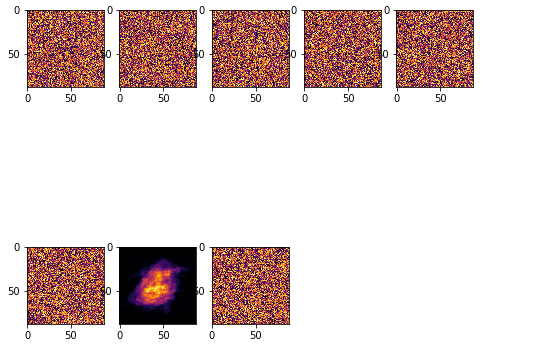
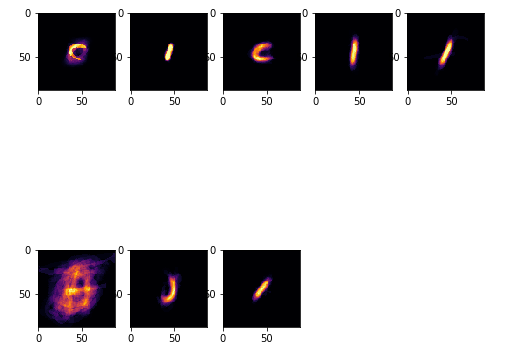
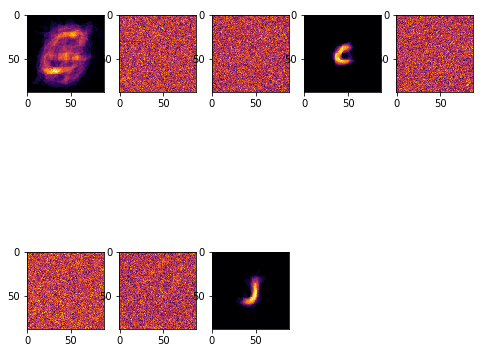
**Competitive Learning Assignment notes:**

Steps to take:

1. Normalised input or initial weights.
2. Noise addition on the weights.
3. Decaying learning rate.
4. Leaky learning: update the weights of the losers as well as winners but with a much smaller learning rate.
5. Update the winners and neighbouring losers.

* Original prototype graphs looked like this:
* 
  + This is when using only the lab5 solutions with no changes
* When theres no normalization, this is how it looks like:
* 
  + Normalization eliminates the repetition of data (https://stackoverflow.com/questions/1102590/what-exactly-does-database-normalization-do), so in this case normalization removes the repeated C, I and J letters
* Added noise, which has improved the result of the prototypes. Less dead cells, and more letters are forming. It seems the noise has also darkened the dead cells
  + Code:
    - noise = np.random.normal(0,1,(letters,n)) # add noise
    - W = W + noise
* 
  + The 3 letters I get are ‘C’, ‘I’ and ‘J’ and rarely do I get one letter repeated.
* So I found out that in the counter that records how many times a particular output neuron won, theres one neuron that’s increasing too much, which is the one with all the letters mixed up in it, while theres 5 dead ones with 0 as the count the entire time.
  + Implementing a decay to the weight change could fix this and as a result, distribute the winning neuron counts to the other dead neurons

TODO:

* Normalize the train data as well. Check PCA (Lab 4)
  + When normalizing, change x = train[:,i] to x = train[i,:]