Implementation Report

Team: Skeletal Insights – Emily Coppess, Jay Ong, Shahbaz Chaudhary

**Research Purpose**

The focus of this project is to decide the classification basis[[1]](#footnote-1) for our proposed yoga learning application. Our two proposed methods are a one-step end-to-end deep learning model and a two-step joint coordinate classification model. The one-step model would take in images and output a pose label – all feature engineering would be done by the hidden layers of the neural net. The two-step model would first use a deep learning model to locate the coordinate points of the joints in the image. Then, those joint coordinates would be fed into a classification model which would output the pose label.

We will evaluate these two methods on the basis of the following metrics: accuracy, speed, and flexibility. In order for this proposed learning application to function, the model needs to be able to correctly identify yoga poses. Specifically, it needs to be sensitive to both small and large differences between poses, while still allowing for some individual variation among yoga practitioners. Speed is critical because this system is intended to give real time feedback. Finally, we would like to minimize domain-specific data requirements, i.e., training flexibility, wherever possible to allow for the future expansion outside of the yoga space.

**Research Hypotheses**

The first method will be highly domain-specific and may require more time to train – resulting in a reduction in speed and flexibility[[2]](#footnote-2). Post-training, we predict that this method will see a speed increase since classification is done in a single step. Due to the lack of explicit feature engineering, this method may show a higher sensitivity to the small inter-pose differences than the two-step model, while placing a higher importance on individual variation than may be desired (affecting the overall accuracy). On the other hand, the two-step model relies on domain-neutral data for the lengthiest part of the training process, i.e., the joint coordinate neural net. This neutrality results in a system flexible enough to be easily extended to new sports. Additionally, we think that the explicit feature-engineering might also assist the system in handling noise reduction – thus improving the overall accuracy. There may be a reduction in the system’s ability to handle small inter-pose differences. Finally, this method may be slower at classification than the first method because the image information needs to travel through two models rather than just one. These predictions are summarized in Table 1.

Table 1. Method Performance Predictions

|  |  |  |  |
| --- | --- | --- | --- |
|  | Accuracy | Speed | Flexibility |
| One-step method | * Better at determining small inter-pose differences | * Slow on training * Fast on classification | * Highly-domain specific data requirements |
| Two-step method | * Higher overall accuracy due to feature engineering | * Faster on training * Slower on classification | * Domain-neutral data required for joint model * Domain-specific data needed for classification |

**Methodology**

*DATA COLLECTION*

We utilize two types of datasets: domain-neutral and domain-specific. Only the joint coordinate neural net uses the domain-neutral dataset which requires joint tagging. For our domain-neutral dataset, we use the “MPII Human Pose” dataset (MPII) from the Max Planck Institut für Informatik. We chose this dataset because it is the training set used for DeeperCut, the neural net we are using for the joint coordinate model. MPII consists of 25,000 images of over 410 human activities and has been coded with joint tagging for the following joints: ankles, knees, hips, wrists, elbows, shoulders, neck, and head.

Unfortunately, there is not a pre-existing domain-specific corpus, so we have to build our own. Since there are over a thousand, possible yoga poses, we need to select a subset to test our methods. To this end, we identify three large groups which we feel encapsulate the large inter-pose differences that our system needs to handle: standing, sitting, and inverted[[3]](#footnote-3). Within each of these three groups, we select a couple of representative poses (which will allow us to test how the system handles small inter-pose differences). We settled on the eight poses seen in Figure 1. Using the Sanskrit names[[4]](#footnote-4) for these poses, we build a web-scraper to pull yoga images from Google images using search words. To build our yoga corpus and to limit the amount of time we spend tagging, we only scrape for one pose at a time - using the search term as the pose label for the resulting scraped images.



Figure 1. Selected poses used to create tagged yoga corpus.

Once we have run the web-scraper, we need to clean up the search results. An image is removed from the corpus if it has any of the following attributes:

* 1. Multiple people
  2. Body parts that are cut-off or obstructed
  3. Non-human subject
  4. Poor resolution and/too small

The number of images remaining post cleaning can be seen in Table 2. Poses will be referred to using their English names for convenience throughout the rest of the paper.

Table 2. Web-scraping result counts by pose, 1,350 images

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pose | downward dog | Buradvaja’s twist | noose | shoulder stand | headstand | warrior I | warrior II | warrior III |
| Count | 163 | 133 | 118 | 153 | 194 | 150 | 238 | 201 |

We normalize our images to ensure that they are all the same size. To further expand our dataset, we add a layer of artificial noise to each of our images. We selected three types of noise: Gaussian blur, rotation by 180°, and greyscale. The results of these manipulations can be seen in Figure 2 and the final size of our dataset can be seen in Table 3.



Figure 2. Every image had their dimensions adjusted so that each image was a 64x64 square (see leftmost image). We then added three types of noise (from left to right): Gaussian blur, flip, and greyscale.

Table 3. Yoga pose corpus distribution, corpus contains 5,400 images.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pose | downward dog | Buradvaja’s twist | noose | shoulder stand | headstand | warrior I | warrior II | warrior III |
| Count | 652 | 532 | 472 | 612 | 776 | 600 | 952 | 804 |

*CLASSIFICATION*

We randomly divided our dataset into three groups: training, test, and holdout. The test set consists of 25% of the dataset (1,352 images), the training set consists of 69% of the dataset (3,728 images), and the holdout set consists of 6% of the dataset (320 images). Models are compared using the accuracy scores from running the classification model on the test set.

1. One-step classification

We begin by examining a model architecture originally developed to classify the CIFAR 10 dataset (Krizhevsky, 2009).[[5]](#footnote-5) The model accepts images as input and outputs yoga pose labels. Training is accomplished using the Adam optimization algorithm built into TensorFlow with a learning rate of 0.001 and using categorical cross-entropy as the loss function. We explore whether adding more layers and manipulating filter sizes and number/type of pooling layers increases model performance. Additionally, we examine two other well-known architectures: Inception V3 and ResNet. We settle on the five approaches detailed in Table 4. The winning approach from Table 4 will be compared to the winning model from the two-step method.

Table 4. Attempted Model Architectures

|  |  |
| --- | --- |
| Approach 1: *CNNs* | Combination of convolution layers, max pooling layers, and a fully connected layer |
| Approach 2: *CNNs with large filters* | Same architecture as Approach 1 but with a larger filter size |
| Approach 3: *CNN with no max pooling* | Same architecture as Approach 1 minus the max pooling layers |
| Approach 4: *Inception V3* | Multilayer perceptrons connecting convolutional layers and global average pooling |
| Approach 5: *ResNet* | 100+ layers, shortcut connections, and downsampling layers |

1. Two-step classification

Images are fed as input to the model detailed in Step One, which outputs joint coordinates and confidence levels. The coordinate points are input into the model detailed in Step Two, which outputs a pose label.

*Step One: Joint Coordinate Model*

We considered two possible pose estimation models: DeepCut and DeeperCut. Ultimately, we use DeeperCut due to its faster speed and higher confidence levels. For this model, we used the model weights that produced the highest confidence found by DeeperCut being trained on the MPII dataset. The model pinpoints areas that are estimated to be joints (see Figure 3) and produces an xy-coordinate pair and a confidence level.

Sample visualization of DeeperCut algorithm in recognizing joint coordinates:


Figure 3. Example of joint locations produced by DeeperCut

*Step Two: Pose Classification Model*

In order to make sure that using only joint coordinates does not jeopardize our classification attempts, we build a few binary classification models. The purpose of these binary classification models is twofold. Firstly, we want to make sure that the coordinates being produced from the neural nets run on 2D images can actually be used for classification (e.g. sanity-checking our DeeperCut results). Mostly crucially, we need to see if a yoga-specific layer in the DeeperCut model is necessary to get usable results. Secondly, how the binary models run might give us clues about which multi-class models we should explore next. Since we are eventually going to be using multi-class classification models, we need to check whether joint coordinates could be used to learning big differences (e.g., downward dog versus warrior pose) as well as small differences (e.g. warrior 2 versus warrior 3). We build models looking at the following three poses: downward dog, warrior 2, and warrior 3. (See Appendix B for the results.)

After our DeeperCut sanity check, we build multiclass classification models. We focus specifically on Support Vector Machine and Random Forest as they performed the best for the binary classification task and have the lowest logloss scores across the first attempts at multiclass classification (see Appendix C). We run the classification models first with the default parameter settings (provided by the scikit learn library in Python) and then we search for the optimized parameters using gridsearch.

**Preliminary Results**

*One-step classification*

Based on the accuracy scores below, the ResNet architecture produces the most accurate model by a clear margin. The other models do not exhibit large differences in their accuracy scores. It would appear that the addition of several more layers, a global pooling layer, and shortcut connections are necessary additions for this classification problem.

|  |  |
| --- | --- |
| CNN | 0.620 |
| CNN Large Filter | 0.598 |
| CNN No Pooling | 0.635 |
| Inception V3 | 0.636 |
| ResNet | 0.799 |

Figure 4. Holdout Accuracy for the one-step models

*Two-step classification*

The two models that we end up comparing for the two-step method are a joint-to-random-forest model and a joint-to-SVM model. As mentioned in the methodology section, we first run the models with the default parameter settings provided by Python and then with optimal parameters found using gridsearch. The accuracy scores for the models can be seen in Table 5. The optimal parameters can be seen in Table 6.

Table 5. Accuracy Scores for joints-to-classification models

|  |  |  |
| --- | --- | --- |
|  | Default Parameters | Optimized Parameters |
| Random Forest | 0.7 | 0.74 |
| SVM | 0.29 | 0.72 |

Table 6. Optimal parameters found by gridsearch

|  |  |
| --- | --- |
| Random Forest | max\_depth = 12, max\_features = 4, min\_samples\_leaf = 2, n\_estimators = 90 |
| SVM | C = 10, gamma = 0.0001, kernel = ‘rbf’ |

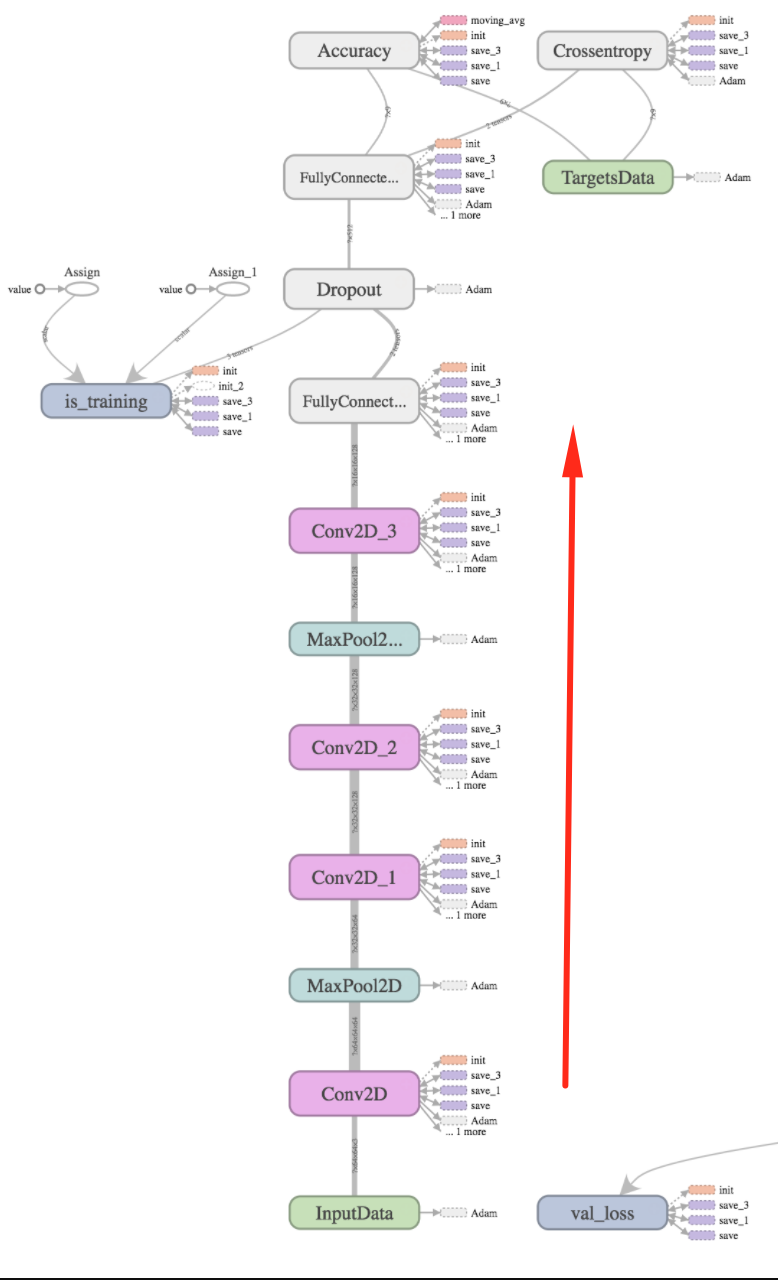
The random forest model is the clear winner (which matches our findings for the binary classification task). However, the margin of improvement is small. When we compare the accuracy scores for the one-step method and the two-step method, the one-step method is the clear winner based on overall accuracy. We need to examine the pose classification confusion matrices for both models to assess the finesse of each model. The model which handles small differences the best would be the more efficient choice for a pose recognition system which contains more than these eight poses.

**Remaining Work**

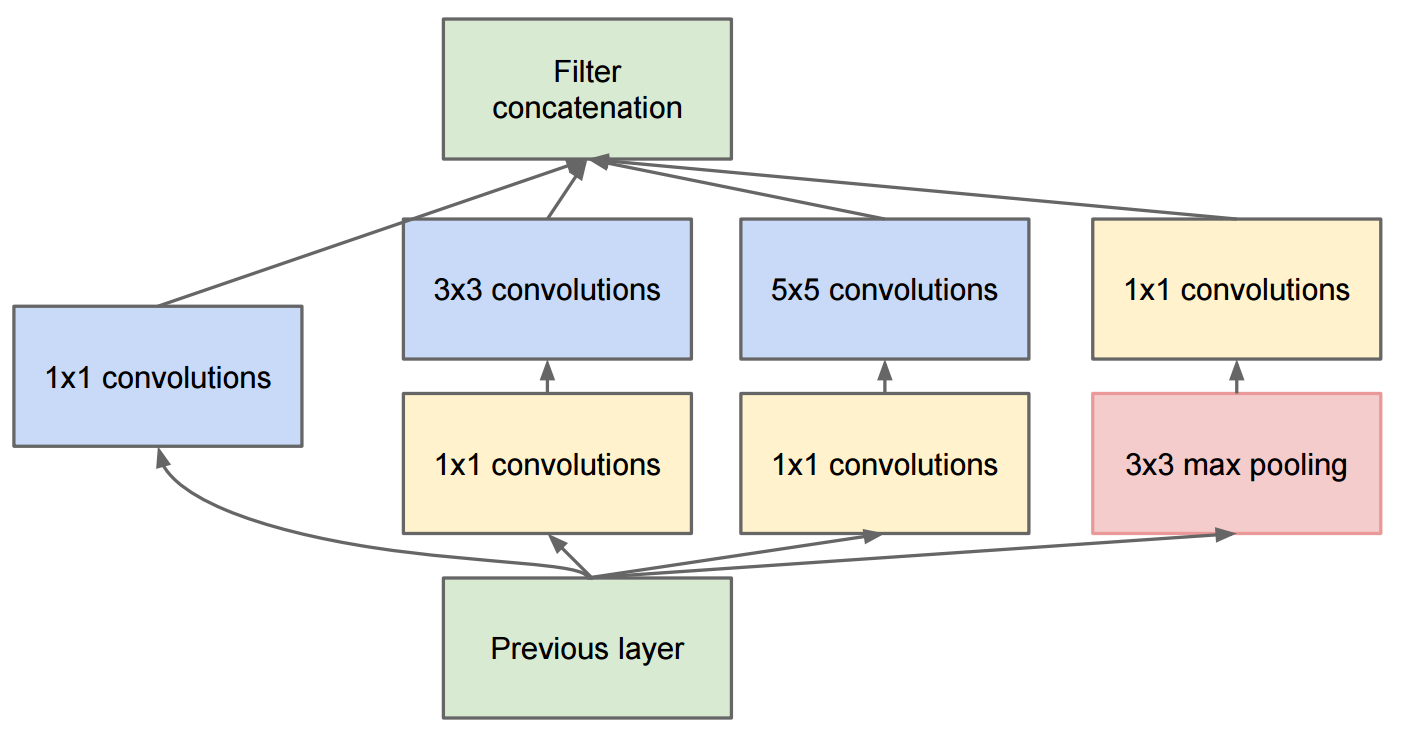
1. Confusion matrices of pose-by-pose accuracy
2. Calculate the duration of the classification tasks for all the models

**Appendix A: Model Architectures**

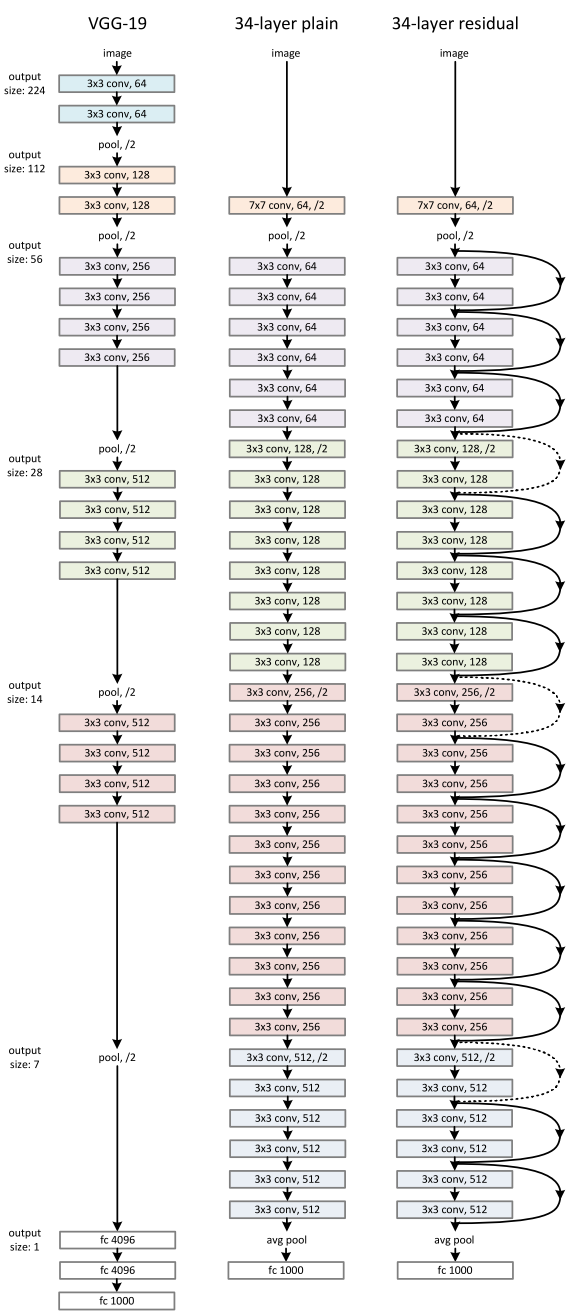
1. **CNN Architecture Basis for Approach 1,2, and 3**



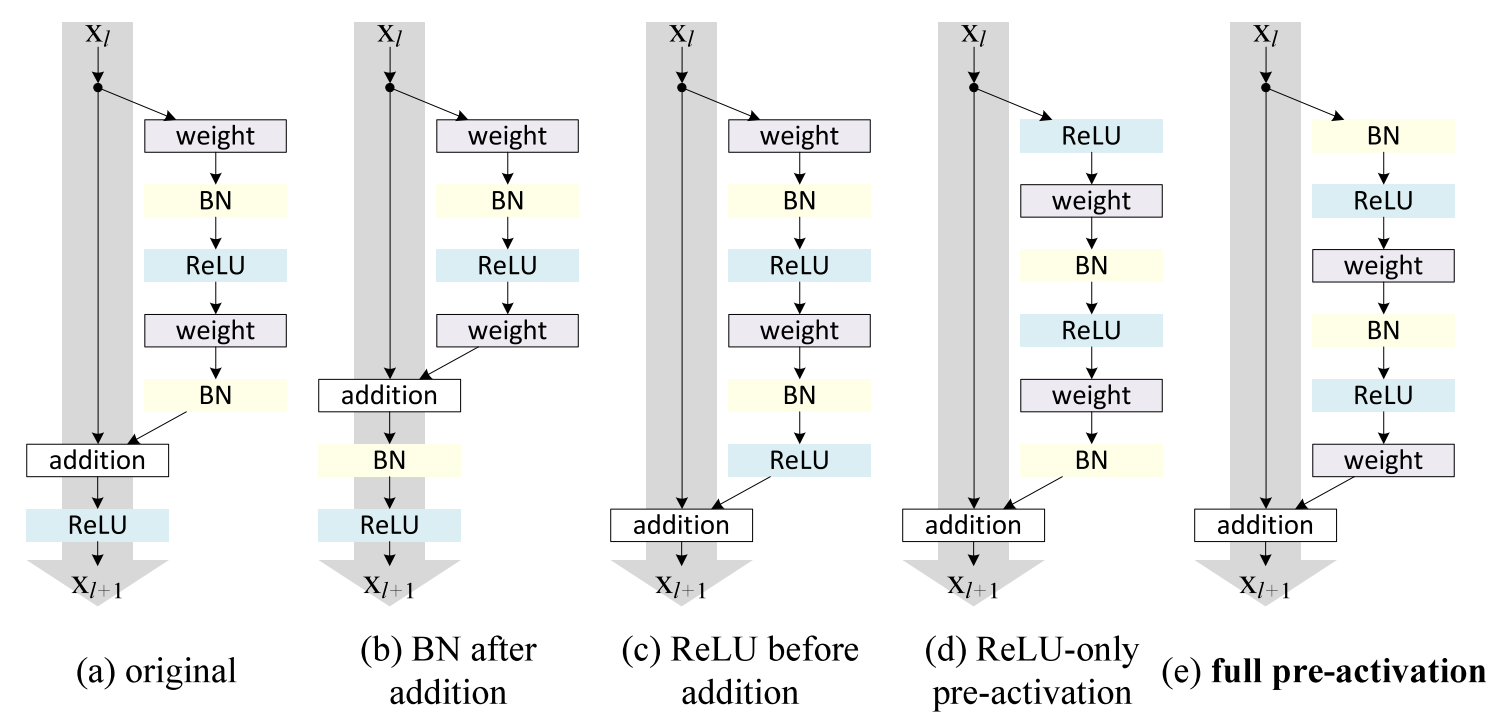
1. **Inception V3 Architecture**



1. **ResNet Architecture**



Variant of residual blocks:



**Appendix B: Binary Classification Results**

1. **Downward Dog versus Warrior II**

| Model | Accuracy | R-squared | RMSE | AUC |
| --- | --- | --- | --- | --- |
| Logistic Regression | 0.88 | 0.67 | 0.29 | 0.95 |
| Random Forests | 0.90 | 0.78 | 0.24 | 0.96 |
| SVM | 0.92 | 0.75 | 0.25 | 0.97 |

1. **Warrior II versus Warrior III**

| Model | Accuracy | R-squared | RMSE | AUC |
| --- | --- | --- | --- | --- |
| Logistic Regression | 0.88 | 0.52 | 0.34 | 0.96 |
| Random Forest | 0.92 | 0.65 | 0.30 | 0.98 |
| SVM | 0.91 | 0.48 | 0.38 | 0.96 |

**Appendix C: Multiclass Classification Preliminary Results**

|  |  |  |
| --- | --- | --- |
| Classifier | Logloss | Duration |
| KNeighborsClassifier | 5.251636 | 0.282790 |
| SVC | 0.944437 | 58.787883 |
| DecisionTreeClassifier | 1.368153 | 0.056639 |
| RandomForestClassifier | 1.024786 | 0.029444 |
| MLPClassifier | 1.454588 | 0.510601 |
| AdaBoostClassifier | 1.518190 | 0.845249 |
| GaussianNB | 3.404759 | 0.011140 |
| QuadraticDiscriminantAnalysis | 2.413621 | 0.019485 |
| LogisticRegression | 0.941902 | 1.168093 |

1. In order for a system to provide feedback for the learner, the system needs to be able to recognize which pose is being attempted. That first step is the focus of this capstone project. [↑](#footnote-ref-1)
2. In other words, using this application to learn another sport would require a complete retraining of the entire system. [↑](#footnote-ref-2)
3. Inverted poses are poses where the head is below the hips. [↑](#footnote-ref-3)
4. Initially, we used the English pose names based on the hypothesis that most images would be tagged in English rather than Sanskrit. This hypothesis proved incorrect. The English scraping results contained too much noise to be useful. [↑](#footnote-ref-4)
5. The specific net architectures used can be seen in Appendix A. [↑](#footnote-ref-5)