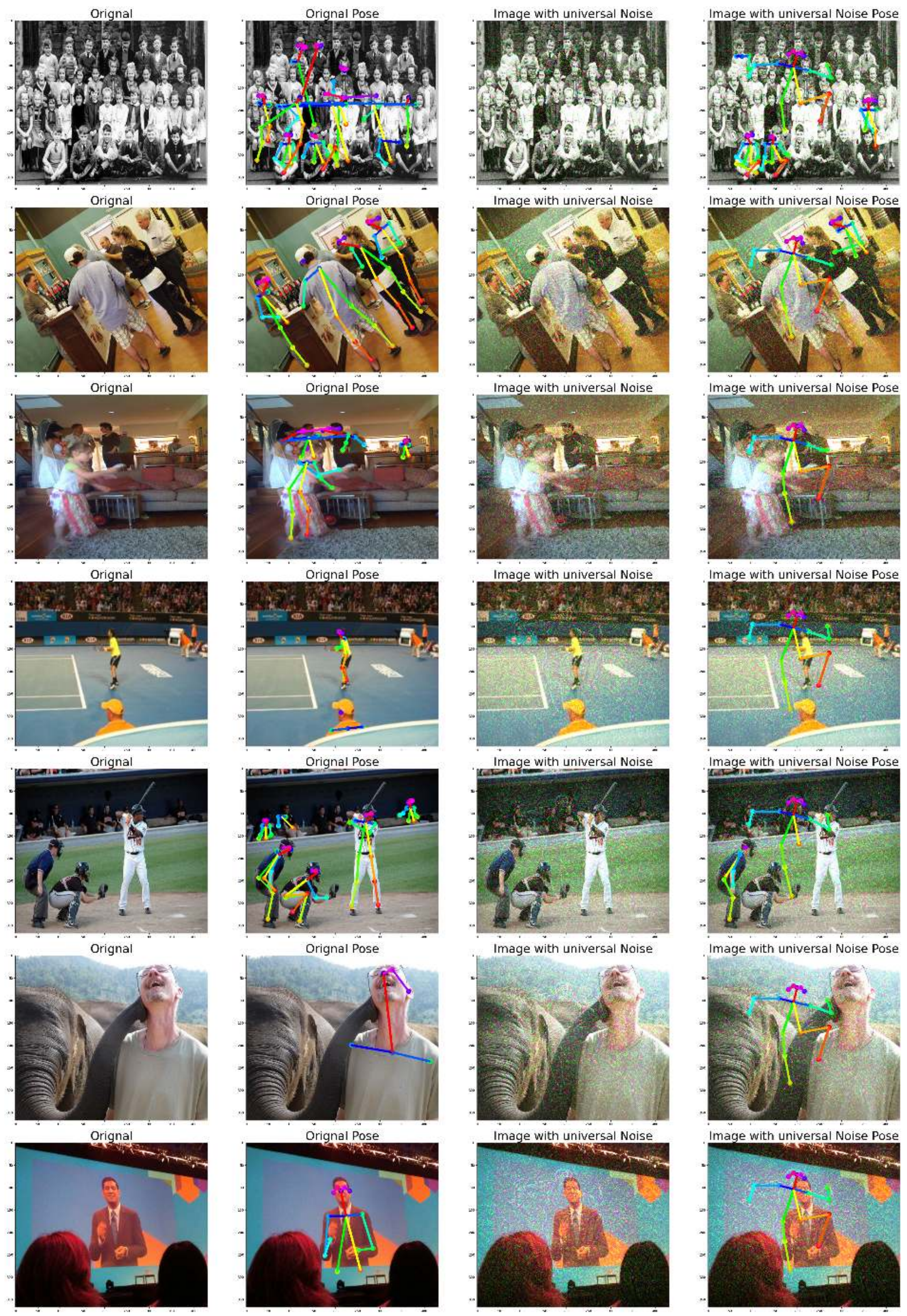

Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose
Orignal
Orignal Pose
Image with universal Noise
Image with universal Noise Pose

Figure A.3.: *Left to right: Input image, detections on input image, image with universal noise and final detections*

A.4. OpenPose translations

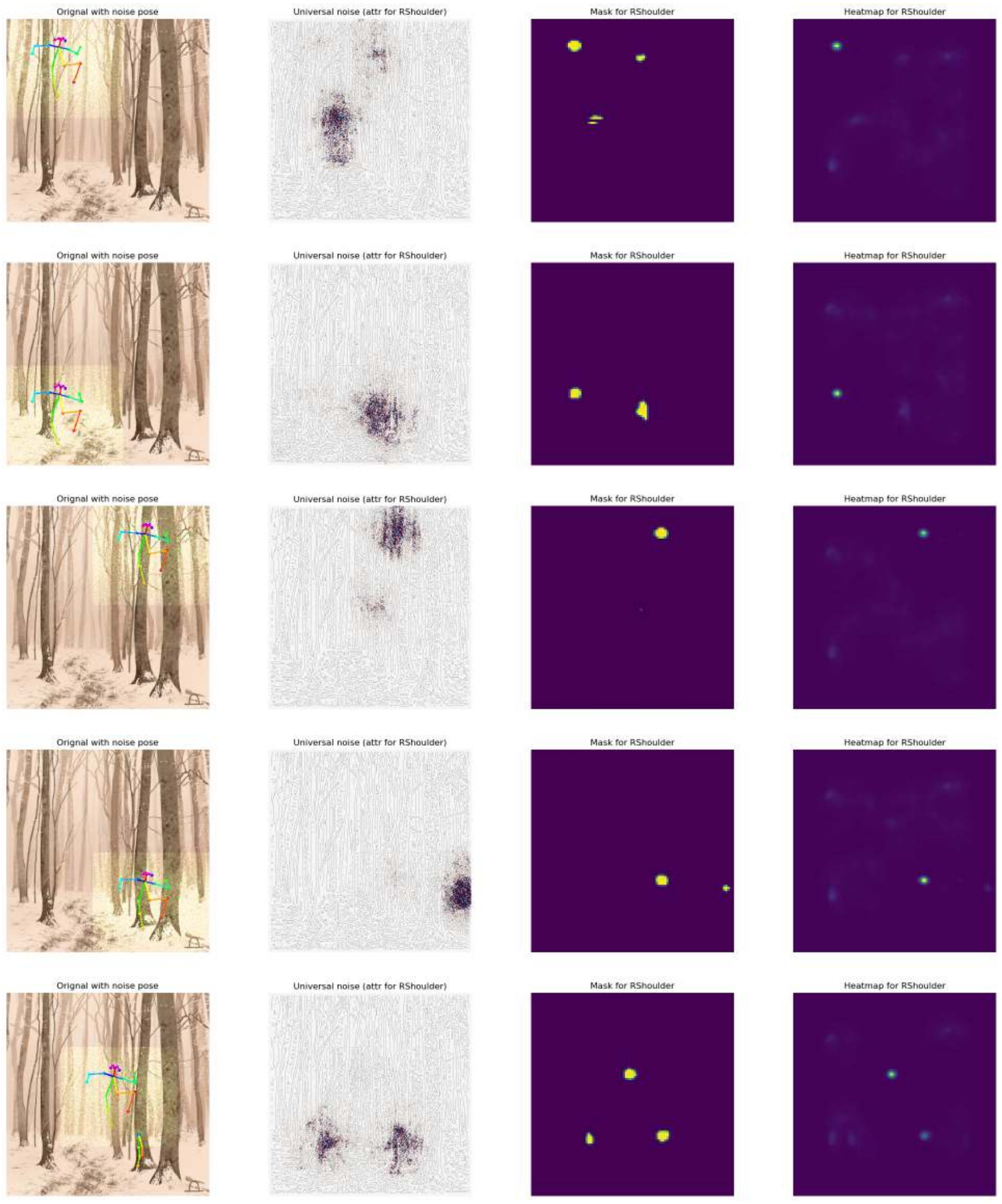


Figure A.4.: *Left to right: image with noise at different positions, attribution for the right shoulder peak, mask for the c-map and c-map*

A.5. OpenPose invisible universal noise results

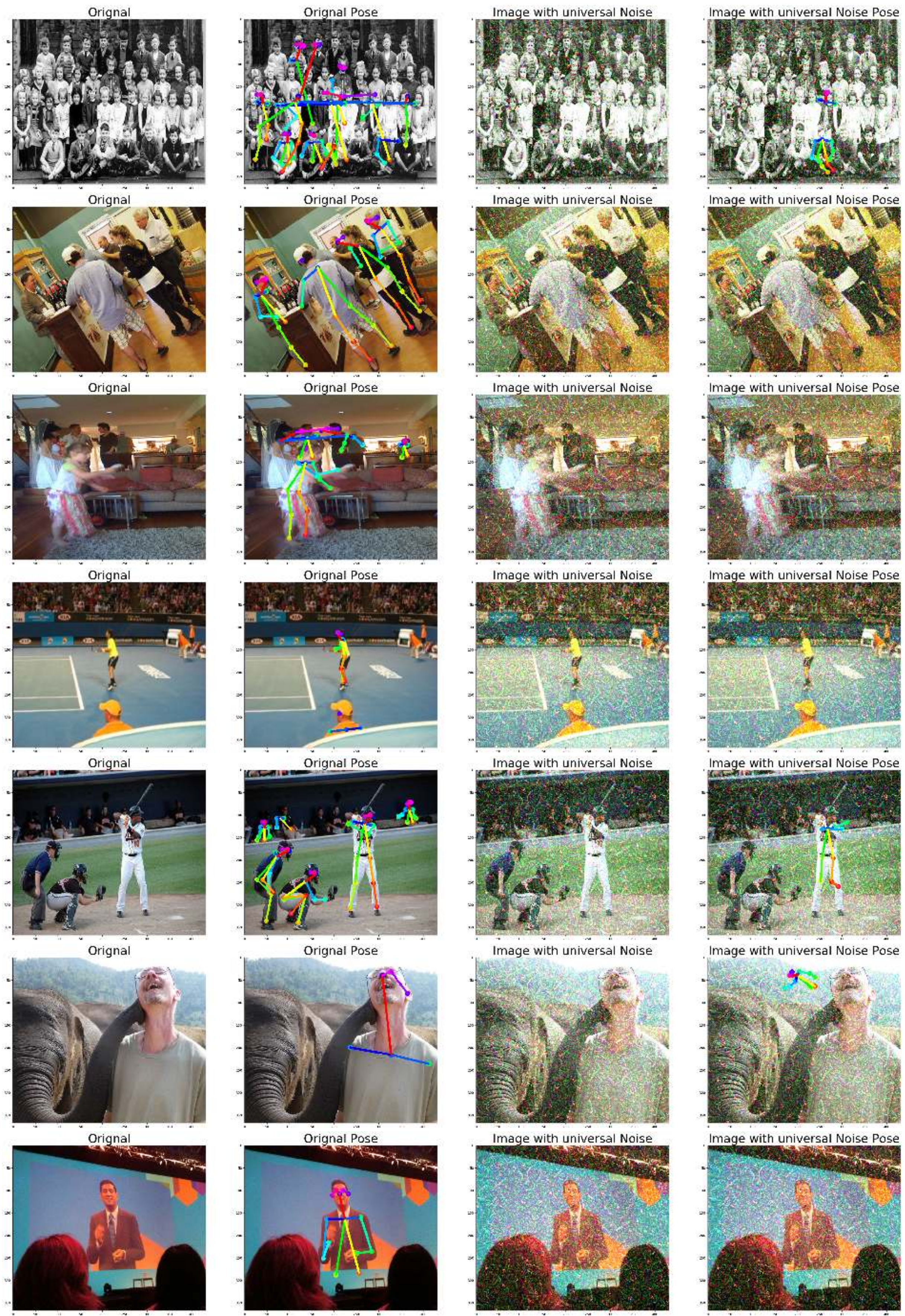


Figure A.5.: Left to right: Input image, detections on input image, image with universal noise and final detections

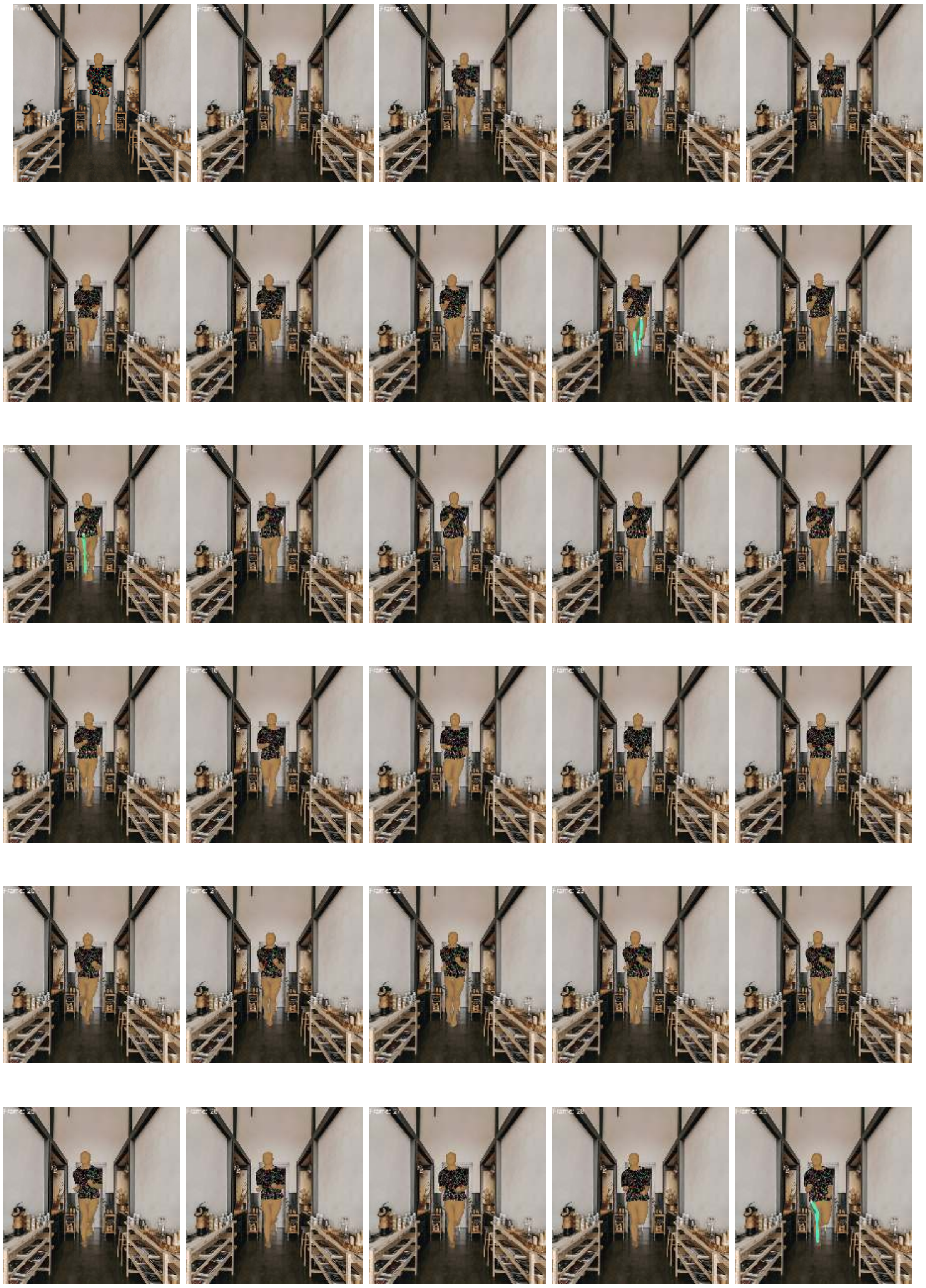
A.6. XML scene description

```xml
<?xml version="1.0" ?><scene version="0.6.0">
  <integrator type="direct"/>
  <shape id="human" type="serialized">
    <transform name="to_world">
      <rotate angle="-11.486546301783648" y="1"/>
            <translate x="-64.92164437446417" y="0" z="0.0"/>
    </transform>
    <string name="filename" value="pose_0_2.serialized"/>
    <integer name="shapeIndex" value="0"/>

    <bsdf type="diffuse">
            <texture type="bitmap" name="reflectance">
            <string name="filename" value="MikeAlger_Texture.jpeg"/>
            <transform name="to_uv">
              <scale x="1"/>
            </transform>
          </texture>
    </bsdf>
  </shape>

  <shape id="rect" type="obj">
     <string name="filename" value="rectangle.obj"/>
    <bsdf type="diffuse">
       <texture type="bitmap" name="reflectance">
        <string name="filename" value="noise_texture.jpg"/>
        <transform name="to_uv">
          <scale x="1"/>
        </transform>
      </texture>
    </bsdf>
    <transform name="to_world">
      <scale x="15.0" y="30.0" z="1.0"/>
      <translate x="0" y="110" z="15"/>
      <rotate angle="-11.486546301783648" y="1"/>
      <translate x="-64.92164437446417" y="0" z="0.0"/>
    </transform>
  </shape>

  <emitter type="constant">
    <spectrum name="radiance" value="1.0"/>
  </emitter>



  <sensor type="perspective">
    <float name="far_clip" value="2500.0"/>
    <float name="fov" value="45"/>
    <string name="fov_axis" value="y"/>
    <float name="near_clip" value="2.0"/>
    <transform name="to_world">
      <lookat origin="-4.21425, 105.008, 327.119" target="-4.1969, 104.951,
          326.12" up="0.0, 1.0, 0.0"/>
    </transform>

    <sampler type="independent">
      <integer name="sampleCount" value="32"/>
    </sampler>

    <film type="hdrfilm">
      <rfilter type="box"/>
      <integer name="width" value="432"/>
      <integer name="height" value="368"/>
    </film>
  </sensor>
</scene>
```

code/ch05/scene.xml

A.7. Frame of video with MoCap running sequence

A.8. Frame of video with noise T-Shirt



Bibliography

[AAB$^+$15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[ACÖG17a] Marco Ancona, Enea Ceolini, A. Cengiz Öztireli, and Markus H. Gross. A unified view of gradient-based attribution methods for deep neural networks. *CoRR*, abs/1711.06104, 2017.

[ACÖG17b] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks, 2017.

[AKB$^+$19] Diego Ardila, Atilla Kiraly, Sujeeth Bharadwaj, Bokyung Choi, Joshua Reicher, Lily Peng, Daniel Tse, Mozziyar Etemadi, Wenxing Ye, Greg Corrado, David Naidich, and Shravya Shetty. End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography. *Nature Medicine*, 25:1, 06 2019.

[Anc20] Marco Ancona. Deepexplain. `https://github.com/marcoancona/DeepExplain`, 2020.

[AÖG19] Marco Ancona, Cengiz Öztireli, and Markus Gross. Explaining deep neural networks with a polynomial time algorithm for shapley values approximation, 2019.

[ASK$^+$05] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: Shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408 – 416, July 2005.

[BCV13] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[BGF17] Facundo Bre, Juan Gimenez, and Víctor Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017.

[BKM$^+$19] Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula, 2019.

[BMR$^+$17] Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch, 2017.

[BP07] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, 26(3):72 – es, July 2007.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[BTD$^+$16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[Bur20] Matt Burns. Bmw magical gesture control finally makes sense as touchscreens take over cars, 2020.

[CDP$^+$19] Robert Challen, Joshua Denny, Martin Pitt, Luke Gompels, Tom Edwards, and Krasimira Tsaneva-Atanasova. Artificial intelligence, bias and clinical safety. *BMJ Quality & Safety*, 28(3):231–237, 2019.

[CHS$^+$18] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields, 2018.

[Cou16] Council of European Union. General data protection regulation (gdpr), 2016. `https://gdpr-info.eu/`.

[CSWS16] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields, 2016.

[DDO$^+$17] R. Daněček, Endri Dibra, Cengiz Öztireli, Remo Ziegler, M. Gross, and Radek Daněček. Deepgarment : 3d garment shape estimation from a single image. *Computer Graphics Forum*, 36:269–280, 05 2017.

[DDS$^+$09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[DK17] Samuel F. Dodge and Lina J. Karam. A study and comparison of human and deep learning recognition performance under visual distortions. *CoRR*, abs/1705.02498, 2017.

[Dre62] Stuart Dreyfus. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, 5(1):30 – 45, 1962.

[DVK17] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning, 2017.

[EEF$^+$17] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models, 2017.

[EVGW$^+$] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[FXTL16] Hao-Shu Fang, Shuqin Xie, Yu-Wing Tai, and Cewu Lu. Rmpe: Regional multi-person pose estimation, 2016.

[Gei19] Claudia Geib. Chinese social credit system, 2019. Accessed 2020-04-09.

[GLYT17] Liuhao Ge, Hui Liang, Junsong Yuan, and Daniel Thalmann. 3d convolutional neural networks for efficient and robust hand pose estimation from single depth images. In *SIGDOC*, pages 5679–5688, 07 2017.

[GSD$^+$17] Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. When will ai exceed human performance? evidence from ai experts, 2017.

[GTI20] GTI. Hand gesture recognition database. `https://www.kaggle.com/gti-upm/leapgestrecog`, 2020.

[GWP$^+$19] Oliver Glauser, Shihao Wu, Daniele Panozzo, Otmar Hilliges, and Olga Sorkine-Hornung. Interactive hand pose estimation using a stretch-sensing soft glove. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 38(4), 2019.

[Hec16] Yotam Hechtlinger. Interpretation of prediction models using the input gradient, 2016.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn, 2017.

[HSSP99] G.E. Hinton, T.J. Sejnowski, H.H.M.I.C.N.L.T.J. Sejnowski, and T.A. Poggio. *Unsupervised Learning: Foundations of Neural Computation*. A Bradford Book. Bradford University Press, 1999.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[Hzz20] Hzzone. Pytorch openpose. `https://github.com/Hzzone/pytorch-openpose`, 2020.

[JSD$^+$14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[JZL$^+$19] Du Jiang, Zujia Zheng, Gongfa Li, Ying Sun, Jianyi Kong, Guozhang Jiang, Hegen Xiong, Bo Tao, Shuang Xu, Hui Yu, Honghai Liu, and Zhaojie Ju. Gesture recognition based on binocular vision. *Cluster Computing*, 22, 11 2019.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[Kel19] Jospeh Keller. Amazon alexa trainig, 2019. Accessed 2020-04-09.

[KGB16] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *CoRR*, abs/1607.02533, 2016.

[Kim18] Ildoo Kim. tf-pose-estimation. `https://github.com/ildoonet/tf-pose-estimation`, 2018.

[KRKP$^+$16] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KUH17] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer, 2017.

[KY55] H. W. Kuhn and Bryn Yaw. The hungarian method for the assignment problem. *Naval Res. Logist. Quart*, pages 83–97, 1955.

[Lab] CMU Graphics Lab. Cmu graphics lab motion capture database converted to fbx.

[LLCL19] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning, 2019.

[LMB$^+$14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollar. Microsoft coco: Common objects in context, 2014.

[LMR$^+$15] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, October 2015.

[LWB$^+$19] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn, 02 2019.

[Mar19] Sherin Mary. *Explainable Artificial Intelligence Applications in NLP, Biomedical, and Malware Classification: A Literature Review*, pages 1269–1292. 07 2019.

[MDBJG16] Tomas Mantecon, Carlos Del-Blanco, Fernando Jaureguizar, and Narciso Garcia. Hand gesture recognition using infrared imagery provided by leap motion controller. volume 10016, pages 47–57, 10 2016.

[MEDM12] Heba Mohsen, El-Sayed El-Dahshan, and Abdel-Badeeh M.Salem. A machine learning technique for mri brain images. pages BIO–161, 01 2012.

[Mic20] Microsoft. Microsoft hololens 2, 2020.

[MKBF17] Jan Hendrik Metzen, Mummadi Chaithanya Kumar, Thomas Brox, and Volker Fischer. Universal adversarial perturbations against semantic image segmentation, 2017.

[MLH03] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55:169 – 186, 2003.

[NDVZJ19] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 38(6), December 2019.

[NPO13] Rahul Narain, Tobias Pfaff, and James O'Brien. Folding and crumpling adaptive sheets. *ACM Transactions on Graphics*, 32:51:1–8, 07 2013.

[NSO12] Rahul Narain, Armin Samii, and James O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Transactions on Graphics*, 31:147:1–10, 11 2012.

[Peg19] Pega. What consumers really think about ai: A global study. Technical report, Pegasystems, One Rogers Street, Cambridge, MA 02142, January 2019.

[Pho75] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311 – 317, June 1975.

[PIT$^+$15] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation, 2015.

[RF16] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[RHDV17] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations, 2017.

[RN09] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[Ros57] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[RRN$^+$20] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Pytorch3d. `https://github.com/facebookresearch/pytorch3d`, 2020.

[SBBR19] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. A general framework for adversarial examples with objectives. *ACM Transactions on Privacy and Security*, 22(3):1 – 30, Jul 2019.

[SHJ+18] Yulong Shi, Xiaoguang Han, Nianjuan Jiang, Kun Zhou, Kui Jia, and Jiangbo Lu. Fbi-pose: Towards bridging the gap between 2d images and 3d human poses using forward-or-backward information. *CoRR*, abs/1806.09241, 2018.

[SjSJ19] Sahil Shah, Naman jain, Abhishek Sharma, and Arjun Jain. On the robustness of human pose estimation, 2019.

[SM56] C. E. Shannon and J. McCarthy. *Automata Studies. (AM-34) (Annals of Mathematics Studies)*. Princeton University Press, USA, 1956.

[SSPH18] Adrian Spurr, Jie Song, Seonwook Park, and Otmar Hilliges. Cross-modal deep variational hand pose estimation. *CoRR*, abs/1803.11404, 2018.

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *CoRR*, abs/1703.01365, 2017.

[SVI+15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision, 2015.

[SVS19] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828 – 841, Oct 2019.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. 12 2013.

[The] The ImageMagick Development Team. Imagemagick.

[TRG19] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection, 2019.

[XZL+19] Kaidi Xu, Gaoyuan Zhang, Sijia Liu, Quanfu Fan, Mengshu Sun, Hongge Chen, Pin-Yu Chen, Yanzhi Wang, and Xue Lin. Adversarial t-shirt! evading person detectors in a physical world, 2019.

[ZF13] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[Zho20] Yuxiao Zhou. Pytorch smpl. `https://github.com/CalciferZh/SMPL`, 2020.

[ZKH18] Nico Zengeler, Thomas Kopinski, and Uwe Handmann. Hand gesture recognition in automotive human machine interaction using depth cameras. *Sensors*, 19:59, 12 2018.

[ZS18] James Zou and Londa Schiebinger. Ai can be sexist and racist âĂŤ itâĂŹs time to make it fair. *Nature*, 559:324–326, 07 2018.