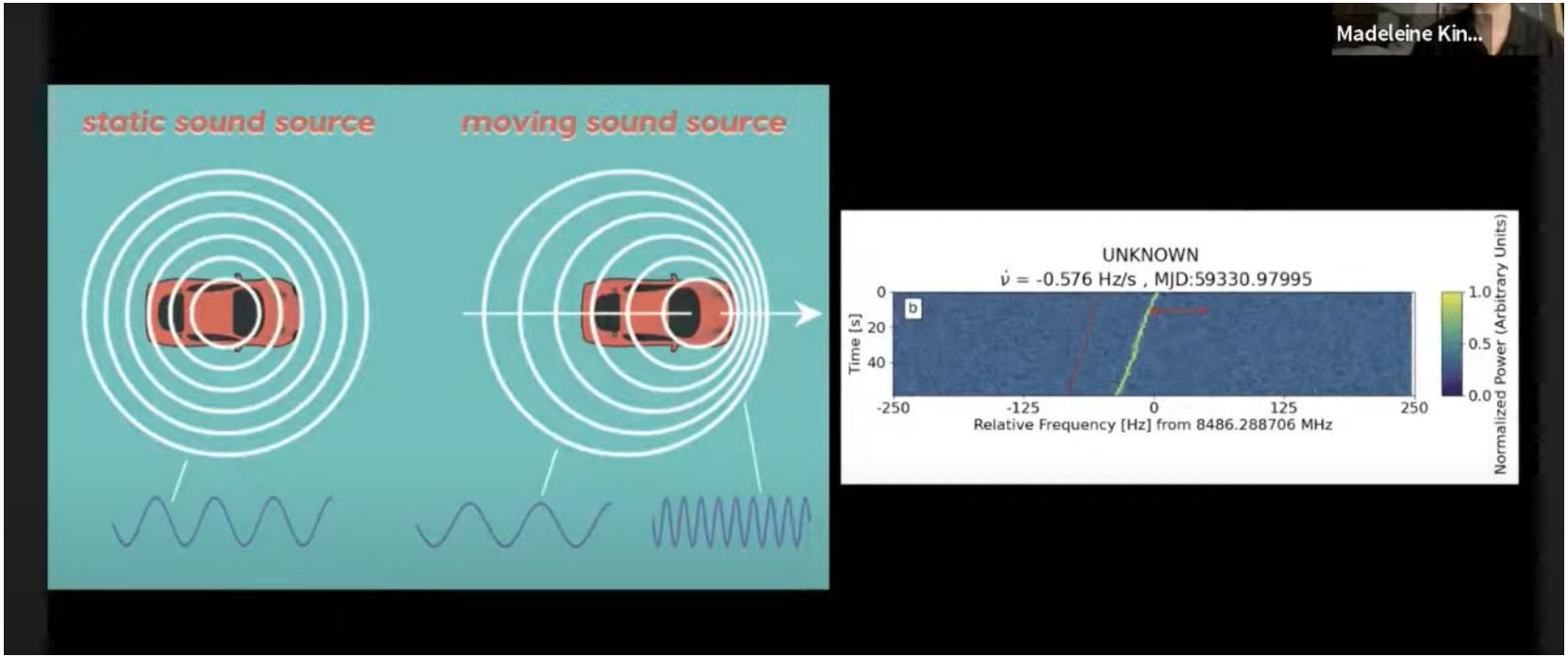
# Problem Statement

SETI Institute is capturing radio signals data and searching for specific patterns in the radio signals to find proofs for extraterrestrial intelligence.

But this data is too large (2 Million Gb per Year) to be handled by explicitly programmed computers and accuracy of results extracted becomes highly important along with the efficiency.

Therefore, SETI Institute is constantly in search of highly optimized Deep learning models and AI solutions to distinguish between different radio signals that may decrease the search time.

# Doppler Effect and Narrow band drd signals



# Proposed Solution

**We propose a Deep learning approach to classify radio signals into one of the 4 given categories. We first analyze the SETI dataset and visualize these radio signals by using 2D Spectrograms. The major goal of the project is to find a robust signal classification algorithm that could further assist the E.T. radio communication.**

**Aim: To Classify the Radio Signals from telescope using Convolutional Neural Networks (CNN)**

# What are we looking for?

- The data from a survey of galactic center performed at the Allen telescope Array.
- The signal captured by ATA in radio frequencies between 3.36 to 9.12 GH.
- Out of all the signals we want to find Narrow band drd: Narrow band doppler drifting radio signals and squiggles: unknown signals.

# Objectives

**While doing this project our main objectives will be :**

- Understanding the Radio-Signal Simulation Data or SETI Data
- Loading & Preprocessing the Kaggle dataset to get data visualisations
- Visualizing the signal data using 2D spectrograms
- Creating a CNN model
- Classifying the SETI signals into various categories.
- Evaluating the CNN model

**Prefer Deep Learning when:**

- Huge training data available for making accurate decisions.
- Possess high-computing power (i.e., CPU, GPU, TPU, etc.) - to allow intensive model training and good application performance.
- Uncertain about the positive feature-engineering outcome (i.e., selecting the most suitable feature(s) yielding the desired outcome), especially in unstructured media (audio, text, images).
- Deployment restricted to high-performance devices (i.e., unsuitable for embedded, micro-controllers). Less/ no domain expertise is available

**Stick to traditional image processing when:**

- Limited (annotated/ labeled) data available.
- Lack of high storage and computing power.
- Cheaper solution desired  Desire flexible deployment over a range of hardware.
- Good domain expertise present.

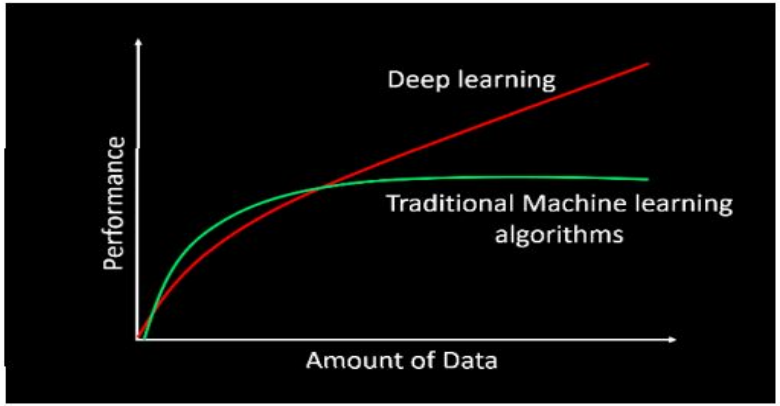
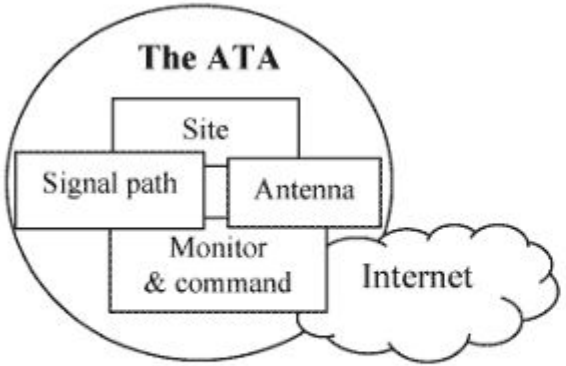
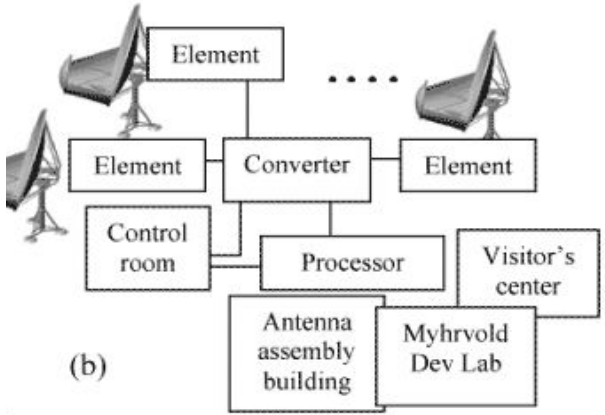



*Figure 2: Data vs Performance Comparison*

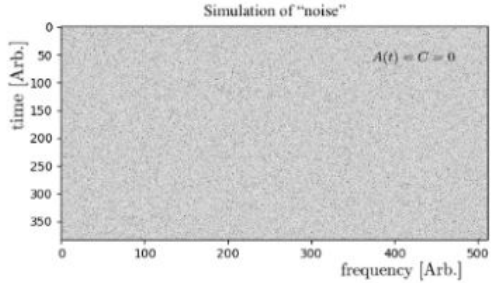
# Signals received by ATA

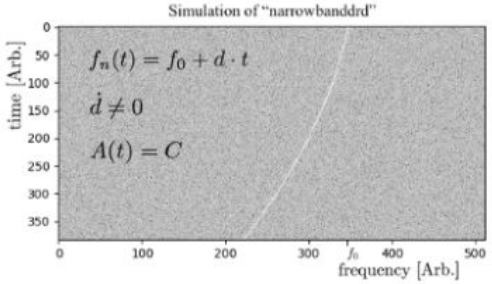
- The antenna collects the radiation from space.
- The signal path brings the radiation from the feed(which is located at the antenna focus)back to the user.
- The monitor and command systems allow the dishes to be accurately moved,and the signal path controlled.
- The site includes the overall antenna configuration, as well as other infrastructure.

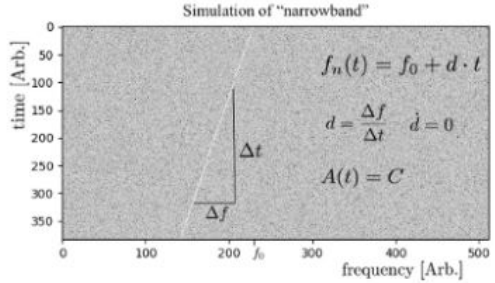


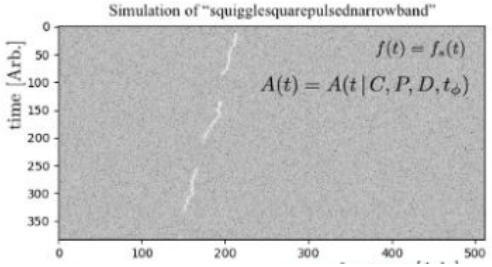



# 4 SIGNAL CATEGORIES:



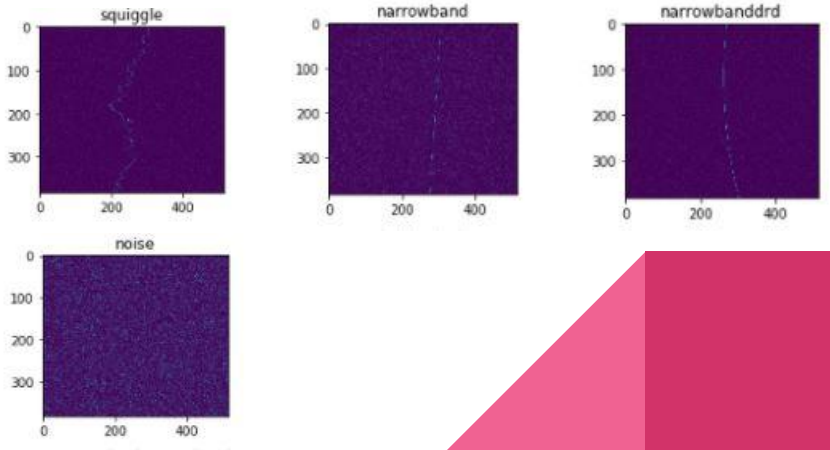



# Dataset

The data we are going to use here consists of 2D spectrograms of deep space radio signals collected at the SETI Institute by using the Allen Telescope Array. We will use that spectrograms as images to train an image classification model to classify the signals into one of four classes.

There were a lot of challenges while using the real data, so a set of simulated signals were built to approximate the real signal data. For this dataset there are 4 diff. classes and they are named on the basis of how they look in spectrogram :

- Squiggle
- Narrowband
- Narrow band drd
- Noise

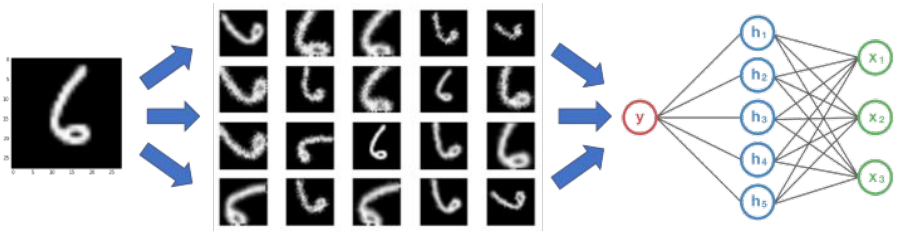


# Image Augmentation

Process of doing minor changes such as flips or translations or rotations to the images in order to increase the relevant data in the dataset

As more is the data available, more accurately CNN can classify the images !

Your neural network is only as good as the data you feed it!




**Types :**

1. Flip
2. Rotation
3. Scale
4. Crop
5. Translation
6. Gaussian Noise

**Code:**

```
# Performing the Keras Preprocessing
# Only one aspect Data Augmentation is Performed
# i.e ->> Randomly fliping the images along the Horizontal axis
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Fitting the DataGenerator so that it can learn
#the Statistics and properties about the training images
datagen_train = ImageDataGenerator(horizontal_flip = True)
datagen_train.fit(x_train)

datagen_val = ImageDataGenerator(horizontal_flip = True)
datagen_val.fit(x_val)
```

# Data Loading & Preprocessing

We have the simulated SETI radio signal dataset available from the kaggle

We will use pandas to read the CSV file where the images are stored

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-drd signal

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.631373 | 0.623529 | 0.713726 | 0.705882 | 0.658824 | 0.666667 | 0.654902 | 0.635294 | 0.647059 | 0.705882 | 0.729412 | 0.72549 | 0.619608 | 0.67451 |
| 2 | 0.72549 | 0.752941 | 0.74902 | 0.701961 | 0.690196 | 0.721569 | 0.709804 | 0.745098 | 0.654902 | 0.721569 | 0.678431 | 0.709804 | 0.713726 | 0.686275 |
| 3 | 0.717647 | 0.701961 | 0.713726 | 0.733333 | 0.705882 | 0.717647 | 0.72549 | 0.682353 | 0.717647 | 0.67451 | 0.690196 | 0.670588 | 0.662745 | 0.666667 |
| 4 | 0.705882 | 0.67451 | 0.654902 | 0.678431 | 0.666667 | 0.662745 | 0.678431 | 0.662745 | 0.686275 | 0.686275 | 0.631373 | 0.65098 | 0.670588 | 0.737255 |
| 5 | 0.647059 | 0.729412 | 0.701961 | 0.67451 | 0.611765 | 0.698039 | 0.713726 | 0.662745 | 0.701961 | 0.67451 | 0.631373 | 0.709804 | 0.694118 | 0.698039 |
| 6 | 0.694118 | 0.682353 | 0.705882 | 0.705882 | 0.666667 | 0.694118 | 0.67451 | 0.713726 | 0.690196 | 0.709804 | 0.756863 | 0.690196 | 0.67451 | 0.721569 |
| 7 | 0.717647 | 0.686275 | 0.760784 | 0.741176 | 0.709804 | 0.72549 | 0.733333 | 0.698039 | 0.654902 | 0.721569 | 0.72549 | 0.729412 | 0.721569 | 0.717647 |
| 8 | 0.713726 | 0.713726 | 0.658824 | 0.690196 | 0.682353 | 0.705882 | 0.709804 | 0.717647 | 0.733333 | 0.733333 | 0.701961 | 0.72549 | 0.72549 | 0.705882 |
| 9 | 0.658824 | 0.678431 | 0.729412 | 0.690196 | 0.705882 | 0.678431 | 0.658824 | 0.670588 | 0.713726 | 0.670588 | 0.654902 | 0.682353 | 0.647059 | 0.654902 |
| 10 | 0.721569 | 0.729412 | 0.764706 | 0.709804 | 0.701961 | 0.658824 | 0.709804 | 0.709804 | 0.698039 | 0.717647 | 0.694118 | 0.72549 | 0.694118 | 0.686275 |
| 11 | 0.635294 | 0.647059 | 0.686275 | 0.705882 | 0.670588 | 0.666667 | 0.666667 | 0.670588 | 0.65098 | 0.705882 | 0.72549 | 0.686275 | 0.72549 | 0.647059 |
| 12 | 0.709804 | 0.713726 | 0.729412 | 0.698039 | 0.647059 | 0.752941 | 0.721569 | 0.694118 | 0.690196 | 0.666667 | 0.694118 | 0.678431 | 0.694118 | 0.709804 |
| 13 | 0.686275 | 0.67451 | 0.717647 | 0.709804 | 0.713726 | 0.709804 | 0.698039 | 0.694118 | 0.721569 | 0.72549 | 0.709804 | 0.72549 | 0.74902 | 0.733333 |
| 14 | 0.666667 | 0.662745 | 0.67451 | 0.690196 | 0.647059 | 0.690196 | 0.764706 | 0.666667 | 0.682353 | 0.701961 | 0.682353 | 0.67451 | 0.694118 | 0.705882 |
| 15 | 0.694118 | 0.737255 | 0.705882 | 0.701961 | 0.709804 | 0.721569 | 0.709804 | 0.67451 | 0.745098 | 0.67451 | 0.682353 | 0.72549 | 0.760784 | 0.709804 |
| 16 | 0.713726 | 0.647059 | 0.65098 | 0.698039 | 0.654902 | 0.694118 | 0.717647 | 0.67451 | 0.670588 | 0.670588 | 0.686275 | 0.694118 | 0.678431 | 0.65098 |
| 17 | 0.756863 | 0.737255 | 0.701961 | 0.658824 | 0.717647 | 0.701961 | 0.729412 | 0.741176 | 0.698039 | 0.694118 | 0.705882 | 0.701961 | 0.701961 | 0.713726 |
| 18 | 0.729412 | 0.721569 | 0.701961 | 0.67451 | 0.662745 | 0.67451 | 0.678431 | 0.72549 | 0.733333 | 0.694118 | 0.682353 | 0.701961 | 0.709804 | 0.678431 |

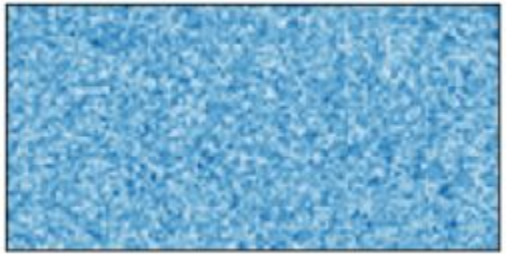
a)

```
train_labels.head(5) # SETI has encode them as OneHot Vectors
```

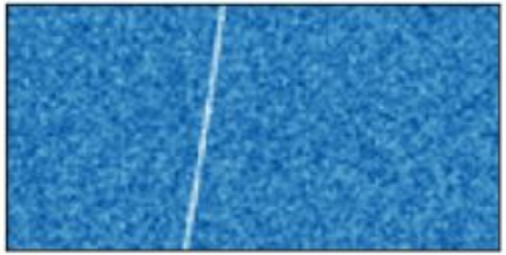
b)

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1.0 | 0.0 | 0.0 | 0.0 |
| 1 | 1.0 | 0.0 | 0.0 | 0.0 |
| 2 | 1.0 | 0.0 | 0.0 | 0.0 |
| 3 | 1.0 | 0.0 | 0.0 | 0.0 |
| 4 | 1.0 | 0.0 | 0.0 | 0.0 |

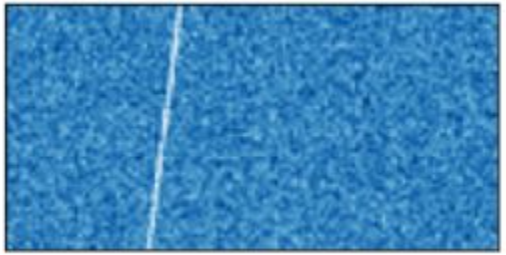
# Data Visualization



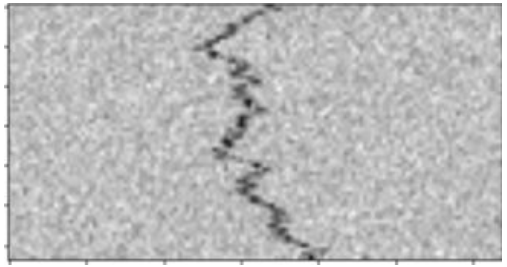
noise



Narrowband drd



narrowband



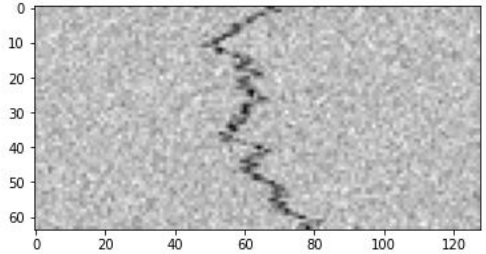
squiggle

# Data Visualization

```
[ ]  plt.figure(0, figsize=(12,12))
     for i in range(1,4):
         plt.subplot(1,3,i)
         img = np.squeeze(x_train[np.random.randint(0, x_train.shape[0])])
         plt.xticks([])
         plt.yticks([])
         plt.imshow(img, cmap='gray')
```

```
# Setting the color_map argument to gray Scale images
plt.imshow(np.squeeze(x_train[3]), cmap='gray')
```

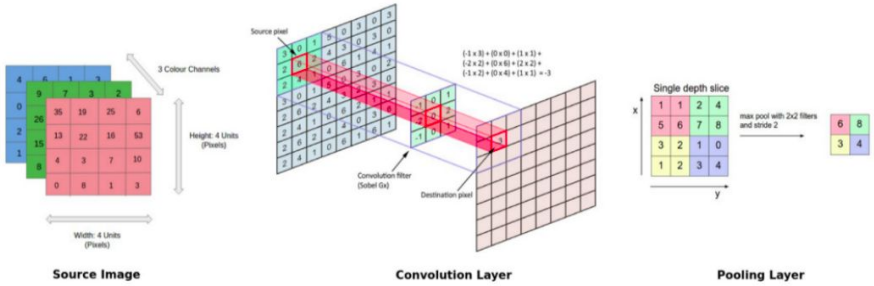
```
<matplotlib.image.AxesImage at 0x7f3b972e8b50>
```

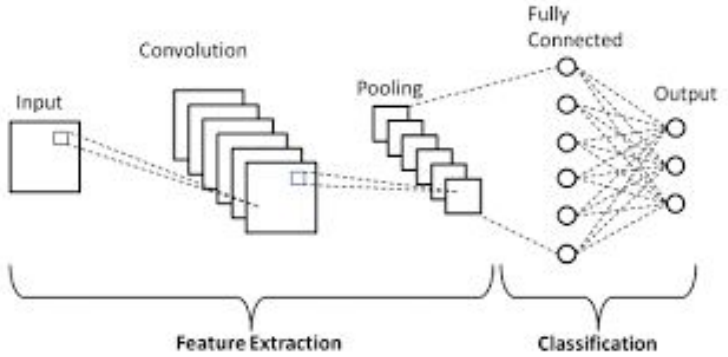


# CNN

In machine learning, a convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks, most commonly applied to analyze visual imagery.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.



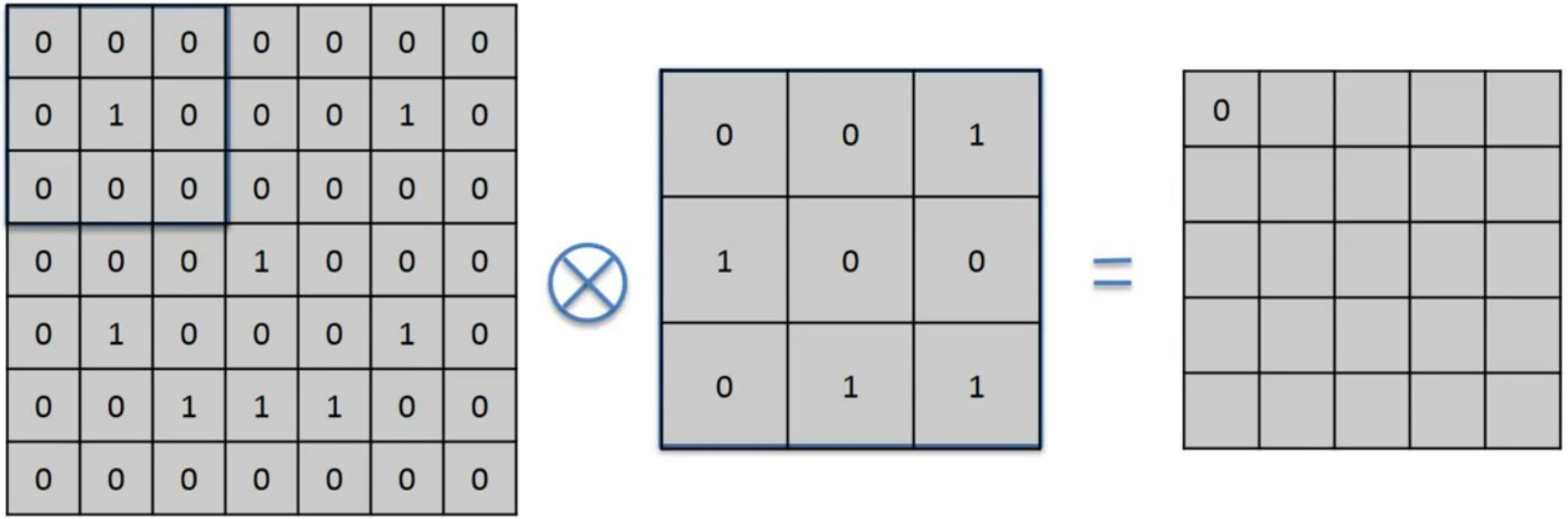

- INPUT  will hold the raw pixel values of the image
- CONV layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- RELU layer will apply an elementwise activation function, such as the max(0,x)
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in decrease in volume
- FC (i.e. fully-connected) layer will compute the class scores. Each neuron in this layer will be connected to all the numbers in the previous volume.

CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

They have applications in image and video recognition, recommender systems and natural language processing.

A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers.

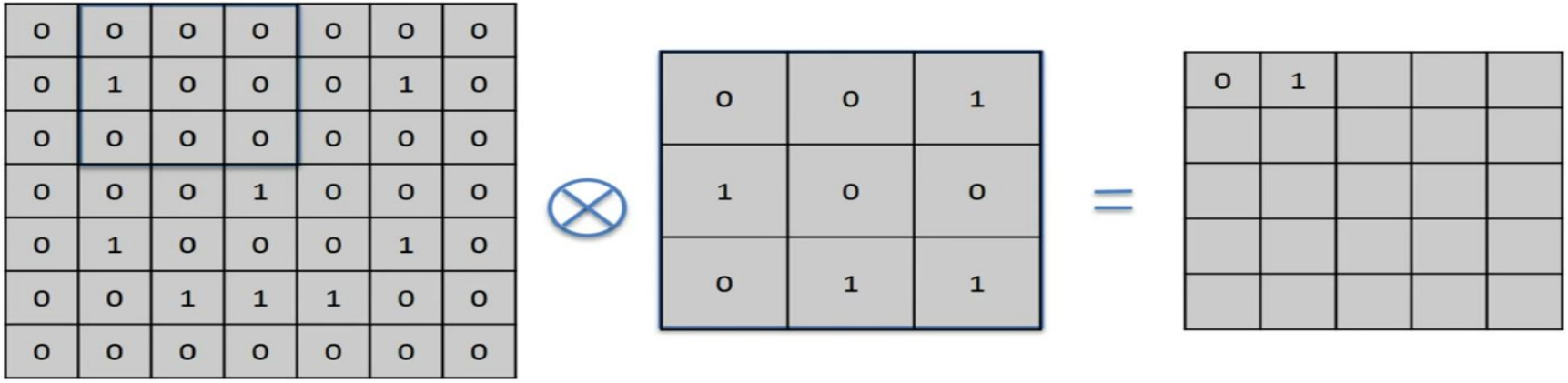
# Step 1: Convolution




Input image

feature detector

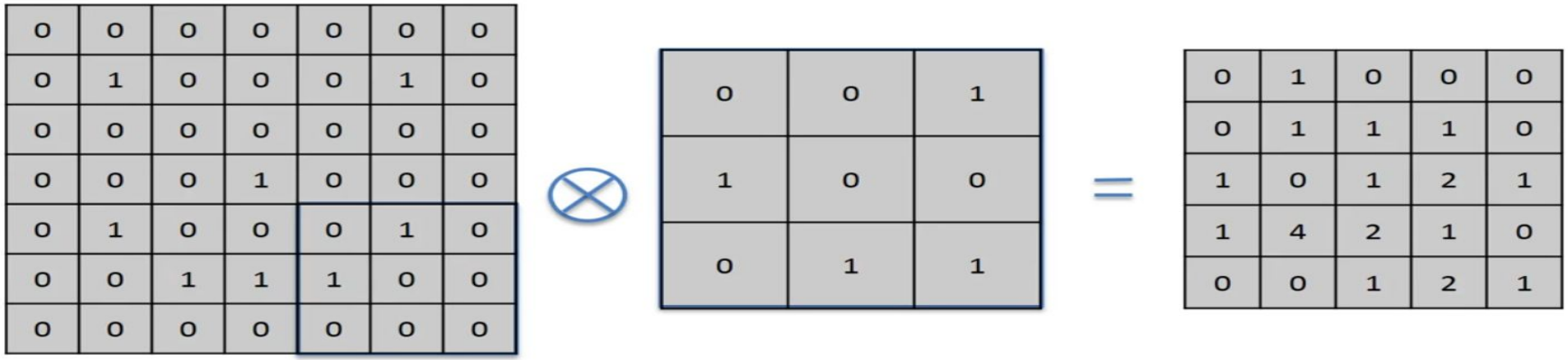
feature map




Input image

feature detector

feature map



# Other kinds of feature detector

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | 5 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Sharpen

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Blur

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 0 | 0 | |
| | -1 | 1 | 0 | |
| | 0 | 0 | 0 | |
| | | | | |

Edge enhancement

| | | | | |
|---|---|---|---|---|
| | | | | |
| | 0 | 1 | 0 | |
| | 1 | -4 | 1 | |
| | 0 | 1 | 0 | |
| | | | | |

Edge detection

*Image Source: docs.gimp.org/en/plug-in-convmatrix.html*

# Step 2: Pooling

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 2 | 1 |
| 1 | 4 | 2 | 1 | 0 |
| 0 | 0 | 1 | 2 | 1 |

Max Pooling

| | | |
|---|---|---|
| 1 | | |
| | | |
| | | |

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 2 | 1 |
| 1 | 4 | 2 | 1 | 0 |
| 0 | 0 | 1 | 2 | 1 |

Max Pooling

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 4 | 2 | 1 |
| 0 | 2 | 1 |

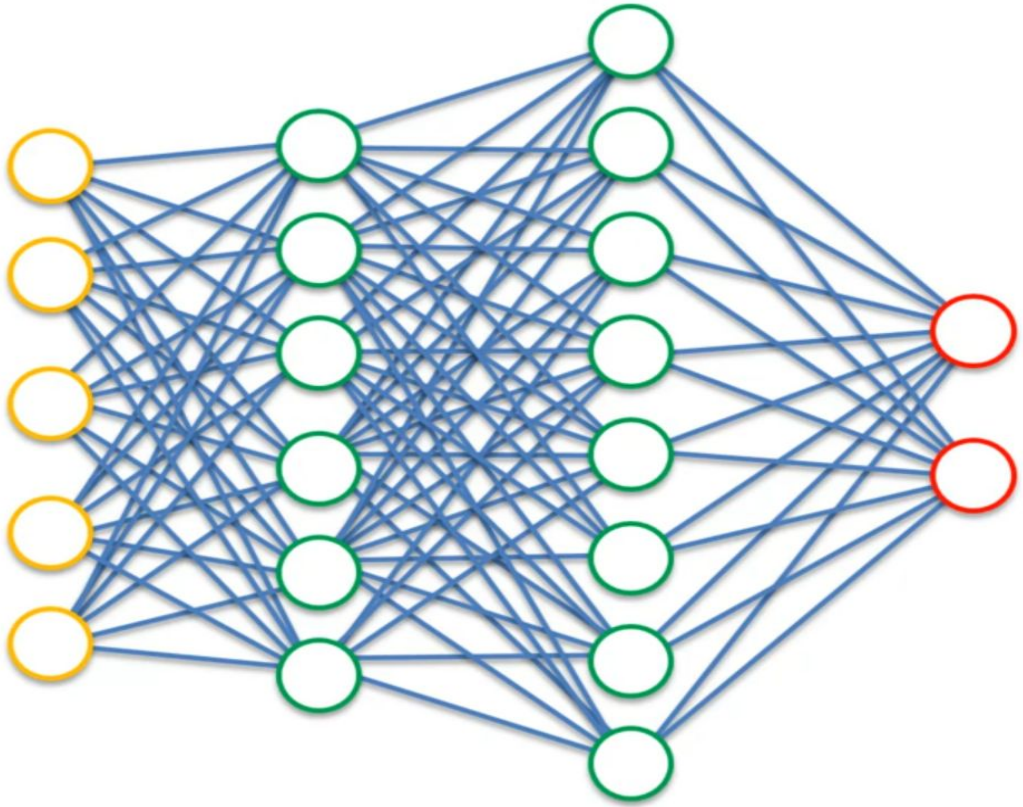
# Step 3: Flattening

| | | |
|---|---|---|
| 1 | 1 | 0 |
| 4 | 2 | 1 |
| 0 | 2 | 1 |

Pooled Feature Map


| |
|---|
| 1 |
| 1 |
| 0 |
| 4 |
| 2 |
| 1 |
| 0 |
| 2 |
| 1 |

# Step 4: Feeding to Neural Network



Flattened input Hidden Layer Output

## CNN Model 1 Architecture:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 64, 128, 32)       832
_________________________________________________________________
batch_normalization (BatchNo (None, 64, 128, 32)       128
_________________________________________________________________
activation (Activation)      (None, 64, 128, 32)       0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 32, 64, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 32, 64, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 64, 64)        51264
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 64, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 32, 64, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 32, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 32, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 1024)              33555456
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
activation_2 (Activation)    (None, 1024)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 4100
=================================================================
Total params: 33,616,132
Trainable params: 33,613,892
Non-trainable params: 2,240
_________________________________________________________________
```

## CNN Model 2 Architecture:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 64, 128, 32)       832

 batch_normalization (BatchN  (None, 64, 128, 32)      128
 ormalization)

 activation (Activation)     (None, 64, 128, 32)       0

 max_pooling2d (MaxPooling2D  (None, 32, 64, 32)       0
 )

 dropout (Dropout)           (None, 32, 64, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 64, 64)        51264

 batch_normalization_1 (Batc  (None, 32, 64, 64)       256
 hNormalization)

 activation_1 (Activation)   (None, 32, 64, 64)        0

 max_pooling2d_1 (MaxPooling  (None, 16, 32, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 16, 32, 64)        0

 conv2d_2 (Conv2D)           (None, 16, 32, 32)        51232

 batch_normalization_2 (Batc  (None, 16, 32, 32)       128
 hNormalization)

 activation_2 (Activation)   (None, 16, 32, 32)        0

 max_pooling2d_2 (MaxPooling  (None, 8, 16, 32)        0
 2D)

 dropout_2 (Dropout)         (None, 8, 16, 32)         0

 flatten (Flatten)           (None, 4096)              0

 dense (Dense)               (None, 1024)              4195328

 batch_normalization_3 (Batc  (None, 1024)             4096
 hNormalization)

 activation_3 (Activation)   (None, 1024)              0

 dropout_3 (Dropout)         (None, 1024)              0

 dense_1 (Dense)             (None, 4)                 4100

=================================================================
Total params: 4,307,364
Trainable params: 4,305,060
Non-trainable params: 2,304
_________________________________________________________________
```

# Loss Function: Categorical cross entropy Loss

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

where $\hat{y}_i$ is the $i$-th scalar value in the model output, $y_i$ is the corresponding target value, and `output size` is the number of scalar values in the model output.

# Optimizer: Adam Optimizer

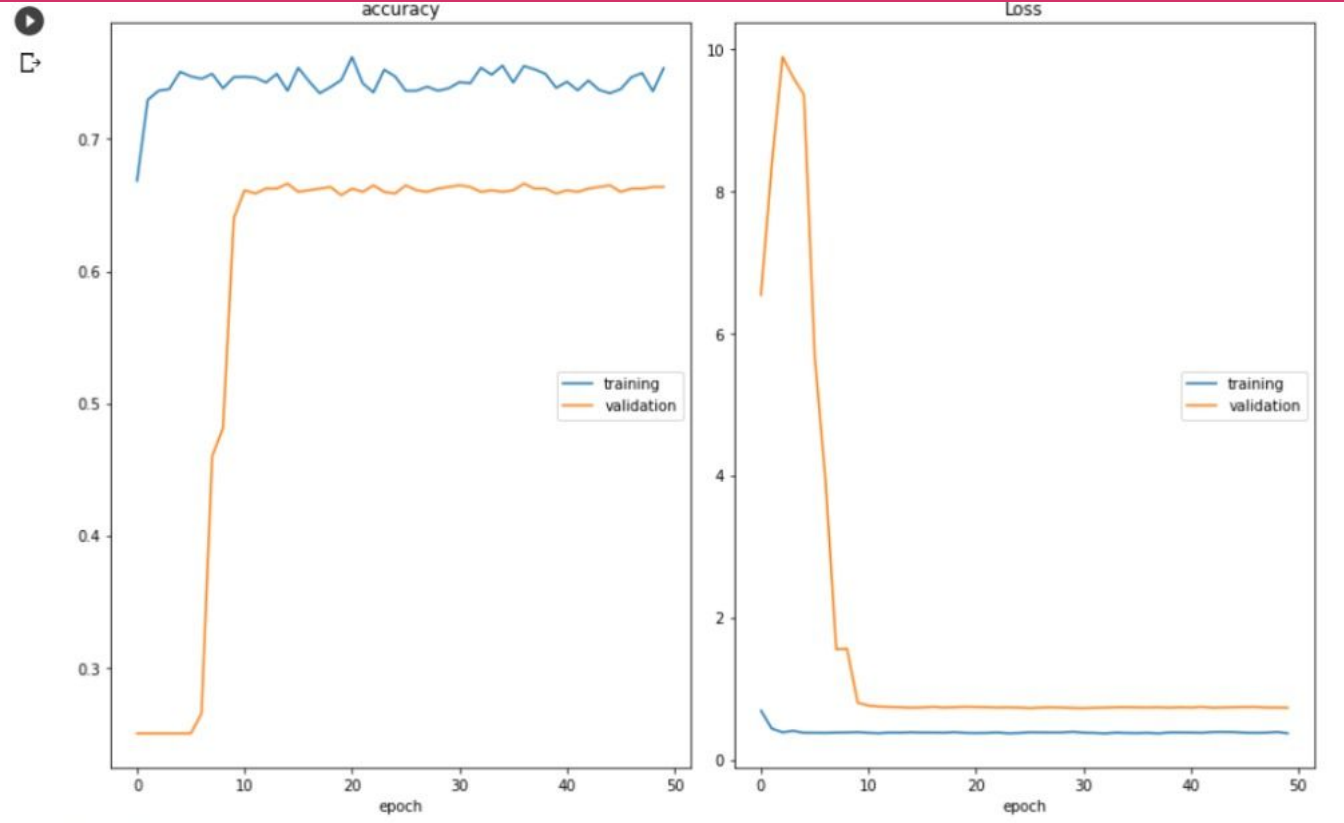
Adam optimizer involves a combination of two gradient descent methodologies and builds upon them to give a more optimized gradient descent.

1. It takes into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.
2. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the ‘exponential moving average’.

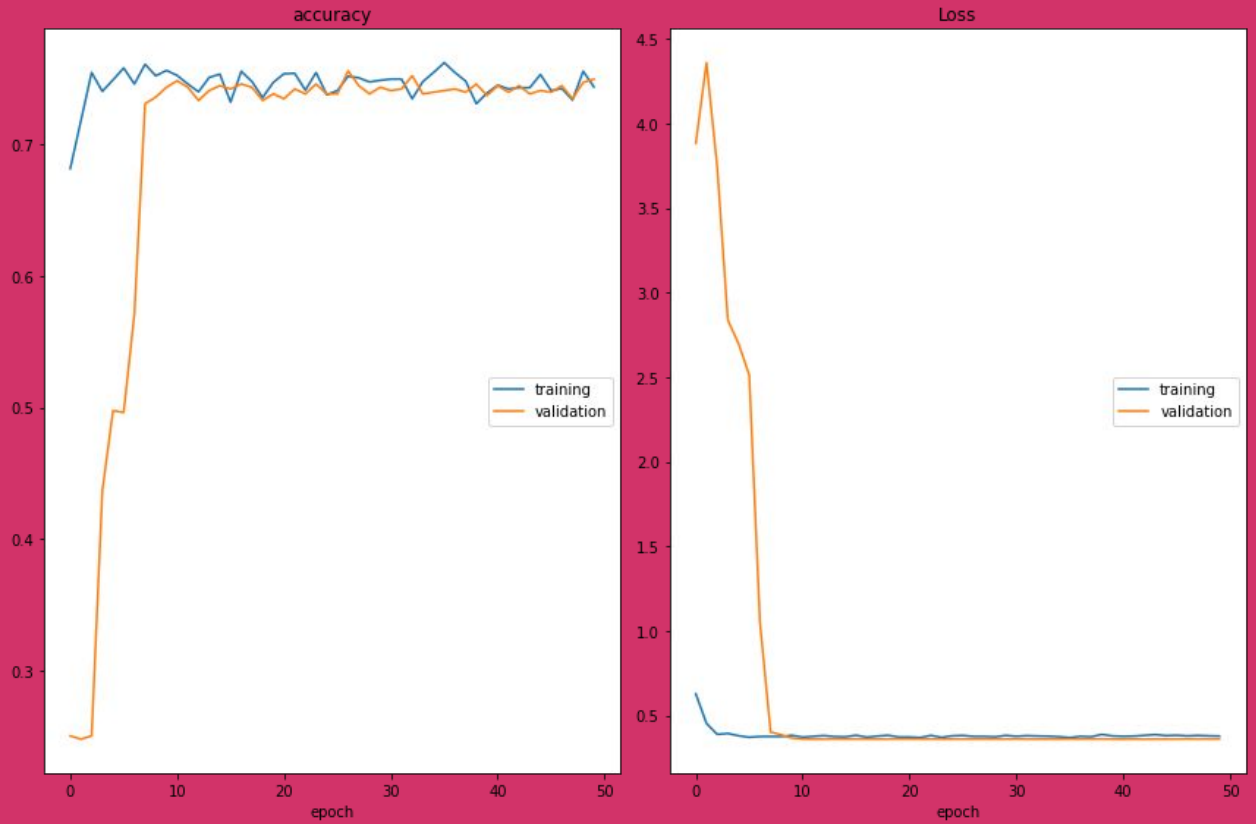
# Model Training:

No. of epochs = 50
initial_learning_rate = 0.005
We have used Exponential Decay for learning rate scheduling with decay rate = decay_rate = 0.96

# Results (Model 1)



# Results (Model 2):



# Performance Parameters:

1. **Precision = (TP) / (FP+TP)**
2. **Recall = (TP)/(FN+TP)**
3. **F1-score is defined as the harmonic mean of precision and recall.**
4. **Accuracy = (TP+TN)/(TP+FP+TN+FN)**

# Confusion Matrix

**It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with different combinations of predicted and actual values.**

# Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.75 | 0.86 | 200 |
| 1 | 0.00 | 0.00 | 0.00 | 200 |
| 2 | 0.47 | 0.99 | 0.64 | 200 |
| 3 | 1.00 | 0.91 | 0.95 | 200 |
| accuracy | | | 0.66 | 800 |
| macro avg | 0.62 | 0.66 | 0.61 | 800 |
| weighted avg | 0.62 | 0.66 | 0.61 | 800 |

Classification accuracy: 0.662500

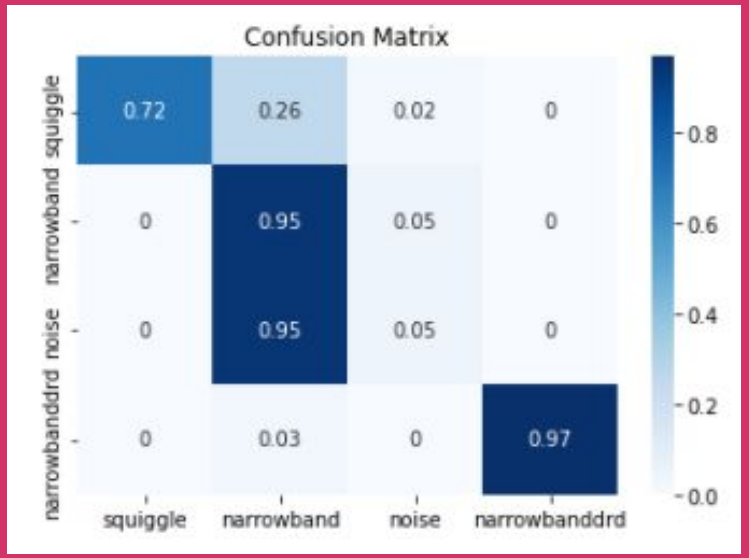
Model 1

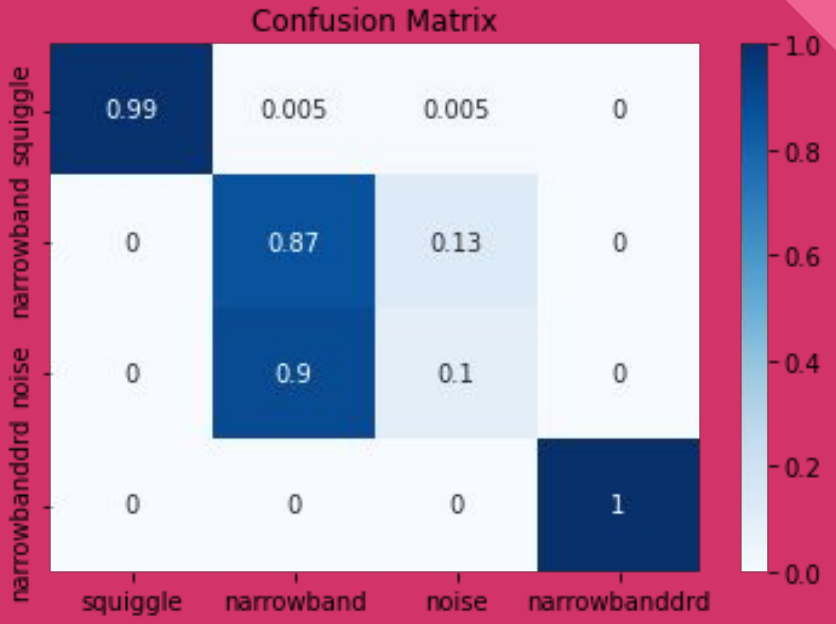
Model 2

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 0.99 | 200 |
| 1 | 0.49 | 0.87 | 0.63 | 200 |
| 2 | 0.44 | 0.10 | 0.17 | 200 |
| 3 | 1.00 | 1.00 | 1.00 | 200 |
| accuracy | | | 0.74 | 800 |
| macro avg | 0.73 | 0.74 | 0.70 | 800 |
| weighted avg | 0.73 | 0.74 | 0.70 | 800 |

Classification accuracy: 0.741250

# Confusion Matrix:




Model 2

# Future Scope:

- **Trying the Transfer Learning mechanism for CNN**
- **Evaluating and comparing the results**

# References

[1] Classification of Communication Signals and Detection of Unknown Formats Using Artificial Neural Networks: Alexander Iversen, Nicholas K. Taylor and Keith E. Brown. 2006.
[2] J. W. Frank Fan, Kenny Smith. Project seti: Machine recognition of squiggles in seti signal data, 2016.
[3] SETI Signal Classification: Identifying Signals Buried Deep in Noise: Akash Mahajan, Guoli Yin, Marc Vaz
[4] https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/
[5]https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/
[6] https://cs231n.github.io/classification/
[7] https://cs231n.github.io/convolutional-networks/
[8] https://cs231n.github.io/understanding-cnn/
[9] https://cs231n.github.io/transfer-learning/

# Google Colab Code Link:

https://colab.research.google.com/drive/1bV_xjuoOZvF14EA5L9QTfqgD05jnnhh1?usp=sharing

# THANKYOU :)