Project Report for EECS 6322: Neural Pose Transfer by Spatially Adaptive Instance Normalization

Peter Caruana*
Department of Electrical Engineering and Computer Science
York University
Toronto, ON M3J 1P3
caruana9@my.yorku.ca

Template for project reports for EECS 6322. This is based on the ML Reproducibility Challenge 2020 report. As part of your project you must submit a document based on this template. Please fill in the sections as described. In addition, you should prepare and record a presentation describing the high level summary of the outcome of your project. The recording should around 5 minutes (no more than 7) and should succinctly describe the elements outlined below (e.g., paper to be reproduced, scope of reproducibility goals, methodology, results, stretch goals and discussion.

Presentation Video: https://drive.google.com/file/d/1i-N7OH5v9rwiu8RQqMsAJkZT2sapjarg/view?usp=sharing

GitHub Repository: https://github.com/PCaruana9/PoseTransfer-6322

Original Repository: https://github.com/jiashunwang/Neural-Pose-Transfer

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

Preprint. Under review.

Reproducibility Summary

Scope of Reproducibility

The goal was to be able to train a deep learning model leveraging a SPAdaIN component in order to transfer the pose of a target mesh to an identity mesh without requiring correspondence between mesh vertices.

Methodology

The central model and evaluation methods were reproduced utilizing the original paper [6], the source code, as well as supplementary works which the original itself was based on, namely the work done by Qi et al. [5]. The deep learning model was implemented in PyTorch just like the original. Google Colab was used as the training platform, however this caused issues of compatibility with the PyMesh library used by the original authors. To circumvent this, a wrapper library was written for the Meshio library which implements some of the functionality needed from PyMesh. As a direct result of this, the data loader, training and evaluation from the original script had to be re-written. Training took approximately three days running on Google Colab Pro.

Results

This work was successful in showing that the methodology of the original paper does result in relatively effective style transfer from one mesh pose to another. Quantitatively a performance of 3.0×10^{-4} was achieved along the PMD evaluation metric with a model trained for 200 epochs.The original paper reports performance of 1.1×10^{-4}, using their maxpool model trained for 200 epochs. Qualitative comparisons show that style is effectively transfered from one mesh to another. Considering that the maxpool version of their model is slightly different, these results are acceptable though they do raise some questions regarding the number of epochs trained for.

What was easy

The model by the authors was fairly simple to implement utilizing the original paper, as a detailed network diagram is given. With some semantic exceptions, the original code-base was fairly usable to test.

What was difficult

Authors made heavy use of the PyMesh library, which caused significant delays in re-implementation in regards to loading and saving data. The standard PyMesh library which can be installed using pip does not support basic functionality. Working versions do exist however they can be very difficult to install on a system, and basically impossible to do on a cloud based service such as Google Colab. Re-implementation of functionality had to be done by creating a wrapper for the Meshio python library. Some specific aspects of model implementation are left out by the authors in the paper. The original github repository is not very well commented making explanations for particular choices scarce.

Communication with original authors

Contact was had with the authors on two occasions to clarify details not mentioned in the paper. First was to inquire about training times and epochs. The authors estimated that training on a single machine took approximately two to three days for 200 epochs, however they recommended to try 300 epochs if results were not adequate. The second correspondence was to clarify the size of the input tensors for the model. There was some confusion regarding the input size for the decoder module, where the code implements it as the number of mesh points $+3$. It was clarified that the "$+3$" term results from concatenating the xyz coordinates of the identity mesh with the pose feature.

1 Introduction and Paper Summary

Pose transfer is defined as the task of transferring the pose of one 3D mesh to another, different 3D mesh. An example of this would be transferring the pose of a 3D cat to the pose of a 3D dog. Typically this would require hand crafted correspondences between the vertices of the target mesh and the identity mesh. This paper by Wang et al. [6] proposes a deep learning model to apply pose transfer without having any built in correspondences between the vertices of either mesh.

The primary contribution to this problem is the integration of image style-transfer techniques, namely through introduction of their novel conditional normalization layer, the Spatialy Adaptive Instance Normalization (SPAdaIN) block. Their network architecture is based on heavily on the Pointnet [5] architecture, and borrows from successful 2D image style transfer implementations of SPADE [2].

Similar to Pointnet, the network architecture mimics the format of an auto-encoder with encoder and decoder segments. The encoder E aims to extract the pose features F_{pose} from the target mesh M_{pose}, in such a way which is invariant to the order of vertices. The encoder consists of three stacked convolutional layers with instance normalization layers in between. The final encoded pose feature is then concatenated with the identity mesh M_{id} and passed to the decoder.

$$Z = E(M_{pose}, M_{id}) = F_{pose} \odot M_{id} \tag{1}$$

Concatenation of M_{id} with F_{pose} produces latent embedding Z. Since vertex orders for input meshes are not consistent, other normalization techniques would not work to preserve features, thus instance normalization is needed to encode the global context.

The decoder introduces the namesake of the paper, the SPAdaIN block. The primary idea of SPAdaIN is to normalize conditionally on spatial position. In the context of 3D pose transfer this would keep the identity of the mesh itself while modifying pose. Adaptive normalization for style transfer [2] and spatially adaptive normalization for image synthesis [4] are already in use in the domain of 2D images. SPAdaIN is a generalization of this idea to deal with point data, in particular 3D points. The functioning behind SPAdaIN is similar to standard instance normalization in that data is normalized along spatial dimensions with learnable parameters β and γ as bias and scale respectively. SPAdaIN takes input from the previous layer X as well as the identity mesh M_{id}. M_{id} is fed through two parallel 1×1 convolutional layers producing $\gamma(M)$ and $\beta(M)$. The final output of the block can then be given as:

$$\text{SPAdaIN}(X, M) = \gamma(M) \cdot X_{IN} + \beta(M) \tag{2}$$

where X_{IN} is the instance normalized input X. The decoder itself consists of three SPAdaIN ResBlocks with 1×1 convolutional layers preceding and following each ResBlock and finally going through a $\tanh(x)$ activation to give the final output. The SPAdaIN residual blocks consist of three SPAdaIN blocks each followed by a 1×1 convolutional layer, modeled after the work of He et al.'s [1] residual blocks.

The loss function used for training is a composite of two separate losses, $\mathcal{L} = \mathcal{L}_{rec} + \lambda_{edge} \cdot \mathcal{L}_{edge}$ Where $\mathcal{L}_{rec}$ is the reconstruction loss and $\mathcal{L}_{edge}$ is the regularized edge loss with coefficient λ_{edge}. The aim of reconstruction loss is to regress vertices to their correct position relative to the ground truth, realized as simply the L2 distance between the meshes. Edge loss is introduced to penalize long edges with the intent to ensure smooth surfaces are generated implemented simply as minimizing the distance of each vertex with respect to its immediate neighbouring vertices.

Training data was generated using SMPL model, which allows for meshes to be generated by sampling a parameter space controlling the pose of a human character mesh. For 16 identities with 400 poses defined by SMPL's parameters. Two meshes are picked at random to be the identity and target inputs for the network. Ground truth is generated by applying the target pose's parameters on the identity mesh in SMPL model. To ensure that the network learns invariance to vertex order, vertices for each mesh are shuffled randomly each time they are selected. To evaluate performance, 14 identities are created which are not in the original dataset and form 72 mesh pairs with randomly selected poses.

The evaluation metric used is the Point-wise Mesh Euclidean Distance (PMD) defined as the mean of distances between vertices $P_v \in M$ and $Q_v \in \hat{M}$, $\forall v \in V$. Along this metric, the model is able to achieve PMD scores of $1.1e-4$ for seen poses and 9.3 for unseen poses. They compare to Deformation Transfer (DT) a non deep learning method which requires a third auxiliary mesh as well as control points in order to transfer pose. DT with 13 control points achieves a PMD score of $6.7e-4$ on the unseen training set and $7.7e-4$ on the seen training set. These results show a comparable performance to the current state of the art with less auxiliary information required. Qualitative results show that this method is able to effectively transfer pose while maintaining the structure of the identity mesh.

2 Scope of reproducibility

This report aimed to reproduce the following claims:

- The proposed SPAdaIN model is able to learn effective 3D pose transfer
- The model learned is invariant to mesh vertex ordering i.e. is permutation invariant

3 Methodology

This project started with implementing data loading and testing the original model. I worked from the original data loading code and used their training data set. The original data loader was dependent on the PyMesh library which required it to be re-written using other 3D mesh loading libraries. I created a wrapper library called MyMesh which implemented the needed functions from Pymesh by using numpy and Meshio. The reason for this is that I was unable to install a working version of the PyMesh library on a Google Colab instance. I took that opportunity to review the code to understand what it does and compared that with the paper. The primary detail I was searching for was that the data loader was randomizing the vertex orders of meshes before returning them. This would ensure that the learned pose feature extractor would be permutation invariant as claimed by the paper. Authors made a trained maxpool variant of their architecture available for use. Some brief testing was done to ensure that data loading was functioning, and that their trained model does indeed work. With a functioning data loader, the next step was to re-implement the model from the paper in PyTorch. This was done largely based off of the network diagrams and descriptions from the paper. Details regarding the sizes of various layers were taken by looking at the original code provided by the authors. To ensure congruency with the original paper, the authors training script was used to train my implementation. This included the implementation of the reconstruction loss $\mathcal{L}_{rec}$ and edge loss $\mathcal{L}_{edge}$. However, similar to the data loading this script was dependent on PyMesh and as such needed to be modified in the same way. Training my implementation of the original model was done using the Google Colab pro service, which gives access to T4 and P100 GPUs. Memory usage did not exceed 12Gb.

3.1 Division of Labour

This project was done alone.

3.2 Datasets

Authors provided their training dataset, which was generated using SMPL model. They used 16 identity models and sampled SMPL's parameter space to generate meshes in certain poses. The dataset contains meshes defined by their identity and pose. In total there are 24027 .obj meshes, however not all are used for training. Each mesh is labeled as "id_i_p" where 'i' and 'p' are integer values with $i \in [0, 16)$ indicating the mesh identity (in this context referring to the character model itself) and $p \in [0, 600)$. To select a pair, a random (i, p) is generated to select an identity mesh, and a separate (i, p) is generated to select a target mesh. The ground truth is the mesh with the identity's 'i' (identity) and target's 'p' (pose). The points for each mesh are loaded into a numpy array and randomly shuffled. They are then turned into tensors before being sent to the model for evaluation/training.

3.3 Hyperparameters

The same hyperparemeters as described in the original paper were used. ADAM optimizer was used for training with learning rate $5e-5$. An edge loss coefficient $\lambda + edge$ of $5e-4$ was used. Batch size was set to 8, and training was done for 200 epochs.

3.4 Experimental setup

Experiments were run on Google Colab pro, a cloud based computing service. Google Colab gives access to T4 and P100 GPUs for training and 24Gb of RAM. The Jupityr notebook used can be found at https://colab.research.google.com/drive/1-oRm1p-1Oun6Tj4zvMEfg629AR_qC6nL?usp=sharing

3.5 Computational requirements

Training for 200 epochs required approximately three days using Google Colab Pro, with a GPU runtime environment. Memory requirements were not recorded however they were well below 12Gb, the maximum allotment on Google

Colab. Colab pro has maximum use period of 24 hours, meaning that training had to be restarted a handful of times. Models were saved every 10 epochs, meaning there was some loss of training time due to timeouts.

4 Results

The work done in this report does support the claims made by the authors. The proposed architecture is able to effectively learn pose transfer, and learns to do so invariant to the order of mesh vertices. However the evaluation metric used, point-wise mesh euclidean distance, does not appear to be useful for fine-grained comparison of results.

4.1 Pose Transfer

I trained a model based on the SPAdaIN architecture proposed in the paper utilizing the same test dataset. Training for 200 epochs, the model achieves $3.0e-4$ score in the PMD metric on previously seen data. The authors using a max-pool version of the model are able to achieve $1.1e-4$ score for seen data. Observing the outputs for both models in comparison to ground truth (shown in Fig. 1 it can be seen that both the original and re-implemented models are able to transfer pose effectively. The authors model is able to do so with noticeably less artifacts, in particular around the legs and feet however this can be attributed to their use of a slightly different max-pooling architecture. These artifacts are also present in the SPAdaBIN model implemented and trained for the stretch goal.

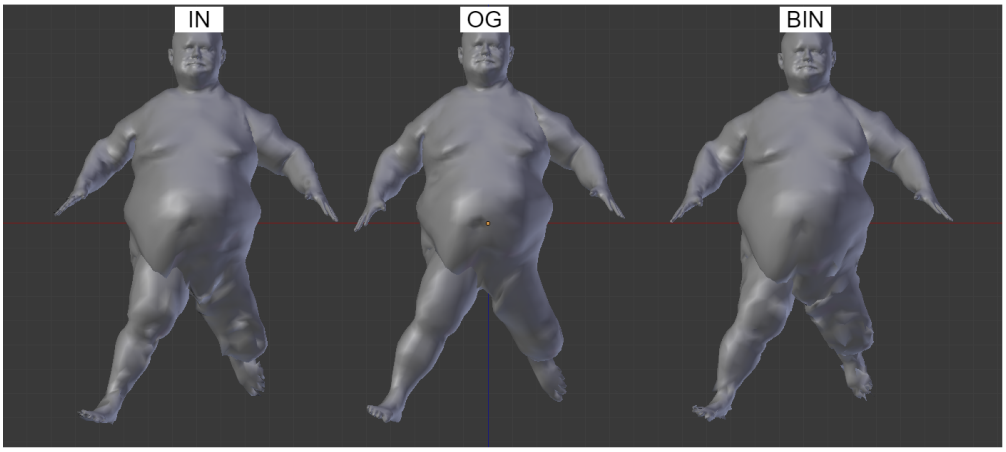

Figure 1: Comparison of re-implemented model (IN), original model (OG) and SPAdaBIN implementation (BIN) using the same input identity and target meshes

4.2 Permutation Invariance

This claim was simply tested by enforcing non-uniform ordering of mesh vertexes when loading data into the model. It is validated through the success of the trained model described above.

4.3 Point-wise Mesh Euclidean Distance (PMD)

While not an explicit claim by the authors, they do use this metric as a means to compare with other state of the art pose transfer techniques, namely Deformation Transfer. Qualitatively their original max-pooling model achieves smoother results than the ones trained for this report, however consistently achieves poorer PMD scores than the trained models, averaging $7.0e-4$. This suggests that this metric is perhaps not too informative for values less than some amount. It should also be noted that authors claimed to achieve scores of $1.1e-4$ for previously seen data.

5 Stretch Goals

The authors main contribution is the SPAdaIN block which uses spatially conditional normalization in order to generate a mesh which maintains the original identity but with the pose features of the target. This task is (by the authors own recognition) heavily inspired by 2D image style transfer. In that same spirit the stretch goal was to apply batch-instance normalization (BIN) to the pose transfer task inspired by Nam and Kim [3] who proposed using BIN for image style

transfer in residual networks. The theory is that BIN selectively normalizes unnecessary styles or features making the useful styles more salient. Using the original dataset, a network was trained which uses SPAdaBIN instead of SPAdaIN blocks. Batch-Instance normalization was implemented using code from Nam and Kim [3] which can be found in their Github repository. Both models were trained for 200 epochs and were compared quantitatively and qualitatively. Using the PMD metric evaluating on seen data, the instance normalization model scored on average $3.0e-4$, versus $5.0e-4$ for the batch-instance normalization model. These differences are virtually negligible when qualitatively looking at the resulting outputs shown in Fig. 2. Both IN and BIN models suffer from artifacts around the feet of the output mesh, something not found in the original model provided by the authors.

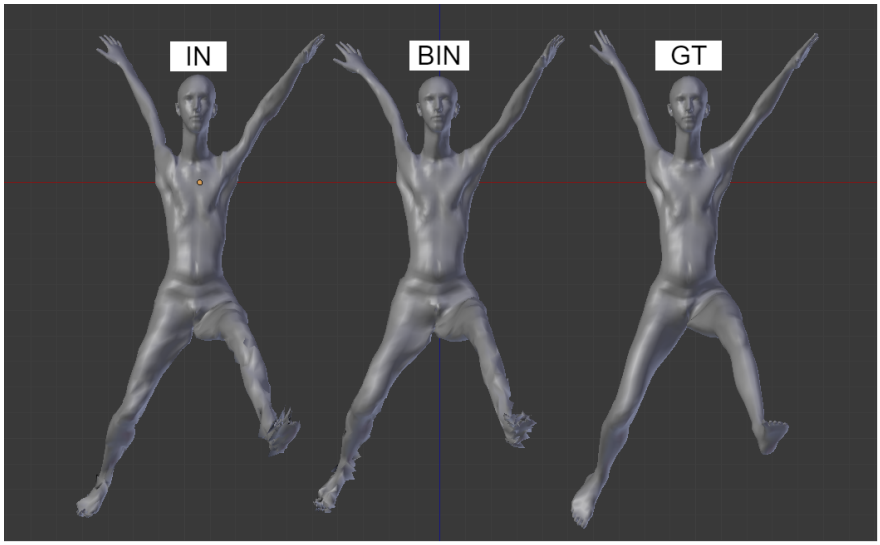

Figure 2: Sample outputs for models using SPAdaIN blocks (IN), SPAdaBIN blocks (BIN) and ground truth (GT)

6 Discussion

Overall, the results of this report do support the claims of the original paper. Their proposed network utilizing SPAdaIN blocks does appear to be capable of learning to transfer pose effectively from one 3D mesh to another while requiring no correspondence between either mesh. Since the paper describes the non max-pooling model, that is what was re-implemented for testing. If given more time a max-pool version would also have been trained and evaluated. Training two similar models for 200 epochs each yield results which do transfer pose effectively, however do not match the quality of the original model. This is explained either through the use of max pooling or the authors perhaps trained for more than the 200 epochs stated in the paper. Though it should be emphasized that this is really a minor detail and quite inconsequential to the paper as a whole.

6.1 What was easy

The evaluation and implementation were fairly straightforward. The network architecture is clearly described, with detailed network diagrams for each component. Implementing the general shell of the model was simply a matter of following instructions laid out in the paper. The same can be said for the loss functions and evaluation metrics. The codebase was divided into logicial scripts according to their function. No complex maths were particularly needed, and even so examples for everything could be found in the authors repository.

6.2 What was difficult

Where the paper lacks in detail is in regards to the specific parameters of network implementation, namely the size of layers through the network. The only way to determine this is through looking at the original code for their model. The difficulty arises in trying to understand the justification for certain numerical values as the authors did not leave many (if any) comments in their code. It took some work to determine the values for in-channels, out-channels and kernel sizes (without blatantly copying code).

The lack of comments also contributed to problems with dependencies on the PyMesh library. When it was determined that installing a working version of PyMesh on Google Colab was not feasible, the lack of comments made it difficult to exactly understand what the library was being used for so that the needed functionality could be replicated, which required digging through the PyMesh API. Determining and re-implementing PyMesh functionality using other 3D mesh libraries was the most difficult and time consuming portion of this project, also including the many hours (and frustration) spent simply trying to install this library in a Google Colab instance. The library can be installed though python's pip tool, however this is deceptive as the version installed does not have any working functionality.

6.3 Communication with original authors

Contact with original authors was fairly brief, but they were very helpful in answering some questions I had. I inquired about the choice for initial layer sizes for the decoder to be $1024 + 3$. I was informed that this was a result of concatenation of pose features F_{pose} and the identity mesh M_{id}, Where the $+3$ term is for the xyz channels of M_{id} and 1024 is the number of vertices in each mesh. I also inquired generally about training times so that appropriate time could be taken to train models. Authors stated that they trained for approximately 1 to 2 days on a single GTX 1080ti GPU, though they suggested training for 300 epochs for better results. That last point gives reason to think that the 200 epochs in the original paper is probably inaccurate, though inconsequential.

References

[1] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[2] Xun Huang and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1501–1510.

[3] Hyeonseob Nam and Hyo-Eun Kim. "Batch-instance normalization for adaptively style-invariant neural networks". In: *arXiv preprint arXiv:1805.07925* (2018).

[4] Taesung Park et al. "Semantic image synthesis with spatially-adaptive normalization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2337–2346.

[5] Charles R Qi et al. "Pointnet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[6] Jiashun Wang et al. "Neural pose transfer by spatially adaptive instance normalization". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 5831–5839.