Classify radio signals from space

Tarit Sengupta
info.taritsengupta01@gmail.com
Techno Main Salt Lake, Kolkata, West Bengal

ABSTRACT

The data we are going to use consists of 2D spectrograms of deep space radio signals collected by the Allen Telescope Array at the SETI Institute. We will treat the spectrograms as images to train an image classification model to classify the signals into one of four classes. By the end of the project, you will have built and trained a convolutional neural network from scratch using Keras to classify signals from space.

Keywords— Radio Signals, Space Signals, Seti Data, Cnn Model, Model Train

1. INTRODUCTION
In this project we use 2D Spectrograms of deep-space radio signals collected by the antennas at the SETI (The search for extra-terrestrial intelligence) Institute.
So the original signals were not 2-D spectrograms but were time-series data collected and downloaded by SETI. We’re going to work with 2-D spectrograms which were created by transforming the input time-series data. So, we’re going to use the spectrograms as images to train a classification model. By the end of this blog, you will learn to build & train a CNN by scratch to classify these radio signals from space.

I’m expecting that you have a prior knowledge about coding in Python and have a basic idea about how a neural network performs under the hood especially convolutional neural network as I won’t be getting into the maths. We’ll begin with importing the libraries.

We’ll be using Tensorflow v 2.2.0.

2. LOAD AND PRE-PROCESS SETI DATA

We use pandas to read the CSV file where the images are stored. But how can images be stored in a CSV file?

The spectrograph images were converted into their raw pixel intensity values and were normalized so the values lie between 0 and 1. They are then converted into an array by stretching them. Therefore, each row of the CSV file corresponds to a single image.

The label was found to be one hot encoded in to a vector of 1,4.(no. of classes).
- 1,0,0,0 is squiggle
- 0,1,0,0 is Narrow-band signal
- 0,0,1,0 is Noise
- 0,0,0,1 is Narrow-band-drpd signal
We then try to see the shape of the DataFrames and we find that the output:

```python
train_images.shape
```

```
(3200, 8192)
```

Since all of the images were converted into 2-D spectrograms we didn’t have information about RGB channels therefore we put the 1 in the last dimension or else we would have put a 3 had it been coloured or colour had an importance here. Also, we converted the data frames into the array format(.values) because our neural network can’t take in the Data Frame format and our image classifier wants the data to be in that specific format which was achieved after reshaping.

Let’s try to visualize the images using matplotlib:

**Task 3: Plot 2D Spectrograms**

Generate batches of tensor image data with real-time data augmentation.

```
plt.figure(figsize=(10,10))
for i in range(9):
    pt.imshow(x_train[i, ...], cmap='gray')
plt.show()
```

Looks like all three of the randomly selected images are of the type Narrow-band.

You can re-run the function to see a different set of images. The real benefit of deep learning comes when we train on batches of data, so let’s generate batches of tensor image data with real time data augmentation. Feel free to experiment with your techniques of augmenting your data. For now I’ll just be flipping them horizontally.
3. CREATE TRAINING AND VALIDATION DATA GENERATORS

As the data was already processed and normalized, we don’t need to do a lot of pre-processing. We are finally ready to build our model.

Let me give you a brief introduction to CNN if you’re a bit rusty of the knowledge on that. So, CNNs are just a type of feed-forward neural network consisting of multiple layers of neurons that have learnable weights and biases, so each neuron in a layer receives an input from a proceeding layer processes the input and optionally follows it with non-linearity. So that your model not just learns a linear sum of the data but also complex non-linear functions.

The network for CNN has multiple layers such as convolution layers, maxpool layers for down-sampling, dropout for regularization, and all these layers followed by one or more fully-connected layers at the end. So, at each layer, a small neuron process portion of input images and output of these collections are then tiled so that the input region overlap to obtain a high-resolution representation of the input image, and this process is repeated for every such layer.

The bottom line is that CNN takes a complex pattern in images and breaks them down to simple patterns through multiple hierarchical layers. Max-pooling is simply a non-linear down sampler, it partitions the input image into a set of rectangles and then finds the max value of that region.

We will now use Keras to implement our CNN. We’ll first begin by importing all the util files from Keras and tf:

Here’s the implementation of the model in Keras:

```python
In [11]: # Initialising the CNN
model = Sequential()
    # 1st Convolution
    model.add(Conv2D(32, (3,3), padding='same', input_shape=(64, 128,1)))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # 2nd Convolution layer
    model.add(Conv2D(64, (3,3), padding='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    # Flattening
    model.add(Flatten())
    # Fully connected layer
    model.add(Dense(1024))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='softmax'))

_initializer (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.
Instructions for updating:
call initializer instance with the dtype argument instead of passing it to the constructor.
```
Before we compile the model let’s define a learning rate scheduler. Use of a scheduler is to decay the learning rate after some ‘time’.

4. LEARNING RATE SCHEDULING AND COMPILE THE MODEL

This will keep mitigating the learning rate like $0.005 \times (0.96)^5 = 0.004076$ after each of the specified step. Here’s the summary of the model:

5. TRAIN THE MODEL

Before we start training our model, we need to define some call-backs if we are interested in saving our model at certain checkpoints or on certain optimizations to have the least validation loss.
History object is the place we need to pass in our data generator that we created in the previous step. The model train for about 10 mins on Google Colab without GPU or TPU.

Let’s now evaluate our model. Its accuracy is around 74% which was considered a benchmark back in 2017 when this dataset was launched in a SETI hackathon. Let’s build a confusion matrix to analyse it better:
This was a basic model and it achieved an accuracy of around 74%. You can definitely increase its accuracy by using transfer learning and fine tune it on pre-trained CNN networks such as VGG and Image-net.