This Is My Title
Along With More Clarification

by

Anthony J La Barca

Submitted in Partial Fulfillment of the

Requirements for the Degree

Bachelor of Science in Physics and Astronomy

Supervised by Segev BenZvi, Ph.D.

Department of Physics and Astronomy
School of Arts and Sciences

University of Rochester

Rochester, New York

2023

For someone a long time ago, in a galaxy far, far away...

Acknowledgments

Consider adding acknowledgements.

Abstract

This is an abstract! Read the paper :)

Contents

List of Tables

List of Figures

1. Introduction

1.1 Citation styles

These are the different citation styles for author-year.

The standard `\cite` command produces the following output: Cybenko 1989.

The `\textcite` command produces the following output: Cybenko (1989).

The `\parencite` command produces the following output: (Cybenko 1989).

The `\footcite` command produces a footnote citation[1].

[1] Cybenko 1989.

2. Deep Learning Techniques

2.1 Multi-Layer Perceptrons

To aid in the classification of supernovae, deep learning techniques can provide flexibility and scalability that have been previously unattainable by manual methods. The simplest and most widely used architecture in deep learning are the feed-forward fully connected neural networks, or multi-layer perceptrons (MLPs) (Popescu et al. 2009). These networks are composed of series of "layers" of neurons, or nodes, that are "fully connected" to the previous layer (Figure 2.1). Given an input vector $\vec{x}^{(0)}$ with n values, an MLP applies a series of weights to each element, resulting in a linear transformation from $\vec{x}^{(0)}$ to $\vec{x}^{(1)}$, where $\vec{x}_{(1)}$ is a vector of length m. This transformation is then followed by a bias added to each element, and an activation function applied to each element. The transition from layer i to $i+1$ can be represented mathematically as

$$\vec{x}^{(i+1)} = \sigma(\mathbf{W}\vec{x}^{(i)} + \vec{b}^{(i+1)}), \tag{2.1}$$

where $\mathbf{W}$ is an $m \times n$ matrix of weights, $\vec{b}^{(i+1)}$ is a vector of length m representing the bias, and σ is the activation function. The activation function can be any function, but

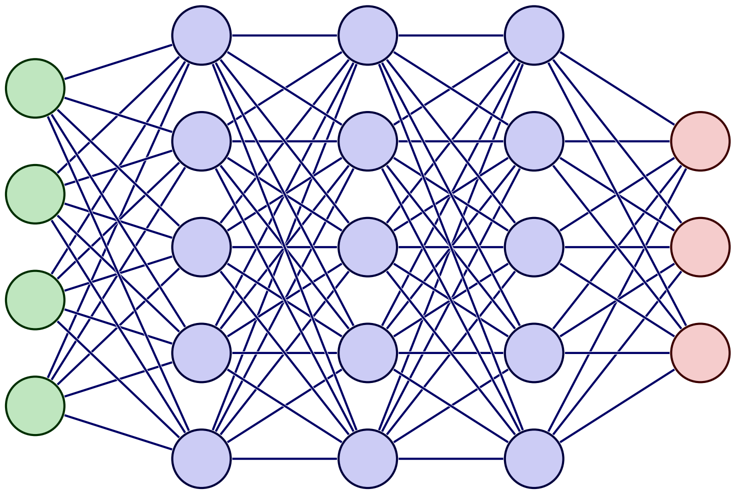


Figure 2.1: A simple MLP with an input dimension of 4 (green nodes), three hidden layers (blue nodes), and an output dimension of 3 (red nodes). Each node is connected to every node in the previous and next layer. The lines represent the weights, and each node has an associated bias. Figure adapted from Neutelings (2021).

traditionally they are chosen to be continuous, non-linear, monotonically increasing, and differentiable. The most common activation functions are the sigmoid function, $\sigma(x_i) = \frac{1}{1+e^{-x_i}}$, hyperbolic tangent, $\sigma(x_i) = \tanh(x_i)$, and the rectified linear unit, or ReLU, $\sigma(x_i) = \max(0, x_i)$, all of which are applied element-wise. The output layer of the network is a vector tailored for the particular purpose of the MLP. I.e. for classification, the output is a vector of normalized probabilities that the input data belongs to.

The MLP architecture, while simplistic, is very powerful. Cybenko (1989) showed that for any boolean function (i.e. 2-class classification), an MLP with a single hidden layer could approximate any function arbitrarily well; this is known as the Universal Approximation Theorem. The theorem is the basis for the success of MLPs in many fields, but it is also the reason that MLPs are not the best choice for all problems.

MLPs begin to show many complications during training large models, such as the exploding/vanishing gradient problem, a lack of invariance under translation, and extremely expensive backpropagation (Naskath, Sivakamasundari, and Begum 2022). These problems inhibit this architecture's abilities to scale to larger datasets and more complex problems, such as image recognition, time series analysis, and natural language processing. Since the mid-2000s, these problems have been addressed by the continued development of alternative architectures, such as convolutional neural networks (CNNs) and transformers.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) were originally developed to address the lack of invariance under translation in input data (Fukushima 1979). CNNs designed for classification are traditionally composed of a series of convolutional layers, usually followed by a pooling layer, and a fully connected layer. The convolutional layers are composed of a series of filters, which take a certain number of inputs from the previous layer and apply a kernel to them, resulting in a single output. These filters are across the entire input space, and are commonly referred to as feature maps. Inputs are traditionally padded to preserve the dimensionality of the original tensor size. Multiple filters are applied in each convolutional layer, resulting in a tensor of shape $N \times x$, where x is the shape of the original input (which can be one or multi-dimensional). To reduce the dimensionality of the output, a pooling layer is applied. The pooling layer takes a certain number of inputs from the previous layer,

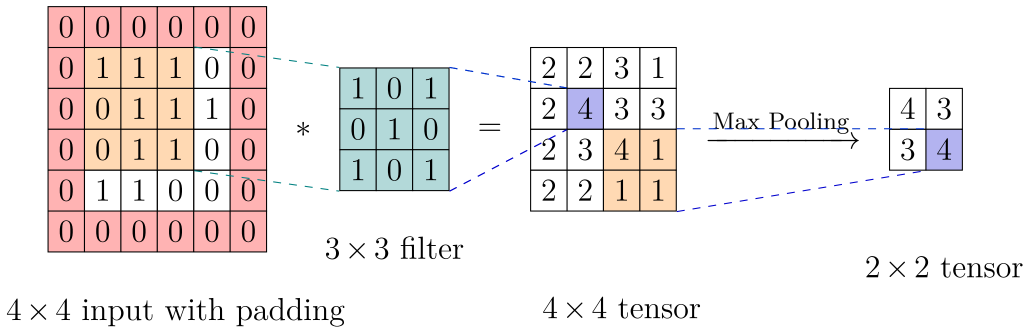


Figure 2.2: A convolutional filter and a 2×2 pooling applied to a 2D input. As the filter is passed over the input space, the kernel is applied. The input is padded with zeros to preserve dimensionality, which is then reduced by the max pooling. Figure adapted from Neutelings (2022).

and reduces them to a single output, similar to the convolutional filters, but without the kernel application. The most common pooling layer is the max-pooling layer, which takes the maximum value in each set as the reduced value. The process of passing an input through a convolutional filter, and then pooled is shown in Figure 2.2. These convolution and pooling layers are applied in series until the desired output dimensionality is reached.

From here, depending on the task, the output of the pooling layer can be fed into a fully connected layer, which is the same as the MLP architecture described in Section 2.1, or it can be fed into another convolutional layer. An MLP is usually applied for classification tasks, while another convolutional layer is applied for segmentation tasks. An example of a CNN architecture used for classification is shown in Figure 2.3.

This architecture was shown to be effective in image recognition tasks by LeCun, F. J. Huang, and Bottou (2004) and then popularized after the success of AlexNet `figures/Typical-CNN-architecture.png`

Figure 2.3: A convolutional network archetitcture designed for classification (adapted from Kumar 2022). Only one convolutional/pooling layer is shown, but there can be multiple depending on the application.

in the ImageNet challenge Krizhevsky, Sutskever, and Hinton (2012). AlexNet was the first of many CNN architectures popularized in the 2010s for vision classification, such as VGG (Simonyan and Zisserman 2015), ResNet(He et al. 2015), and DenseNet (G. Huang et al. 2016). This also spurred a sudden increase in deep networks designed for segmentation tasks as well, such as the FCN architecture by Shelhamer, Long, and Darrell (2016). In addition to vision tasks, it also found great succes in applications such as natural language processing (Kim 2014), engineering, and medicine (Kiranyaz et al. 2021).

2.2.1 CNNs on Spectroscopic Data in Astronomy

CNNs excel at identifying patterns throughout their input space, stemming from their spacial invariance. Theoretically, this should translate to spectral classification quite well. In fact, CNNs have been applied to spectroscopic data in the past, including on the DESI dataset. Parks et al. (2018) used a CNN to detect strong emission lines in DESI spectra with great success. * Talk about 1D CNN currently running, and figures/cnn/cnn_rocfull.png

figures/cnn/cnn_cmfull.png

Figure 2.4: CNN Diagnostics: ROC Curve (left) and Confusion Matrix (right)

preprocessing used *

This work provides a baseline for the use of CNNs on supernovae classification, **but leaves more to be desired.** Therefore, an alternate architecture was proposed by * Cite Eddies work * which augments the preprocessing of the spectra with a conversion to a 2D image. This 2D image was then fed to a more traditional vision CNN architecture (Appendix A.2). This architecture was shown to train quickly (less than 1 hour on a single GPU), but it was not able to achieve astonishingly high accuracy. Figure 4.10 shows the ROC curve and confusion matrix for the CNN trained on synthetic spectra.

figures/cnn/cnn_max_ypred.png

Figure 2.5: Max value of the output vector from the CNN.

figures/cnn/cnn_roc99.png

figures/cnn/cnn_cm99.png

Figure 2.6: CNN Diagnostics: ROC Curve (left) and Confusion Matrix (right) with a 99% confidence cut

The maximum value of the output vector was used to determine the predicted class, **which may not be the best choice for this problem. why not?** Figure 4.9 shows the distribution of the maximum value found. As shown, there are approximately 20 thousand spectra that are classified very confidently, but there are many more that are not.

A **much better ROC curve and confusion matrix** are produced by evaluating the diagnostics for only highly confident classifications (Figure 2.6). Therefore, it is clear that the CNN, when confident in its classification, produces accurate results. This cut, however, is not ideal, removing *insert percent*% of the data.

The next step in training would be to either increase the confidence of the CNN or move to a different architecture, with the hopes of increasing not only the overall accuracy of the networks, but also the number of confident classifications.

2.3 Introduction of Transformers

Transformers are a relatively new architecture introduced in 2017 by Vaswani et al. (2017) for natural language processing (NLP) tasks. The original architecture consists of an encoder-decoder system. The encoder accepts a series of tokens and produces a series of vectors representing the input data via a series of self-attention layers and feed-forward layers. The decoder then takes the output of the encoder, and produces a series of tokens, one for each input token.

Once transformers were shown to have remarkable success in NLP tasks, they were quickly adapted to other fields, such as vision. Dosovitskiy et al. (2020) developed a vision transformer (ViT) architecture that differed from the original transformer encoder by replacing the tokenized input with a more involved preprocessing step. In short, the input image is broken into a series of patches, which are then flattened into a vector. These vectors, along with positional encodings, are then fed into the transformer architecture. For classification tasks, the first input token is replaced with a class token. After passing through the transformer, the class token is then run through a fully connected layer to produce the final probabilities.

2.3.1 ViT on Spectroscopic Data

Previous implementations of transformers have been shown to have characteristics beneficial to the classification of spectral data. A transformer's ability to learn contextual information is essential in spectral classification. A broad absorption line, for example, may be indicative of a Type Ia supernova if in one part of the spectrum, but may be indicative of a Type II supernova if in another part of the spectrum. This contextual information is not easily learned by a CNN, as the convolutional layers are not able to learn the importance of certain parts of the input space. Attention can also play a role in identifying the purpose of features that are not in a standard location. For example, a **non-k corrected s**pectrum might have a continuum pattern at different locations in the spectrum, but the overall shape would be similar. This change in sizing would be difficult to learn with fixed filters in a CNN, but would be identified based on their positional importance by a transformer. In addition to this, ViTs have been shown to outperform CNNs on vision tasks, which shows they are capable of focusing on learned features.

* Include caveat about the fact that ViTs take longer to train than CNNs *

3. Creation and Training of Spectral ViT

In order to examine the effectiveness of the transformer architecture for supernovae spectral classification, the following series of steps were taken. First, a custom transformer architecture was created based on the traditional ViT architecture (Dosovitskiy et al. 2020), and trained on a simple synthetic dataset to validate our approach. Next, a synthetic dataset using DESI spectra was created under a variety of preprocessing conditions. The previously created CNN and new transformer architectures were trained on the preprocessed data. Finally, these training sessions were then used to determine the optimal training conditions for non-redshift corrected data.

3.1 Creation of Spectral ViT

The Spectral ViT architecture was coded using the `PyTorch` deep learning framework. The architecture is shown graphically in Appendix A.3.1. The Spectral ViT is composed of three main components: the pre-processor, the encoder, and the classifier.

The pre-processing component is responsible for taking the input spectra extracted from the DESI CCDs and converting them into a series of vectors that the transformer can interpret. Consider a sample of N spectra, each with 10000 pixels corresponding to a bin containing a wavelength between 3600Å and 9800Å. Each group is split into a set number of patches (approximately 100 pixels in length). Each patch is then linearly mapped via a fully connected network to a vector 4 times smaller than the patch size. The sample is now a set of size $N \times 100 \times 25$. A classification token of the same dimension as the patches is initialized with uniformly distributed random values between 0 and 1, and then concatenated to the beginning of each sample. In order for the transformer to properly understand the positional relationship between each patch, an embedding is added to each patch. This embedding is a scalar function based on the size of the patch and is calculated as follows:

$$\text{Embedding}_{ij} = \begin{cases} \sin\left(\frac{i}{10000^{(j/\text{patch size})}}\right) & \text{if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{((j-1)/\text{patch size})}}\right) & \text{if } j \text{ is odd} \end{cases}, \tag{3.1}$$

where i is the patch index, j is the index in the hidden dimension, and the patch size is the number of pixels defined in each patch (Vaswani et al. 2017). In a CNN, the positional relationship between each pixel is understood by the convolutional layers and the pooling layers. However, without the positional embeddings, there is no difference between the first and last patch in the sample. The sinusoidal embeddings were chosen in alignment with the original transformer architecture (Vaswani et al. 2017), as well as providing a simple way to encode the positional information. A visual representation of the embeddings for the sample is shown in Fig. 3.1. Once figures/embeddings_new.png

Figure 3.1: Visual representation of the embeddings for a sample of spectra. The x-axis represents the index of the patch, the y-axis represents the index of the hidden dimension, and the color represents the value of the embedding.

the embeddings are added element-wise to the patches, the resulting tensor (of size $N \times 101 \times 300$) is then passed through the encoder.

The encoder is the main component of any ViT architecture, as it contains the multi-head attention and feed-forward layers. The encoder is composed of a user-defined number of transformer blocks, each of which contains layer normalization, multi-head attention, another layer normalization, and finally a feed-forward layer. The multi-head attention layer used in this work is a scaled dot-product attention layer found in Vaswani et al. (2017). Each patch is passed through three separate linear layers, each with a different set of weights, resulting in three sets of vectors: the query (Q), key (K), and value (V).

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \tag{3.2}$$

The dot-product between the query and key vectors is then calculated, scaled by the square root of the dimensionality of the query vector, and then passed through a softmax function, defined as follows:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}. \tag{3.3}$$

The resulting attention weights are then multiplied by the value vectors (Equation 3.2). This is simultaneously done for each head in the multi-head attention layer. Each result is then concatenated together, and then passed through a linear layer to reduce the dimensionality to the size of the query vector. This resulting vector is then added element-wise to the original patches, normalized, and then passed through a MLP, resulting in a tensor of size $N \times 101 \times 300$. This again is added element-wise to the original patches. Each step is repeated for a user-defined number of transformer blocks, resulting in a tensor of size $N \times 101 \times 300$.

The final component of the Spectral ViT architecture is the classifier. The classifier takes in only the first patch of each sample, which has been designated as the classification token. The classification token is passed through a linear layer to reduce the dimensionality to the number of classes, and then passed through a softmax function to produce a probability distribution over the classes. Therefore, the resulting tensor is of size $N \times 1 \times 6$, for our 6 classes of supernovae.

3.1.1 Validation of Spectral ViT Architecture

After creating the Spectral ViT architecture, we tested our ability to train it effectively. A synthetic dataset, consisting of a continuum with broad Gaussian peaks placed at predetermined locations. Certain combinations of peak locations were chosen to figures/synth_data_new.png

Figure 3.2: Synthetic spectra used to verify performance of SpectralViT. Red and green vertical lines represent features used to seperate the data into disticnt classes for SNR=2 and SNR=10, respectively. Two classes are shown for each SNR.

represent an arbitrary 'class' of supernovae. These peaks, our synthetic emission lines, were given random amplitudes and widths, simulating variability, with a maximum allowed value. *This maximum allowed value was then used to add Gaussian noise to the signal: either 2 or 10 times the signal to simulate noisy ($S/N = 2$), or very good data ($S/N = 10$).* Examples of the synthetic spectra with different continuum profiles are shown in Fig. 3.2.

These datasets with SNR=10 were trivially separable by a Spectral ViT architecture with two transformer blocks, two transformer heads, eight hidden layers, and a patch size of 16 (spectra were 1024 pixels long). An accuracy of 100% on the testing set was achieved within 5 epochs of training (Figure 3.3b). Trained on the same architecture, the data with SNR=2 was more difficult to separate, only achieving a 33.975% test accuracy (Figure 3.3a).

3.2 Creation of Synthetic DESI Spectra

Once we demonstrated the Spectral ViT architecture was training effectively on simple synthetic data, the architecture was trained on DESI data. In order to create `figures/pre_testing/SNR2_training_epoch.png`

(a) SNR = 2

`figures/pre_testing/SNR10_training_epoch.png`

(b) SNR = 10

Figure 3.3: Training of Spectral ViT on synthetic datasets. The model was able to reach 100% accuracy on the test set of SNR=10 (top), but failed to reach an accuracy of 50% on the test set of SNR=2 (bottom).

a large enough training set, observed DESI spectra were used as a base to create synthetic spectra.

** Talk to BenZvi about what to put in this section / who to cite for all of the work ** ** Couldn't find Eddies thesis online – ask BenZvi fora copy **

3.2.1 Pre-Processing of DESI Spectra

Once the spectra was created and saved as DESI files, they needed to be extracted, preprocessed, split into training, testing, and validation sets, only then could they be saved, and used to train the Spectral ViT architecture. The preprocessing method `figures/cnn/cnn_training_history.jpg`

Figure 3.4: Training of the CNN on synthetic DESI spectra.

developed by ** Cite eddies paper **, and is split into 3 main steps: redshift correction, rebinning / down sampling, and normalization.

The reduced spectra, a table of best fit redshifts, and a set of fitting templates and coefficients are provided by the DESI pipeline in the form of a FITS file (Pence, W. D. et al. 2010; Guy et al. 2023). The z correction step was used to move the spectra back into the rest frame using the best-fit redshift identified by the DESI pipeline. Next, all artifacts in the spectra (masks, bad pixels, etc.) were removed. The spectra were then re-binned and downsampled to a user-defined resolution (default 3600). The spectra were then normalized to redefine the maximum and minimum flux values to 1 and 0 respectively. Finally, for the CNN datasets, the spectra were split into a 2D array of equal height and width (i.e. for the spectra of length 3600, the array would have a shape of 60×60). After this, the spectra were split into training, testing and validation datasets that comprised 60%, 20%, and 20% of the total dataset, respectively.

3.3 Training of Neural Networks

3.3.1 CNN Training

The CNN architecture was developed by ** Cite eddies paper **, and has properties shown in Table A.1. This CNN was trained for a maximum of 50 epochs, with a batch size of 50, and non-variable hyperparameters (Table 3.1). The timeline of the training of the CNN architecture is shown in Fig. 3.4.

3.3.2 Transformer Training

Two Spectral ViT architectures were trained on the synthetic DESI spectra. The first architecture was smaller, while the second was larger. The properties of both are shown in Table 3.2. Both were trained initially on spectra downsampled to 3600 bins. As shown in Figure 3.5, the smaller architecture was able to train effectively, reaching a test accuracy of 60.94% after approximately 31 epochs. The larger architecture was unable to train effectively, only reaching a test accuracy of around 26% before overfitting, as shown in Figure 3.6. Therefore, the smaller architecture was chosen for

Hyperparameter	Value
Learning Rate	0.00004135238172950965
Regularization	0.06
Dropout Rate	0.40904759925886294
Batch Size	50

Table 3.1: Hyperparameters of the CNN used to classify DESI spectra. CNN adapted from Sepeku (2022).

Hyperparameter	**Smaller Architecture**	**Larger Architecture**
Learning Rate	0.0001	0.01
Batch Size	16	16
Blocks	4	4
Heads	5	12
Hidden Dimension	25	72

Table 3.2: Hyperparameters of the smaller and larger Spectral ViT architecture used to classify DESI spectra.

further testing.

In addition, the smaller architecture was trained on spectra downsampled to 1800 and 900 bins to test its performance on spectra of varying resolution. As shown in Figure 3.7 and Figure 3.8, neither variation was able to train as effectively as the original binning.

The smaller architecture, with a resolution of 3600 bins, was then trained on the non redshift-corrected spectra. As shown in Figure 3.9, the model was able to train effectively, reaching a test accuracy of 49.82%.

figures/v1_real/vit_model_V1_original_redotraining_new.png

Figure 3.5: Training of the small architecture on redshift-corrected spectra downsampled to 3600 bins. Over-fitting was determined to have occurred by Epoch 31.

figures/v1_real/vit_model_V1_bigtraining_new.png

Figure 3.6: Training of the large architecture on Redshift-corrected spectra downsampled to 3600 bins. No convergence above random guessing was observed.

figures/vit_model_V1.2training_new.png

Figure 3.7: Training of the small architecture on Redshift-corrected spectra downsampled to 1800 bins. No convergence above random guessing was observed.

figures/vit_model_V1.3_muchsmallermodeltraining_new.png

Figure 3.8: Training of the small architecture on Redshift-corrected spectra downsampled to 900 bins. No convergence above random guessing was observed.

figures/v2_real/vit_model_V2training_new.png

Figure 3.9: Training of the small architecture on non Redshift-corrected data downsampled to 3600 bins. Over fitting was determined to have occurred by Epoch 26.

4. Results

4.1 V1

figures/v1_real/vit_model_V1_original_redoroc_curve_v1.png

figures/v1_real/vit_model_V1_original_re

Figure 4.1: Spectral ViT V1 Diagnostics: ROC Curve (left) and Confusion Matrix (right)

figures/v1_real/vit_model_V1_original_redomax_ypred_binary_31.png

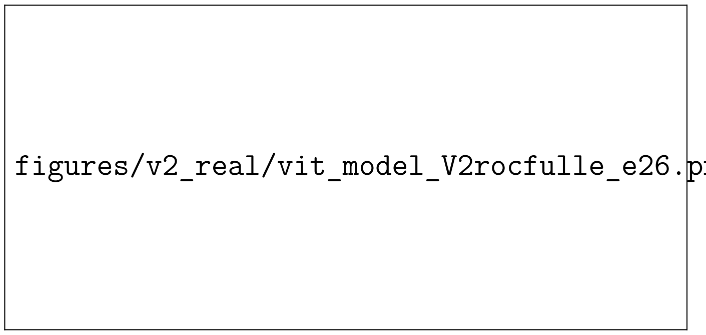
Figure 4.2: Max value of the output vector from the Spectral ViT V1.

figures/v1_real/vit_model_V1_original_redoroc99_e31.png figures/v1_real/vit_model_V1_original_re

Figure 4.3: Spectral ViT V1 Diagnostics: ROC Curve (left) and Confusion Matrix (right) with a 99% confidence cut

figures/v1_real/vit_model_V1_original_redoroc999999_e31.png figures/v1_real/vit_model_V1_original_re

Figure 4.4: Spectral ViT V1 Diagnostics: ROC Curve (left) and Confusion Matrix (right) with a 99.9999% confidence cut



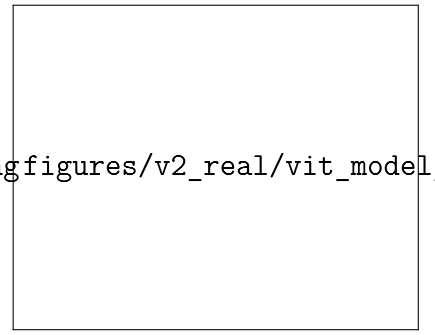


Figure 4.5: Spectral ViT V2 Diagnostics: ROC Curve (left) and Confusion Matrix (right)

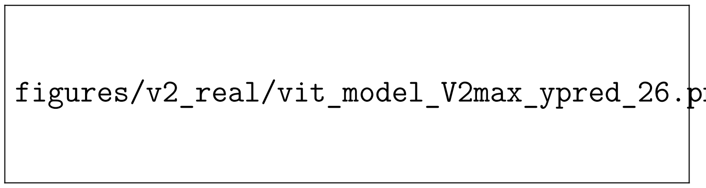


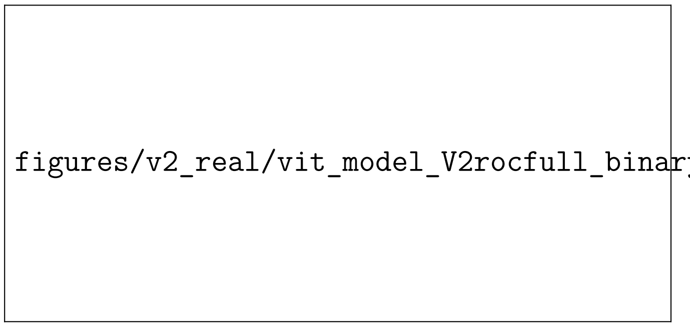
Figure 4.6: Max value of the output vector from the Spectral ViT V2.

4.2 V2

figures/v2_real/vit_model_V2roc99_e26.png

figures/v2_real/vit_model_V2cm99_e26.png

Figure 4.7: Spectral ViT V2 Diagnostics: ROC Curve (left) and Confusion Matrix (right) with a 99% confidence cut



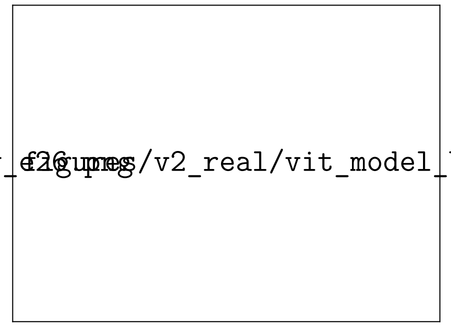


Figure 4.8: Spectral ViT V2 Binary Diagnostics: ROC Curve (left) and Confusion Matrix (right)

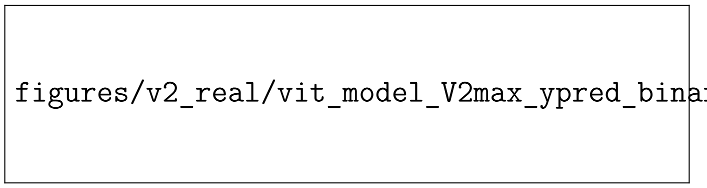


Figure 4.9: Max value of the output vector from the Spectral ViT V2 Binary Classification.

4.2.1 Binary Classification

figures/v2_real/vit_model_V2roc9999_binary_e26.png figures/v2_real/vit_model_V2cm9999_binary

Figure 4.10: Spectral ViT V2 Binary Diagnostics: ROC Curve (left) and Confusion Matrix (right) with a 99.99% confidence cut

Bibliography

Cybenko, G. (Dec. 1, 1989). “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274.

Dosovitskiy, Alexey et al. (2020). “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: *CoRR* abs/2010.11929. arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

Fukushima, Kunihiko (1979). “Self-Organization of a Neural Network Which Gives Position-Invariant Response”. In: *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI’79. Tokyo, Japan: Morgan Kaufmann Publishers Inc., pp. 291–293. ISBN: 0934613478. DOI: 10.5555/1624861.1624928.

Guy, J. et al. (Mar. 2023). “The Spectroscopic Data Processing Pipeline for the Dark Energy Spectroscopic Instrument”. In: *The Astronomical Journal* 165.4, p. 144. DOI: 10.3847/1538-3881/acb212. URL: https://doi.org/10.3847/1538-3881/acb212.

He, Kaiming et al. (2015). *Deep Residual Learning for Image Recognition*. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

Huang, Gao et al. (2016). *Densely Connected Convolutional Networks*. DOI: 10.48550/ARXIV.1608.06993. URL: https://arxiv.org/abs/1608.06993.

Kim, Yoon (2014). *Convolutional Neural Networks for Sentence Classification*. DOI: 10.48550/ARXIV.1408.5882. URL: https://arxiv.org/abs/1408.5882.

Kiranyaz, Serkan et al. (2021). “1D convolutional neural networks and applications: A survey”. In: *Mechanical Systems and Signal Processing* 151, p. 107398. ISSN: 0888-3270. DOI: https://doi.org/10.1016/j.ymssp.2020.107398. URL: https://www.sciencedirect.com/science/article/pii/S0888327020307846.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kumar, Ajitesh (Apr. 2022). *Different types of CNN Architectures explained: Examples.* URL: https://vitalflux.com/different-types-of-cnn-architectures-explained-examples/.

LeCun, Y., Fu Jie Huang, and L. Bottou (2004). “Learning methods for generic object recognition with invariance to pose and lighting”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2, II–104 Vol.2. DOI: 10.1109/CVPR.2004.1315150.

Naskath, J., G. Sivakamasundari, and A. Alif Siddiqua Begum (Oct. 2022). “A Study on Different Deep Learning Algorithms Used in Deep Neural Nets: MLP SOM and DBN”. In: *Wireless Personal Communications* 128.4, pp. 2913–2936. DOI: 10.1007/s11277-022-10079-4. URL: https://doi.org/10.1007/s11277-022-10079-4.

Neutelings, Izaak (Sept. 26, 2021). *Neural networks.* URL: https://tikz.net/neural_networks/.

— (Apr. 9, 2022). *Neural networks.* URL: https://tikz.net/conv2d/.

Parks, David et al. (Jan. 2018). “Deep learning of quasar spectra to discover and characterize damped Lyα systems”. In: *Monthly Notices of the Royal Astronomical Society* 476.1, pp. 1151–1168. ISSN: 0035-8711. DOI: 10.1093/mnras/sty196. eprint: https://academic.oup.com/mnras/article-pdf/476/1/1151/24261051/sty196.pdf. URL: https://doi.org/10.1093/mnras/sty196.

Pence, W. D. et al. (Nov. 2010). “Definition of the Flexible Image Transport System (FITS), version 3.0”. In: *A&A* 524, A42. DOI: 10.1051/0004-6361/201015362. URL: https://doi.org/10.1051/0004-6361/201015362.

Popescu, Marius-Constantin et al. (July 2009). “Multilayer perceptron and neural networks”. In: *WSEAS Transactions on Circuits and Systems* 8.

Sepeku, Edmund (May 2022). –. Senior Thesis. Rochester, NY.

Shelhamer, Evan, Jonathan Long, and Trevor Darrell (2016). *Fully Convolutional Networks for Semantic Segmentation.* DOI: 10.48550/ARXIV.1605.06211. URL: https://arxiv.org/abs/1605.06211.

Simonyan, Karen and Andrew Zisserman (2015). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *International Conference on Learning Representations.*

Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

A. Detailed Diagrams of Various Neural Networks

A.1 Multi-Layer Perceptron (Fully Connected Feed-Forward Networks)

A.2 Convolutional Neural Networks

A.3 Transformers

A.3.1 Spectral ViT

A.3.2 Multi-Head Attention

```
Model: "model"
__________________________________________________________________________________________________
Layer (type) Output Shape Param # Connected to
==================================================================================================
input_spec1 (InputLayer) [(None, 60, 60, 1)] 0
__________________________________________________________________________________________________
conv2d_1 (Conv2D) (None, 53, 53, 5) 325 input_spec1[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D) (None, 29, 29, 5) 5125 input_spec1[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D) (None, 5, 5, 5) 15685 input_spec1[0][0]
__________________________________________________________________________________________________
conv2d (Conv2D) (None, 57, 57, 5) 85 input_spec1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 53, 53, 5) 212 conv2d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 29, 29, 5) 116 conv2d_3[0][0]
__________________________________________________________________________________________________
batch_normalization_4 (BatchNor (None, 5, 5, 5) 20 conv2d_4[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 57, 57, 5) 228 conv2d[0][0]
__________________________________________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 17, 17, 5) 0 batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D) (None, 45, 45, 5) 1285 input_spec1[0][0]
__________________________________________________________________________________________________
max_pooling2d_3 (MaxPooling2D) (None, 9, 9, 5) 0 batch_normalization_3[0][0]
__________________________________________________________________________________________________
max_pooling2d_4 (MaxPooling2D) (None, 1, 1, 5) 0 batch_normalization_4[0][0]
__________________________________________________________________________________________________
flatten (Flatten) (None, 16245) 0 batch_normalization[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten) (None, 1445) 0 max_pooling2d_1[0][0]
__________________________________________________________________________________________________
flatten_2 (Flatten) (None, 10125) 0 conv2d_2[0][0]
__________________________________________________________________________________________________
flatten_3 (Flatten) (None, 405) 0 max_pooling2d_3[0][0]
__________________________________________________________________________________________________
flatten_4 (Flatten) (None, 5) 0 max_pooling2d_4[0][0]
__________________________________________________________________________________________________
concatenate (Concatenate) (None, 28225) 0 flatten[0][0]
flatten_1[0][0]
flatten_2[0][0]
flatten_3[0][0]
flatten_4[0][0]
__________________________________________________________________________________________________
dense (Dense) (None, 64) 1806464 concatenate[0][0]
__________________________________________________________________________________________________
dropout (Dropout) (None, 64) 0 dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense) (None, 6) 390 dropout[0][0]
==================================================================================================
Total params: 1,829,935
Trainable params: 1,829,647
Non-trainable params: 288
__________________________________________________________________________________________________
```

Table A.1: Architecture of the CNN used to classify DESI spectra. CNN adapted from Sepeku (2022).

B. Title of Appendix B