Application of Deep Learning to Terrain Classification with Bayesian Optimization-Based Hyperparameter Search

Nathan Starliper
Email: nstarli@ncsu.edu Rafael Luiz da Silva
Email: rdasilv2@ncsu.edu

***Abstract*—Visual terrain identification can significantly enhance the efficiency of autonomous control mode switching in a lower limb robotic prosthesis. In this work we evaluate various deep learning architectures for image classification and dimensionality reduction and compare with standard machine learning methods such as random forest (RF) and Principal Component Analysis (PCA) for dimensionality reduction. We also asses the impact of data augmentation methods in the performance of classification. Finally, we explore methods for hyperparameter selection and their impact on performance.**

I. Motivation

Current state of the art research in robotic lower limb prostheses have shown these devices enable amputees to efficiently perform activities such as rising from chairs and climbing stairs or ramps. [1]–[3]. The transition between distinct locomotor tasks (i.e.: stair ascent/descent, ground walking, etc) requires prostheses with environmental and human kinematic awareness. Specific techniques have been proposed to ensure that images with reasonable quality are able to be captured for terrain identification even with the shaking support inevitable when capturing images from a moving leg [4]. Deep learning techniques, in particular Convolutional Neural Networks (CNN) have shown extraordinary success in image classification. These techniques could be explored for the specific application of terrain identification in a lower limb prosthesis. However, deep neural networks have been shown to be particularly susceptible to over-fitting and input perturbations especially with a small dataset. Also, data collection can be very difficult and restrict the size of the training set. For this reason we investigate the use of data augmentation to improve performance. Further, CNNs have been found to be very sensitive to hyperparameter choice and can require the tuning of many different hyperparameters. Therefore, we will investigate techniques for efficient hyperparameter search using bayesian optimization.

II. Dataset

The dataset consists of 566 images of asphalt, 648 images of carpet, 1041 images of cobblestone, 984 images of grass, 168 images of mulch and 585 of tile (see Figure 1 for samples), which reflect the terrains which the subjects walked over.

We rescale the images to 64x64 grayscale images. The reason for this is 2-fold. Firstly, this will eventually be implemented in an embedded platform where computational capacity is limited and computational speed is critical in real time decision making. Secondly, specifically for using grayscale images, we hope that this will increase generalization performance of the model with out-of-model data. We are currently training on a relatively limited dataset with a high degree of homogeneity within classes. For example, all of the sidewalks are the same material found around campus and all of the carpets and tile are found in campus buildings. However, in the real world, one would expect to encounter many different colors of tile, carpet, and sidewalks. By removing the color feature, we hope to build more robust models.

We split the data into 80% training and 20% testing. We reserve 20% of the training data to be used as validation data during iterative training of the networks.

Fig. 1. Samples of the different type of terrains observed from the camera on the leg. (A) Asphalt, (B) Carpet, (C) Cobblestone, (D) Grass, (E) Mulch and (F) Tile.

III. Methodology

A. Data Augmentation

Data augmentation can be a powerful tool for extending a training data set. Much research has shown the potential of data augmentation to improve performance in deep neural networks (DNNs) especially for image classification and object recognition [5]. Due to the relatively small size of our current data set, over-fitting is inevitable which will lead to drastically reduced generalization. We believe that data augmentation will assist in synthetically extending our data set and preventing over-fitting. When augmenting data, it is important to only apply data transformations that make sense to the application and do not have the potential to cause a change in class, therefore hand-designed data transformations are required. In

our application we decided on a few linear transformations that we could realistically expect to encounter. These include:

- Rotation range (-20, 20)°
- Horizontal flip
- Vertical shift range ±0.15
- Horizontal shift range ±0.15

These transformations allow us to mimic real world situations. For example someone may be walking along a slanted surface or may simply angle their leg during a side step or stepping while turning. The flips and shifts are used to make sure that geometric and noisy features such as the lines of the brick or the glares on the tiles respectively are evenly distributed around the areas of the images. We randomly apply our augmentations online on each training batch. This allows for us to essentially create an infinite flow of training images. We also experimented with other transformations such as image zoom and brightness, however these degraded performance. Impact of our hand designed augmentation method is discussed in the results section.

B. CNN for Classification

We use a deep CNN for classification of the various terrains from grayscale images. We select parameters of our architecture using bayesian optimization based hyperparameter tuning as detailed in section III.D. The final structure used for classification is shown in figure 2. The structure is straightforward. We use two sets of double-stacked convolutional layers as smaller stacked convolutions have been shown to be more efficient than single larger filter size convolutions. Each individual convolutional layer is followed by nonlinear activation, the activation functions for each layer are shown in table I. After each convolutional stack we perform batch normalization and then utilize max pooling for regularization. We then flatten the output of the second maxpooling layer to a fully connected dense layer. This fully connected layer applies dropout for further regularization. The final hyperparameters of the model were obtained through bayesian optimization search, as shown in table I.

Our CNN uses the Adam optimizer [6] for stochastic gradient descent. Many studies show the effectiveness of Adam and Ruder et. al [7] recommend Adam as the best overall choice compared to many different modern techniques. Adam also provided a solution to an issue we were facing in our initial testing of our network, namely instability. This instability is shown in fig 3. Here we can see that the validation loss is extremely unstable and never converges like training does. This issue was resolved by implementing learning rate decay with the Adam optimizer. We use a learning rate decay of 0.001 which is the widely used recommendation in most studies (including the previously cited ones). Results implementing learning rate decay can be seen in the final training accuracy and loss curves figures 17 and 18 respectively.

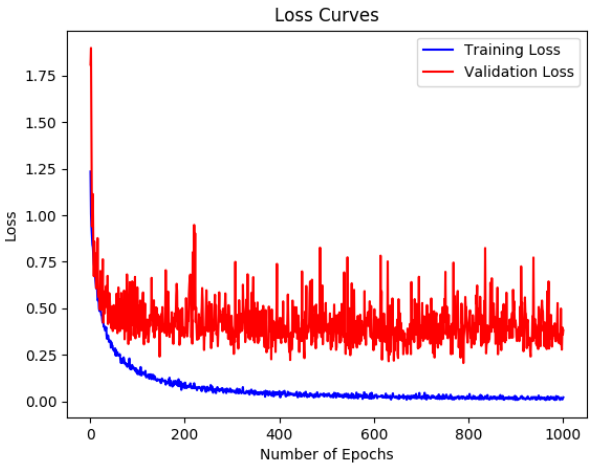

Fig. 3. Initial training of CNN without learning rate decay.

TABLE I
FINAL HYPERPARAMETERS AND STRUCTURE OF THE CNN. FOR EACH HYPERPARAMETER THE SEARCH SPACE AND BEST FOUND VALUE ARE SHOWN.

Hyperparameter	**Search Space**	**Best**
Conv1 Num Filters	32, 64, 128, 256	64
Conv2 Num Filters	32, 64, 128, 256	256
Conv3 Num Filters	32, 64, 128, 256	64
Conv4 Num filters	32, 64, 128, 256	32
Conv1 Filter Size	2, 3, 5	5
Conv2 Filter Size	2, 3, 5	3
Conv3 Filter Size	2, 3, 5	3
Conv4 Filter Size	2, 3, 5	3
Conv1 Activation	relu, tanh, elu	elu
Conv2 Activation	relu, tanh, elu	relu
Conv3 Activation	relu, tanh, elu	relu
Conv4 Activation	relu, tanh, elu	relu
Max Pool 1 Size	2, 4, 8	2
Max Pool 2 Size	2, 4, 8	8
Dense Num Neurons	64, 128, 256, 512	256
Dense Activation	relu, tanh, elu	elu
Dropout	uniform{0, 1}	0.02076
Learning Rate	uniform{0.00001, 0.01}	0.007563
Batch Size	16, 32, 64	16

C. Variational Autoencoder for Efficient Dimensionality Reduction

Dimensionality reduction methods such as principal component analysis (PCA) and Kernel based PCA (KPCA) will be compared with a Variational Autoencoder (VAE) [8] for the same purpose. The output of these methods will then be applied to an RF classifier [9] and performance will be quantified. As it will be explained in the next section, the aforementioned algorithms will be explored with an automatic hyperparameter search method in order to find the best model given a defined space of parameter values to search on.

VAE has been proposed as an probabilistic approach for dimensionality reduction using neural networks to encode a higher dimension input to a lower dimensional one denominated as latent vector or latent space. Analogously, the decoder structure of the VAE performs a lossy recontruction

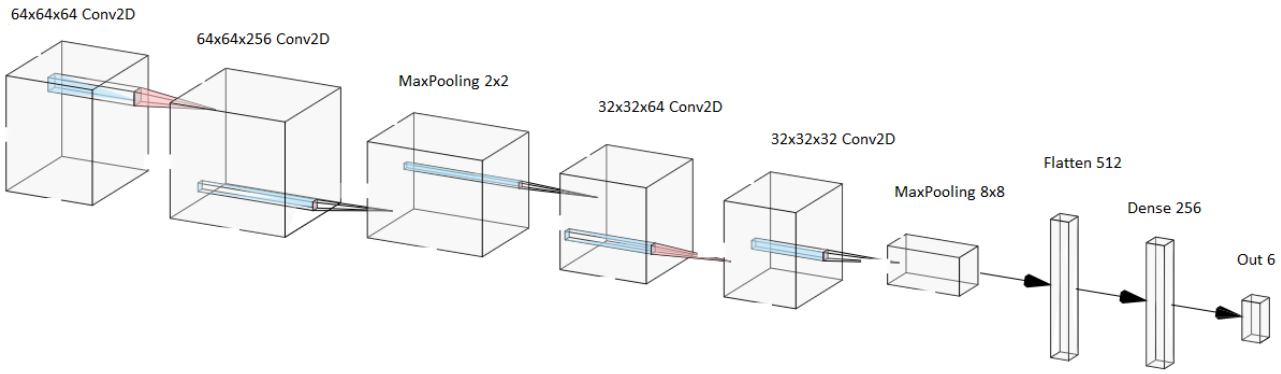

Fig. 2. The structure of our CNN model with dimensions.

of the input from the encoded data. In order to evaluate its effectiveness with terrain images, the PCA method will be used as a benchmark since it has been applied for multiple types of application for dimensionality reduction. Moreover, KPCA will be part of the study since it is a nonlinear dimensionality reduction method, what makes it a natural candidate for comparison against VAE.

The VAE evaluation is done by training (80% of total available data) this model with a given space of parameters over 400 epochs and then using the latent vector to be classified with the RF algorithm. The evaluation of PCA is done by fitting the training data to its eigenspace and then projecting the training data onto this space for the training of RF classifier, the same is done with the test data. Initially, it is found the explained variance of PCA to find how many principal components explains up to 90% of the data as shown in figure 4, which is nearly 80. This procedure is repeated 400

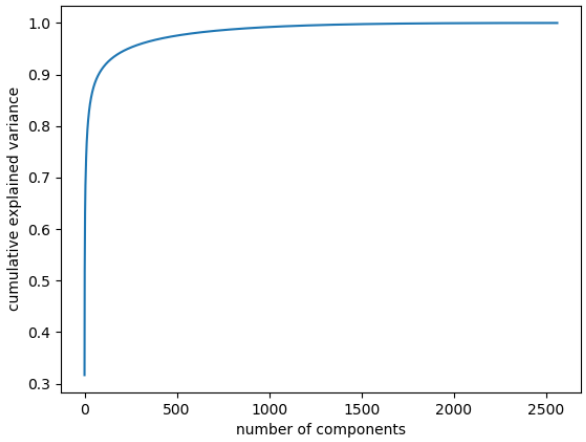

Fig. 4. PCA: Explained variance according to the number of principal components utilized.

times and at each time, the split between training and testing is kept but the samples are shuffled to provide more realistic model fitting. In the sequence, the RF classifier is trained and tested with the compressed data respective to each set. The chosen number of components is the one that provides the best classification accuracy for the RF model. With KPCA, the split between training and testing is kept static since the space of hyperparameters is wide, thus we preferred to do not shuffle the samples at each evaluation of the model. The different evaluations of KPCA are performed by the Bayes hyperparameter search method [10] that will be presented next which makes 400 evaluations. In the same way as it is done for PCA, the compressed vector is applied to the RF model for classification and quality assessment.

TABLE II
FINAL HYPERPARAMETERS AND STRUCTURE OF THE VAE. FOR EACH HYPERPARAMETER THE SEARCH SPACE AND BEST FOUND VALUE ARE SHOWN.

Hyperparameter	Search Space	Best
batch_size	16, 32, 64, 128, 256	32
intermediate_dim	32, 64, 128, 256, 512	128
latent_dim	2, 4, 16, 32, 64, 256	32
activation functions	relu, tanh, elu	elu
learn_rate	uniform(0.00001, 0.005)	0.002661
lr_decay	uniform(0.0001, 0.005)	0.002172

TABLE III
FINAL NUMBER OF COMPONENTS OF PCA. THE SEARCH SPACE AND BEST FOUND VALUE ARE SHOWN.

Hyperparameter	Search Space	Best
n_components	range(72,79)	79

TABLE IV
FINAL HYPERPARAMETERS AND STRUCTURE OF THE KPCA. FOR EACH HYPERPARAMETER THE SEARCH SPACE AND BEST FOUND VALUE ARE SHOWN.

Hyperparameter	Search Space	Best
kernel	linear, poly, rbf, sigmoid, cosine,	poly
gamma	uniform(0.001, n_features)	2034.993
degree	randint(0, 10)	3
n_components	randint(0, 128)	14
coef0	uniform(0, 1)	0.392872

D. Bayesian Optimization Hyperparameter Tuning

There has been much research into automatic hyper parameter tuning methods. Recently bayesian optimization based

approaches have shown promising results for finding the best subset of hyperparameters in the shortest amount of evaluations, and in many cases finding the combination of parameters that gives a lower loss that traditional methods do not find at all [10]. We use the Tree Parzen Estimates (TPE) algorithm which is shown in [10] to give superior results to grid and random search. Bayesian based approaches are particularly useful in our case with limited computational resources and the sheer number of hyperparamters we are tuning. We utilize Bayes optimization to not only find the ideal hyperparameters but also to select the best combinations of these hyperparameters and architecture configurations (such as number of convolutional filters, size of filters, size of max pooling, number of fully connected neurons, kernels, etc). The full list of the hyperparameters and configuration parameters tuned as well as their search space and optimal value are listed in table I for the CNN model, and tables II, III and IV for VAE, PCA and KPCA respectively. For the CNN, without even taking into account the continuous distribution of our dropout and learning rates, this list gives us 544,195,584 different possible combinations of parameters. Obviously an exhaustive grid search is nearly impossible and random search would require a large amount of iterations to give a decent probability of finding the optimum values.

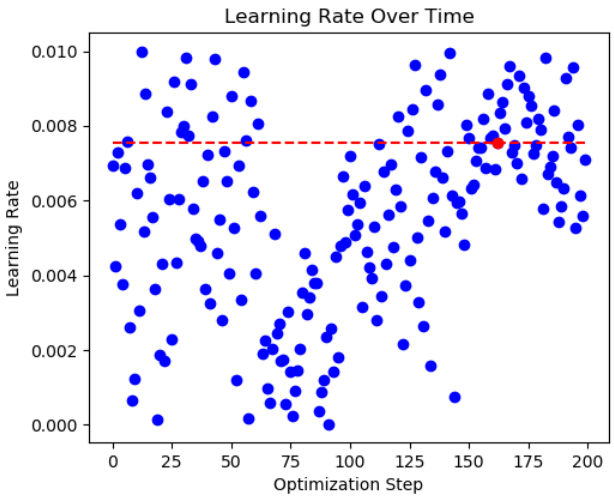

Fig. 5. CNN: Learning rate chosen at each optimization step. Optimum value shown in red.

The optimization runs as follows. The optimization runs for a preset amount of steps (200 for the CNN, 400 for the VAE). Each optimization step includes training the network for a specified number of epochs (100 for CNN, 400 for VAE, which we determined by observing training bahavior with some initial training runs) on a chosen subset of hyperparameters. The first optimization steps are mostly random while taking into account the initial distributions of the parameter search spaces (in our case these are all uniform so the initial searching is uniformly random). We return the validation accuracy during training which we use as the optimization metric. After each optimization step, the algorithm updates it's probablistic search space with the new information obtained. In this way the

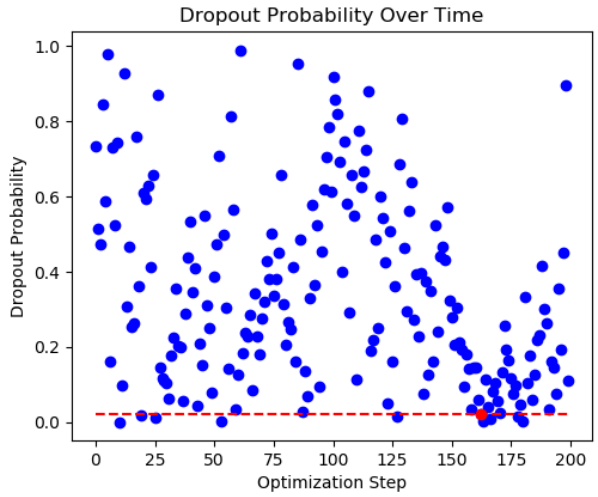

Fig. 6. CNN: Dropout probability chosen at each optimization step. Optimum value shown in red.

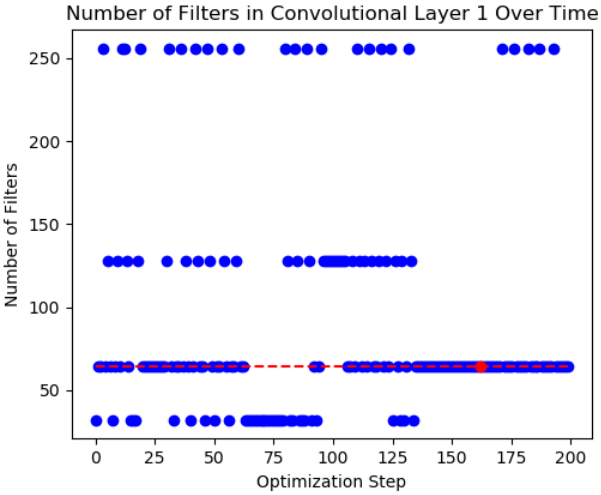

Fig. 7. CNN: Number of filters in the conv2D layer 1 chosen at each optimization step. Optimum value shown in red.

algorithm is able to efficiently narrow it's search to the parameters that are most likely to return a high validation accuracy. These can be seen in figures, 6, 5, 7, 8 for the CNN, which shows the chosen value for learning rate, dropout probability, number of filters in first convolutional layer, and number of neurons in dense layer at each optimization step respectively. For the dimensionality reduction methods, figures 9 to 12 show the same pattern of search occurring for VAE and figure 13 and 14 represent PCA and KPCA respectively. The dashed red line represents the optimum value for the parameter found over the entire optimization. The dot on the dashed line shows the step at which this value was found. We can clearly observe that the algorithm is converging around a certain value as it advances into later steps. You will also notice that the algorithm continues to search the wider space (although less frequently). This is because the algorithm tries to balance exploration vs exploitation in order to avoid getting

stuck in local minima (for lack of a better term).

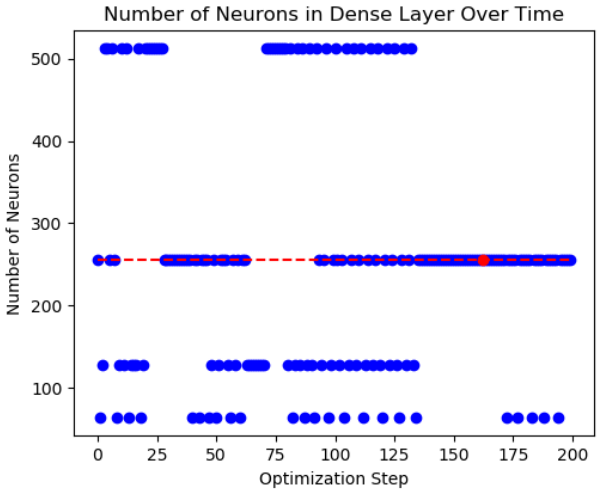

Fig. 8. CNN: Number of neurons in the fully connected layer chosen at each optimization step. Optimum value shown in red.

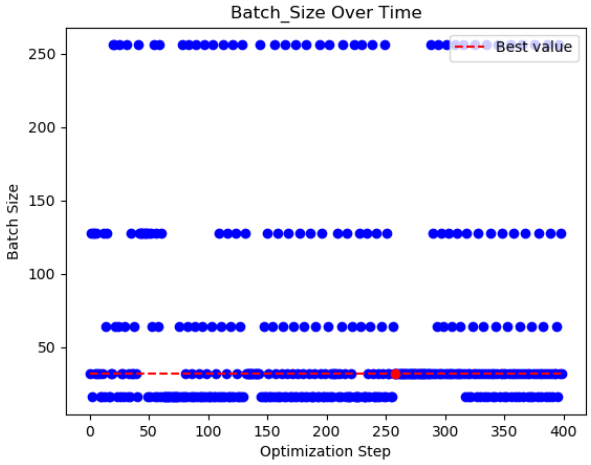

Fig. 9. VAE: Batch size search for each optimization step. Optimum value shown in red.

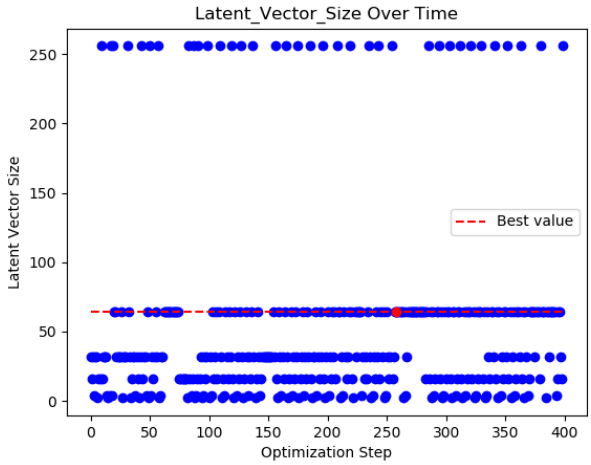

Fig. 10. VAE: Latent vector size search for each optimization step. Optimum value shown in red.

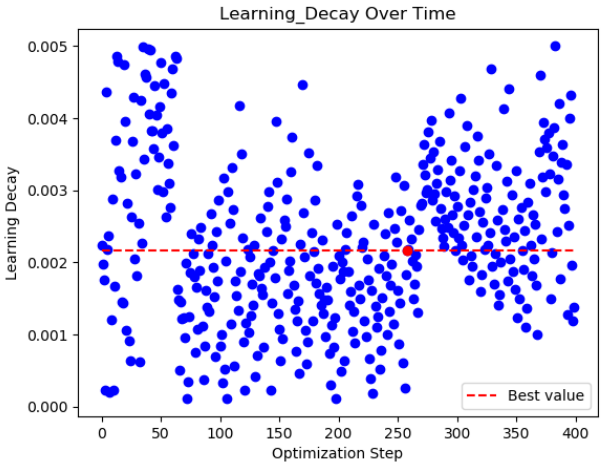

Fig. 11. VAE: Learning Decay values search for each optimization step. Optimum value shown in red.

IV. Results

A. Image Classification with CNN

We first explore the affect of using data augmentation to extend our data set. Figures 15 and 16 show the training and validation accuracy and loss curves respectively. Figures 17 and 18 show the same curves using the augmented data set. The curves without data augmentation show clear over-fitting. Training accuracy and loss each reach 100 and 0 and stay there while the validation accuracy and loss peak much lower and actually begin to degrade as time passes which is a typical characteristic of over fitting the training data. Data augmentation reduces the gap between training and validation accuracy and loss and improves performance on the validation set by up to 7%! The effect of data augmentation on testing performance is shown is shown in table V. Confusion matrices can give us a better idea of what specific issues the network is having. Figures 19 and 20 show the confusion matrix and normalized confusion matrix respectively. One thing that stands out right away is that our "grass" class is imbalanced which seems to be correlating with decreased prediction accuracy. This tells us that we could potentially increase performance by balancing our classes and acquiring more images of grass to better train that class. Final overall testing of our model is shown in table V.

TABLE V
Final testing accuracy of model with and without data augmentation

Testing Set Up	Loss	Accuracy
Final Test with Data Augmentation	0.094	0.965
Final Test without Data Augmentation	1.056	0.882

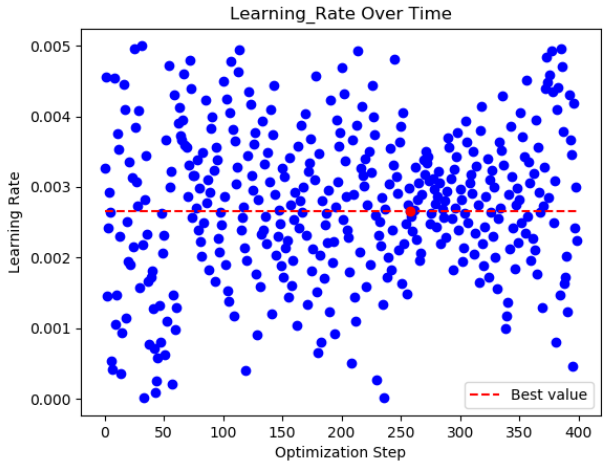

Fig. 12. VAE: Learning Rate values search for each optimization step. Optimum value shown in red.

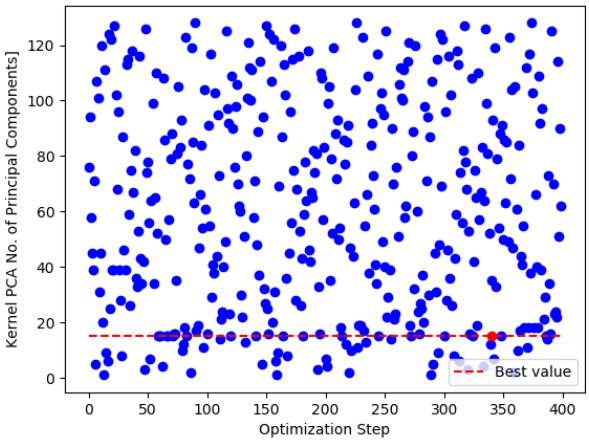

Fig. 14. KPCA: Number of components chosen at each optimization step. Optimum value shown in red.

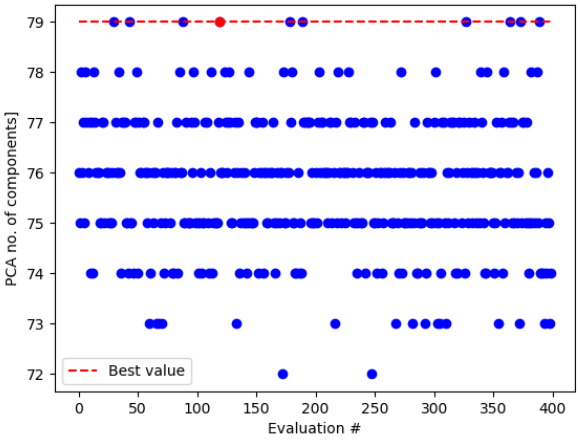

Fig. 13. PCA: Number of components chosen at each optimimzation step. Best value shown in red.

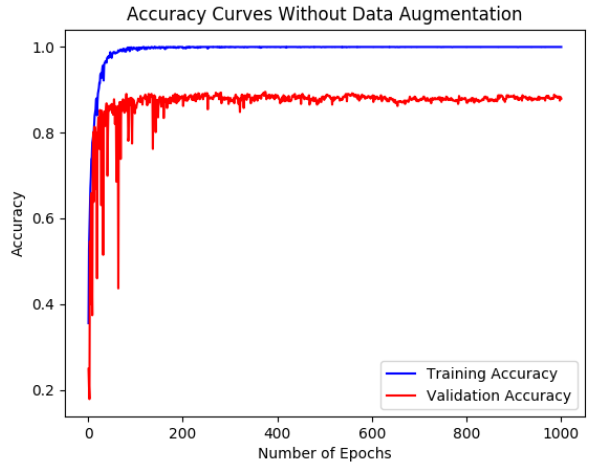

Fig. 15. Training and validation accuracy curves without data augmentation.

B. VAE

As one can see in table VI, out of the three models employed, VAE performs slightly better with the cost of using a larger latent vector to encode the data. Although KPCA has a worse performance compared to the others, it needs the least number of elements to represent the data with a penalty of around 2% in accuracy. What could drive the decision of which model to choose would be really the components that would have the classification accuracy as the input for control of the leg for example.

After finding the set of hyperparameters for the VAE, it was evaluated how stable the compression would be in terms of classification accuracy performed by the RF. Therefore, the model found was ran through 5000 epochs and the loss result is shown in figure 21. One can see that after around 300 epochs the loss in the trainning is stable, indicating that the model could be evaluated with the Bayesian optimization with around 300 epochs and enables further searches with the hyperparameter search. Although this evaluation was not performed due to time constraints, it was performed an experiment to see how stable the obtained accuracy with RF classification is with 250 epochs of training and 50 different runs. The result is shown in figure 22. This figure shows that the model performance is unstable, indicating that further epochs of training would be needed. Therefore, care needs to be taken since if one evaluates figure 21 only, it could be concluded that the model is stable after 300 epochs whereas several different runs with further epochs are actually needed to enable such statement or the refuse of it.

TABLE VI
MODEL COMPARISON FOR DIMENSIONALITY REDUCTION

Algorithm	Dimensionality Reduction	Accuracy[%]
VAE	32	83.6
PCA	79	81.7
KPCA	14	80.8

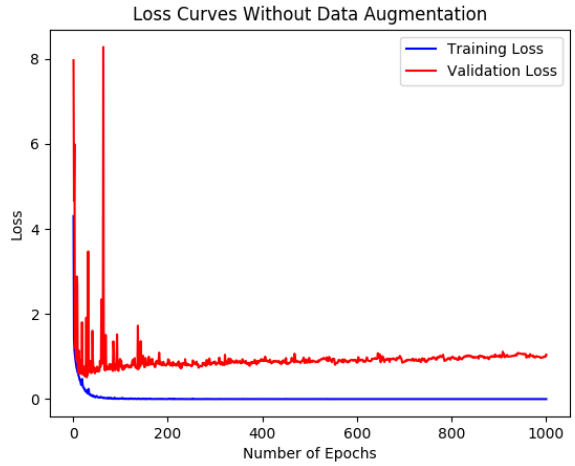

Fig. 16. Training and validation loss curves without data augmentation.

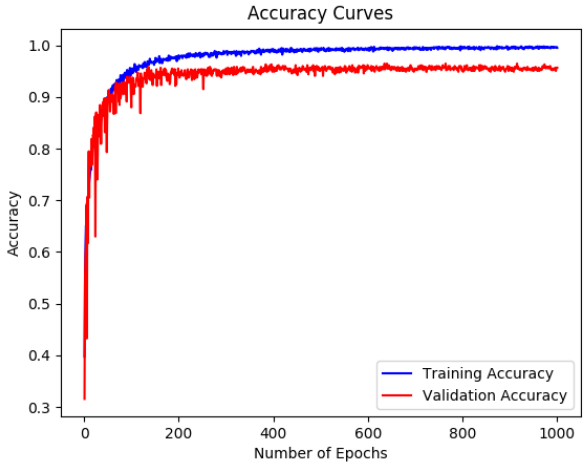

Fig. 17. Training and validation accuracy curves with data augmentation.

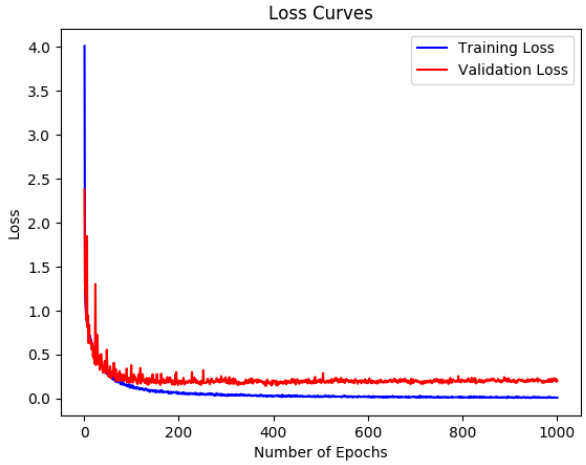

Fig. 18. Training and validation loss curves with data augmentation.

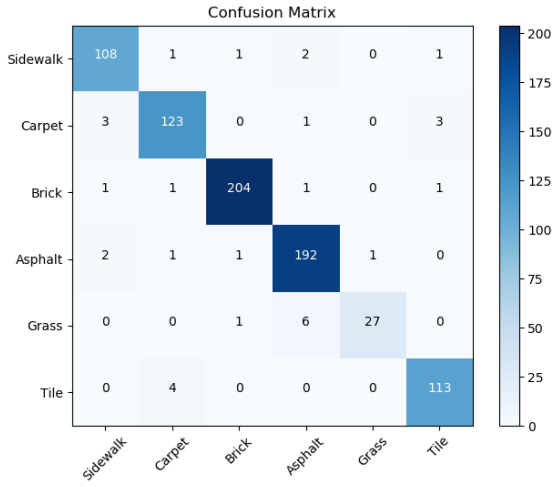

Fig. 19. CNN Confusion Matrix.

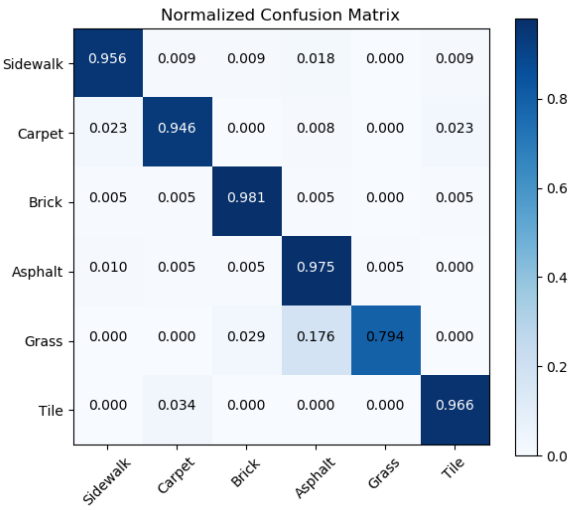

Fig. 20. CNN Normalized Confusion Matrix.

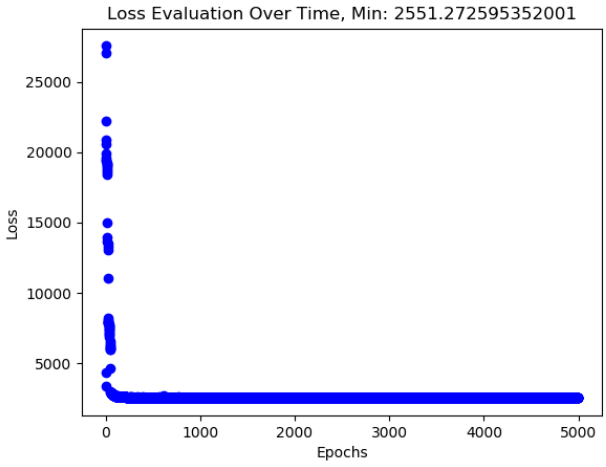

Fig. 21. VAE: Loss evaluation during training as a measure to see how stable the found hyperparameters are.

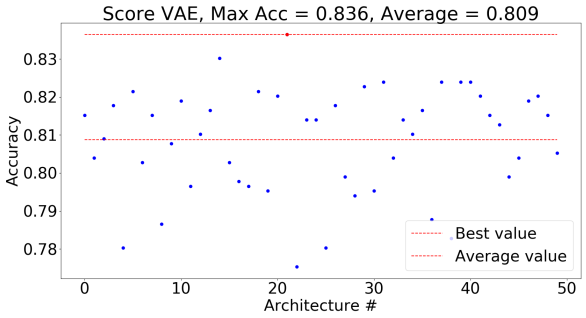

Fig. 22. VAE: Accuracy evaluation throught different executions with the model, 50 different executions were performed where each of them had the model trained for 250 epochs.

V. Appendix

A. How to run the codes

1) The Data: We have provided all of the data necessary in pickle format. The images are stored in *features.pkl* and the corresponding labels are stored in *labels.pkl*. Make sure that these two files are in the same directory as all scripts. To see the preprocessing code we used to create the pickle files please refer to the script *load_images.py*.

2) CNN:

- Ensure *labels.pkl* and *features.pkl* are in the same directory as scripts
- *cnn.py* contains all of the functions used by the run scripts. If hyperparameters are re-optimized, they need to be updated in this code.
- *run_cnn_datagen.py* is the main code to run our final CNN model with online data augmentation. Training parameters need to be updated if parameters are re-optimized.
- *run_cnn_datagen.py* is the script to run our model without data augmentation. Note that this is just for analysis and should not be used to test our final model.
- Run *run_hyperopt_cnn.py* to perform Bayesian hyperparameter optimization if needed.
- *visualizations.py* creates the plots and charts used in this report.

3) VAE/Dimensionality reduction:

- Make sure that the generate files *labels.pkl* and *features.pkl* are in the same directory
- run *main_vae.py* for hyperparameter search on VAE
- copy the found hyperparameters and replace them in the file *test_vae.py*, make sure you set number of epochs to 400
- run *main_kpca.py* for hyperparameter search on KPCA
- run *main_pca.py* for evaluation of PCA
- in order to get the graphs make sure all previous steps were ran then you are able to execute *plot_results.py*

REFERENCES

[1] F. Sup, H. Varol, J. Mitchell, T. Withrow, and G. M., "Preliminary evaluations of a self-contained anthropomorphic transfemoral prosthesis," *IEEE ASME Trans Mechatron*, vol. 14, no. 6, pp. 667–676, 2009.

[2] S. Au, M. Berniker, and H. Herr, "Powered ankle-foot prosthesis to assist level-ground and stair-descent gaits," *Neural networks : the official journal of the International Neural Network Society*, vol. 21, pp. 654–66, 06 2008.

[3] J. K. Hitt, R. Bellman, M. Holgate, T. G. Sugar, and K. W. Hollander, "The sparky (spring ankle with regenerative kinetics) project: Design and analysis of a robotic transtibial prosthesis with regenerative kinetics," *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 1, no. DETC2007-34512, pp. 1587–1596, 2007.

[4] J. P. Diaz, R. L. da Silva, B. Zhong, H. H. Huang, and E. Lobaton, "Visual terrain identification and surface inclination estimation for improving human locomotion with a lower-limb prosthetic," *IEEE Engineering in Medicine and Biology (EMBC)*, 2018.

[5] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *CoRR*, vol. abs/1712.04621, 2017.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[7] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[8] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2013.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[10] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *ICML*, 2013.