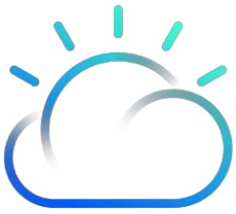


Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

(http://cocl.us/pytorch_link_top)

(http://cocl.us/pytorch_link_top)

Convolutional Neral Network Simple example

(http://cocl.us/pytorch_link_top)

(http://cocl.us/pytorch_link_top)
(http://cocl.us/pytorch_link_top)

(http://cocl.us/pytorch_link_top)

Table of Contents (http://cocl.us/pytorch_link_top)

In this lab, we will use a Convolutional Neral Networks to classify horizontal an vertical Lines

Estimated Time Needed: **25 min**

Helper functions

In [1]:

In [2]:

Out[2]:

```
<torch._C.Generator at 0x7fee50071350>
```

function to plot out the parameters of the Convolutional layers

In [3]:

`show_data` : plot out data sample

In [4]:

create some toy data

In [5]:

`plot_activation` : plot out the activations of the Convolutional layers

In [6]:

Utility function for computing output of convolutions takes a tuple of (h,w) and returns a tuple of (h,w)

In [7]:

Prepare Data

Load the training dataset with 10000 samples

In [10]:

Load the testing dataset

In [11]:

Out[11]:

```
<__main__.Data at 0x7fedcbf65c50>
```

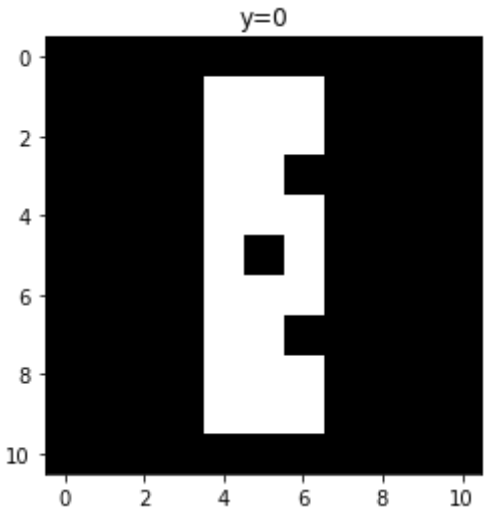
we can see the data type is long

Data Visualization

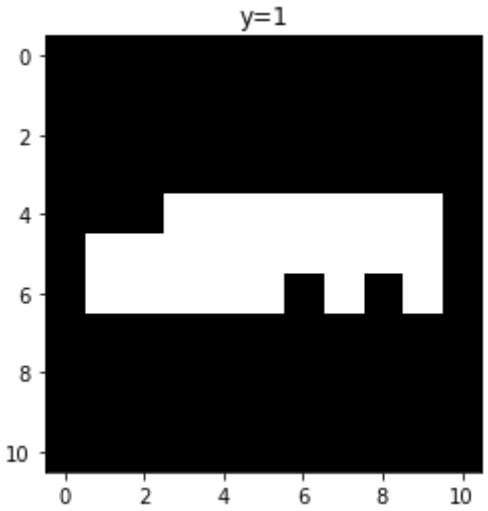
Each element in the rectangular tensor corresponds to a number representing a pixel intensity as demonstrated by the following image.

We can print out the third label

In [12]:



In [13]:



we can plot the 3rd sample

Build a Convolutional Neral Network Class

The input image is 11 x11, the following will change the size of the activations:

convolutional layer

max pooling layer

convolutional layer

max pooling layer

with the following parameters `kernel_size` , `stride` and `pad` . We use the following lines of code to change the image before we get tot he fully connected layer

In [14]:

```
(10, 10)
(9, 9)
(8, 8)
(7, 7)
```

Build a Convolutional Network class with two Convolutional layers and one fully connected layer. Pre-determine the size of the final output matrix. The parameters in the constructor are the number of output channels for the first and second layer.

In [15]:

Define the Convolutional Neral Network Classifier , Criterion function, Optimizer and Train the Model

There are 2 output channels for the first layer, and 1 outputs channel for the second layer

In [16]:

we can see the model parameters with the object

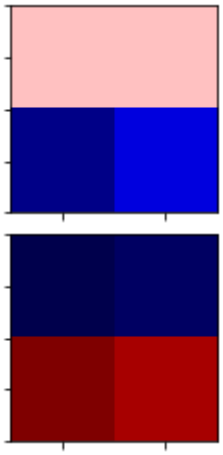
In [17]:

Out[17]:

```
CNN(
  (cnn1): Conv2d(1, 2, kernel_size=(2, 2), stride=(1, 1))
  (maxpool1): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=
1, ceil_mode=False)
  (cnn2): Conv2d(2, 1, kernel_size=(2, 2), stride=(1, 1))
  (maxpool2): MaxPool2d(kernel_size=2, stride=1, padding=0, dilation=
1, ceil_mode=False)
  (fc1): Linear(in_features=49, out_features=2, bias=True)
)
```

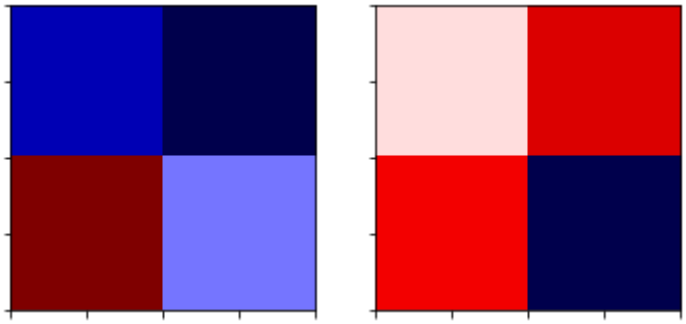
Plot the model parameters for the kernels before training the kernels. The kernels are initialized randomly.

In [18]:



Loss function

In [19]:



Define the loss function

In [20]:

optimizer class

In [21]:

Define the optimizer class

In [22]:

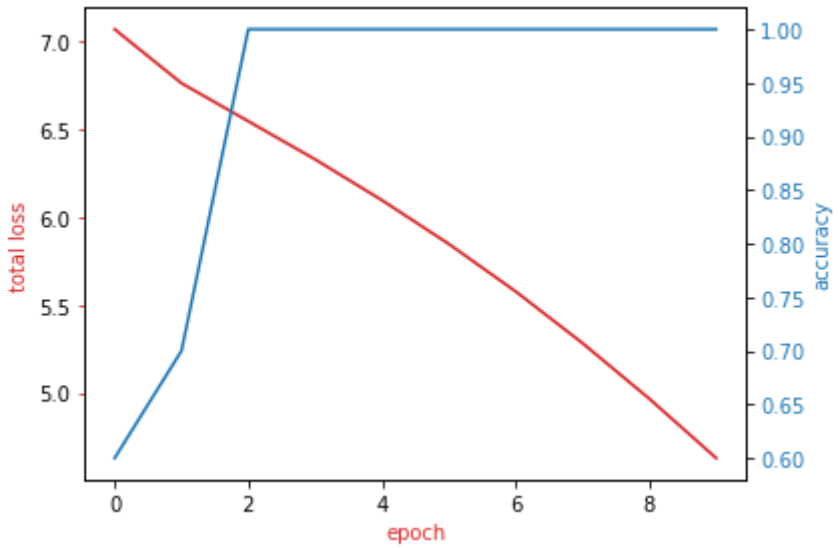
Train the model and determine validation accuracy technically test accuracy **(This may take a long time)**

In [23]:

Analyse Results

Plot the loss and accuracy on the validation data:

In [24]:



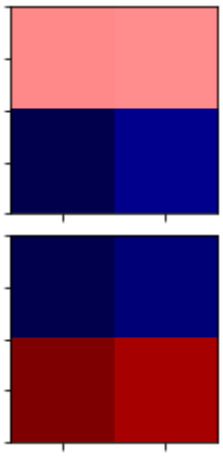
View the results of the parameters for the Convolutional layers

In [25]:

Out[25]:

```
tensor([[[[ 0.1661,  0.1605],
          [-0.4581, -0.3664]]],


        [[[-0.4613, -0.4014],
          [ 0.5564,  0.4737]]]])
```

In [26]:

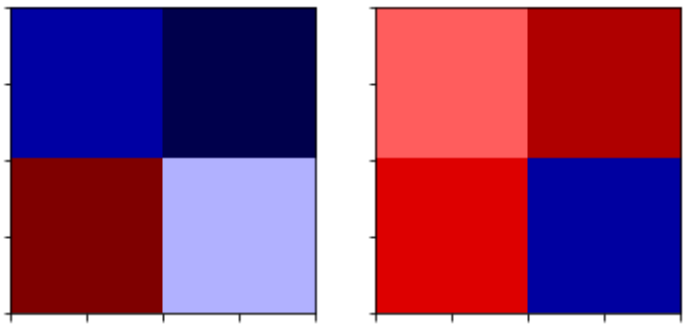


In [27]:

Out[27]:

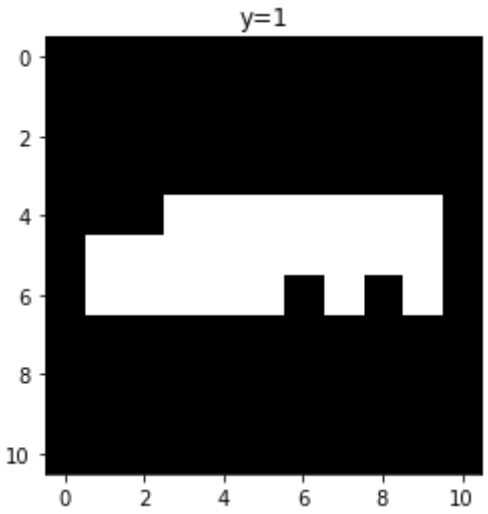
```
tensor([[[[ 0.1661,  0.1605],
          [-0.4581, -0.3664]]],


        [[[-0.4613, -0.4014],
          [ 0.5564,  0.4737]]]])
```

In [28]:



Consider the following sample

In [29]:



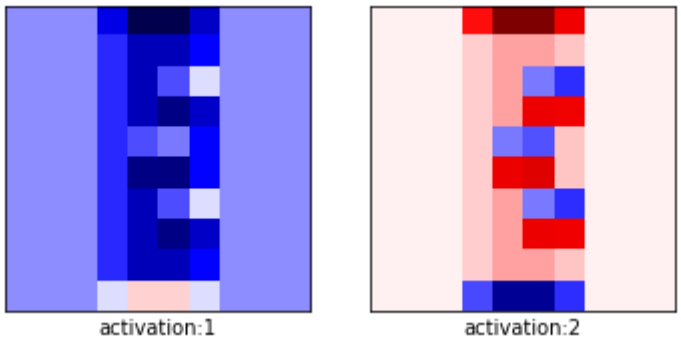
Determine the activations

In [30]:

Plot them out

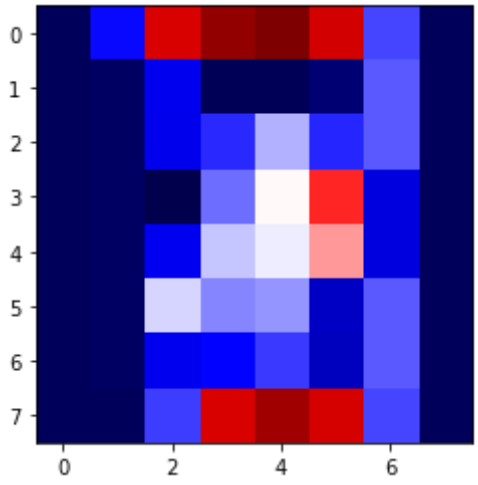
In [31]:

2



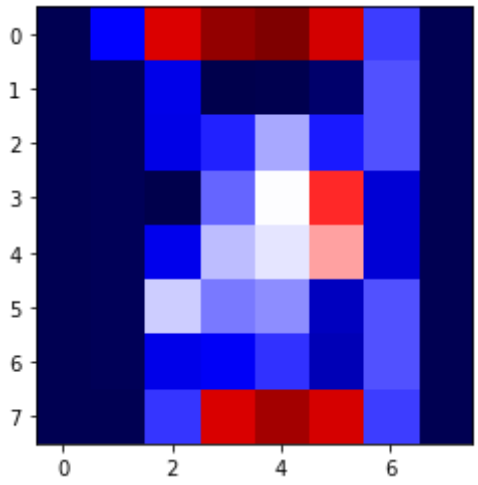
In [32]:

1



In [33]:

1



we save the output of the activation after flattening

In [34]:

we can do the same for a sample where y=0

In [36]:

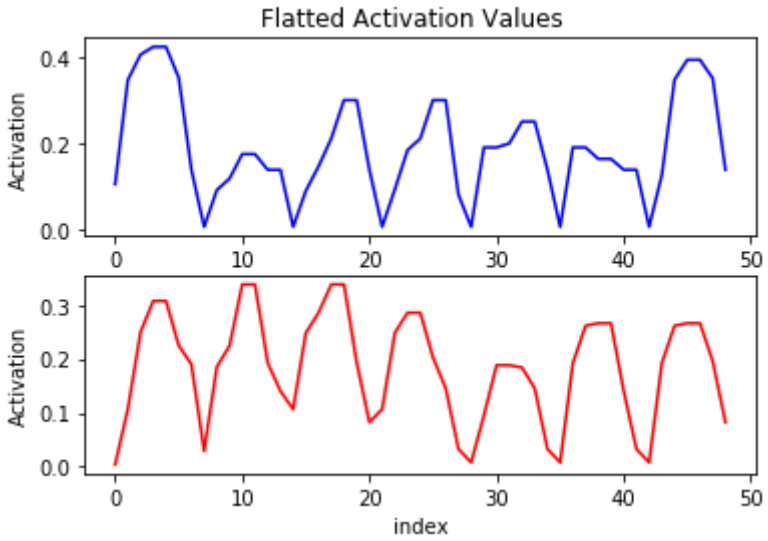
Out[36]:

```
array([0.00415926, 0.10685794, 0.24987417, 0.3076138 , 0.3076138 ,
       0.22501388, 0.18994048, 0.02890191, 0.18508461, 0.22431666,
       0.33811432, 0.33811432, 0.19279845, 0.14028351, 0.10685794,
       0.24826197, 0.2856608 , 0.33811432, 0.33811432, 0.19279845,
       0.0824036 , 0.10685794, 0.24826197, 0.2856608 , 0.2856608 ,
       0.20249416, 0.14528106, 0.03288748, 0.00793505, 0.09491169,
       0.18829235, 0.18829235, 0.18393406, 0.14528106, 0.03288748,
       0.00793505, 0.19222237, 0.26141855, 0.26613277, 0.26613277,
       0.14033706, 0.03288748, 0.00793505, 0.19222237, 0.26141855,
       0.26613277, 0.26613277, 0.19597782, 0.08254446], dtype=float32)
```

In [37]:

Out[37]:

```
Text(0, 0.5, 'Activation')
```



About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering. His research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition.

Other contributors: Michelle Carey (https://www.linkedin.com/in/michelleccarey/)

In [ ]: