MNRAS 000, 1–8 (2015) Preprint 5 December 2021 Compiled using MNRAS LATEX style file v3.0

HERA Memo #108
Received 4 Dec 2021

Using Deep Learning to Identify RFI in HERA Data

Zhile (Lister) Chen,[1]★ Paul La Plante,[1,2]
[1]Department of Astronomy, University of California, Berkeley
[2]Berkeley Center for Cosmological Physics, University of California, Berkeley

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

Radio Frequency Interference (RFI) refers to the anthropogenic noise mixed in the data captured by the radio telescopes. RFI creates a sharp discontinuity in the data captured by the Hydrogen Epoch of Reionization Array (HERA) telescope. These sharp discontinuities can affect the overall sensitivity of HERA's measurements in two main aspects: the magnitude of RFI is much greater than the intrinsic sky signal, and unflagged RFI introduces errors due to the Fourier transform in HERA's data processing procedure. Therefore, identifying RFI in HERA data sets is crucial. We present an RFI detection Convolutional Neural Network (CNNRFI) that identifies RFI in HERA's data sets to address this issue. The network is trained using simulated HERA visibility data and artificially modeled RFI, where we treat the modeled RFI as the "ground truth" in the training data set. We evaluate the performance of CNNRFI both by comparing the accuracy and precision of RFI found by that CNNRFI to an unseen test dataset of the simulated RFI, as well as predictions made by the Sky-Subtracted Incoherent Noise Spectra (SSINS), an established method for identifying RFI. CNNRFI achieves a precision of 99% in comparison with the HERA simulated data, and a agreement of 67.7% compared to SSINS. This larger discrepancy with SSINS can be explained by the aggressive flagging nature of SSINS, which tends to flag pixels beyond the ones identified in the algorithm as a means of reducing contamination from data below the noise floor.

Key words: dark ages, reionization, first stars — methods: data analysis — methods: observational

1 INTRODUCTION

The Hydrogen Epoch of Reionization Array (HERA) telescope is a radio interferometer seeking to detect the 21cm signal from the Epoch of Reionization. Detecting this signal potentially provides us with clues to understand the ionization process of the first stars and galaxies in the early universe (DeBoer et al. 2017). HERA captures sky visibilities, and pairs of visibilities are combined to produce a power spectrum. HERA's visibility data consists of four dimensions: baseline (defined as a pair of antennas, and measured in meters), time (measured in seconds), frequency (measured in MHz), and instrumental polarization. Anthropogenic noise is mixed within the HERA data, and we refer to this human made noise as the radio frequency interference (RFI). Examples of RFI includes FM radio and digital TV signals. RFI is usually two to three orders of magnitude larger in amplitude compared to the sky signal, and this RFI is typically narrow in frequencies. We visualize both the sky data and the RFI using a so-called waterfall plot (Figure 1).

Because of its high amplitude and narrow frequency, RFI creates sharp discontinuities in HERA data. These sharp discontinuities create disruptions in the accuracy of HERA's data in two aspects. The first aspect is that the magnitude of these signals swamps the the sky signal. The second aspect is related to HERA's data processing procedure: the raw data captured by the telescope is sent through a Fourier transform. Consequently, the sharp discontinuities at narrow frequencies created by the RFI get transformed into high magnitudes

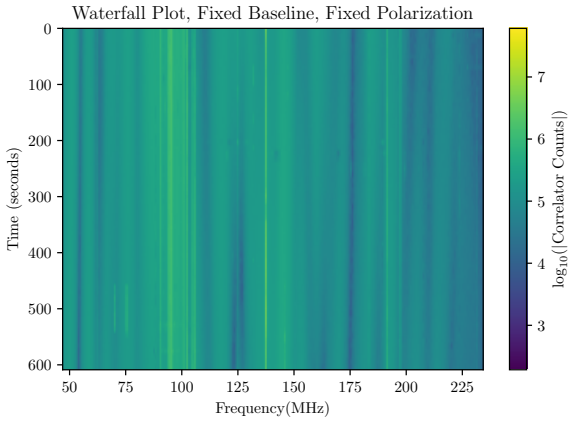

Figure 1. A waterfall plot of HERA data. Because HERA's raw data is a complex quantity, we have taken the absolute value of the visibilities. We have also taken the base-10 logarithm of the resulting amplitude to better visualize the sky data because RFI is about 2 to 3 orders of magnitude brighter than the background. The x-axis of the plot represents frequency in units of MHz, and the y-axis represents time in units of seconds. The narrow yellow stripes significantly brighter than the background are the RFI. The bright strips at around 100 MHz are the FM radio signals, and at 137 HZ are the low Earth orbit communications satellites (ORBCOMM) signals.

of power at all modes, and therefore creating significant inaccuracies in the data (Offringa et al. 2019). Further complications arise when one tries to categorize whether a signal is a RFI or not: one may

★ E-mail: listerchen319@berkeley.edu

falsely categorize a non-RFI signal as a RFI, which is commonly referred to the false positive case, or one may falsely categorize a RFI as a non-RFI, referred to as the false negative case. Compared with the false-positive case, if the false-negative case remains in the raw data and gets sent through the Fourier transform, the false-negative RFI will end up adding high power magnitudes to all modes. Therefore, comparatively, the false-negative scenario hurts the data accuracy more than the false-positive case, and optimally, we should try to minimize the number of false negative cases when detecting RFI.

In this paper, we present our machine learning RFI detection algorithm developed in `TensorFlow` (Abadi et al. 2016) – Convolutional Neural Network RFI detector (CNNRFI). Machine learning is a branch of artificial intelligence, and in our use case the machine learning algorithm improves the accuracy of predictions through the use of training data. Our model relies heavily on the concept of deep learning, specifically the convolutional neural network (commonly known as CNNs) (Fukushima & Miyake 1982; LeCun et al. 1999; Krizhevsky et al. 2017). Instead of using a single layer to encode information, we use multiple layers to encode and decode data, which allows for extracting higher-level, detailed RFI structures from the raw input HERA data. CNNRFI's development is based on the architecture of the U-Net RFI implementation (Ronneberger et al. 2015; Akeret et al. 2017) in the sense that we use max pooling layers to condense the size of the input space, followed by deconvolution layers to return to the original size of the dataset. However, instead of implementing a uniform number of filters for each convolutional layer, CNNRFI has a increasing number of filters in convolutional layers, and therefore, the neural network can capture more features of the RFI in one image in each proceeding layer. After reaching the deepest layer, the networks then invert the process until it reaches the output prediction layer. CNNRFI is also based on the architecture of DFCN (Kerrigan et al. 2019).

Efforts have been made to automate the RFI detection process using more traditional statistical methods, such as Sky-Subtracted Incoherent Noise Spectra, which subtracts the slowly varying sky signal via numerical calculation to separate RFI (Wilensky et al. 2019). In Sec. 3.2 of this paper, we compare the RFI detection results from CNNRFI to those provided by SSINS to evaluate the performance of CNNRFI on real data.

This paper is outlined as follows. Sec. 2 introduces the architecture of CNNRFI in detail. Sec. 3 discusses the performance of CNNRFI in two aspects. We first generated a data set unseen by CNNRFI but similar to the structure of the training data set– the evaluation data set – from `hera_sim`, HERA's simulation data. We then ran CNNRFI on this evaluation data set and compared its RFI predictions to the "ground truth" – the RFI indicated in the input waterfalls. We also compared CNNRFI's detection result to that produced by SSINS. Sec. 3 discusses the procedures and results of these two evaluations. Finally, in Sec. 4, we conclude with discussing possible future improvements on CNNRFI.

2 METHOD

2.1 CNNRFI architecture

The architecture of CNNRFI is similar to the structure of a U-Net RFI (Ronneberger et al. 2015; Akeret et al. 2017): both these networks uses multiple convolutional 2D layers to extract features. When passing through a convolutional 2D layer, the data goes through a filter that responses distinctly to data with different scales, and the layer provides a feature map that summarizes the RFI picked out. However, while the U-Net architecture uses the same convolutional layer structures throughout, CNNRFI increases the number of features by using an increasing number of filters on convolutional layers as the network learns the data deeper. After finally applying 128 filters, the architecture inverts the process by decreasing the number of filters as the layers approaches the output prediction layer. This architecture style is commonly known as the image pyramid (Lin et al. 2017). The structure of CNNRFI is also similar to the DFCN's architecture (Kerrigan et al. 2019), but implements the network architecture in `Keras` (Chollet et al. 2015) instead of `TensorFlow`.

The neural network starts with three convolutional layers. Compared to a typical Artificial Neural Network (ANN) (Lecun et al. 1998), the standard 2D convolutional neural network (CNN) preserves the spatial dependence of our input image (Lecun et al. 1998; Kerrigan et al. 2019). This spatial dependence preservation is important because we want to preserve the relative position of RFI and sky data on the input waterfall image. Following these three convolutional layers is a batch normalization layer, which re-standardize the input to a neural network. This layer is intended to accelerate the data training process, as well as regularize the data coming out of the three convolutional 2D layers. Using this layer reduces generalization error (Ioffe & Szegedy 2015). The output from the batch normalization layer is then down-sampled with a max pooling layer. The max pooling layer picks out the highest pixel value in a small region of the image (a square of 2×2 pixels in our case) and reduces the size of the input image accordingly. The combination of: (1) three convolution 2D layers; (2) a batch normalization layer; and (3) a max pooling layer "bundle" is put together as a "stack layer". There are four stack layers in total in our model, and each stack layer is concatenated to a up sampling 2D layer of corresponding size later in the network. The up sampling 2D layer doubles the input dimension and performs the inverse of the convolutional 2D layer operation. We then take the output from our stack layer and concatenate it with the up sampling 2D layer, and this up sampling 2D has equal time and frequency dimension with with stack layer output. This technique is named "skip connections" (Long et al. 2015; He et al. 2016), and has been shown to prevent overfitting the deep layers near the center of the U-Net. Concatenating the convolutional layer and up sampling 2D layer speeds up the training process. This layer concatenation also takes into account higher-order non-linear effect in the network. When the higher-order nonlinear effect in the network becomes dominant in layers, the network tends to over-fit data during the training process. Concatenating our stack and up sample layers avoids data overfitting in when training the network (Kerrigan et al. 2019). Finally, the activation function "softmax" is applied on the layer before the network makes a prediction. Before applying the softmax activation function to the final layer, some of the data components could be negative or larger than one. The softmax function normalizes the function into values between [0, 1] (Bridle 1989). We used categorical crossentropy as our loss function, which quantifies the differences between two probability distributions. In our case, there are two "categories" for our input data (either "flagged pixel" or "unflagged pixel"), so we use an activation and loss function suitable for classification problems. We also used adaptive moment estimation, *Adam*, as our network optimizer (Kingma & Ba 2017).

A detailed description of CNNRFI's architecture can be found in Table 1, and Figure 2 presents a visualization the neural network.

Traditional convolutional neural networks commonly restricts the dimension of input data (Krizhevsky et al. 2017). In contrast with traditional architectures, we passed in (None, None, None, None) as the shape of the input data to train the network. Through not providing the shape of the training data for the network, we open up the flexibility for data's shape to put in CNNRFI – as long as the input

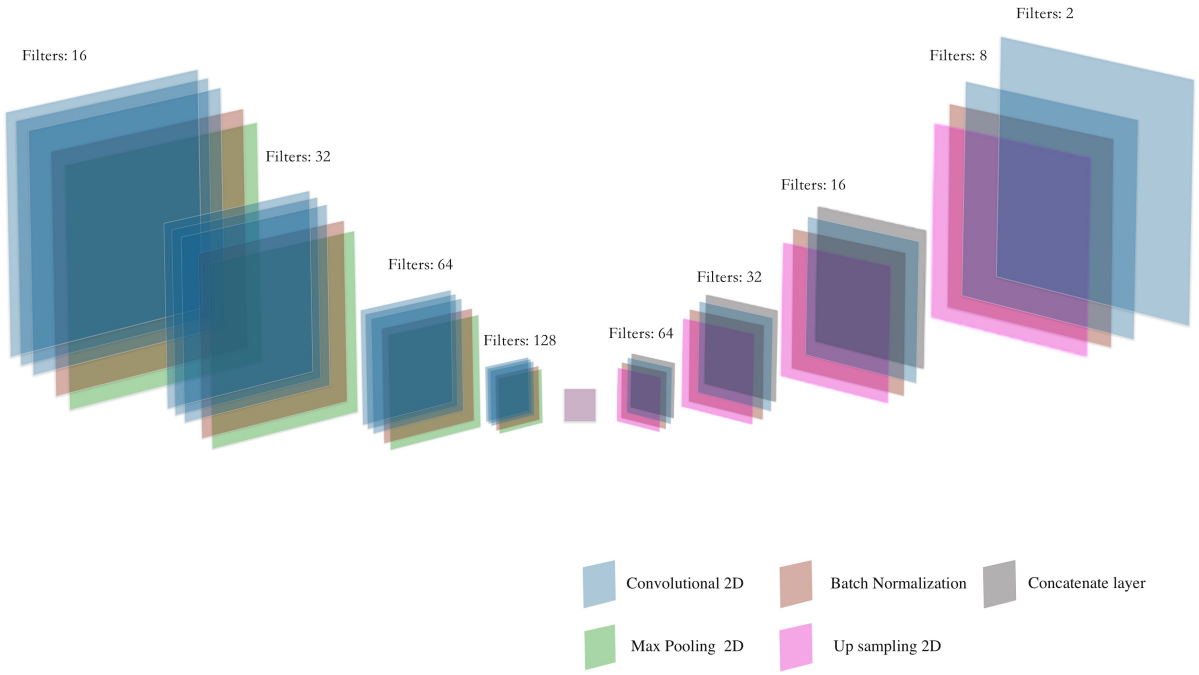

Figure 2. A visualization of the CNNRFI's architecture. As shown in the figure, there are many layer structures between the waterfall input and output, and these layers increase the network's accuracy to capture RFI in an image (Schmidhuber 2015). Due to our "flip and cut" method of pre-processing the data mentioned in section 2.2, CNNRFI has no restriction on the dimensions of the input image. For waterfalls with a non-16 multiple dimension of frequency or time, the immediate waterfall output from the neural network will have their extended dimensions, with both frequency and time a multiple of 16, but during our data post-processing stage, we cut the waterfall to its original input shape. Section 2.1 and Table 1 provides a detailed description of the network's structure.

data are multiples of 16, the CNNRFI is able to make a prediction on the data. To make the model more flexible, we designed a special way of preparing the raw data before passing the data to the architecture, which we will discuss in the data preparation section.

2.2 Data Processing

The input data has four dimensions: baseline, time, instrumental polarization, and frequency (DeBoer et al. 2017). For simplicity, we fix one baseline and one polarization in the input data. After down-selecting the baseline and the polarization in the data set, the original four-dimensional data captured by HERA is reduced to an array with two dimensions: time and frequency. The next data preparation step is to adjust the input data's time and frequency into appropriate dimensions. Although CNNRFI restricts every input data dimension to be multiples of 16, we allow the input data shape in any dimension by applying the "flip and cut" method on the raw waterfall image. When a raw data set is passed in, our data pre-processing algorithm first determines whether the time and frequency dimensions are multiples of 16. If a dimension is not a multiple of 16, our algorithm will first flip the entire data set across the last column (or row, depending on whether the non-16 multiple dimension is frequency of time) of this dimension, and then concatenate the flipped array with the original array on the flipping axis. Then, the algorithm finds the minimum multiple of 16 larger than the current dimension, and cuts the concatenate array at the next multiple of 16. For example, if the input waterfall takes the shape of (45, 1536) after down selecting one baseline and one polarization, the algorithm will recognize that 1546, the frequency dimension, is a multiple of 16, whereas the time dimension, 45, is not. The smallest multiple of 16 larger than 45 is 48. The algorithm will then use the 45th column of data as the "flipping axis" to flip the entire waterfall across the 45th time column, and concatenate the two arrays together. Following the concatenation, the algorithm with cut the time axis of the expanded array, which is in the shape of (89, 1536), at 48, resulting in an array of shape (48, 1536). See figure 3 for a visual illustration of this procedure. The same procedure applies when the frequency axis is not a multiple of 16: the algorithm will flip the data across the last frequency row and cut off the frequency dimension at the next multiple of 16 of the raw waterfall frequency dimension. If both the time and frequency dimensions are not multiples of 16, the algorithm will apply this procedure twice: the algorithm will adjust the dimension of both time and frequency to multiples of 16.

This "flip and concatenate" data-expanding method is the smoothest way to introduce the adjusted waterfall to the neural network, because flipping across the last time column and/or frequency row avoids introducing new discontinuities to the waterfall. If new discontinuities are introduced to the waterfall at the data pre-processing stage, CNNRFI can potentially interpret this artificially introduced discontinuity as a RFI, and consequently, this discontinuity introduced by waterfall pre-processing can negatively influence the CNNRFI's training process to find the real noise. Due to this benefit, we adopt the "flip and concatenate" method to adjust the dimensions of the raw waterfall array.

Furthermore, to reduce the complications introduced by complex phases, we take the absolute value of the data set to make the entire waterfall real and positive. Because RFIs are, in general, two to three

Table 1. The complete CNNRFI model. In the table, Conv2D stands for convolutional 2D layers, Max Pool stands for Max Pooling 2D layers, Batch Norm stands for batch normalization layers, and the stack layer stands for the bundle of 3 convolutional 2D layers - batch normalization layers - Max Pooling 2D layer

Layer Type	Parameter Number	Layer Connected
Input Layer	0	N/A
Conv2D_0	160	input_1
Conv2D_1	2320	Conv2D_0
Conv2D_2	2320	Conv2D_1
Batch Norm_0	64	Conv2D_2
Max Pool_0	0	Batch Norm_0
Conv2D_3	4640	Max Pool_0
Conv2D_4	9248	Conv2D_3
Conv2D_5	9248	Conv2D_4
Batch Norm_1	128	Conv2D_5
Max Pool_1	0	Batch Norm_1
Conv2D_6	18496	Max Pool_1
Conv2D_7	36928	Conv2D_6
Conv2D_8	36928	Conv2D_7
Batch Norm_2	256	conv2D_8
Max Pool_2	0	Batch Norm_2
Conv2D_9	73856	Max Pool_2
Conv2D_{10}	147584	Conv2D_9
Conv2D_{11}	147584	Conv2D_{10}
Batch Norm_3	512	Conv2D_{11}
Max Pool_3	0	Batch Norm_3
Up Sampling2D_0	0	Max Pool_3
Batch Norm_4	512	Up Sampling2D_0
Conv2D_{12}	73792	Batch Norm_4
Concatenate_0	0	Stack_0, Up Sampling2D_0
Up Sampling2D_1	0	Concatenate_0
Batch Norm_5	512	Up Sampling2D_1
Conv2D_{13}	36896	Batch Norm_5
Concatenate_1	0	Stack_1, Up Sampling2D_1
Up Sampling2D_2	0	Concatenate_1
Batch Norm_6	256	Up Sampling2D_2
Conv2D_{14}	9232	Batch Norm_6
Concatenate_2	0	Stack_2, Up Sampling2D_2
Up Sampling2D_3	0	Concatenate_2
Batch Norm_7	128	Up Sampling2D_3
Conv2D_{15}	2312	Batch Norm_7
Conv2D_{16}	18	Conv2D_{15}

orders of magnitudes brighter than the sky background, we take the log-10 of the entire array to make the background sky more visible, both for human and the algorithm.

There is only one step to post-process the data after CNNRFI makes a prediction on the waterfall: our data post-processing algorithm takes the predicted data array and cut the array back to its original shape. After multiple testing, we found out cutting the flag array does not alter the relative position of those flags.

In summary, in the data pre-processing stage, if any or both of the time and frequency axes are not in multiples of 16, the non-16 multiple axis will be adjusted to a expanded waterfall array with multiple of 16 on both axes' dimensions. The adjusted waterfall is then taken the absolute value and log-10 before being passed into the CNNRFI. In the data post-processing stage, we cut the dimension of the flag array to its original data input shape.

2.3 Training Dataset

We used two packages to build our training data: the Radio Interferometric Measurement Equation solver (RIMEz) [1] and *hera_sim*[2]. We first used RIMEz to simulate our background sky signal, and we used *hera_sim* to paint RFI on top of the waterfall background.

RIMEz simulate the background sky data by numerically compute the fundamental visibility measurement equation for every correlation and a single polarization:

$$V_{jk}(\nu, t) = \begin{bmatrix} V_{jk}^{pp} & V_{jk}^{pq} \\ V_{jk}^{qp} & V_{jk}^{qq} \end{bmatrix} \tag{1}$$

$$= \int \mathcal{J}_j \mathcal{C} \mathcal{J}^{\dagger}{}_k \exp(-2\pi i \nu \vec{b} \cdot \hat{s}/c) d\hat{s} \tag{2}$$

where the coherency matrix is:

$$C = \begin{bmatrix} I(\hat{s}, \nu) + Q(s, \nu) & U(s, \nu) - iV(s, \nu) \\ U(s, \nu) + iV(s, \nu) & I(\hat{s}, \nu) - Q(s, \nu) \end{bmatrix} \tag{3}$$

The integration in equation 2 is across the entire sphere. A more detailed discussion of the background sky simulation is discussed in section 4.2.1 of Aguirre et al. (2021)

Upon obtaining the background sky data, we then use *hera_sim* to paint three classes of RFI on top of the simulated sky data: base RFI, dtv RFI, and scatter RFI. The base RFI includes bright bands discretely occurring at 0-100 MHz, 400 Hz, and 800-1000 Hz. Hera_sim empirically derives and simulates these base RFI (Parsons et al. 2012). Scatter RFI can occur at every frequency on the spectrum. After experimenting with several different choices, we found that giving scattering a 0.01 probability of occurring and 10% of the base RFI's amplitude most resembles the real data. Dtv RFI stands for digital TV signal, and dtv appears discretely in the range of 800 - 1000 MHz in shapes of rectangular blocks. Similarly, after comparison with real data, we determined that the reasonable parameters for the dtv signal are: 0.05 probability of occurring and 1% of base RFI's magnitudes. All components of RFIs are stretched by a coefficient of 7.5 to achieve a higher real data RFI resemblance. Figure 4 is a example simulated data waterfall slice.

Each data waterfall is created by adding up the background sky signal, base RFI, scatter RFI, and dtv RFI. This process is looped over for 1000 times so that 1000 waterfalls of data sets are created. The 1000 waterfalls are then divided into 700 waterfall sets and 300

[1] https://github.com/UPennEoR/RIMEz
[2] https://github.com/HERA-Team/hera_sim

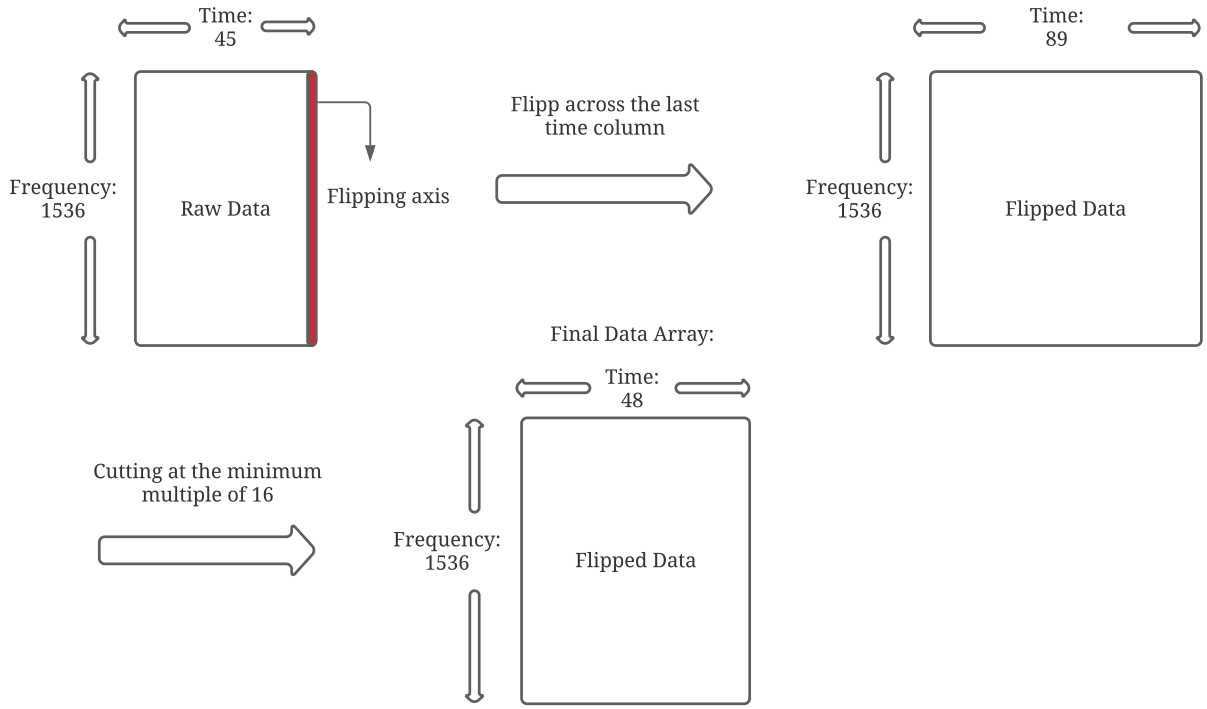

Figure 3. Data Preparation Procedure for a raw data shape (45, 1536), where 45 is time axis and 1536 is the frequency axis.

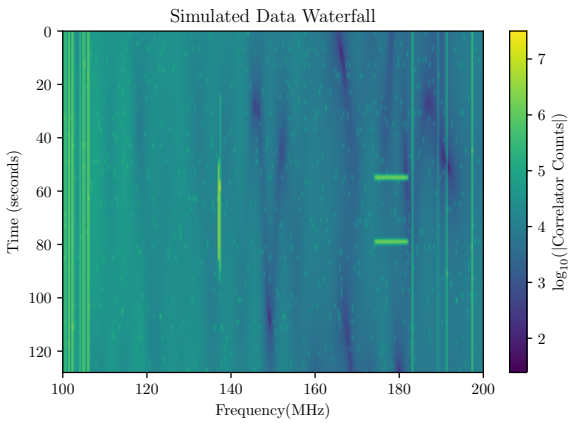

Figure 4. This is an example training data waterfall. The bright stripes appearing from 0-100 MHz, 400 MHz, 800 - 1000 MHz is the base RFI from hera_sim. The soporatic bright dots spread across the spectrum is the scatter RFI, and the blocks appearing at 800 MHz is DTV RFI.

waterfall sets.The 700 waterfall sets are used to train the CNNRFI model, while the rest of the 300 waterfalls are used as validation data to evaluate the performance of the CNNRFI architecture. Within each waterfall, there are 64 time axis and 1024 frequency axis. The training data for CNNRFI is in the shape of (700, 64, 1024), and the validation data for CNNRFI takes the shape of (300, 64, 1024).

3 EVALUATION

3.1 Training Results

For the first evaluation, we generated 300 new waterfalls the same way we generated the training and validation data set. The 300 new data sets are composed of the same elements as the training and validation data: background sky signal, base RFI, scatter RFI, and dtv RFI. We retained the training data set's chance of occurring and the amplitude for both scatter RFI and dtv RFI. Despite the newly generated waterfall arrays' resemblance to the network's training and validation data, these 300 new waterfalls are completely unique from the 1000 training and the validation waterfalls, meaning that CNNRFI has never "seen" these 300 data waterfalls prior to making a prediction. We evaluate the quality of the predicted data in two aspects: the number of false negative cases, where the data point is a RFI signal in the simulation data but CNNRFI did not flag the RFI signal, and the number of false positive cases, meaning that the data point is a sky signal in the simulation data, but CNNRFI flagged the data point as RFI.

Figure 5 provides a visualization of CNNRFI's prediction result on one waterfall slice. To quantify CNNRFI's model prediction accuracy, we calculated the average occurrence percentage of true positive, false positive, and false negative cases. The single waterfall occurring percentage is calculated by the number of specific cases divided by the total number of points on the waterfall plot: 64*1024 = 65536:

$$P_{occuring} = \frac{N_{cases}}{65536} \tag{4}$$

where N_cases stands for the number of specific cases (true positive, false positive, false negative) in one waterfall plot. Because we used a total of 300 waterfall plots to test the accuracy of the architecture, we

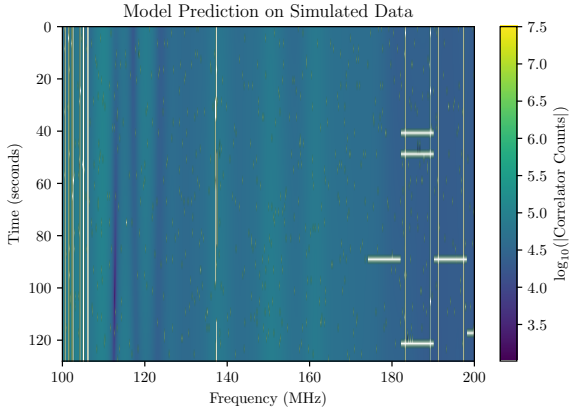

Figure 5. The true positive, false positive, and false negative cases of CNNRFI's prediction on one simulated data waterfall. Data points marked by white stands for true positive cases, blue points stands for false negative cases, and black points stands for true positive cases. There are very few false negative and false positive cases, so the blue and black points are not very visible in the plot.

Table 2. The average occurrence percentage of the true positive, true negative, false positive, and false negative cases of CNNRFI's prediction on simulated data

Cases	% Average Occurrence
True Positive	4.4%
True Negative	95.5%
False Positive	0.0046%
False Negative	0.003%

define the N values the average number of each cases occurring. The average number of each cases' occurrence is calculated by summing up the number of one specific case throughout the 300 waterfalls and dividing by 300.

$$\overline{N_{cases}} = \frac{\sum N_{wf}}{300} \tag{5}$$

where N_{wf} stands for the number of cases in each category in each waterfall. Table 2 shows a result of the average occurrence percentage for each cases.

As mentioned in the introduction (section 1) of the paper, the false negative cases damages data accuracy the most. We can see that the % of average occurrence of false negative cases only consists 0.003% in each of the data waterfall, and this result indicates that when making predictions on the simulated data, CNNRFI is efficient at minimizing the false negative cases to produce high quality flag predictions. The % average occurrence of false positive cases is 0.0046%, which is marginally more than the percentage of the false negative cases. However, the percentage still remains small compared to all the true cases. Aside from the small percentage of occurrence rate, the false positive cases harm the data's accuracy less than the false negative case. Therefore, it is safe to conclude that our model has high precision in predicting flags in the simulated data.

Table 3. The average occurrence percentage of the true positive, true negative, false positive, and false negative cases of CNNRFI's prediction compared to SSINS's flags

Cases	% Average Occurrence
True Positive	3.6%
True Negative	64.1%
False Positive	0.3%
False Negative	32.0%

3.2 Comparison with SSINS result

Evaluating the performance of CNNRFI on real HERA data introduces one complication: there is no "ground truth" indicating which data point is RFI. To resolve this complication, we used another non-machine learning RFI detection algorithm: Sky-Subtracted Incoherent Noise Spectra (SSINS) to predict RFI in the same set of real HERA data (Wilensky et al. 2019). We then treat the result from SSINS as the "ground truth" and compare CNNRFI's RFI prediction result against the prediction produced by SSINS.

For both SSINS and CNNRFI, we passed in 1840 raw HERA data files. Due to computing memory limitation, we divided 1840 files into groups of 8, and we programmed SSINS to produce one RFI prediction array on every 8 data files. For each prediction, we set the stream significance (s) to be 20, other significance (o) to be 5, and threshold (t) as aggressive as 0.1. SSINS made a total of 230 predictions, and we stacked all 230 arrays together to produce one RFI flag prediction on all raw data files. Similarly, we divided 1840 files into groups of 8 files for CNNRFI to make a prediction. Within each group of files, we iterate across all the data file to generate arrays of data for each antenna pair and polarization combinations. However, we filtered through all the arrays and only selected data with auto-correlation on antenna pairs and cross-correlation on polarizations due to how CNNRFI is trained. After filtering through data in each group, we then use CNNRFI to make a prediction on each filtered data array. We added all the predicted RFI arrays together within each group and divided the sum by the number of array samples to normalize the RFI prediction array to be between 0 and 1. We also added a threshold component when producing the final array: the final produced RFI prediction array will only flag a pixel is the pixel is predicted as RFI in more than 10% of the data arrays.

We evaluated the result from CNNRFI to that of SSINS using four matrices: true positive, true negative, false positive, and false negative. The occurrence percentage is calculated using the same method as section 3.1: counting the number of each cases and divide them by the total number of pixels on the waterfall, as show in equation 6

$$P_{occuring} = \frac{N_{cases}}{3680 * 1536} = \frac{N_{cases}}{5652480} \tag{6}$$

Table 3 shows the occurring percentage of each cases in this comparison, and figure 6 presents a visualization of all four cases.

The difference in RFI predictions between SSINS and CNNRFI can be explained by SSINS's aggressiveness when flagging RFI. Compared to CNNRFI, SSINS tends to overflag RFIs to make sure that most of the RFI in the data gets captured. The large-area RFI flagging is likely to produce a safe dataset that excludes the majority of RFIs in the raw data. CNNRFI, on the other hand, is rather

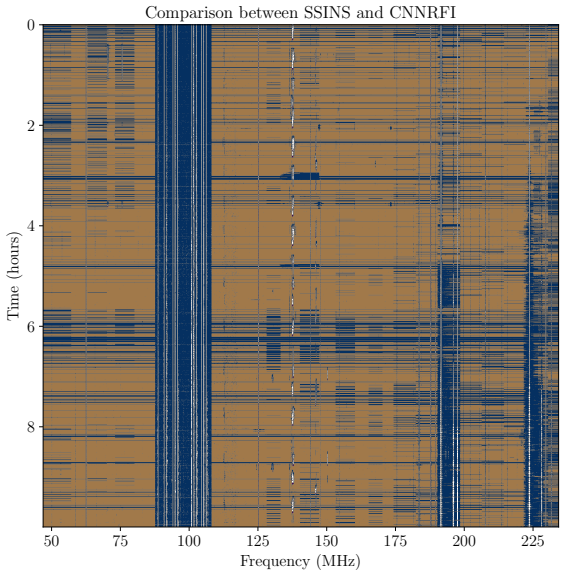

Figure 6. The true positive, true negative, false positive, and false negative cases of CNNRFI's prediction compared to that produced by SSINS (Wilensky et al. 2019). In the context of this figure, the "true" and "false" are relative to the SSINS output instead of the measure of the "real" underlying RFI. Because there are not many false positive cases, the red part are not very visible in the figure.

conservative at flagging pixels as RFI, because our machine-learning algorithm tends to target only the pixels that "looks like" RFIs. Therefore, a future test for CNNRFI is that we can train our neural network using flags produced by SSINS, and compare the result from SSINS and CNNRFI using a new, raw data set from HERA. Furthermore, it is worth noting that unlike CNNRFI, SSINS produced flags across some of the time dimensions, defying the definition of RFI. This "RFI" across time dimensions potentially resulted from the discontinuity at the boundary of each files – there are some discontinuities when concatenating each file together across the time dimension, and SSINS detected these discontinuities in time and interpreted them as time "RFI".

Although non-machine learning methods to detect RFIs like SSINS already exist, it is still significant to implement RFI detection via machine learning, because according to empirical data, machine-learning algorithms reduces the wall-clock time for data processing compared to non-machine learning methods (Kerrigan et al. 2019). Therefore, we suspect that it takes less wall clock time for CNNRFI's to predict flags in raw data compared to non-machine learning algorithms. This time advantage is crucial for HERA's data processing because HERA records around 4TB of raw data for every 12 hours of observing (DeBoer et al. 2017), and in the future, HERA will be fully constructed with 350 antennas and we will then expect HERA to record around 50 TBs of data per day. Under current circumstances, if an algorithm takes more than 12 hours to process all the raw data recorded in 12 hours of observation, unprocessed raw data will end up piling up. Consequently, the algorithm will fail to keep up with HERA's data recording overtime. Thus, using CNNRFI to filter RFI contributes to developing a faster data cleaning process under HERA's time constrain, and CNNRFI's fast data-processing nature is even more crucial for HERA's data processing procedure in the future when HERA will be fully connected with 350 antennas and record a larger amount of data.

4 CONCLUSIONS

In this paper, we demonstrated our machine-learning RFI detection algorithm – CNNRFI. CNNRFI is constructed through deep convolutional neural network combined with layer concatenation, and is trained on 700 waterfalls constructed using RIMz (Aguirre et al. 2021) and *hera_sim*[3]. The mock RFI from *hera_sim* establishes the "ground truth" for evaluating the performance of CNNRFI. We then evaluated CNNRFI's prediction by comparing the neural network's RFI prediction to the mock RFI in 300 newly generated data waterfalls. We calculated the true positive, true negative, false positive, and false negative numbers for each waterfall, averaged the number of each cases over 300 waterfalls, and calculated the average percentage of the occurrence of each cases. The result indicates that the false negative case, the error that damages data accuracy the most, only occurs in 0.003% in a waterfall on average, and thus demonstrating that CNNRFI is able to predict RFI on simulated data with high accuracy.

We also used CNNRFI to predict RFI in real data captured by HERA. Because we do not know which data point is RFI in real data, we compared CNNRFI's result to that produced by SSINS. In this comparison, we treated SSINS's result as the "ground truth", and found out that CNNRFI agrees with 67.7% of the RFI detected by SSINS. We also found out that the false positive percentage is 0.3%, and the false negative percentage is 32%. The discrepency between the two model's RFI prediction can be explained by "overflagging" nature of SSINS: SSINS tends to minimize false negative cases by spreading flags around dtected RFI. Compared to SSINS, CNNRFI tends to target only the RFI-like pixels. SSINS and CNNRFI's difference in flagging aggressiveness is discussed in detail in section 3.2. Furthermore, empirical data suggests that CNNRFI has a advantage reducing the wall clock time in processing the raw HERA data compared to other non machine-learning methods (Kerrigan et al. 2019), and therefore CNNRFI contribute to a building a faster raw data processing procedure for HERA.

In the future, we would like to expand the category of mock RFI in our training data set. Real data captured by HERA may include RFI that are not scatter RFI, dtv RFI, or base RFI. To make our training data set more realistic, we will add in random RFI aside from base RFI, dtv RFI, and scatter RFI. A good test for CNNRFI's ability to capture RFI accurately is to train CNNRFI using the flags produced by SSINS in section 3.2. We will then pass in another set of raw data files for both SSINS and CNNRFI to make predictions, and compare the prediction results from two algorithms. We will also test CNNRFI's performance by comparing the results produced by CNNRFI to other RFI detection algorithms aside from SSINS. We also envision expanding CNNRFI's function. Specifically, we would like to add data calibration functions to CNNRFI. In the real data sets captured by HERA, abnormal sections not caused by RFI can appear – for example, a defect data point may be caused by antenna malfunctioning. We would like to revise CNNRFI's structure so that CNNRFI can also detect these defect data points aside from predicting RFI.

The github repository for CNNRFI described in this paper can be found at: `https://github.com/listerchen319/cnn_rfi`

[3] `https://github.com/HERA-Team/hera_sim`

ACKNOWLEDGEMENTS

We thank the South African Radio Astronomy Observatory for hosting HERA. We gratefully acknowledge the support of the HERA machine learning research group for providing valuable suggestions and inspirations for this project.

REFERENCES

Abadi M., et al., 2016, arXiv e-prints, p. arXiv:1605.08695
Aguirre J. E., et al., 2021, Validation of the HERA Phase I Epoch of Reionization 21 cm Power Spectrum Software Pipeline (arXiv:2104.09547)
Akeret J., Chang C., Lucchi A., Refregier A., 2017, Astronomy and Computing, 18, 35
Bridle J. S., 1989, in Proceedings of the 2nd International Conference on Neural Information Processing Systems. NIPS'89. MIT Press, Cambridge, MA, USA, p. 211–217
Chollet F., et al., 2015, Keras, https://keras.io
DeBoer D. R., et al., 2017, PASP, 129, 045001
Fukushima K., Miyake S., 1982, in Amari S.-i., Arbib M. A., eds, Competition and Cooperation in Neural Nets. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 267–285
He K., Zhang X., Ren S., Sun J., 2016, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp 770–778
Ioffe S., Szegedy C., 2015, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (arXiv:1502.03167)
Kerrigan J., et al., 2019, MNRAS, 488, 2605
Kingma D. P., Ba J., 2017, Adam: A Method for Stochastic Optimization (arXiv:1412.6980)
Krizhevsky A., Sutskever I., Hinton G. E., 2017, Commun. ACM, 60, 84–90
LeCun Y., Haffner P., Bottou L., Bengio Y., 1999, Object Recognition with Gradient-Based Learning. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 319–345, doi:10.1007/3-540-46805-6_19, https://doi.org/10.1007/3-540-46805-6_19
Lecun Y., Bottou L., Bengio Y., Haffner P., 1998, Proceedings of the IEEE, 86, 2278
Lin T.-Y., Dollár P., Girshick R., He K., Hariharan B., Belongie S., 2017, Feature Pyramid Networks for Object Detection (arXiv:1612.03144)
Long J., Shelhamer E., Darrell T., 2015, Fully Convolutional Networks for Semantic Segmentation (arXiv:1411.4038)
Offringa A. R., Mertens F., Koopmans L. V. E., 2019, MNRAS, 484, 2866
Parsons A. R., Pober J. C., Aguirre J. E., Carilli C. L., Jacobs D. C., Moore D. F., 2012, ApJ, 756, 165
Ronneberger O., Fischer P., Brox T., 2015, in Navab N., Hornegger J., Wells W. M., Frangi A. F., eds, Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015. Springer International Publishing, Cham, pp 234–241
Schmidhuber J., 2015, Neural Networks, 61, 85–117
Wilensky M. J., Morales M. F., Hazelton B. J., Barry N., Byrne R., Roy S., 2019, PASP, 131, 114507

This paper has been typeset from a TEX/LATEX file prepared by the author.