**Build a Custom GAN Model (Optional): Part 1**

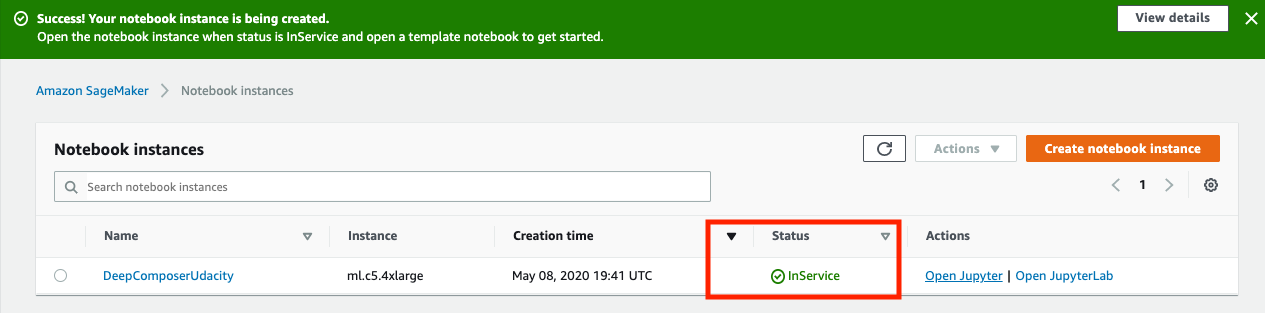
To create the custom GAN, you will need to use an instance type that is not covered in the Amazon SageMaker free tier. Y**ou may incur a cost if you want to build a custom GAN**.

You can learn more about SageMaker costs in the [Amazon SageMaker pricing documentation](https://aws.amazon.com/sagemaker/pricing/).

**Important:** This is an ***optional*** exercise.

**Setting Up the AWS DeepComposer Notebook**

1. Go to [the AWS Management Console](https://aws.amazon.com/console/) and search for *Amazon SageMaker*.
2. Once inside the SageMaker console, look to the left-hand menu and select **Notebook Instances**.
3. Next, choose **Create notebook instance**.
4. In the Notebook instance setting section, give the notebook a name, for example, DeepComposerUdacity.
5. Based on the kind of CPU, GPU and memory you need the next step is to select an instance type. For our purposes, we’ll configure a ml.c5.4xlarge
6. Leave the **Elastic Inference** defaulted to none.
7. In the Permissions and encryption section, create a new IAM role using all of the defaults.
8. When you see that the role was created successfully, navigate down a little way to the Git repositories section
9. Select **Clone a public Git repository to this notebook instance only**
10. Copy and paste the public URL into the Git repository URL section: https://github.com/aws-samples/aws-deepcomposer-samples
11. Select **Create notebook instance**
12. Give SageMaker a few minutes to provision the instance and clone the Git repository
13. When the status reads "InService" you can open the Jupyter notebook.



**Exploring the Notebook**

Now that it’s configured and ready to use, let’s take a moment to investigate what’s inside the notebook.

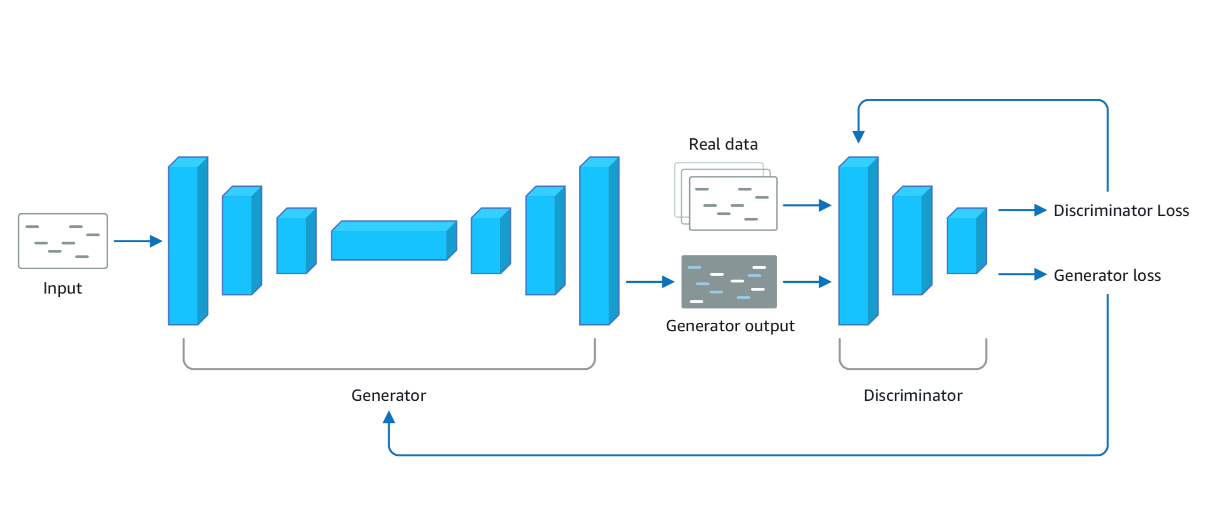
**Open the Notebook**

1. Click **Open Jupyter**.
2. When the notebook opens, click on "**gan**".
3. When the lab opens click on **GAN.ipynb**.

**Review: Generative Adversarial Networks (GANs).**

GANs consist of two networks constantly competing with each other:

* **Generator network** that tries to generate data based on the data it was trained on.
* **Discriminator network** that is trained to differentiate between real data and data which is created by the generator.

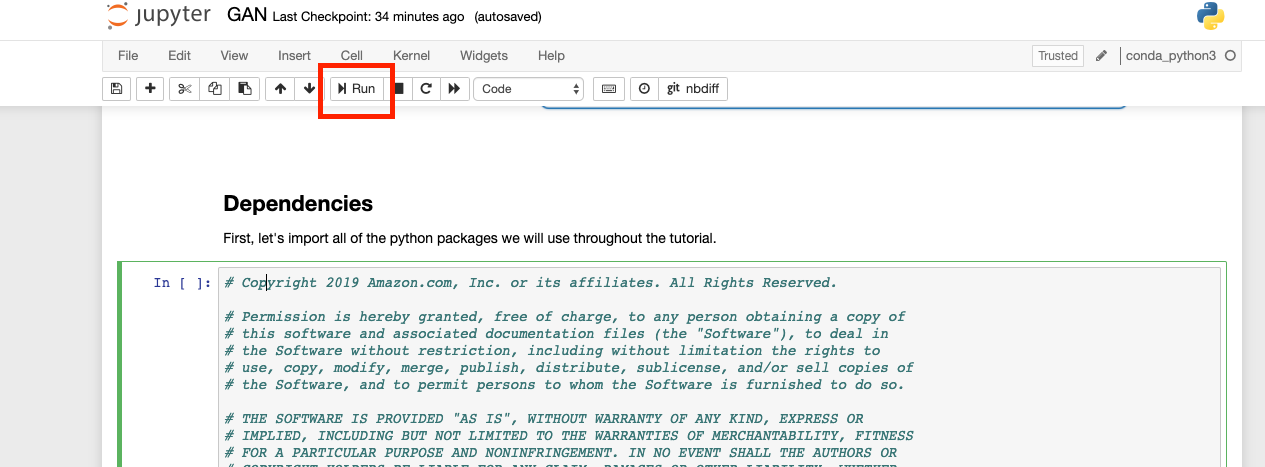


A diagram of generator and discriminator.

**Set Up the Project**

1. Run the first **Dependencies** cell to install the required packages
2. Run the second **Dependencies** cell to import the dependencies
3. Run the **Configuration** cell to define the configuration variables

***Note***: While executing the cell that installs dependency packages, you may see warning messages indicating that later versions of conda are available for certain packages. It is completely OK to ignore this message. It should not affect the execution of this notebook.



Click **Run** or Shift-Enter in the cell

**Good Coding Practices**

* Do not hard-code configuration variables
* Move configuration variables to a separate config file
* Use code comments to allow for easy code collaboration

**Data Preparation**

The next section of the notebook is where we’ll prepare the data so it can train the generator network.

**Why Do We Need to Prepare Data?**

Data often comes from many places (like a website, IoT sensors, a hard drive, or physical paper) and it’s usually not clean or in the same format. Before you can better understand your data, you need to make sure it’s in the right format to be analyzed. Thankfully, there are library packages that can help! One such library is called [NumPy](https://numpy.org/), which was imported into our notebook.

**Piano Roll Format**

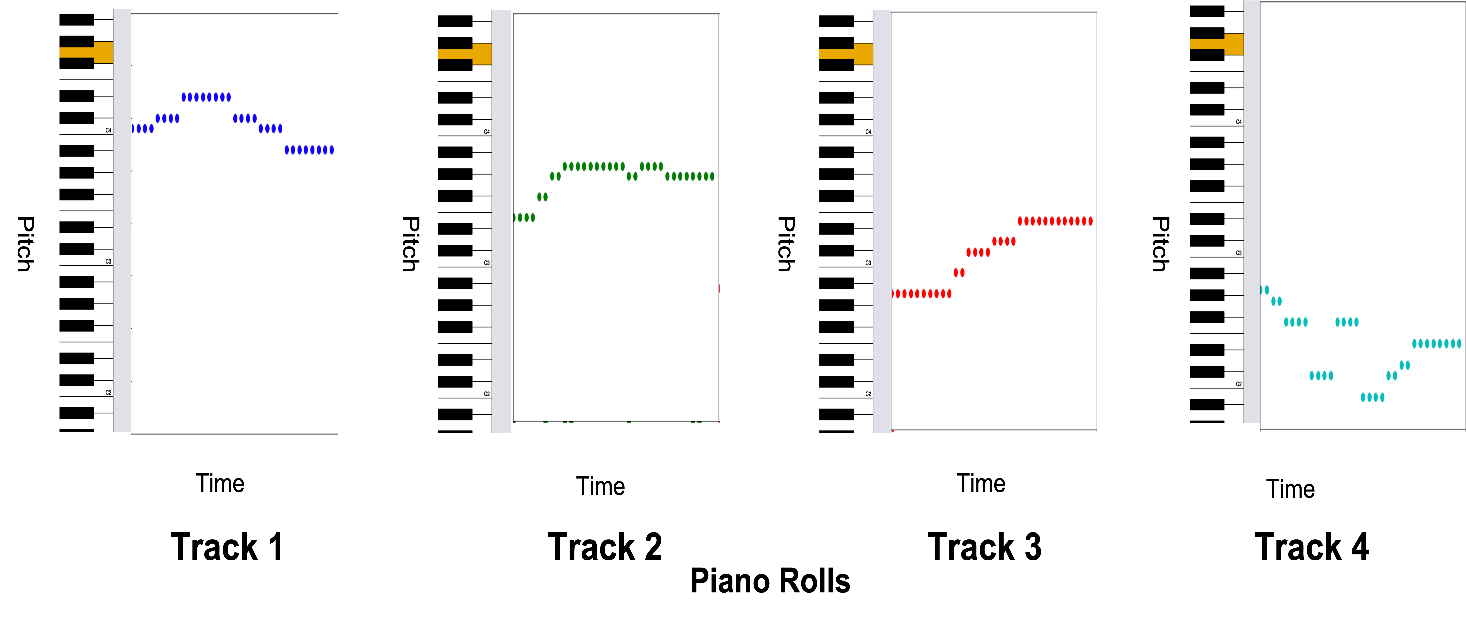
The data we are preparing today is music and it comes formatted in what’s called a “piano roll”. Think of a piano roll as a 2D image where the X-axis represents time and the Y-axis represents the pitch value. Using music as images allows us to leverage existing techniques within the computer vision domain.

Our data is stored as a NumPy Array, or grid of values. Our dataset comprises 229 samples of 4 tracks (all tracks are piano). Each sample is a 32 time-step snippet of a song, so our dataset has a shape of:

(num\_samples, time\_steps, pitch\_range, tracks)

or

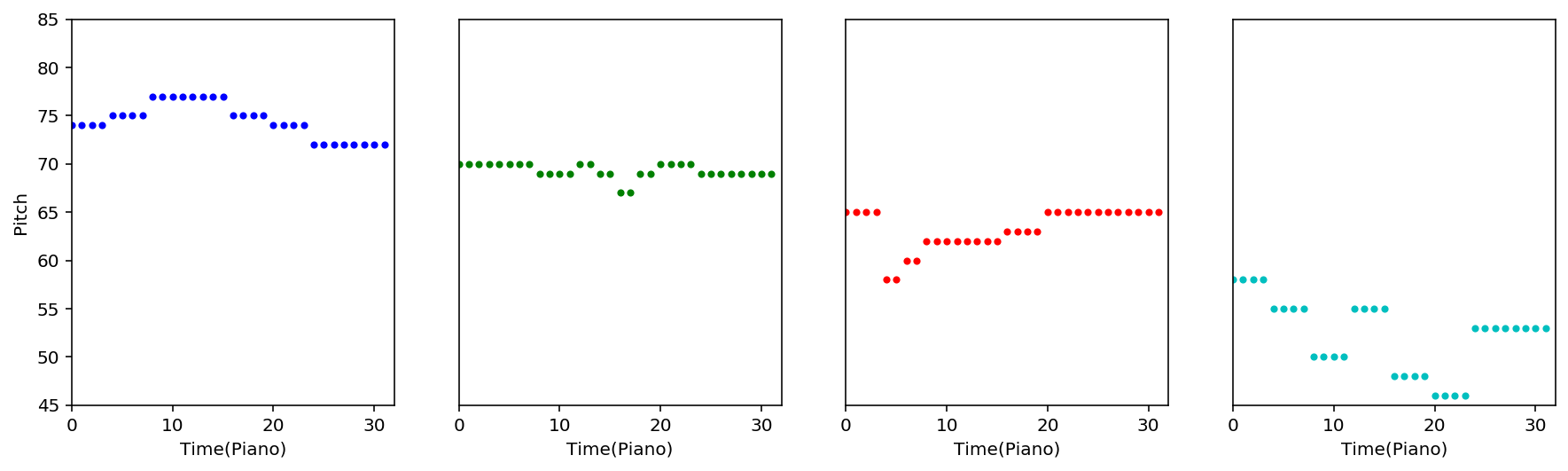
(229, 32, 128, 4)



Each Piano Roll Represents A Separate Piano Track in the Song

**Load and View the Dataset**

1. Run the next cell to play a song from the dataset.
2. Run the next cell to load the dataset as a nympy array and output the shape of the data to confirm that it matches the (229, 32, 128, 4) shape we are expecting
3. Run the next cell to see a graphical representation of the data.



Graphical Representation of Model Data

**Create a Tensorflow Dataset**

Much like there are different libraries to help with cleaning and formatting data, there are also different frameworks. Some frameworks are better suited for particular kinds of machine learning workloads and for this deep learning use case, we’re going to use a [Tensorflow](https://www.tensorflow.org/) framework with a [Keras](https://keras.io/) library.

We'll use the dataset object to feed batches of data into our model.

1. Run the first Load Data cell to set parameters.
2. Run the second Load Data cell to prepare the data.

;