**Installation and use of Shiny\_PNW-Cnet app for desktop audio processing**

Updated December 2022.

Direct comments and questions to Zack Ruff, [zjruff@gmail.com](mailto:zjruff@gmail.com).

**Note:** If you have used a previous version of this app, please be advised that the setup instructions have changed a bit. This is due to changes in the relationship and interdependency between Keras and TensorFlow and their corresponding packages in Python and R.

For purposes of this guide we assume that you are using a computer running a 64-bit version of Windows on which you have at least some ability to install programs. The app will not function on 32-bit operating systems. The app is not designed to run on MacOS, Linux or other operating systems, although it might work with some modification (attempt at your own risk).

**Part 1. Installing required software**

**Step 1.** Install the most recent version of R. If it is already installed, go to step 2.

Go to [https://cran.RStudio.com/](https://cran.rstudio.com/) and click the link labeled “Download R for Windows,” then follow the instructions to download the installer for the latest versions. Open your Downloads folder and find the installer; it will be called something like R-4.2.2-win.exe. Run the installer. The default settings should be fine.

**Step 2.** Install RStudio. If it is already installed, go to step 3.

Go to <https://posit.co/download/rstudio-desktop/> and click the big blue button labeled “Download Rstudio Desktop for Windows”. Again, go to your Downloads folder and run the installer; the default settings will be fine for our purposes.

**Step 3.** (Optional) Update R and RStudio.

If you just installed both programs, skip this step. This also may not be strictly necessary as long as your packages are reasonably up to date, but if you have been using the same version for a year or more, you should probably update regardless. Updating regularly is good practice.

To update R, open the R console (it is recommended to not do this through RStudio). Run the following commands:

install.packages("installr")  
library(installr)  
updateR()

This will install the installr package, load the package, and run its updateR function, which will search for the latest version and install it if there is one available. It will generate various popups; just agree to the default options and it should all be fine.

To update RStudio, open RStudio and click Help > Check for Updates in the menu bar. If there is a newer version available, click “Quit and download.” Run the installer.

**Step 4.** Install SoX.

SoX (aka Sound eXchange) is a command-line program that the app uses to process audio data. You will not need to do anything directly with it, but it needs to be installed on your computer. The most recent release is version 14.4.2, which works well for our purposes. Go to <https://sourceforge.net/projects/sox/files/sox/> and click “Download latest version” to download the executable installer for Windows. Open your Downloads folder and run the installer. Make a note of where it is installed; the default is something like C:\Program Files (x86)\sox-14-4-2.

**Step 5.** Set up the r-reticulate conda environment.

This is the most complicated step, so some explanation might be worthwhile. The PNW-Cnet model was built and trained using the TensorFlow software library, developed by Google, and the keras Python package, developed by François Chollet et al. In order to run the PNW-Cnet model from RStudio, we first have to install a compatible version of Python and tell RStudio where to find it. For consistency across different systems, we recommend using Miniconda, an environment manager that can be installed from within RStudio.

[ **Note:** If you are on a slower internet connection, you may run into problems when trying to install packages in RStudio. RStudio has a default timeout limit of 60 seconds; if a download takes longer than this, you will get a timeout error and the installation will fail. You can tell RStudio to allow more time by simply entering

options(timeout=600)

in the console pane. This will increase the time allowance to 600 seconds (10 minutes). Note that this change will only affect the current R session. If you restart R or close and reopen RStudio, the time limit will reset, but you can set it again as needed. ]

Open RStudio and use the console pane to enter the command

install.packages("reticulate")

This will install the reticulate R package and its dependencies. Next, enter the command

reticulate::install\_miniconda()

The default installation path for Miniconda might vary depending on the specifics of your system, but it generally will be in your home directory rather than a system directory, so it should not require elevated permissions. The installation path on my machine is C:\Users\zjruf\AppData\Local\r-miniconda.

This path will be shown in the RStudio console once the installation is successful. You can also check it from RStudio later with the reticulate::miniconda\_path() function.

Once Miniconda is installed, RStudio will attempt to create a conda environment called r-reticulate. This is essentially a self-contained Python installation with its own set of packages, which RStudio will use to load and run the PNW-Cnet model. However, lately (ca. Sept 2022), reticulate has developed a habit of successfully installing Miniconda but then failing to create the r-reticulate environment. This is not a big deal, it just means you will need to take a brief detour to create the r-reticulate environment from outside RStudio.

Close RStudio, then use the Windows search bar to search for "anaconda" and click on the "Anaconda Prompt" option. This should spawn a command prompt window that looks like this:



The (base) indicates that the base conda environment is active, while C:\Users\zjruf> indicates the current working directory (this will be different on your machine). We can run commands from this window just like from the RStudio console. Enter the command

conda env list

This will show a list of the conda environments that currently exist and the path to the folder in which each environment lives. The base environment lives in the Miniconda installation folder (the r-miniconda folder) and is always the active environment when the Anaconda prompt launches. When other environments are created, they will live in a subdirectory of r-miniconda called envs.

If there is no entry in this list for the r-reticulate environment, then you will need to create it. Enter the command

conda create -n r-reticulate python=3.8

This tells Miniconda that we want to create a new environment called r-reticulate and install Python version 3.8 in that environment. The prompt will show you a preview of the default packages that will be installed and ask if you want to proceed. Hit Enter to go ahead with the installation.

Once the r-reticulate environment has been created, enter

conda activate r-reticulate

The (base) at the beginning of the prompt should change to (r-reticulate) to indicate that our new environment is now active.

Text

Description automatically generated

The ability to create multiple conda environments and switch between them as needed is useful in many contexts. For our purposes, the main benefit is that it allows us to set up a stable, reproducible execution environment for the app. We can set up the r-reticulate environment with specific versions of Python and the packages that we need, which allows us to stick with a configuration that we know works well. This ensures consistent behavior across different machines, and it means we don’t need to worry that one of our packages will be updated in a way that breaks the overall functioning of the app.

Now that the r-reticulate environment exists and is active, we can configure it to work correctly by installing the right versions of Python and its associated packages. The first thing we need to do is verify that Python version 3.8 is installed. Enter the command

python --version

and note the version indicated by the output.

A screenshot of a computer

Description automatically generated with medium confidence

If the version is 3.8, then all is well. (The patch number, in this case 13, should not matter.) If the version is 3.9 or higher, then we will need to downgrade Python for compatibility with the packages we want to use. Enter

conda install python=3.8

You will get a warning that several packages will be downgraded; type y and hit Enter to proceed.

Next we are going to install some Python packages that are necessary for the neural net to work correctly. We will install these using the pip tool.

[Note: Anaconda has recently changed to a licensing model which requires a paid license for “Commercial” users, i.e. those within an organization with >200 users, to access the Anaconda Repositories. Using conda install for most packages will by default install from the Anaconda Repositories. By contrast, pip installs from the Python Package Index ([www.PyPI.org](http://www.PyPI.org)) and therefore requires no license.]

First we will install the tensorflow-cpu Python package. Enter

pip install tensorflow-cpu==2.2

(Note that there are two equals signs between cpu and 2.2. This is the correct syntax for pip.) The tensorflow-cpu package depends on a number of other packages which will be installed automatically alongside it, so this installation can take a few minutes; the Anaconda Prompt window will be flooded with semi-informative text and progress bars to indicate that things are happening. Ultimately you will be returned to a prompt with a blinking cursor, at which point it should be safe to proceed.

Text

Description automatically generated

Next we will downgrade the protobuf package to avoid compatibility issues with this version of TensorFlow.

pip install --upgrade protobuf==3.20

We will then install the pillow package, like so:

pip install pillow==8.2

Type y and hit Enter when you are prompted to proceed. pillow is the currently maintained fork of the Python Image Library and is used by TensorFlow to handle the spectrograms, which are stored as PNG image files. Again, following some output you will return to a blinking cursor.

Finally, we will install pandas.

pip install pandas==1.4

pandas is a data science-oriented package that introduces a lot of R-like functionality to Python, including the ability to create data frames that R can handle natively, which is helpful in ensuring smooth interaction between the two languages.

Once these steps are complete, the r-reticulate environment should be all set up, and you can close the Anaconda Prompt window.

**Troubleshooting the r-reticulate environment**

*(Note: These steps are not part of the normal installation process but can be helpful when things go awry. If everything appears to be working normally, you can skip to Step 6 on page 9.)*

A major benefit of using an environment manager like Miniconda is that conda environments are highly self-contained. As a result, it is actually very easy to delete the r-reticulate environment and recreate it, and this can be very useful in troubleshooting and resolving issues, especially on machines where the app has previously been installed.

The simplest way to remove the environment is to open the Anaconda prompt and enter

conda env remove --name r-reticulate

Enter y when you are prompted to proceed. The r-reticulate environment and all its packages will be deleted. Other environments will not be affected, nor will your installation of Miniconda itself.

However, each conda environment lives in its own folder, and running conda env remove will not actually delete this folder. If you want to be really thorough, you can find and remove the folder manually in File Explorer. (This is not necessary, but it makes everything slightly neater.) Before removing the environment through the Anaconda prompt, run

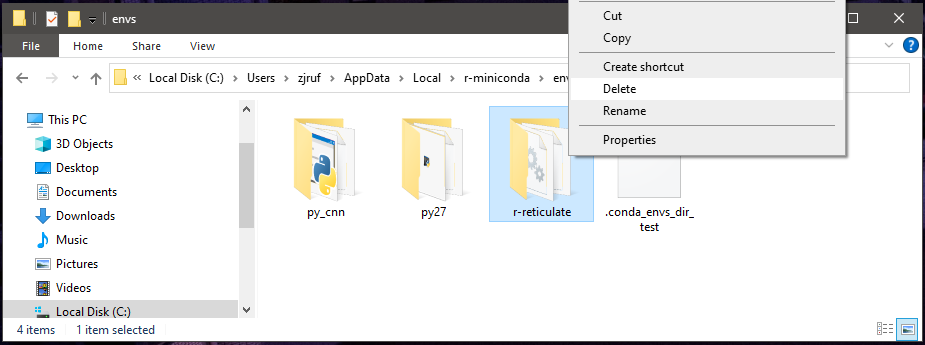
conda env list

You will get a list of your conda environments and where they are located.

Text, chat or text message

Description automatically generated

Highlight the path to the r-reticulate environment, use Ctrl + C to copy it, then open File Explorer and paste the path into the address bar. Then go up one level to the envs folder. You will see there is a folder for each environment (except base, which lives in the r-miniconda folder, one level up). Once you have removed the environment using conda env remove in the Anaconda prompt, select the r-reticulate folder and delete it.



Once the environment has been deleted, reopen the Anaconda Prompt and enter

conda create -n r-reticulate python=3.8

(If you only ran conda env remove to get rid of the old environment, you will get a warning that a directory already exists at the target location. Enter y to proceed anyway.) This will create a new conda environment called r-reticulate in the same location, with Python 3.8 installed, but with only default packages. From here, activate the environment and reinstall the packages as outlined above, beginning with installing tensorflow-cpu.

**How to make sure R is using the right conda environment**

If you are encountering errors suggesting that you are missing a package that you know you installed, it is likely that R is not using the correct version of Python (for instance, it may be defaulting to the base conda environment or the system’s default Python rather than r-reticulate). This can occur if e.g. you created an R project by copying the GitHub repository, or if you are using a preexisting installation of Anaconda or Miniconda.

You can check this using the reticulate::py\_config() function. The output should show that python.exe is located in the r-reticulate environment folder. If it is not, you can force R to use a specific version by setting the RETICULATE\_PYTHON environment variable.

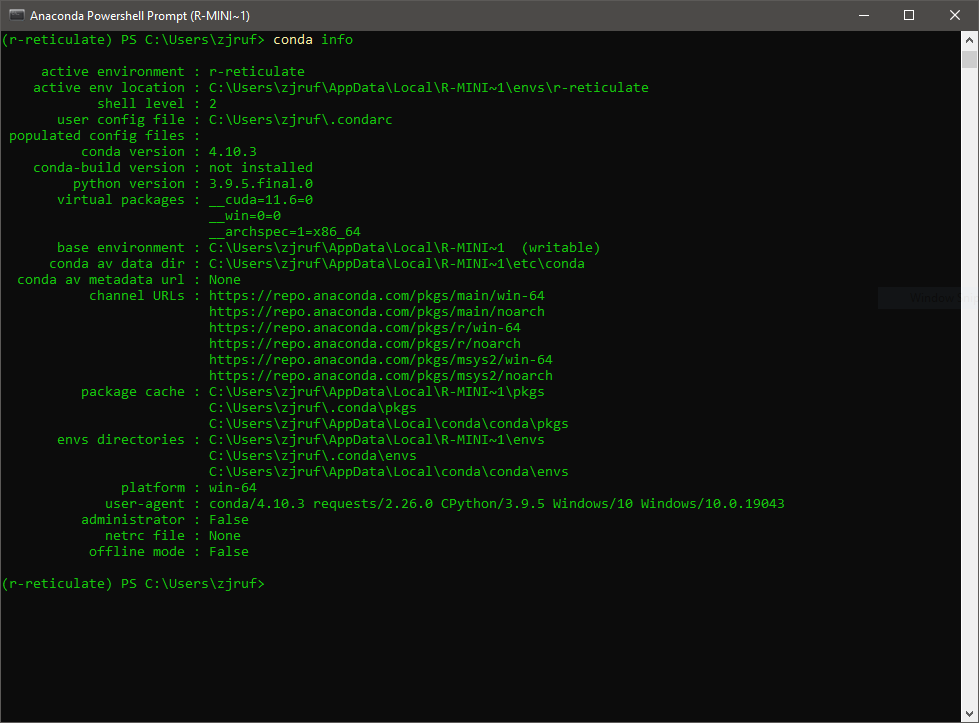
First, you will need to find the path to the correct Python executable. Open the Anaconda prompt, run

conda activate r-reticulate

to activate the correct environment, then run

conda info

The output shows the path to the r-reticulate environment folder on the line labeled active env location. Highlight this path and copy it.

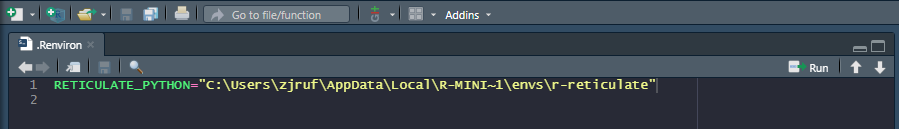


Next we will create a configuration file that defines this path as the default Python that RStudio should assume we want to use. In RStudio, Click File > New File > Text File. This will open a blank, unnamed file for editing in the code editor.

In the code editor, add the line

RETICULATE\_PYTHON=""

and paste the path to the r-reticulate environment folder between the quotation marks, like so:



Now click File > Save As... and enter .Renviron as the filename. You may get a warning that the new file will be hidden. Click OK to save the file anyway. The color of the text in the code editor may change as RStudio realizes the new file is a configuration file and applies the appropriate syntax highlighting. Finally, click Session > Restart R.

At the beginning of each new session, the R interpreter will read the .Renviron file and will set the RETICULATE\_PYTHON environment variable to the path you provided. Now if you run reticulate::py\_config(), the correct Python version should show up; there will be a note reading Python version was forced by RETICULATE\_PYTHON, and the app should behave correctly.

**Step 6.** Install the other necessary R packages.

Close the Anaconda prompt. Reopen RStudio and run the command

install.packages(c("tidyverse", "shiny", "shinyjs", "tuneR"))

The tidyverse is a family of packages that provide some useful data-handling functions. tuneR is a package for R-based analysis of audio and provides useful functions for dealing with wav files. shiny is a package for building interactive apps like this one, and is used to create the graphical user interface. shinyjs adds some useful JavaScript-based functionality to the base Shiny framework.

(If you have used previous versions of this app, you may notice that it is no longer necessary to install the keras or tensorflow R packages. If you already have these packages installed, you can leave them alone; they should not interfere with the app or prevent it from functioning.)

At this point you should have all the software that is required to use the Shiny\_PNW-Cnet app.

**Step 7.** (Optional) Install Bulk Rename Utility.

As the name implies, this is a simple utility program for quickly and efficiently renaming many files and is available from <https://www.bulkrenameutility.co.uk/>. The Shiny\_PNW-Cnet app assumes that your WAV files are named according to our naming conventions, in which each filename encodes some important information about where and when the recordings were made. The naming conventions are described in the next section.

As of November 2022 you can use the app itself to rename your files to match our naming conventions; Bulk Rename Utility will simply give you finer control over the process.

**Filename formatting**

Your audio files should have filenames structured as follows:

Area\_Site-Station\_Date\_Time.wav

This filename consists of four components separated by underscores: Study area or project code, location, date, and time. The Study area or project code can be alphanumeric; we use three-letter abbreviations for each of our study areas. Site and station ID should both be alphanumeric and should be separated by a single hyphen (-). The date should be in YYYYMMDD format. The time should be in HHMMSS format in 24-hour time. (This time and date format matches the default timestamp on files produced by Wildlife Acoustics ARUs.)

Here is an example of the filenames used in our spotted owl monitoring program:

TYE\_19317-A\_20210516\_201530.wav

The filename indicates that this file was recorded in the Tyee study area (TYE), in hexagon 19317, at station A, and the recording began on May 16, 2021 at 8:15:30 PM.

Note that both the spatial and temporal information in this filename is arranged in order from most general to most specific – from study area to hexagon to station within the hexagon, and from year to month to day, and finally from hour to minute to second. This is a useful property for keeping files organized.

Please avoid using spaces in the names of your files and folders, as these can cause issues with processing. Underscores are a safe alternative.

**Using the app’s renaming function**

In the filename shown above, there are two components: a location and a timestamp. The timestamp component is usually set by the ARU, so we typically just need to worry about the location. The app will construct the location component based on the names of the folders containing each file; you just need to structure and name your folders so that this information is available. Basically, each file’s parent folder should be named for the recording station or individual ARU. The folder containing *that* folder (the file’s grandparent folder) should be named for the field site, including the study area or project code.

As an example, say we have a set of files on SD cards that we have just retrieved from ARUs in the field. These were recorded using Song Meter ARUs with the default prefix (the ARU serial number), so the filenames all look like this:

S4A16302\_20210516\_201530.wav

The timestamp information is correct, but we need the filenames to include the location as well. We organize these files into folders corresponding to the site and the recording station. For instance, if ARU number S4A16302 was deployed at recording station A at site 19317 in the Tyee study area, then we would place the files from this ARU in a folder called Stn\_A within a folder called TYE\_19317, so the full path to this file might be

D:\TYE\_19317\Stn\_A\S4A16302\_20210516\_201530.wav

Similarly there would be files from other ARUs in folders called Stn\_B, Stn\_F, etc, all within the TYE\_19317 folder. And on the same external hard drive we might have folders for other sites, e.g. D:\TYE\_18163, D:\TYE\_18255, etc, each containing its own set of station folders.

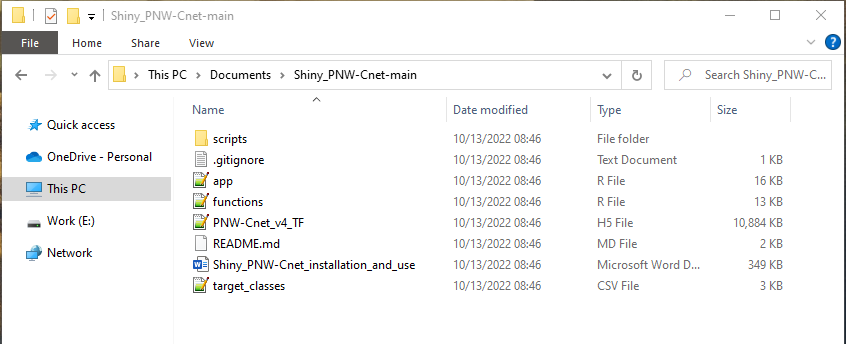
**Part 2. Processing audio using the Shiny interface**

These instructions assume you have installed all the necessary software as outlined above.

Note: This application works for our purposes, and we have tried to make it somewhat user-friendly, but it was not made by professional software engineers, so its operation is somewhat finicky. It makes some assumptions in order to work efficiently, and it may not work well (or at all) if those assumptions are broken. Be careful!

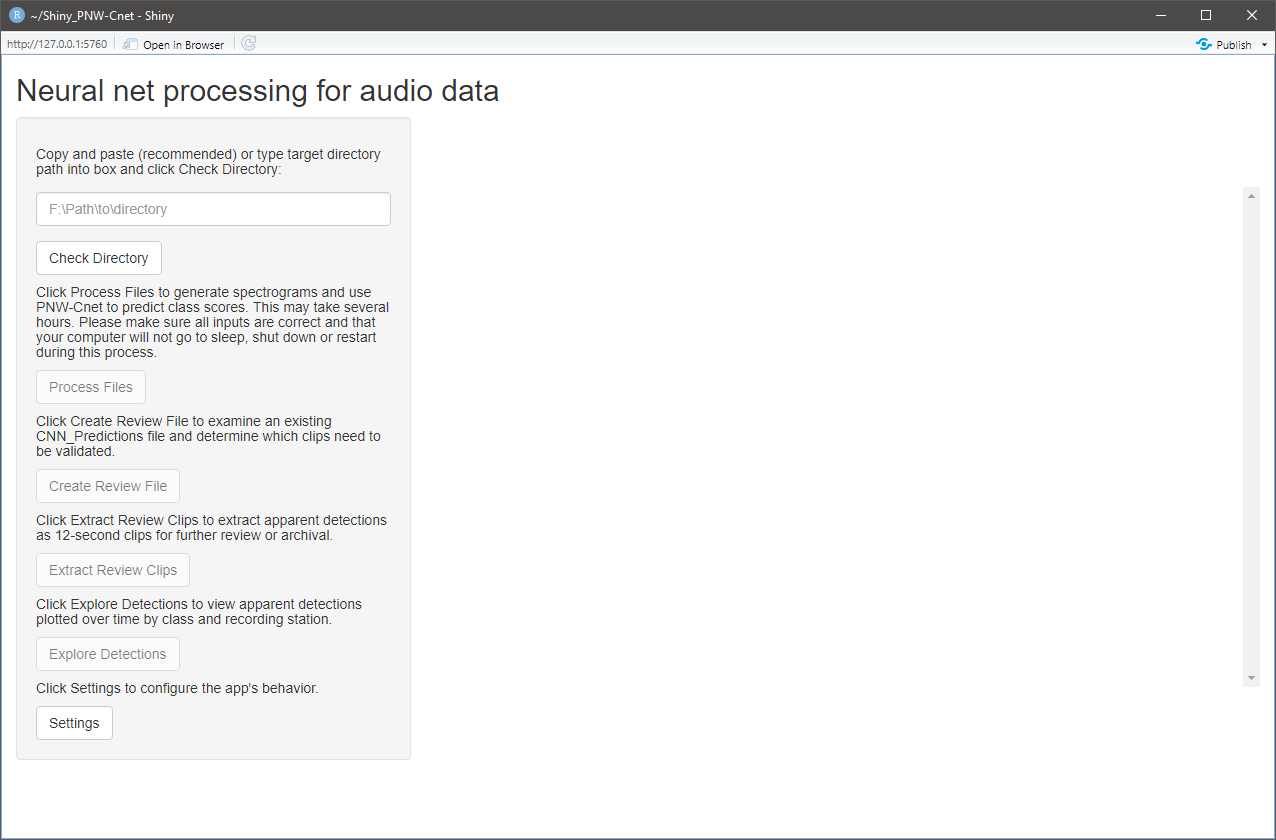
You will launch this application from within RStudio by typing a command into the Console pane; further operation will use the Shiny window that appears once the program is running. The Shiny window will take over RStudio’s input and output as long as it is open; you will not be returned to the normal prompt until this window is closed. However, some of the program’s output (progress bars, messages, etc.) will appear in the console pane, so it is helpful to keep an eye on RStudio while the application is running.

**Step 1.** Copy this folder (Shiny\_PNW-Cnet-main) to your RStudio home directory, i.e. whatever folder comes up if you open RStudio and run getwd(). The default in Windows is C:\Users\[username]\Documents. This home directory needs to contain a folder called Shiny\_PNW-Cnet-main, which should look like this:



If you installed SoX in a non-default location in Part 1 (Step 4), you will need to edit the app.r file so that the app knows where to find SoX. Open the app.r script file in RStudio and edit line 19, replacing C:/Program Files (x86)/sox-14-4-2 with the path to the correct folder.

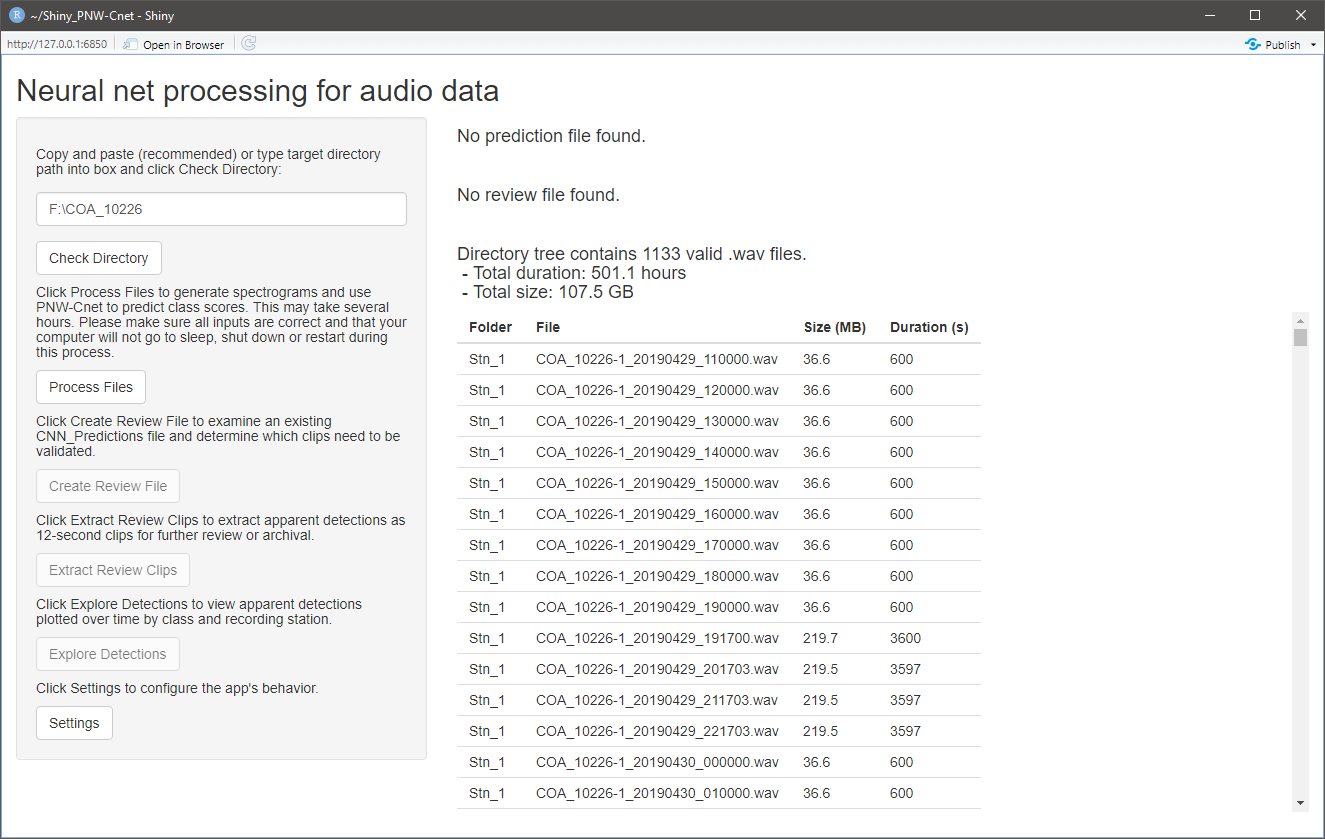
**Step 2.** In the RStudio console window, type shiny::runApp("Shiny\_PNW-Cnet-main") and hit Enter. (If you type "Shiny" within the parentheses and hit Tab, the full name of the folder should auto-complete.) RStudio will search the current working directory for a folder called Shiny\_PNW-Cnet-main containing a file called app.r; if it does not find these items, you will get an error. Assuming the folder and app file are found, a Shiny application window will appear. There will also be some output in the console pane as the app loads some R packages and establishes communication with RStudio. When first launched, the Shiny window will look like this:



This window consists of a side panel containing controls and a main panel containing information and output. Most of the controls in the side panel will be disabled on launch; these will become active as the app locates data to be processed and as various steps in the processing workflow are completed.

**Step 3.** Enter the path to the target directory and click Check Directory.

In File Explorer, copy the path to the directory containing the audio data that you want to process, then click the Check Directory button. The app will search the directory recursively (i.e., including subfolders) for files with a .wav extension. WAV files that are smaller than 500 KB or that have “\_part\_” in the filename will be ignored. In the main part of the app window you will see a verbal summary of the number of WAV files that were found and their total duration and size on disk. There will also be a table showing more detailed information on each file, including filename, parent folder (relative to the target directory), duration in seconds, and file size in megabytes.



**Step 4.** (Optional) Configure settings.

Click the button marked Settings to switch to another view showing various options that can be adjusted to change the app’s behavior. These settings are described in the Appendix.

The default options should be fine, but we recommend setting a different

**Step 5.** Click the button labeled Process Files.

[Note: The first time you attempt to process data, you may get a Windows Firewall pop-up saying that features of the program have been blocked. This should not actually prevent the app from working; you can close the pop-up and the program should continue to run.]

The app will create spectrograms in the form of image files in Portable Network Graphic (.png) format, representing non-overlapping 12-second segments of the audio. The images will be created in a new directory called Temp/images within the folder you specified in step 4. There will be 300 images for every hour of audio, so if you have a substantial dataset there will be a lot of them. We have not implemented a progress bar for this step, but as a crude workaround you can open the Temp/images folder in Windows Explorer and watch as the files are created.

This is usually the longest step; processing time will vary depending on your computer’s hardware capabilities and where the data are stored. We recommend using a computer with multiple cores, although with many cores the speed can also be limited by the read and write speeds of your storage device. As a very general rule of thumb, with data stored on external USB hard drives, you should be able to process approximately 100 hours of recordings per hour of processing time.

Once all the images have been generated, the app will load the trained PNW-Cnet model and start classifying the images in batches. (Sadly, due to changes in the internal workings of the app, it is no longer possible to create a progress bar here either. So if you don’t see one, don’t worry.) Once the full set of images has been classified, the app will write the results to a comma-separated values file called CNN\_Predictions\_[site name].csv and will open a dialog box indicating that audio data processing has finished.

In addition to the file containing the raw predictions, the app will create a summary file called [site name]\_detection\_summary.csv showing the number of apparent detections of each target class by station and date across a range of detection thresholds. Apparent detections are defined as images (equivalently, 12-second clips) to which the model assigned a score higher than the detection threshold for a given class.

If the model predictions were successfully written to file, the Create Review File and Explore Detections buttons will become active.

**Step 5.** [Optional] Click the button labeled Explore Detections.

The interface will change to look something like this:



The plot to the right is generated directly from the detection summary file generated in step 4. You can use the controls in the sidebar to visualize apparent detections of any of the 51 target classes at thresholds ranging from 0.05 to 0.99, summarized daily or weekly over the recording period.

Please note that these counts are intended **only** for data exploration purposes and should **not** be treated as objective evidence of species presence or absence without further human review. Put simply, AI is far from perfect and lots of weird stuff happens in the woods. While the false positive rate is quite low for many of these classes, especially at higher thresholds, verification of apparent detections by knowledgeable human reviewers is an important and necessary part of the process (which brings us to the next step).

Click the button labeled Back to inputs to return to the previous view.

**Step 6.** Click the button labeled Create Review File.

This will search the model prediction file for lines where either a) the score for one of the spotted owl classes (STOC or STOC\_IRREG) was ≥ 0.25 or b) the score for any of the other target classes was ≥ 0.95, and write these lines to a new file. This is the set of clips that are flagged for human review under our current validation procedure. The app will write a message to the console pane that says Attempting to create review file based on CNN\_Predictions\_[location].csv… Once the review files have been created the console will report the number of lines written to file. This button will actually create two files containing similar information; one ends in \_review.csv and is simply a filtered version of the CNN\_Predictions file, while the other ends in \_review\_kscope.csv and is intended to be used with Kaleidoscope. See the section below titled "Using the review\_kscope file with Kaleidoscope" for details.

**Step 6.** [Optional] Click Extract Wav Files.

[Note: We do not recommend this as part of the normal review process. You can easily review apparent detections of your species of interest using the \_review\_kscope.csv file in combination with Kaleidoscope without having to extract them as short clips first, which saves a fair amount of time and disk space. See the following section for details.]

This will create a new folder called Review in the target directory specified in step 3. The Review folder will have subfolders for each of the target classes, which will contain the clips of that class that warrant further review. The nested directory structure that is created is divided up by target class, then by recording station, then by week.

The app will print a message to the console that says Extracting wav files beginning at [time]… Extracting the clips may take some time. When all the new clips have been made there will be another message that says Finished at [time]. [n] clips extracted. Once you get the message that review clips have been extracted, you can close the Shiny window and exit RStudio. Below is an example of console output from the processing of a very small test set.

Clips are extracted as .wav files with the same parameters (bit depth, sample rate, number of channels) as the input data. Depending on the dataset and how much other data is on the same hard drive, storage space may be an issue when extracting clips for review. For instance, our data are single-channel .wav files with a sample rate of 32 kHz and a bit depth of 16 bits, which works out to roughly 225 MB per hour of audio. This means that every 12-second clip takes up roughly 750 KB, and 10,000 such clips will take up about 7.5 GB. You can use the number of lines in the Review file to judge how much free space you will need before extracting clips for review.

Once clips for review have been extracted, you can close the Shiny window by clicking the X in the upper right-hand corner or by pressing Esc from the RStudio console window.

**Using the review\_kscope file with Kaleidoscope**

*Note: We are not affiliated with Wildlife Acoustics, Inc. This guide has not been reviewed or endorsed by Wildlife Acoustics and may not represent specifically authorized or intended usage of Kaleidoscope.*

*For detailed instructions on using Kaleidoscope, see the User's Manual available at* [www.wildlifeacoustics.com](http://www.wildlifeacoustics.com).

When you click the "Create Review File" button, the app will create two different files. Both have a .csv extension and contain comma-separated text; the name of one file ends with \_review.csv and the other ends with \_review\_kscope.csv. The review\_kscope file is meant to be used with Kaleidoscope, an audio processing program available from Wildlife Acoustics, Inc. Kaleidoscope is available in a free (Lite) or paid (Pro) version; the free version is sufficient to review apparent detections.

The review\_kscope file is a table which points to specific locations within the original audio in a format that Kaleidoscope can understand, allowing you to browse these locations and apply species tags. This approach has a few advantages: You do not need to take the time to extract thousands of short clips or allocate extra disk space for them. You can easily examine the audio surrounding a given clip for additional context. You can sort the review file in different ways, allowing for more flexibility in how you browse the clips. Kaleidoscope also seems to be slightly faster at loading short portions of a long audio file than many shorter clips.

**Setting up**

When you launch Kaleidoscope you will see the main window, which has a number of tabs; the Batch tab will be in focus. This lets you specify an Input and an Output directory. In both of these fields, enter the path to the target directory that you specified when you processed the audio using the app, and it should contain the CNN\_Predictions file, the review file, and the review\_kscope file. Now click the Signal Params tab. In the field marked "(s) Maximum inter-syllable gap," enter 0. By default Kaleidoscope will add "padding" equal to this number of seconds before and after each clip. It is better to avoid this so that you are seeing the same things that the model sees.

In the Kaleidoscope main window, click File > Open Results. This will bring up a file selection dialog. By default it will show you the folder specified in the Output directory field. Select the review\_kscope file and click Open. This will bring up two additional windows: the Viewer window, which features waveform and spectrogram representations of the audio, and the Results window, which contains a table representation of the review\_kscope file.

In the Results window, click File > Edit columns. In the window that pops up you can use the Show, Hide, Move Up and Move Down to change which columns are visible and in what order. The columns that we recommend having visible, and their interpretation, are as follows:

IN FILE: The name of the original audio file.  
PART: The segment of the original audio file that we are looking at. An hour of audio is divided into 300 12-second segments, labeled part\_001 to part\_300.  
MMSS: Offset from the start of the file in minutes and seconds. Related to PART, e.g. part\_005 represents the fifth 12-second segment of the audio file, which begins at an offset of 00:48. This is included for human readability.  
TOP1MATCH: The most likely class according to the model. Same as "Sortlabel" in the regular review file. Note that the app prioritizes spotted owls and applies a lower threshold to this class, so clips labeled STOC are those which had a score of 0.25 or more for the STOC or STOC\_IRREG class, regardless of the scores for the other classes.  
TOP1DIST: The actual score assigned to the most likely class by the model, indicating the strength of the match. Ranges from 0 (no match) to 1 (very close match).  
MANUAL ID: If you apply species tags to a clip, they will appear in this column. When the file is originally created this field will be blank.  
SORT: A combination of apparent label (same as TOP1MATCH), Station, and Week. This is included so the user can quickly confirm the presence of each species at each station within each week, facilitating the efficient generation of weekly encounter histories.

The other columns are mostly irrelevant for our purposes and can be ignored. Click OK in the Edit Columns window to apply the selection to the results window.

In the Viewer window, click File > FFT Settings. Change FFT SIZE to 1024 and WIN SIZE to 256 and click OK. The sliders under the spectrogram control brightness / gain (which also controls playback volume) and contrast / dynamic range; I generally keep them at 0 and -96, respectively.

You can browse through clips in the results\_kscope file by clicking a row in the Results window and using the up and down arrow keys to move to the next and previous row, or by clicking on the spectrogram in the Viewer window and using the up and down arrows, or using the < and > buttons below the spectrogram on the right side. If the Auto next file box is checked, you will automatically advance to the next clip after applying a tag.

Tags can be applied in a few ways as well. In the Viewer window, you can type a tag into the Identification field and hit Enter. You can use the buttons under this field as well; the button on the left will automatically populate with the value in the TOP1MATCH field, while the button on the right will always apply the Noise tag. There are four rows of rectangular buttons at the bottom of the Viewer window; these can be reprogrammed by right-clicking them and typing the tag you want them to apply. Buttons in the top row are linked to number keys 1 through 8 so you can use these keys to apply tags from this row, which is often convenient. You can also select a range of rows in the Results window and click File > Bulk ID selected rows to tag all the selected rows with a particular tag.

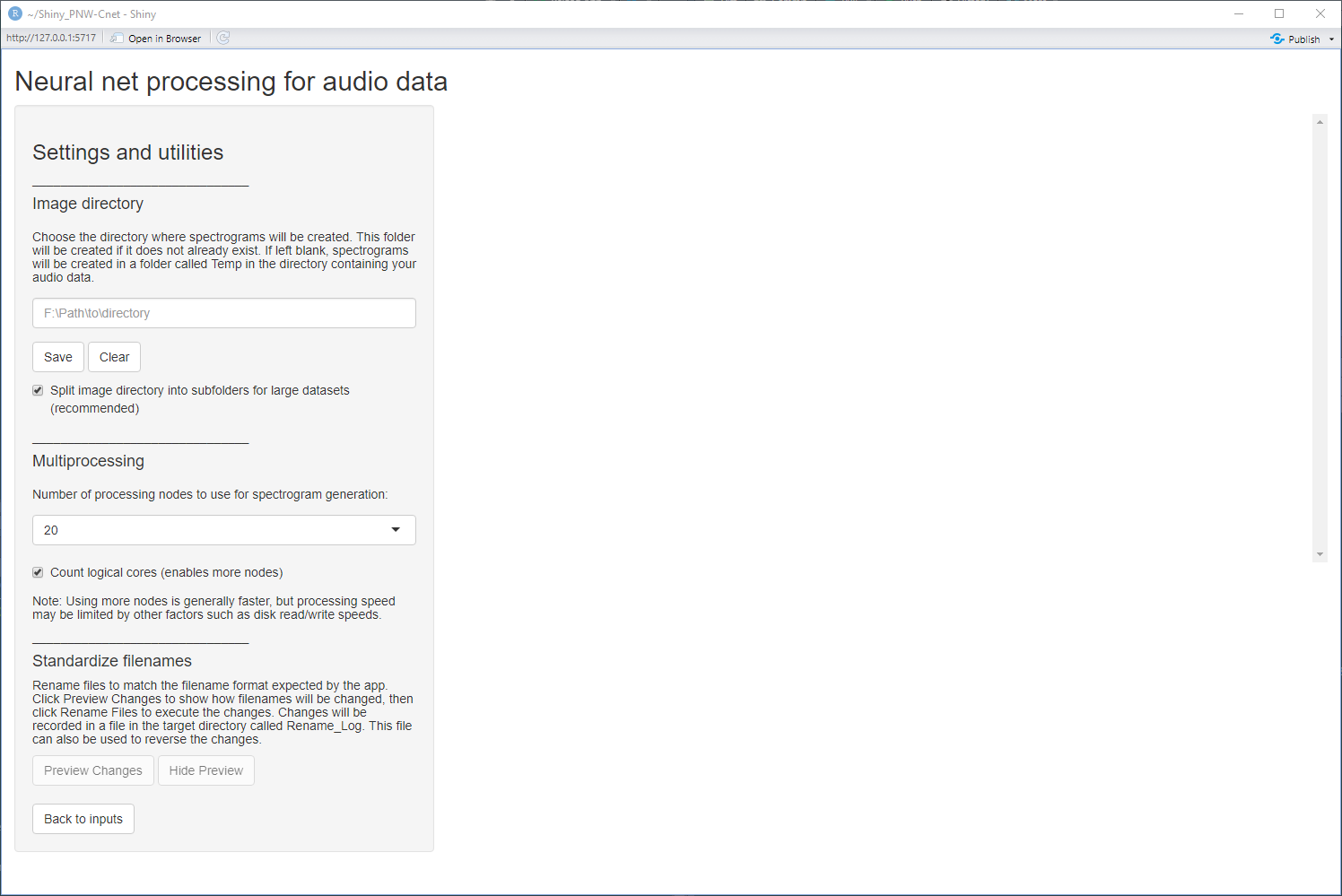
You can sort the Results table by clicking on the column headers to browse the clips in different ways. Clicking a column once will sort the table by that column in ascending order, and clicking it again will reverse the order. To sort by multiple fields, click the column headers in order from the finest to the broadest level of the sort. For instance, sorting by TOP1DIST in descending order and then by SORT will organize the table by class-station-week and show you the best matches first within those groupings. Sorting by TOP1DIST in descending order and then by TOP1MATCH will allow you to browse the best matches for each class within the whole dataset. Use the << and >> buttons in the Viewer window to move to the previous or next "folder," defined as the next value of the broadest level of the sort. For instance if you have sorted the table in the manner just described, clicking >> will take you from a row with BUVI in the TOP1MATCH field to the first row with COCO in that field.

Note that you must click File > Save in the Results window for any tags you have applied to be preserved in the review\_kscope file. We recommend saving frequently to avoid losing work in the event that Kaleidoscope crashes or you lose power.

**Appendix**

*Settings and Utilities*

From the app's main window, if you click the button labeled Settings, the window will change to look like this:



This view allows you to configure the app's behavior and perform ancillary tasks related to processing. The configurable options are described below.

*Image Directory*

When the app processes audio data, it generates spectrograms which are then saved as image files. By default, these files will be saved in a folder called Temp in the directory containing your audio data. You can use the Image Directory field to tell the app to save the spectrograms in a different folder. The folder you specify will be created if it does not already exist. If you click "Save," the app will remember this folder and use it for future processing. The app will create subfolders within this folder to hold spectrograms from different sets of audio data.

Setting a custom image directory is useful for a couple of reasons. First, these image files take up disk space – generally about 50 MB per hour of audio, which adds up quickly. Saving these files to a different drive means you don’t need to worry about running out of space on the drive containing your audio data. Second, it can make processing faster, especially if your audio files are stored on an external device like a USB hard drive. Saving spectrograms to an internal hard drive, especially a solid-state drive, will typically be much faster than saving them to a USB drive, which can be limited both by the transfer speed of the USB connection and by the write speed of the drive itself.

The check box labeled “Split image directory into subfolders for large datasets (recommended),” when checked, tells the app to divide the spectrograms into multiple folders instead of lumping them all into one directory. Piling up hundreds of thousands of files in the same folder can make operations dealing with those files sluggish; dividing them up helps to avoid this issue. We recommend keeping this box checked, as there is no real downside to dividing up the image folder.

*Multiprocessing*

Generating spectrograms from audio files is an example of a task that can benefit from parallel computation, in which multiple processing cores can work on several tasks simultaneously (or *in parallel*) to get work done faster. Virtually all computers being sold today have multiple cores, and the app takes advantage of this when generating spectrograms. The dropdown allows you to set the size of the multiprocessing cluster that the app will use to generate spectrograms; the number of nodes in this cluster roughly corresponds to the number of cores that will be used. By default, the app will use as many nodes as your computer has cores; use fewer nodes if you want to leave some headroom for doing other work on the same machine.

Uncheck the box labeled "Count logical cores" if you want to restrict the size of the cluster to the number of physical cores in your CPU. Hyperthreading allows many CPUs to behave as if they had a larger core count than they actually do (e.g. a CPU might be advertised as having 8 cores and 16 threads). The app takes this into account and will default to using the number of threads, or logical cores, to set the size of the multiprocessing cluster.

As noted, using more cores will generally result in the spectrograms being generated at a faster rate, but the increase is not linear since there are other factors involved, including the speed of the data connection and the write speed of the storage device.

*Standardize filenames*

The app can automatically rename your files to match the naming conventions outlined earlier in this document, eliminating the need for external programs like Bulk Rename Utility.

**Please note:** Editing filenames in bulk can save a ton of time, but it is also possible to destroy a lot of data very quickly if you are not careful. This feature is very new, and while we have tried to build in some basic safeguards, they may not be sufficient in all cases. Please exercise caution when using this feature and make sure you have another copy of your data in case things go catastrophically wrong.

When using this feature, the app will construct a prefix for your files using the names of the two lowest levels of folders containing each .wav file. If there is not already a timestamp present in the filename, the app will also construct its own timestamp in the proper format using the file's time of modification. The filename will consist of the prefix and the timestamp separated by an underscore, plus the .wav file extension.