Project F3 - Gait Recognition

Mahmoud Abdelkhalek (maabdelk), Jeffrey Barahona (jabaraho), Dinesh Bhosale (dbhosal)

I. Generative Model

A. Latent Variable Models

Latent variable models are statistical models that relate a set of observable variables to a set of latent variables. For latent variable models, we have a set of functions $f(z;\theta)$ parameterized by $\theta \in \Theta$ where $f : Z \times \Theta \rightarrow X$ and z is a random variable in Z. We are maximizing the probability of each X in the training set according to the function

$$P(X) = \int P(X|z;\theta)P(z)dz \tag{1}$$

The main takeaway here is that if we can train a model that can produce a set of training samples, then we can generate similar samples using our random variable z [1]. This is the general approach we are taking for creating synthetic data with our model.

B. Variational Autoencoder

This section aims to provide a conceptual basis of what a variational autoencoder (VAE) is. A VAE is a powerful generative model having applications as diverse as from generating fake human faces to producing purely synthetic music. VAE is used to generate an output that looks similar to the input data, where we can alter, explore the variations on the data in a specific direction. It is similar to traditional autoencoder in the sense that it has an encoder and a decoder stage. However, the purpose of the VAE is to model the distribution of a latent vector z as a stochastic process with pdf $P(z)$. From Equation 1, the term $P(X|z;\theta) = N(X|f(z;\theta), \sigma^2 I)$ for our latent variable model where $f(z;\theta)$ is our mean and we have a scalar multiple σ^2 for the covariance matrix. The latent spaces are continuous which allow easy random sampling and interpolation. There is a problem, however, the equation 1 can be intractable, so we approach this from a different angle in the VAE. Instead we use an alternative distribution Q to encode our z latent space. This also enables us to use ELBO (the variational lower bound) to approximate $P(X)$. We can also reparametrize z into a linear form to allow for gradient descent as shown in figure 1.

C. Data Preprocessing

To prepare our data, we used a sample wise mean centered and a min-max normalization to bound the data to values in the range $[0, 1]$. This turned out to be ideal for our case since other preprocessing methods actually reduced performance on the test set and our validation set.

Fig. 1. Reparameterization

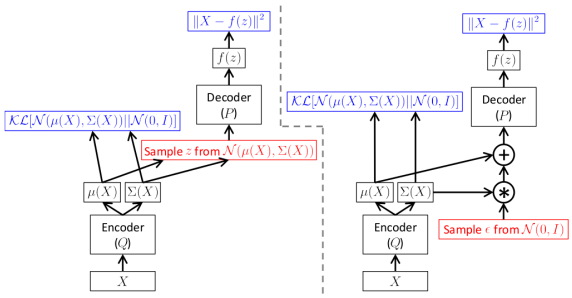
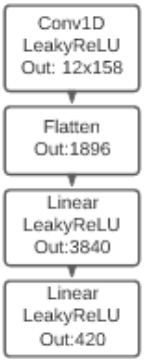


Fig. 2. Encoder

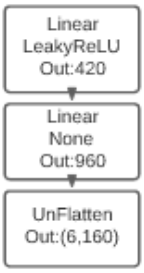


1) Data Augmentation: To augment data for our classifier, we resampled from the VAE to take advantage of how the latent space returns vectors in a Gaussian distributed neighborhood of latent space. Resampling the raw this way generated similar enough samples such that we could assume they had the same label.

D. Model Architecture

For our generative model, we used a variational autoencoder in PyTorch with a convolutional encoder and a standard neural net for the decoder as demonstrated by figures 2 and 3. The first layer of the encoder is a convolutional layer with 12 output channels and LeakyReLU as the activation. We use LeakyReLU to ameliorate the issue of vanishing gradients by preventing the gradients from going to zero [2]. This is used as a feature extractor for the following linear layers. The decoder is composed of two linear layers with no activations on the last layer. The loss function for this VAE was derived from the loss function defined by [3]

Fig. 3. Decoder



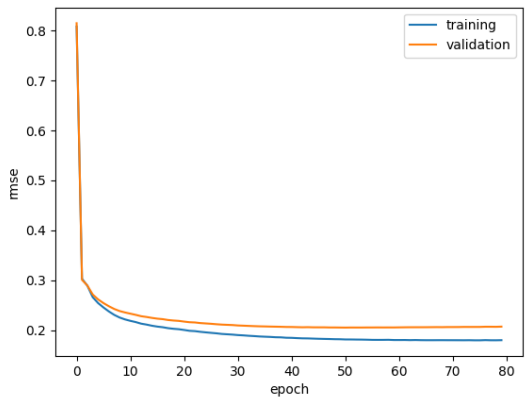
E. Methodology

We trained the model with the RMSprop optimizer for 40 epochs, a batch size of 86, a latent space size of 256, and a learning rate of 1×10^{-3}. Furthermore, we used gradient clipping to reduce the effects of vanishing and exploding gradients. We also used KL divergence annealing to improve model performance [4]. For the KL Divergence annealing. Effectively, KL divergence is a regularizing term for ensuring that the model's latent space represents the true distribution well. The modified loss is depicted in the equation below.

$$ELBO = RMSE + \beta D_{KL}(P||Q)$$

We started with a 0 coefficient, effectively removing the KL divergence term in our loss to allow it to learn a good code set, then slowly increasing it to a maximum of 5×10^{-6} across 40 epochs. Since the aim is to minimize RMSE on the test set, we opted to track validation RMSE rather than the combined RMSE-KL Divergence curve for training. That curve is depicted in figure 4

Fig. 4. Combined Loss Curve

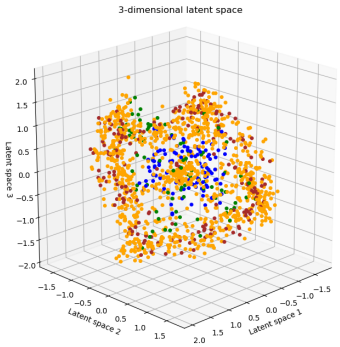


II. Generative Model Evaluation

A. Visualization

In figure 5, the latent space of a model with 3 dimensions is shown. Each color represents a different class. We can see a few different clusters. The 3d latent space represents a horse saddle like structure.

Fig. 5. 3D Latent Space Visualization



orange-0, green-1, blue-2, brown-3

B. Data Generation

A variational auto-encoder with latent space dimension equal to 256 is trained to generate synthetic data (reconstructions). RMSE (root mean squared error) is calculated between the reconstructed signal generated by the VAE and the validation set i.e. original signal (160 samples) to evaluate model generation capability. A minimum RMSE of 0.0811 is observed between one signal (160 samples) and the reconstructions. Figure 6 and 7 shows the reconstructions and original signal for the respective RMSE.

III. Predictive Model

We generated 256-dimensional latent vectors using the VAE for the labelled training and unlabelled testing data. A Hidden Markov Model (HMM) was then trained using the labelled training data to predict a sequence of gait positions given the observed latent vectors. A HMM is a probabilistic graphical model, specifically a Bayesian network, that models the joint distribution of an observed sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_K\}$ and a latent (hidden) sequence $\mathbf{Z} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_K\}$ of random vectors, denoted $p(\mathbf{X}, \mathbf{Z})$. In a HMM, the following conditional independencies are assumed:

$$\mathbf{x}_i \perp \{\mathbf{x}_{i-1}, \mathbf{z}_{i-1}, ..., \mathbf{x}_1, \mathbf{z}_1\} | \mathbf{z}_i$$
$$\mathbf{z}_i \perp \{\mathbf{x}_{i-1}, \mathbf{z}_{i-2}, ..., \mathbf{x}_1, \mathbf{z}_1\} | \mathbf{z}_{i-1}$$

For $i \in \{1, 2, ..., K\}$. A typical HMM is shown in figure 8. For gait identification, the observed sequence $\mathbf{X}$ represents the 256-dimensional latent vectors that are generated by the VAE, and the latent sequence $\mathbf{Z}$ represents the sequence of gait positions associated with each latent vector. Therefore, to predict the most likely gait position given the observed

Fig. 6. Accelerometer Measurements (Reconstructed vs Original)

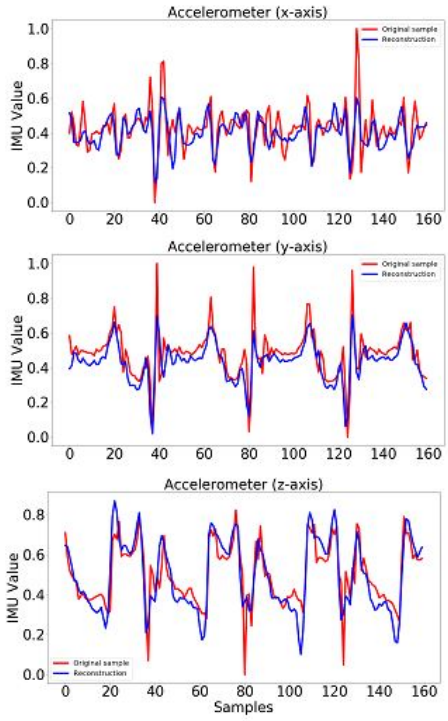
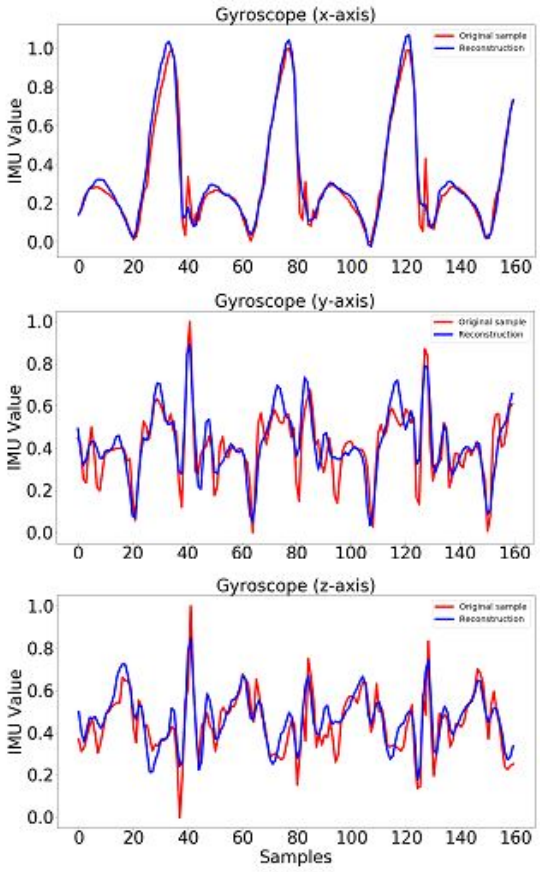


Fig. 7. Gyroscope Measurements (Reconstructed vs Original)



sequence $\mathbf{X}$ of latent vectors, it is necessary to find the latent sequence $\mathbf{Z}^*$ that maximizes $p(\mathbf{Z}|\mathbf{X})$, which is equivalent to finding the latent sequence $\mathbf{Z}^*$ that maximizes the joint distribution $p(\mathbf{X}, \mathbf{Z})$ [5], or:

$$\mathbf{Z}^* = \underset{\mathbf{Z}}{\operatorname{argmax}}\, p(\mathbf{X}, \mathbf{Z})$$

$\mathbf{Z}^*$ can be determined using the Viterbi algorithm, also known as the max-product message passing algorithm, which is very similar to the sum-product message passing (belief propagation) algorithm, but with summations replaced with max terms. However, since the joint distribution $p(\mathbf{X}, \mathbf{Z})$ is parameterized by the state transition probabilities $\{p(\mathbf{z}_2|\mathbf{z}_1), p(\mathbf{z}_3|\mathbf{z}_2), ..., p(\mathbf{z}_K|\mathbf{z}_{K-1})\}$, the emission probabilities $\{p(\mathbf{x}_1|\mathbf{z}_1), p(\mathbf{x}_2|\mathbf{z}_2), ..., p(\mathbf{x}_K|\mathbf{z}_K)\}$, and the initial probability distribution $p(\mathbf{z}_1)$, as shown in figure 8, then it is first necessary to estimate these probability distributions.

While it is usually the case that the latent sequence $\mathbf{Z}$ cannot be observed, and therefore the likelihood function $p(\mathbf{X}; \theta)$, where θ represents the aforementioned probability distributions, needs to be computed by marginalizing the joint distribution $p(\mathbf{X}, \mathbf{Z})$ using the forward-backward algorithm, in the case of gait identification, there exists a set of observations of the latent sequence $\mathbf{Z}$, which are the gait positions associated with each latent vector generated by the VAE.

Therefore, the joint distribution p(X,Z;θ) can be estimated directly. This is done in MATLAB using the `hmmestimate` function. Once this joint distribution is estimated, the Viterbi algorithm can be used to find the most likely sequence of gait positions given a new sequence of observed latent vectors. This is also done in MATLAB using the `hmmviterbi` function. Therefore, the un-labelled testing set will be used to generate the most likely sequence of gait positions, which will be submitted as our predictions.

However, since the latent vectors generated by the VAE exist in $\mathbb{R}^{256}$, then there will exist an infinite number of possible values for the observed sequence $\mathbf{X}$. Instead, these latent vectors need to be quantized, which was performed using K-means clustering. K-means clustering can be used to summarize a set of vectors by clustering nearby vectors together, then using the center of the cluster of these vectors as a code for the rest of the vectors [6]. In MATLAB, K-means clustering was achieved using the `kmeans` function. More precisely, the training data consisting of 256-dimensional latent vectors was encoded into 2048 clusters. These encoded vectors were then used to train the HMM model.

IV. Predictive Model Evaluation

As shown in table I, the HMM performs better for class 1 (going down the stairs) and class 2 (going up the stairs). It is difficult for the HMM to reliably learn the characteristics of class 0 (walking on solid ground) and class 3 (walking on grass) due to somewhat similar and closer points in the latent

Fig. 8. An example of a HMM

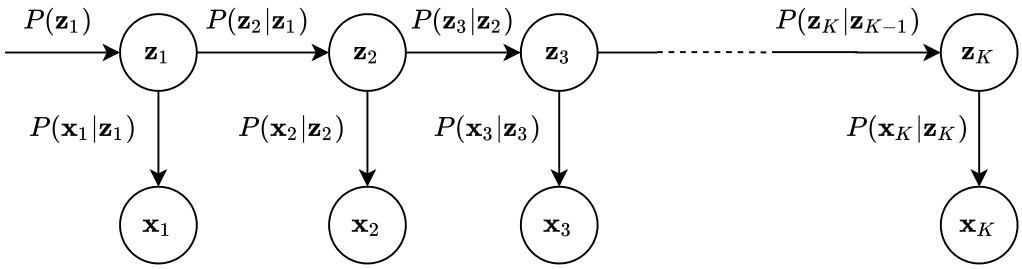


TABLE I
HMM PERFORMANCE METRICS

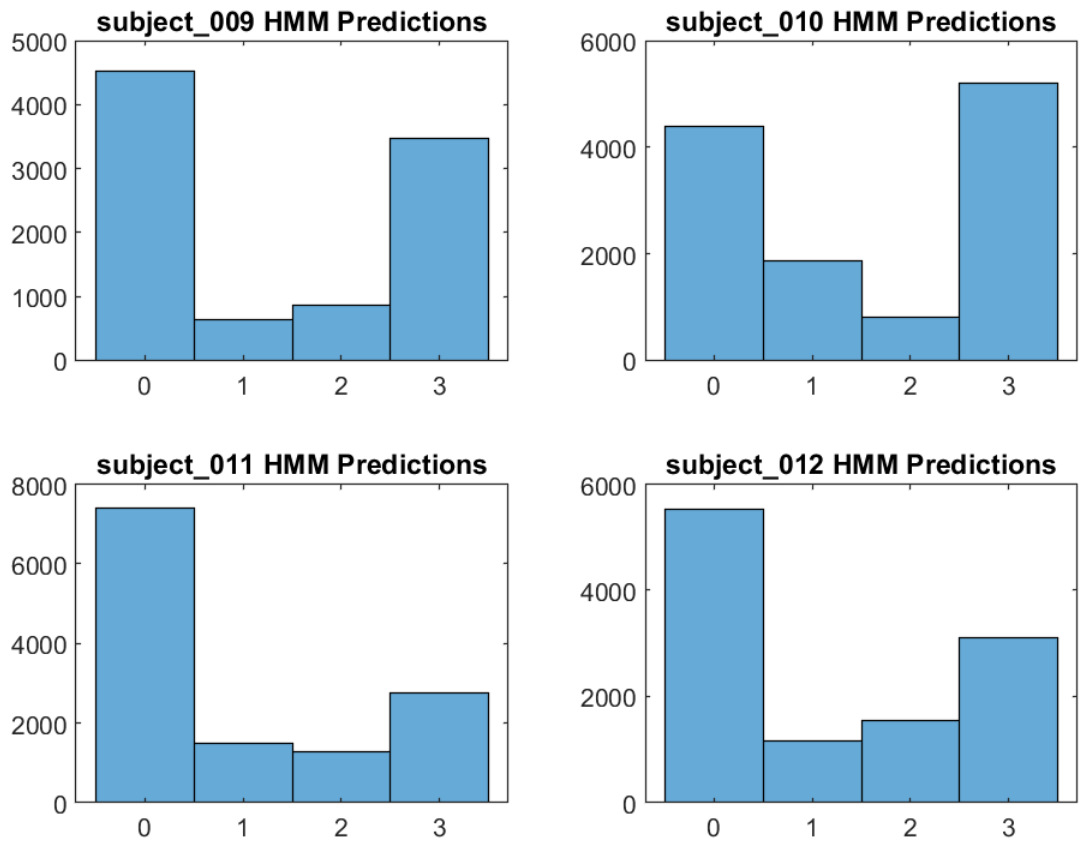
Class No.	**Precision**	**Recall**	F_1 **Score**	**Accuracy**
0	0.5593	0.4597	0.5046	0.5593
1	0.8091	0.8653	0.8363	0.8091
2	0.8717	0.8737	0.8727	0.8717
3	0.5791	0.6384	0.6073	0.5791
Average	0.7048	0.7093	0.7052	0.7048

space. These performance metrics were obtained from a 5-fold cross-validation experiment. This is confirmed by the bias towards predicting classes 0 and 3, as shown in the histogram in figure 9. Table II shows the confusion matrix computed from the true values and the predictions. Again, these were the best predictions based on a 5-fold cross-validation. The bias towards classes 0 and 3 is also clear here.

TABLE II
CONFUSION MATRIX FOR HMM

True \ Predicted	0	1	2	3
0	957	207	157	761
1	105	1831	34	146
2	127	63	1903	85
3	522	162	89	1365

Fig. 9. Histogram of predictions on the unlabelled test set for each subject



REFERENCES

[1] C. Doersch, "Tutorial on variational autoencoders," *arXiv:1606.05908 [cs, stat]*, Aug. 13, 2016.

[2] L. Trottier, P. Giguère, and B. Chaib-draa, "Parametric exponential linear unit for deep convolutional neural networks," *arXiv:1605.09332 [cs]*, Jan. 10, 2018.

[3] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv:1312.6114 [cs, stat]*, May 1, 2014.

[4] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin, "Cyclical annealing schedule: A simple approach to mitigating KL vanishing," *arXiv:1903.10145 [cs, stat]*, Jun. 10, 2019.

[5] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989, Conference Name: Proceedings of the IEEE.

[6] R. Python. (). "K-means clustering in python: A practical guide real python," [Online]. Available: https://realpython.com/k-means-clustering-python/ (visited on 11/22/2020).