GRAVITY SPY: GLITCH CLASSIFICATION FOR LIGO USING DEEP TRANSFER LEARNING

Group 89: Arash Behravesh, Piyoosh Srivastava, and Michael Wibowo

University of California San Diego, La Jolla, CA 92093-0238

***Index Terms*—** transfer learning, image classification, spectrogram, gravitational waves

1. INTRODUCTION

LIGO stands for "Laser Interferometer Gravitational-wave Observatory". It is a large scale physics experiment and observatory which seeks to detect cosmic gravitational waves. There are two observatories located at Livingston, Louisiana, and Hanford, Washington. The first gravitational wave was observed in 2015, due to the merging of two black holes a billion light-years away from the earth. Gravitational waves have a measurable effect on the space they travel by shrinking and stretching distances. LIGO detectors use laser light to measure these effects. If the ultra-precise sensors of LIGO detect any such shrinking or stretching, they create a signal. However, due to the ultra-sensitive nature of these sensors, sometimes noises are generated due to instrumental or environmental factors. By detecting and classifying these noises or glitches, their sources can be identified. This helps LIGO in removing potential sources and to better curate their gravitational wave data. In this project, we used a transfer learning approach to classify these glitches. In transfer learning, a base model is trained on a general dataset, then the learned features are extracted or transferred to the target model which is then trained on the specific dataset. We also employed a convolutional autoencoder to compare results.

2. RELATED WORK

Many attempts have been made to use machine learning algorithms for glitch classification. An algorithm based on distances calculated using Longest Common Subsequence (LCSS) is presented in [1], for unsupervised classification of glitch waveforms. In [2], authors have used Principal Component Analysis for Transients (PCAT), Principal Component LALInference Burst (PC-LIB) and Wavelet Detection Filter with Machine Learning (WDF-ML) for classifying glitches. They found that "WDF-ML can classify lower frequency transients than the other two methods. PC-LIB is better able to classify longer duration transients due to its longer analysis window. PCAT can classify new types of transients as soon as they appear in the data and thus provide transient waveforms for PC-LIB's signal models" [2]. A deep neural-network-based classifier is used in [3] to differentiate between gravitational-wave bursts and glitches. A Self Organizing map (SOM) neural-network architecture is used for unsupervised classification of glitches. In [4], the authors did supervised classification of glitches by utilizing a variety of standard machine learning classification models: logistic regression, support vector machines (linear and kernel), and a custom convolutional neural network. These are combined into an ensemble model, which achieves 98.21 percent accuracy. Most similar to our work is [5], which applies transfer learning using standard state-of-the-art deep CNN classifiers (e.g., Inception, ResNet, VGG) pre-trained on ImageNet, with the best models achieving 98.84 percent accuracy for supervised classification of glitches. We tried to replicate this approach by using several other state-of-the-art CNN architectures.

3. DATASET AND FEATURES

The dataset is provided by Coughlin [6] as a set of pre-generated spectrograms with 22 classes, and 7966 with a split of 5587 training, 1200 validation, and 1179 test images. They are initially provided as a matplotlib image including the axes, thus we have to crop to just the plot area. The dataset also provides multiple window lengths for the spectrograms around the glitch event of 0.5, 1, 2, 4 seconds as a potential hyperparameter. According to Bahaadini et al. [4], the dataset was constructed by using the Omicron algorithm, which uses sine-Gaussian wavelets to search for excess noise with high SNR and within the target frequency range of 10 to 2048 Hz.

The split provided by the dataset was used; it should be noted that this distribution is different than those used by our reference papers [4] and [5], since our version is a slightly smaller curated version (compared to both papers), and not randomly split (compared to the second paper). Furthermore, the images must be resized to the input dimensions of each network, which is usually the ImageNet size (224 x 224), except for Inception v3 which uses 299 x 299. We also normalized the inputs to the ImageNet mean and standard deviation, in accordance with the regular ImageNet preprocessing procedure. Although this dataset is rather small, we were unable to use image augmentation to improve our results, since it does not make sense to scale or rotate spectrograms, and we

did not have the original waveforms to generate synthesized spectrograms either.

4. METHODS

We attempted to replicate the results of George et al. [5], as well as test other former and current state-of-the-art image classification models.

For classifications, these neural networks use the softmax activation function as the output of the final layer. Since the output of the network is a probability of whether the input belongs to each class, they must sum to one. This is represented in the equation

$$S(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{1}$$

where the sum in the denominator ensures the values are normalized to 1. However, to ensure numerical stability, the values are often offset by the maximum z_i, which prevents overflow by making the exponent always semi-negative:

$$S(\mathbf{z})_i = S(\mathbf{z} - max(z_i)) \tag{2}$$

4.1. Models

The following models were used in the evaluation: AlexNet [7], VGG16 [8], ResNet-50 [9], Inception v3 [10], ResNext-50 [11], MobileNet v2 [12], EfficientNet-b4 [13]. For conciseness we will elaborate only on the more recent networks, as we assume most have already encountered the older models.

MobileNet v2 iterates on v1, which implemented depthwise separable convolutional layers. The model introduces the idea of bottleneck layers, which are small 1x1 convolutional layers inserted around the depthwise convolutional layers. These help to establish intermediate features within the models, as they reduce the dimensionality of the representation. Residual connections, initially introduced by ResNet [9], are also inserted connecting the bottleneck layers, as *inverted residuals* since they connect the smaller rather than larger layers. [12]

EfficientNet, released in 2019, is the current state-of-the-art image classifier, which employs compound scaling. This involves training an initial base network, which is found through a neural architecture search with AutoML and aims for maximizing accuracy and FLOPs. Then, this model is scaled through a further grid search of increasing resolutions, depth (number of layers), and width (number of channels). Eight variants were found with increasing size (B0-B8); we chose B4 since it was the largest version that allowed a batch size of 32 to keep in line with the other models. [13]

For each of these models, we obtained the pretrained ImageNet [14] weights through PyTorch [15], and replaced the final fully connected layer with one that had 22 outputs, for classification on our dataset. We enabled retraining for all weights, so that the models could be fully fine-tuned to our task.

4.2. Convolutional AutoEncoder

We propose another solution to tackle the problem of glitch classification using an autoencoder with convolutional learnable filters. This architecture takes advantage of a two modules; an encoder module and a decoder module. The encoder module compresses the data by forcing it through a bottleneck. This layer is known as the latent space which contains necessary information to reconstruct the information on the end of the decoder module.

This autoencoder architecture encodes the input data through 8 convolutional layers, and reduces the dimensionality of the input data by performing max-pooling after every convolution in the encoder module. To reconstruct the data, the network takes in compressed data from the encoder module and upsamples it using bilinear upsampling so that it can be fed into convolutonal layers yet again. The reason behind bilinear upsampling was its better performance compared to the nearest neighbour method. At the output of the network, we expect to see a reconstruction of the input batch with minimal error. For the objective function, we use Mean Squared Error loss to measure error which can be represented as the following equation:

$$Loss = \frac{1}{n}\sum_{i=0}^{i=n}(x_i - t_i)^2 \tag{3}$$

This is to measure the reconstruction quality, but here we're classifying, so we feed the network into a 2 layer fully connected module and use cross entropy as the loss to measure classification performance. Cross entropy loss is defined as the following function where t is the target label and y is the output label:

$$L(t, y) = -1/n\sum_{i=0}^{i=n} t_i log(y_i) + (1 - t_i) log(1 - y_i) \tag{4}$$

Another popular method to use in the decoder module for CNN autoencoders is using Transpose-Convolutional layers, which removes the need of using upsampling by trying to mimic deconvolution and changing the output size. Note that deconvolution is not achieved by Transpose-convolution and it only tries to increase the dimensions of the input size, or in other words, interpolate. This method is more expensive than performing upsampling followed by convolution, so we chose to not use transpose convolution since our network had around 1.1 million learnable parameters.

Model	# Param. (mil)	Test accuracy
AlexNet	57.1	0.964
VGG16	134	0.957
Resnet-50	23.6	0.969
Inception v3	24.4	0.973
ResNext-50	23.0	0.964
MobileNet v2	2.3	0.975
EfficientNet-b4	17.6	0.975

Table 1. Number of parameters and test accuracies for each image classifier

5. RESULTS

5.1. Transfer Learning Classification

To train our dataset, we used the Adam [16] optimizer with categorical cross entropy and a 0.001 learning rate (the default), which was decayed by 0.1 every 7 epochs to ensure convergence. However, this did not appear to matter as the accuracy seemed to saturate at similar epochs with or without the decay. The models were trained for 25 epochs on an NVIDIA GTX 2080 GPU with a batch size of 32. We first did a batch size search in the beginning with Inception v3, using sizes of 4, 8, 16, and 32. As shown in the results of Figure 2, there is no significant difference attained by increasing or decreasing batch size, so we used the largest size to decrease training time. For evaluation, we chose to use basic accuracy as it the papers we referenced it also used it, and it is the best representation of our task of classifying anomalies.

The best performing networks appeared to be MobileNet v2 and EfficientNet, but all of the test accuracies were fairly close to each other.

The validation and test accuracies are shown in Figure 1 and Table 1. The test accuracies are fairly close together, implying that this problem does not necessarily need the advancements provided by each model, and/or that there isn't enough training data to fully make a difference. This can be shown by the surprisingly high results achieved by AlexNet, which was by far the oldest and most basic model tested. On the other hand, VGG16 did not do so well, possibly because of its large number of parameters, twice that of AlexNet. Since we had a small dataset, there wasn't enough data for it to adjust all of the parameters accordingly during fine-tuning. The validation accuracy plot in 1 demonstrates that not very many epochs are needed to achieve the maximum accuracy; in fact, only 7 are needed before it plateaus.

Figure 4 shows an example of a misclassified spectrogram on the right, and an actual example of the predicted class on the left. They are quite similar, with blips around the start time (center of spectrogram) in the higher frequencies, so it is a prediction error which is quite understandable, and even possible for humans to make.

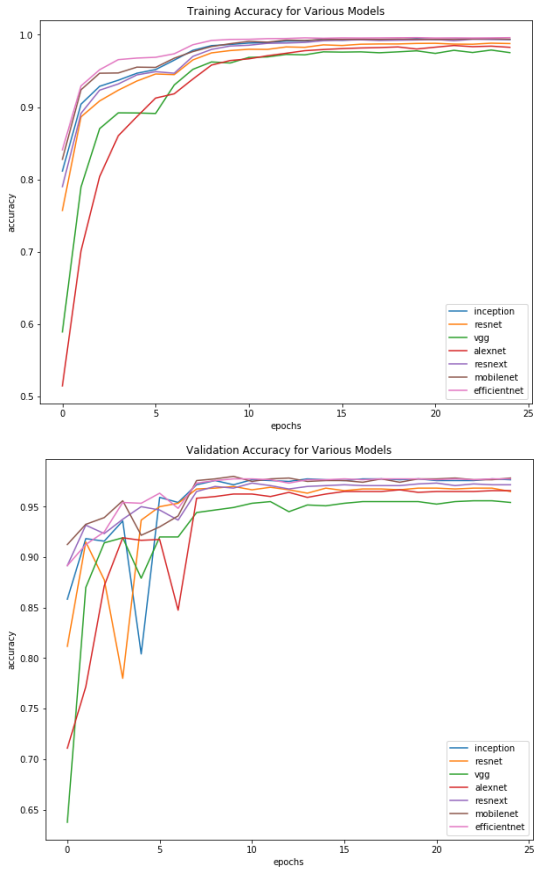

Fig. 1. Train and validation accuracy for each image classifier

5.2. CNN Autoencoder Results

This particular architecture did not perform well and achieved results as the following table. In autoencoders CrossEntropy loss is the best metric to assess the output of a multiclass classification problem. Test loss was obtained after the network was trained and experienced overfitting.

Model	# Param. (mil)	Valid Accuracy
CNN-AE	0.45	0.6

This result is not acceptable for a network with 0.45 million parameters and achieves poor performance not close to state of the art networks that we used with deep transfer learning. This is due to the nature of the dataset. Upon inspecting the dataset, we can see that the network doesn't have to learn tons of features, but the features are spread across the image, thus the network needs to be able to learn locality of the features. This network is only learning small patches on training data, and can't learn sufficient features with only 0.4 million features. This model was trained on the same dataset as before, and used batch sizes of 16. Larger batch sizes could not be handled due to GPU memory allocation and large input sizes of (16x224x224)]. Adam optimizer seemed to perform best with

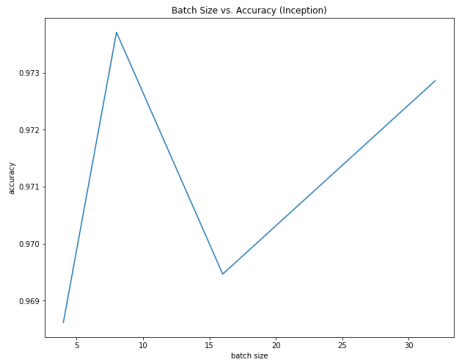

Fig. 2. Test accuracies for different batch sizes for Inception v3[10]

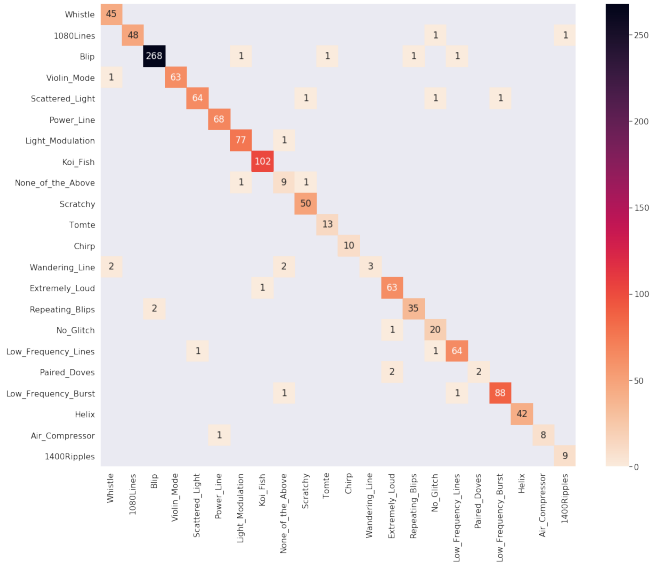

Fig. 3. Confusion matrix for Inception v3[10]

weight decay at 0.01 and learning rate of also 0.01. Weight decay proved to be a significant factor in the network converge in only 5 iterations on the training set. We picked cross-entropy loss to measure the performance of this particular application and it is the best metric since the task is classification. Once the network is trained, the output layer is cascaded into a fully connected module to map the output to a probability space. To map the fully connected layer to probability space we would need to use categorical cross entropy to produce class categorization and calculate error. It's fair to mention that with more iterations and more computational power we think this network is able to perform well.

6. CONCLUSION

To conclude our discussion, it is worthwhile to revisit the results from both experiments on deep transfer learning, our

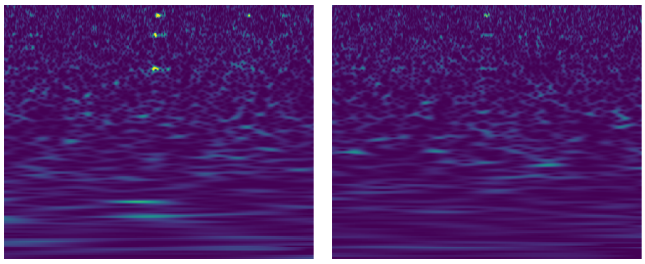

Fig. 4. An example of the "Violin Mode" class on the left, and a misclassified example on the right, actual class "1400 Ripples"

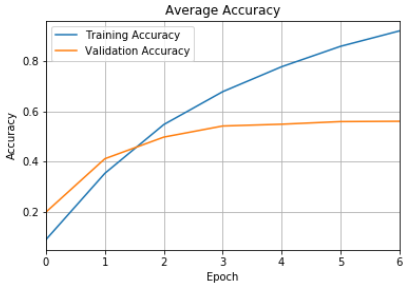

Fig. 5. Accuracy plot for autoencoder

custom autoencoder architecture, and explore a new idea for future work in this field. Deep transfer learning is fast and reliable using pre-trained state of the art network architectures, and they ensure high accuracy in classification, however, they could be too large to deploy for some specific applications. We introduced a CNN based AutoEncoder which has up to 50 times less parameters than the said networks, and are significantly faster to train from scratch. Both approaches yielded acceptable results and achieve good performance with minimum loss, but they exhibit limitations. Our classification models all achieved roughly the same accuracies, which indicate that the problem is not too difficult, since even AlexNet was able to achieve good results.

This particular task can also be approached in the time-domain and classify the tasks in time using attention networks which can be significantly faster and more accurate at classifying the glitches, and more importantly, be able to predict them and ignore their effect from the results. This involves training an attention network to focus on the signal of interest and ignore noise (irrelevant features in time). Attention networks are heavily used in NLP applications and seem to perform exceptionally well as Brown et al. [17] experimented with RNNs with Attention on Anomalies to detect anomalies in system log files. Another solution would be to yet again rely on RNNs in LSTM format to detect anomalies with a much simpler architecture.

7. REFERENCES

[1] S Mukherjee, R Obaid, and B Matkarimov. Classification of glitch waveforms in gravitational wave detector characterization. *Journal of Physics: Conference Series*, 243:012006, aug 2010.

[2] Jade Powell, Alejandro Torres-Forné, Ryan Lynch, Daniele Trifirò, Elena Cuoco, Marco Cavaglià, Ik Siong Heng, and José A Font. Classification methods for noise transients in advanced gravitational-wave detectors II: performance tests on advanced LIGO data. *Classical and Quantum Gravity*, 34(3):034002, jan 2017.

[3] Salvatore Rampone, Vincenzo Pierro, Luigi Troiano, and Innocenzo Pinto. Neural network aided glitch-burst discrimination and glitch classification. *International Journal of Modern Physics C*, 24:1350084, 11 2013.

[4] S. Bahaadini, V. Noroozi, N. Rohani, S. Coughlin, M. Zevin, J.R. Smith, V. Kalogera, and A. Katsaggelos. Machine learning for gravity spy: Glitch classification and dataset. *Information Sciences*, 444:172 – 186, 2018.

[5] Daniel George, Hongyu Shen, and E. A. Huerta. Classification and unsupervised clustering of ligo data with deep transfer learning. *Phys. Rev. D*, 97:101501, May 2018.

[6] Scott Coughlin. Updated gravity spy data set, November 2018.

[7] 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.

[8] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[10] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and ZB Wojna. Rethinking the inception architecture for computer vision. 06 2016.

[11] Saining Xie, Ross Girshick, Piotr Dollar, Z. Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. pages 5987–5995, 07 2017.

[12] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018.

[13] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[14] 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.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[17] Andy Brown, Aaron Tuor, Brian Hutchinson, and Nicole Nichols. Recurrent neural network attention mechanisms for interpretable system log anomaly detection. *ArXiv*, abs/1803.04967, 2018.

8. CONTRIBUTIONS

- Arash handled the autoencoder portion of the experiments, including design, training, and testing of the autoencoder.
- Piyoosh handled a portion of the transfer learning experiments.
- Michael handled the dataset imports into PyTorch, as well as the general training and testing infrastructure of the transfer learning experiments.

9. REPLIES TO CRITICAL REVIEWS

9.1. Group 6

- I would recommend adding a subtitle 'Glitch classification for LIGO spectrum', since 'gravity spy' seems to be a little bit general and confusing.

  Yes, our formal title is different but we forgot to put that on the slides. The current title derives from the Zooniverse project.

- The data part could be more explicit. Where, and how did you get them? Any preprocessing is done to eliminate noise, abnormal data, etc?

  We did not mention the preprocessing in detail since it was already done by the dataset provider [6]. All we have to work with are the generated spectrograms.

- It's interesting to compare the performance of different network structures (and to me they basically perform equally well and the differences seem minor: accuray range from 0.957-0.975). What is your own interpretation of each of them, which one of them physically or intuitively makes more sense? What is your conclusion after comparing these structures?

  This was mentioned in the report, but we think that the dataset is not large or difficult enough for the differences to show, since even AlexNet was able to achieve a decent score.

- Have you tried any new ideas that may improve these models? Or what do you think that can possibly yield a major improvement based on the given models?

  We were thinking of using image augmentation, but quickly scrapped it due to infeasibility since generating synthetic spectrograms is difficult, especially without the original waveforms.

- Is there a particular reason why you want to encode glitches? To me it seems unnecessary.

  Technically this is true, we only need a binary classifier to determine whether it is a glitch or not. However, it may be useful in determining and eliminating particular sources of these glitches to improve wave reception.

9.2. Group 44

- Group 89 uses the ImageNet pertained weights to initialize all the classification models. Considering the gravitational wave sensor dataset, it is not very similar to ImageNet. Though it might not harmful to initialize them with ImageNet weights, it might achieve better accuracy when you try to train from scratch.

  We actually found that training from scratch is harmful. There are not enough samples to fully train all of the parameters; this is why "transfer learning" or "fine-tuning" is so popular, especially with image classification. Although ImageNet does not cover spectrograms, the identification of visual features such as curves and shapes are still useful.

- In order to use VGG or ResNet, group 89 rescales their dataset input to 224x224 to suit the model input. It might cause some feature loss since the original signal data has a size of 800 x 600. You can consider doing some convolution to keep the spatial feature before simply cropping and scaling them into 224x224. (Also it's possible to adjust the input size for VGG or ResNet, but it might takes additional effort)

  Yes, this is possible. However, using smaller window sizes from the dataset achieves the same effect, and we found that it does not play a big role in the accuracy.

- For the accuracy plot, when at epoch 4, the validation accuracy is much lower than epoch 3 or epoch 5. This is not very normal and you might want to check the validation loss as well.

  This is an interesting phenomenon we observed; the validation loss did spike, but returned to its normal trajectory as before. We may need to perform additional investigation to see why this is the case, especially since it occurs for all models. Inception was the worst out of all of them though, highlighting this.

- For the auto-encoder part, group 89 seems to only do the training and reconstruct the training data. Inspired by our last GPU homework, it might be possible to detect anomaly using auto-encoder as well. You might give it a try.

  We tried doing that and learned that the model is not performing well with only 6 iterations. More iterations and more parameters might have helped the model perform better.

9.3. Group 75

- More explanations on how you process data. What is the original data, and how the gravitational waves become pictures? Because the CNN model you use are basically designed for pictures with 3 channels, you should explain how you adjust data to fit in the models.

  Again, we are only given the RGB spectrograms generated by the providers of the dataset [6].

- Because you do not design CNN models, maybe spend more time on presenting their structures.

  We tried to do this but realized that it would take too long. For future presentations we will try to allocate more time for this section.

- Explain how you adjust hyperparameters and model structures.

  We left learning rate to the default and batch size to 32 since it didn't affect accuracy (Fig. 2) and decreased training time. The learning rate decay didn't seem to do much, and 25 epochs was more than enough.

  The only changes to the model we did was to resize the final classification layer to fit the number of classes.

- What is the result of autoencoder? What does the final layer of encoder (I mean pictures of 3*3 grid in the last slide) tell us?

  The results are added in section 5.2. The learnt filter shown shows is representation of the latent space, however, it would have been best to show case a cluster of the latent space to better represent the compressed representation.

- Spend more time on analyzing you results, like why some models produce better results, how hyperparameters influence the results, etc.

  Point taken, we can do better on pacing our presentation.

- Autoencoder training has a bad loss-vs-epoch plot. Do you take any efforts improving it?

  We retrained the model with fully connected layers and produced a smoother graph, but the model experienced overfitting.