Convolutional NN for Vision Problems

Image: 2d Matrix of Pixels

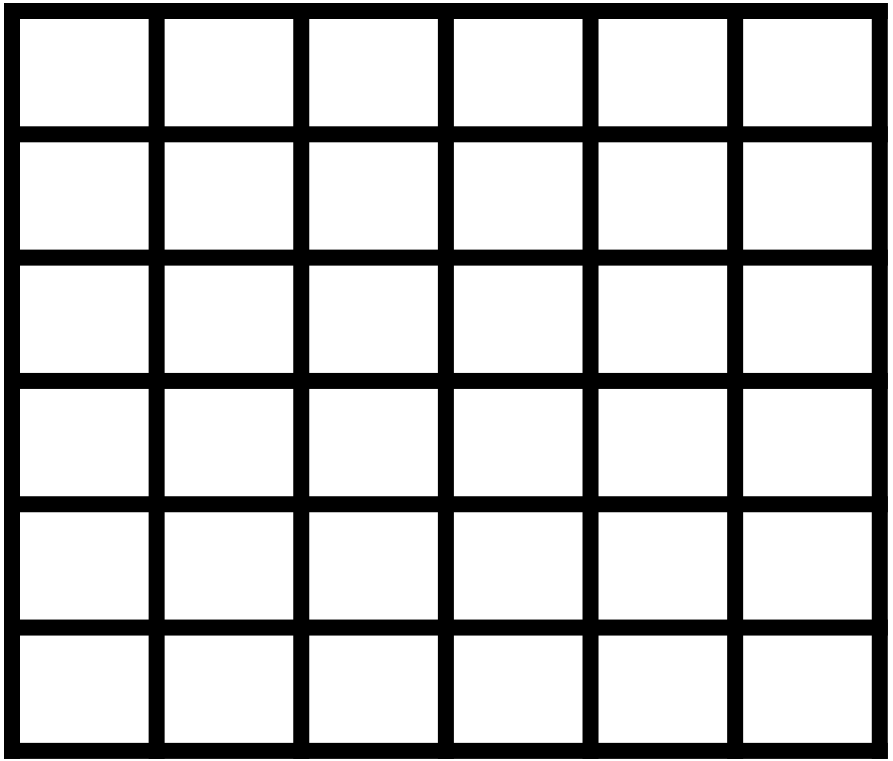
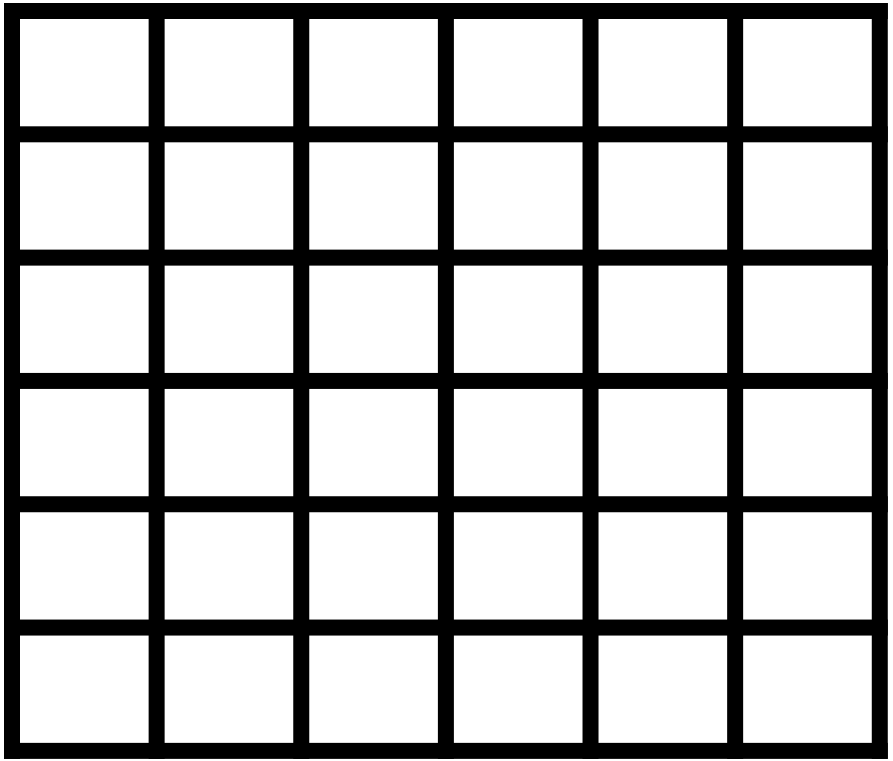


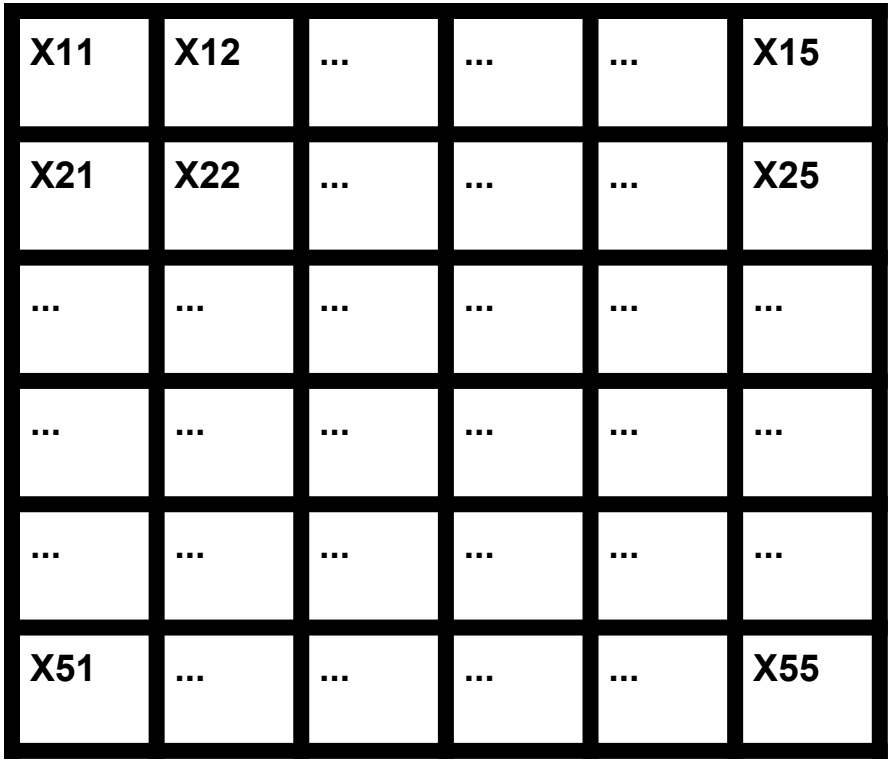
Image: 2d Matrix of Pixels



Old ideas:

- Look within image for interesting areas
- Generate local features from groups of adjacent pixels

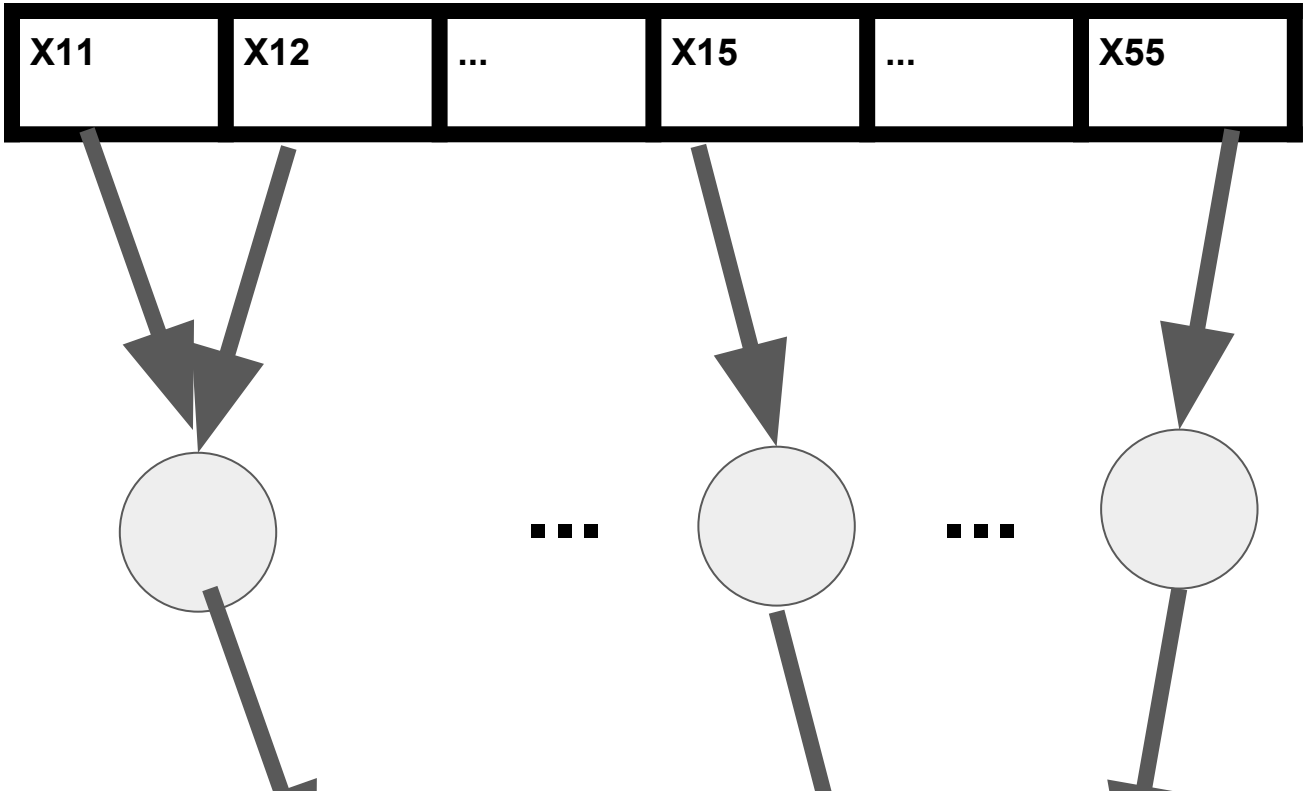
Image: 2d Matrix of Pixels



Neural Nets:

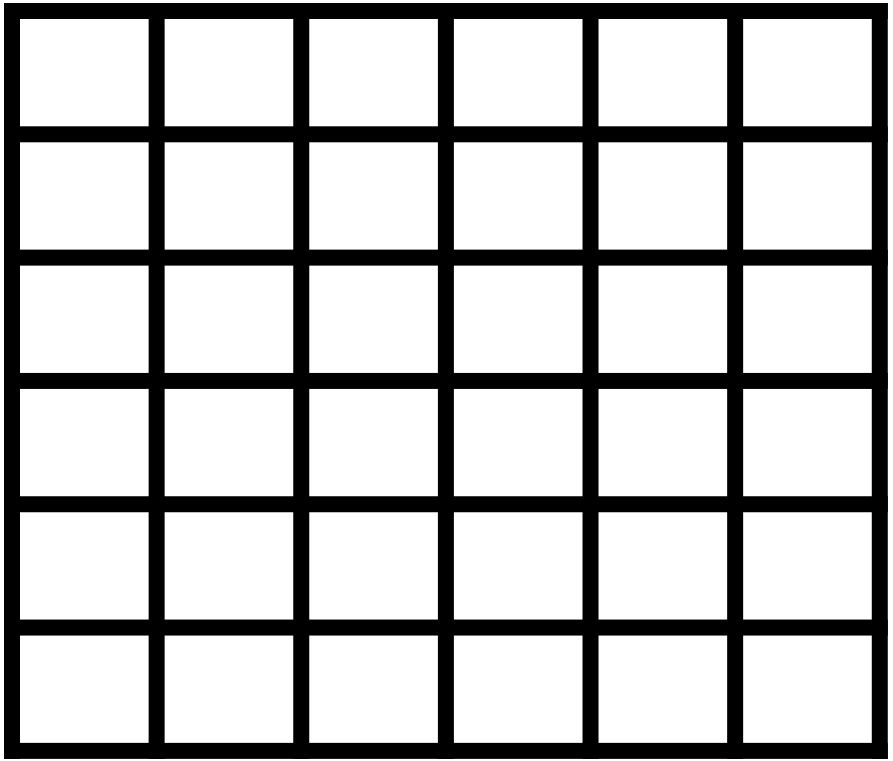
- Pixels are the basic features
- Passed through layers

Image: 2d Matrix of Pixels; Neural Nets



How do we set up these layers?

Image: NN Layers



Idea 1:
Have adjacent pixels map to a hidden node

Image: 2d Matrix of Pixels

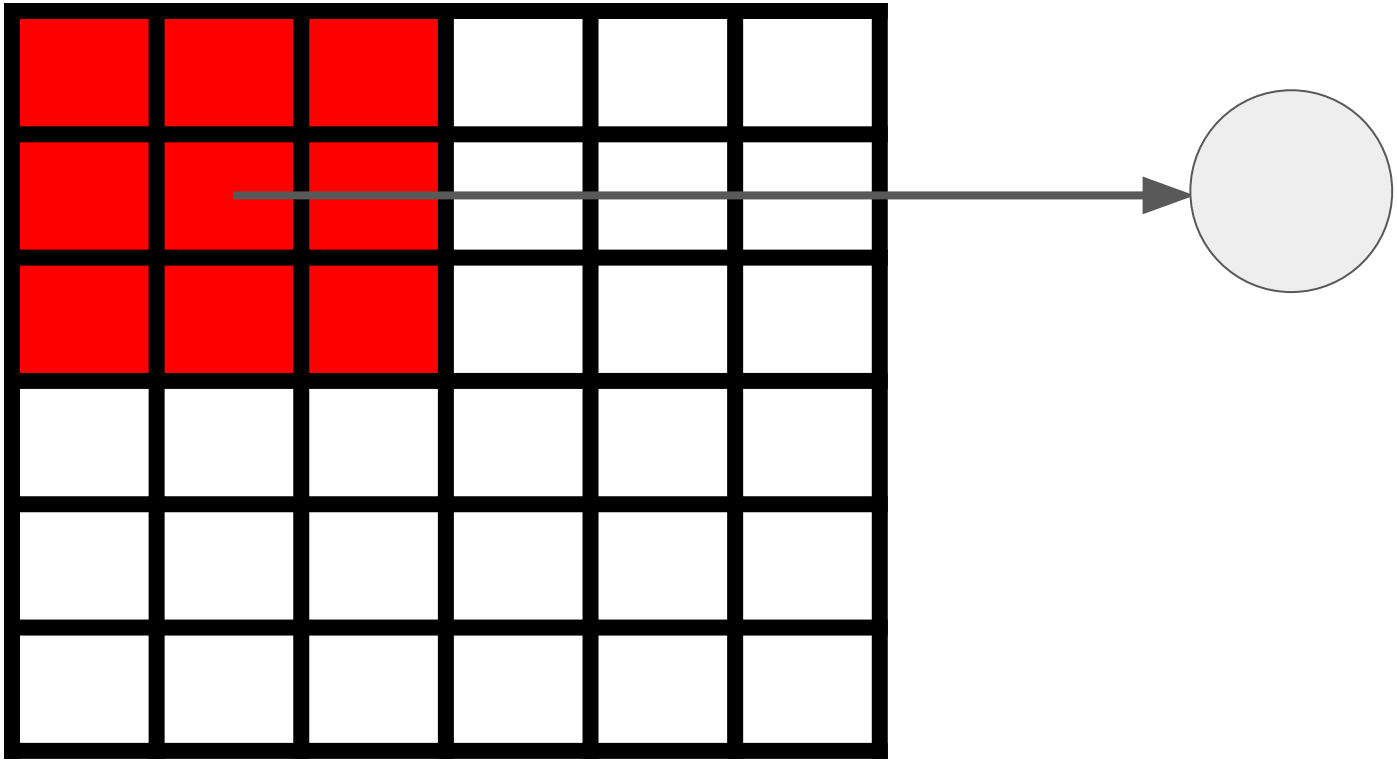


Image: 2d Matrix of Pixels

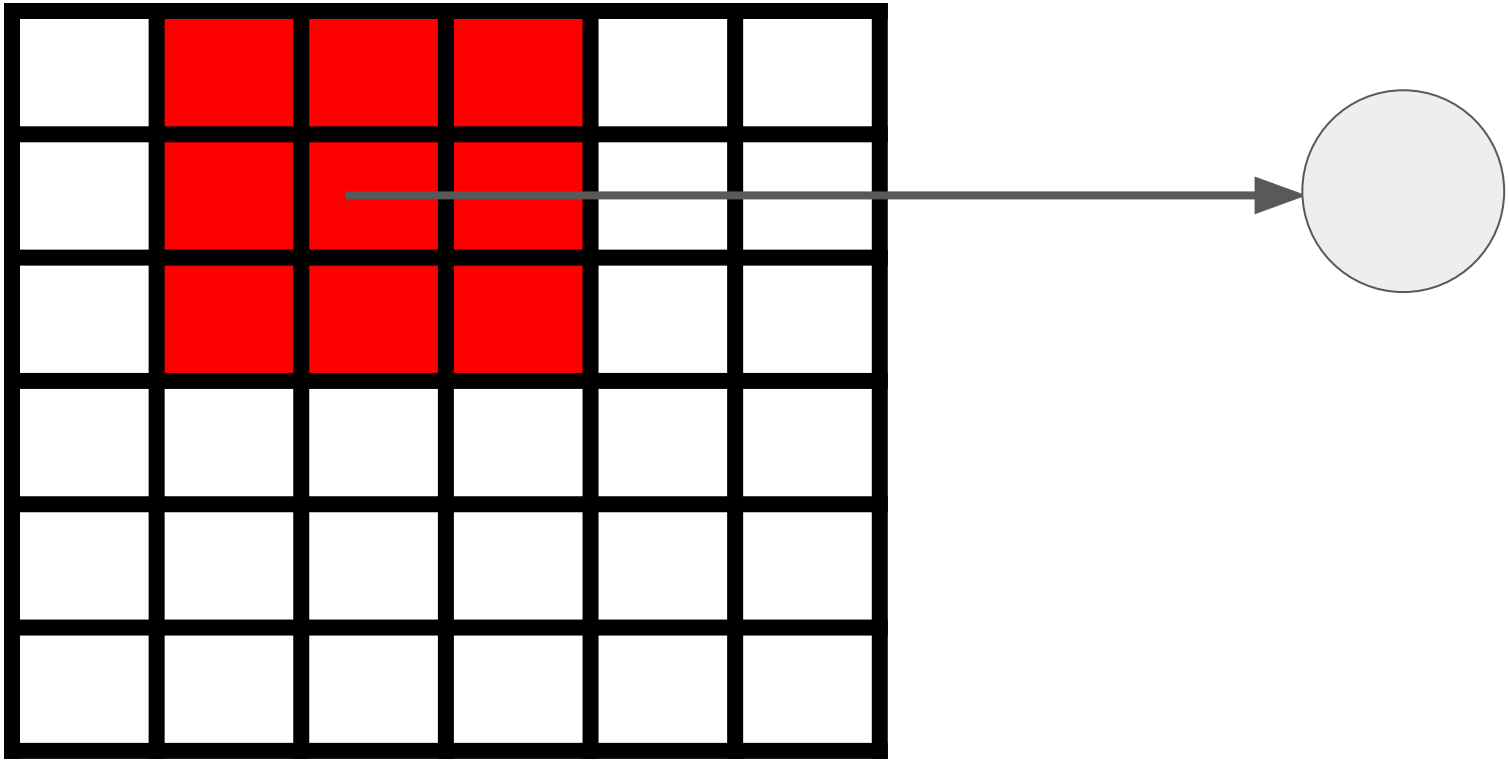


Image: 2d Matrix of Pixels

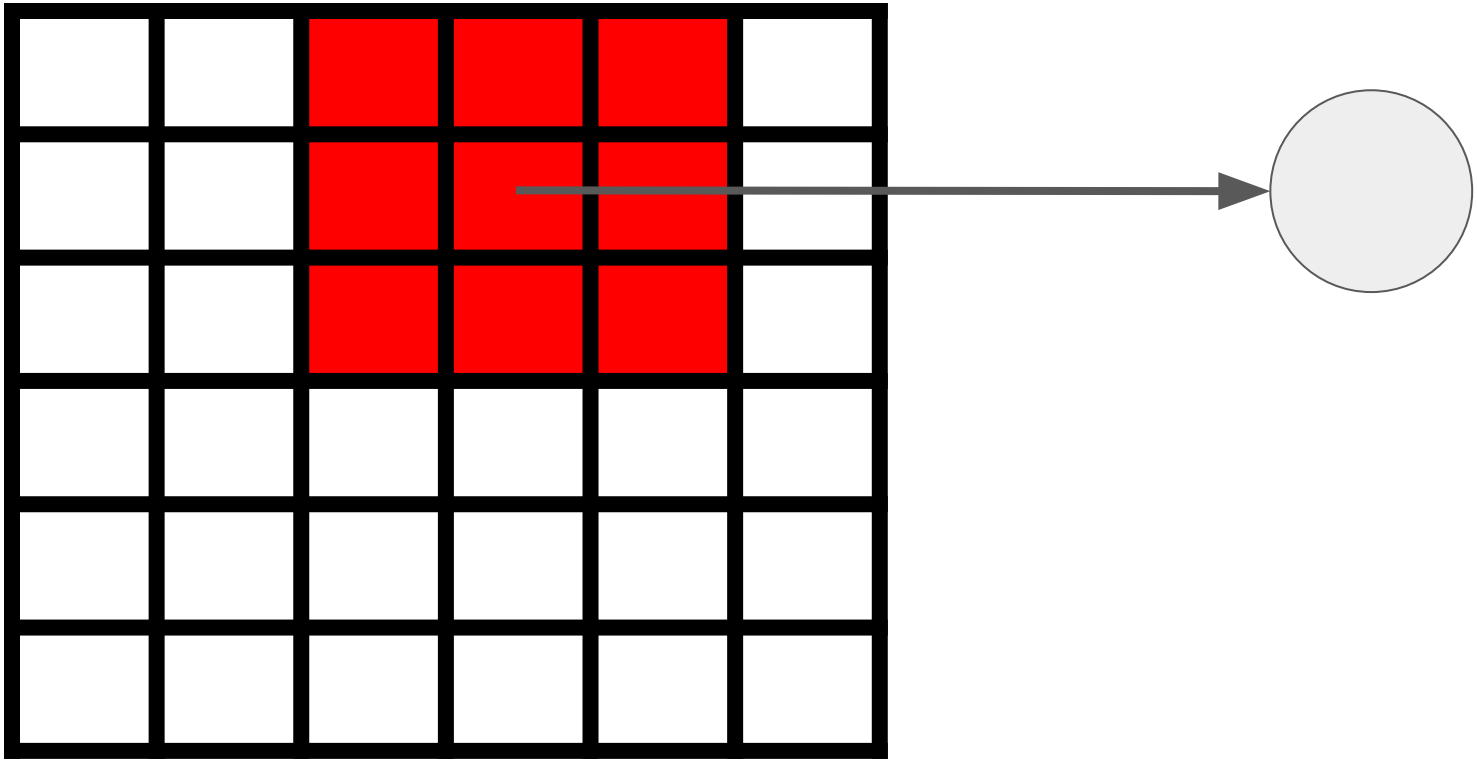


Image: 2d Matrix of Pixels

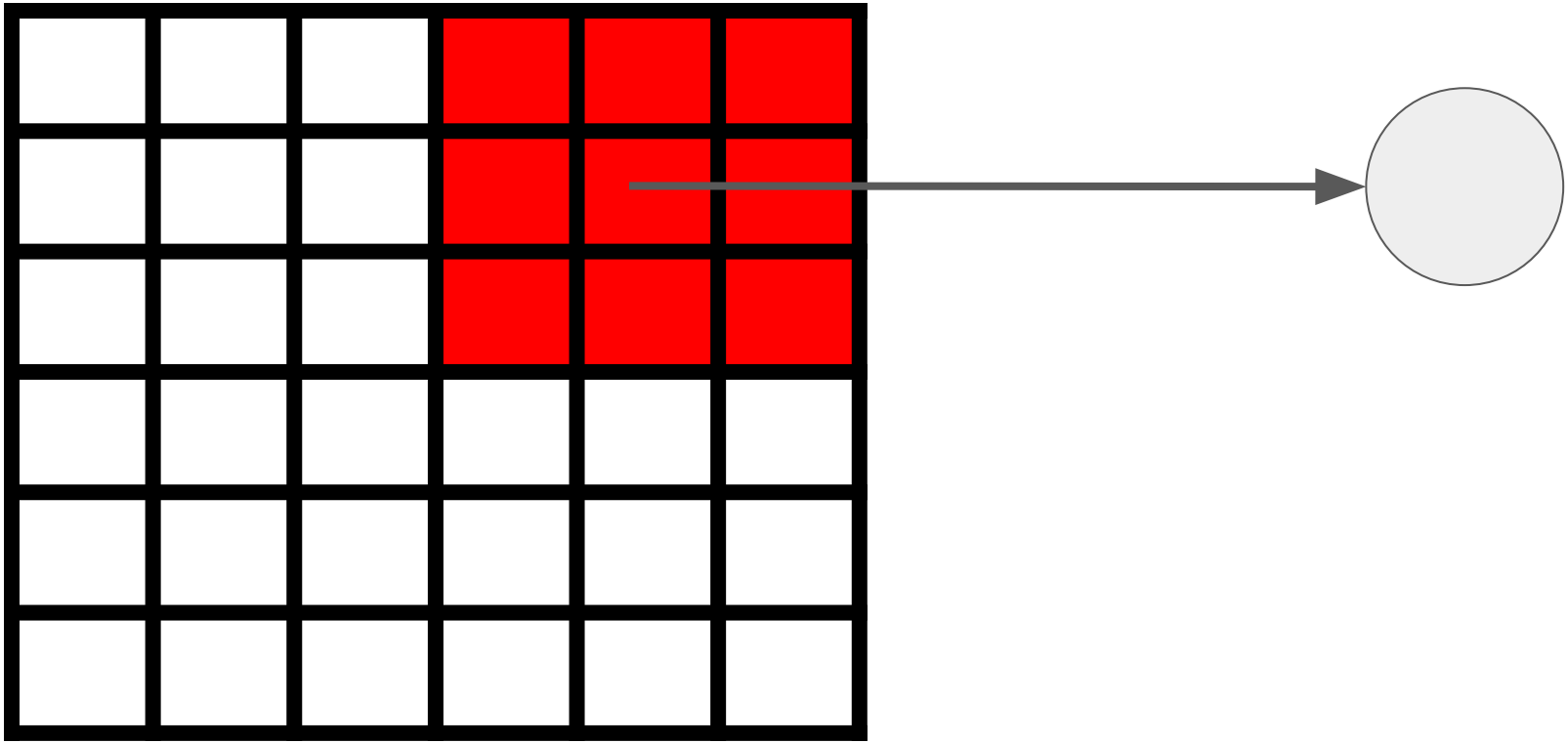


Image: 2d Matrix of Pixels

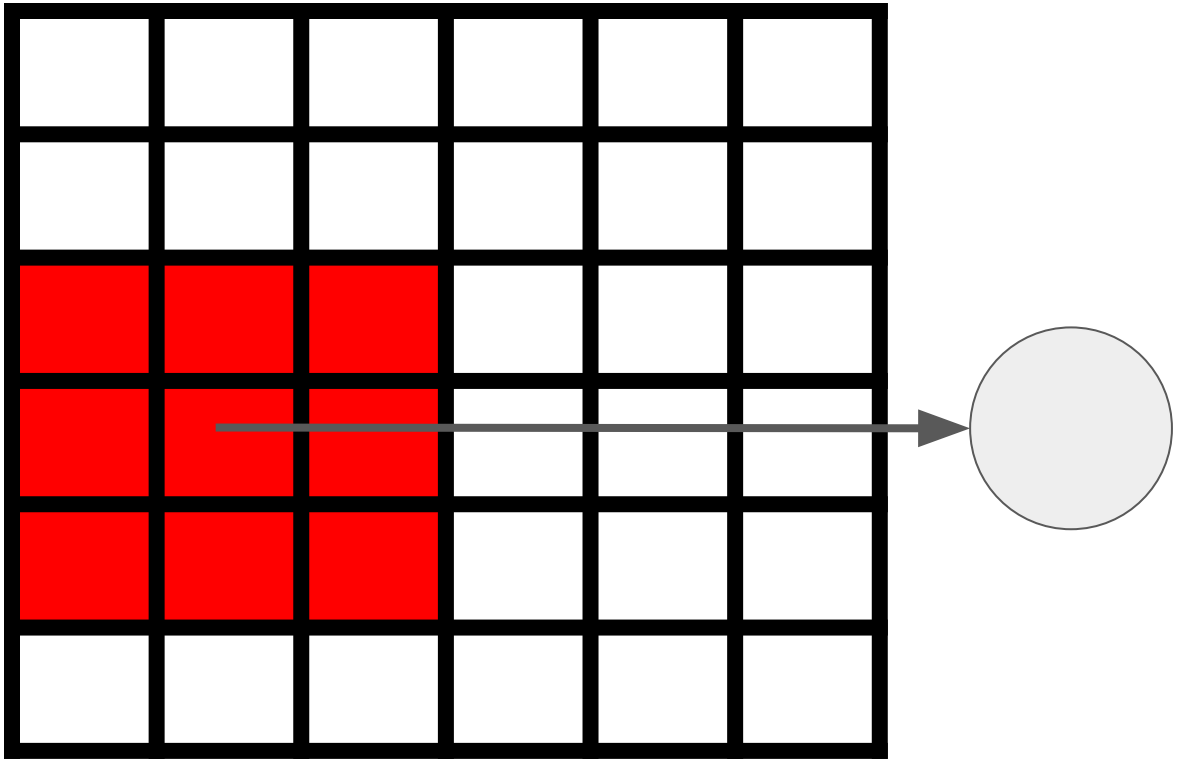
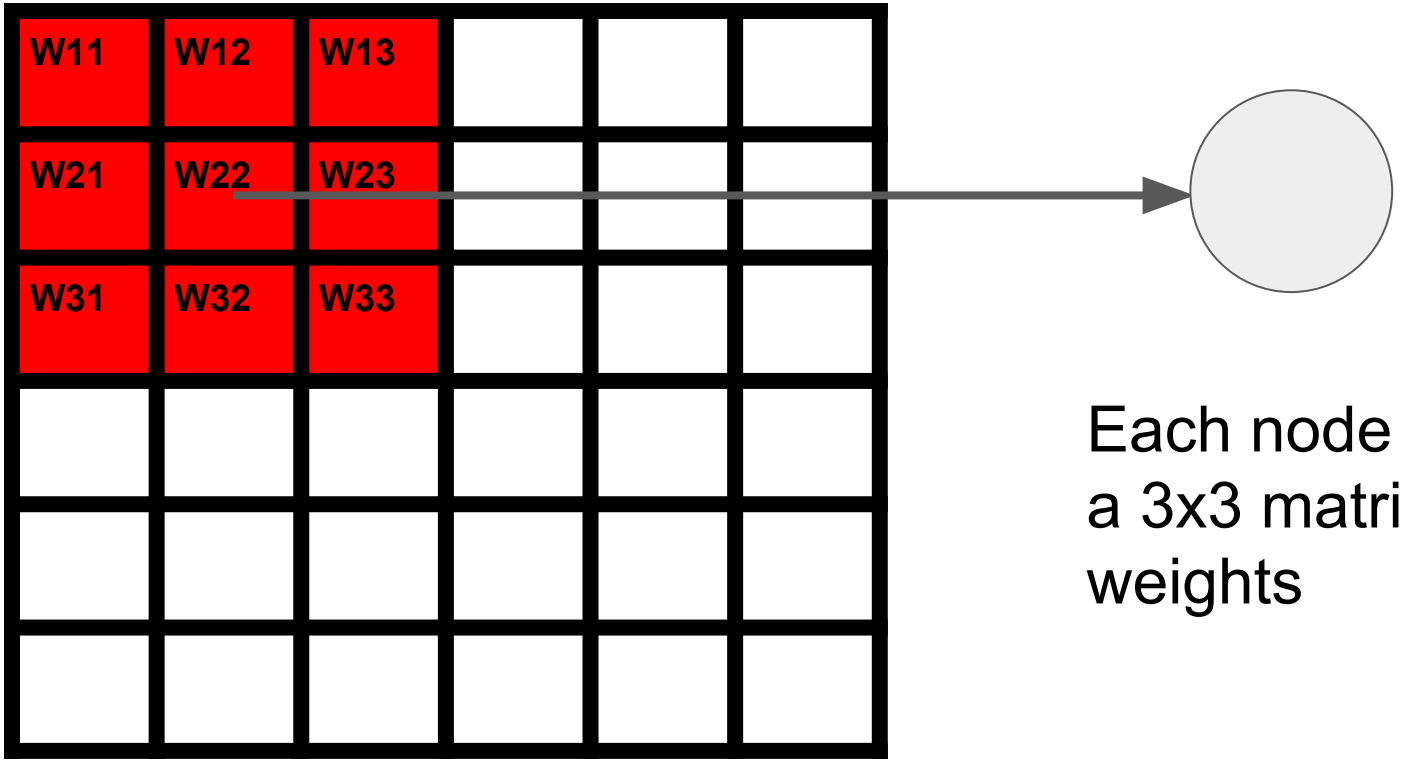
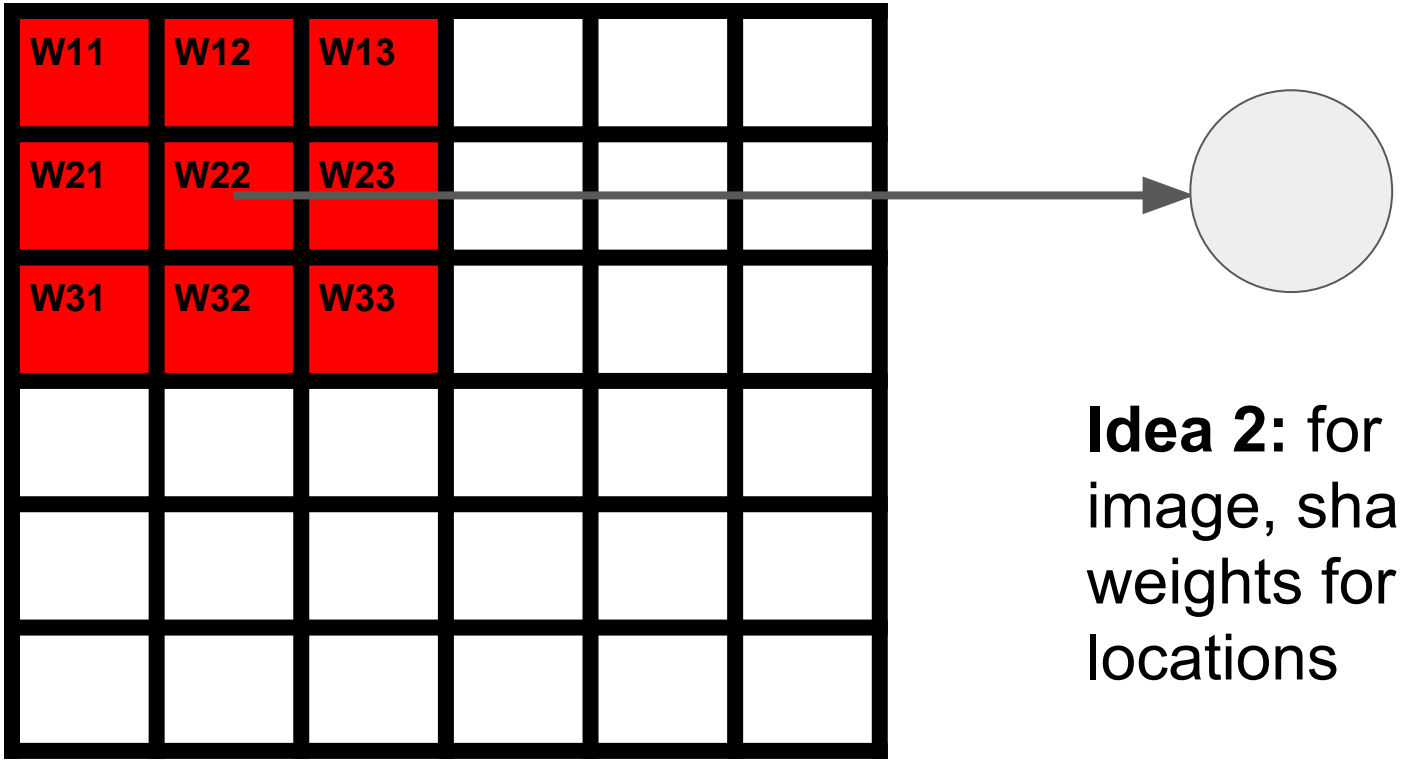


Image: 2d Matrix of Pixels, weights



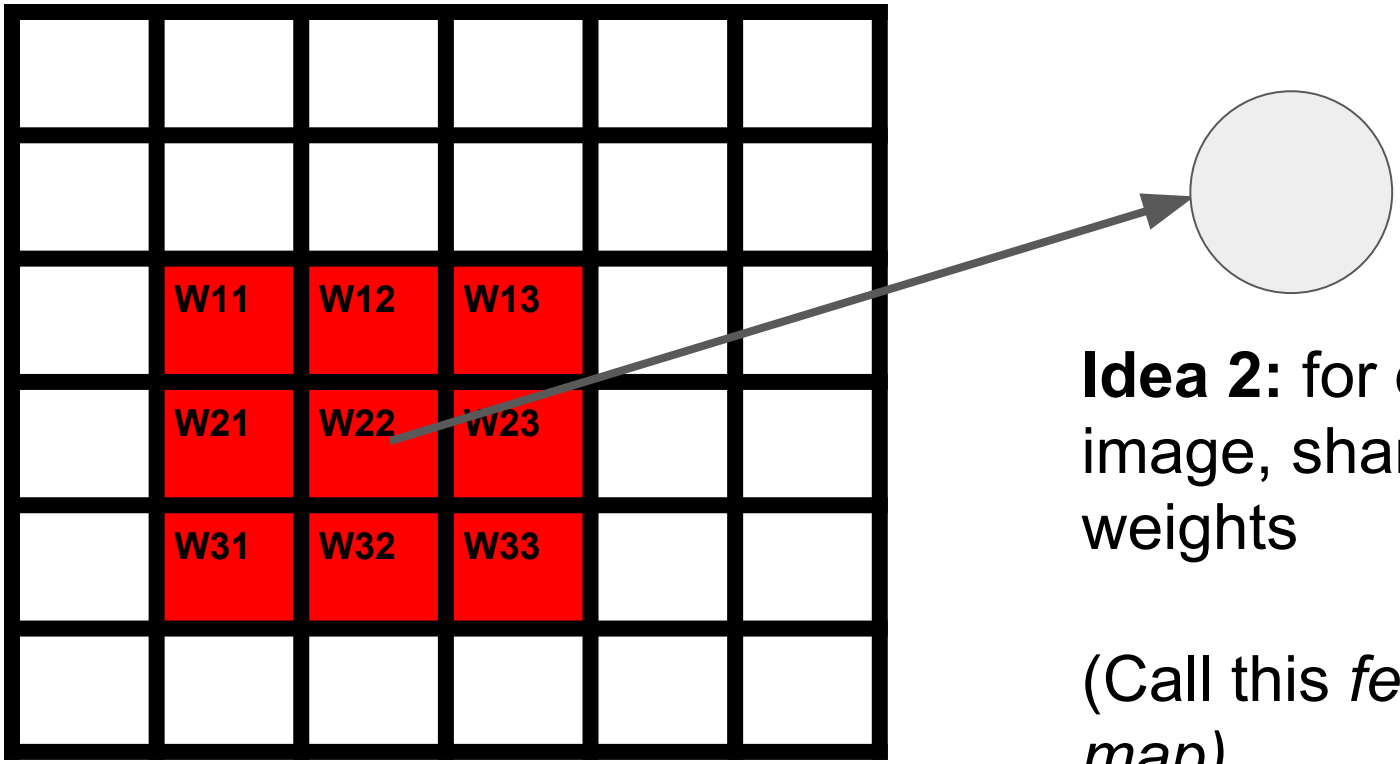
Each node needs a 3x3 matrix of weights

Image: 2d Matrix of Pixels, *shared weights*



Idea 2: for entire image, share the weights for all locations

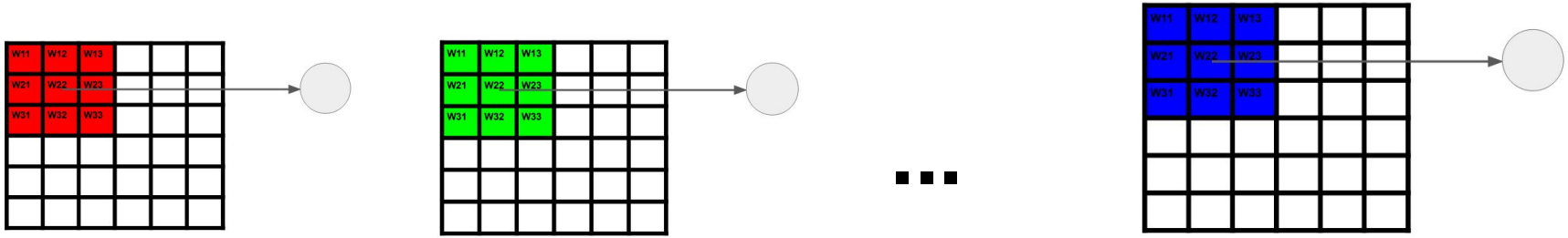
Image: 2d Matrix of Pixels, *shared weights*



Idea 2: for entire image, share the weights

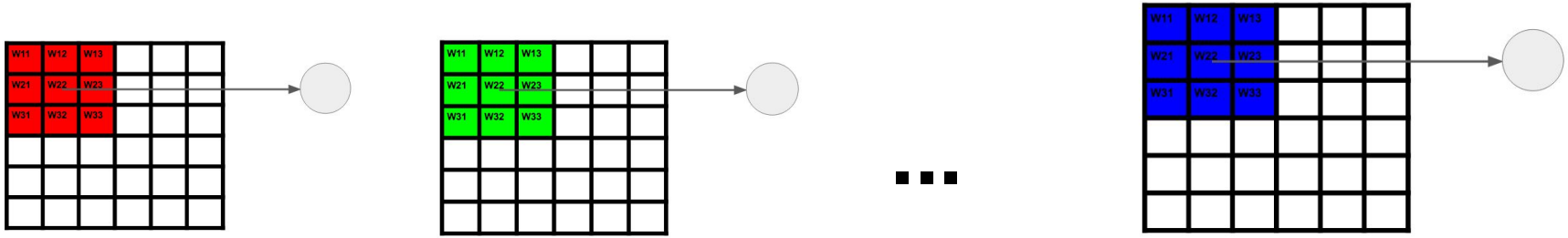
(Call this *feature map)*

Image: 2d Matrix of Pixels, *feature maps*



Idea 3: have many feature maps for the image

Image: 2d Matrix of Pixels, *feature maps*



Idea 4: you can repeat this for hidden layers (feature maps of feature maps)

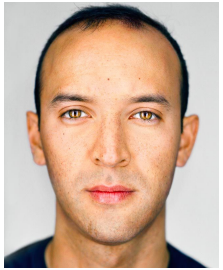
Why? Each feature map output can be arranged like pixels in an image; take output from feature map as an image

`theano.tensor.signal.conv2d`

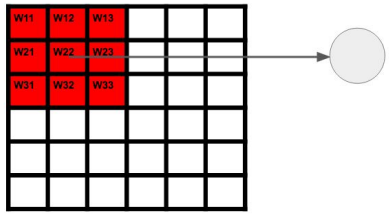
Convenience function for this architecture

Takes as input “4D tensors”, representing the data inputs and weights

4D Tensor for data

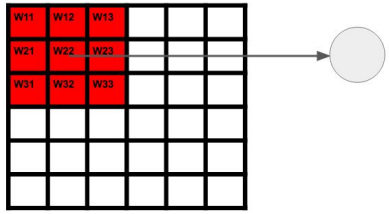


,



, X , Y

,

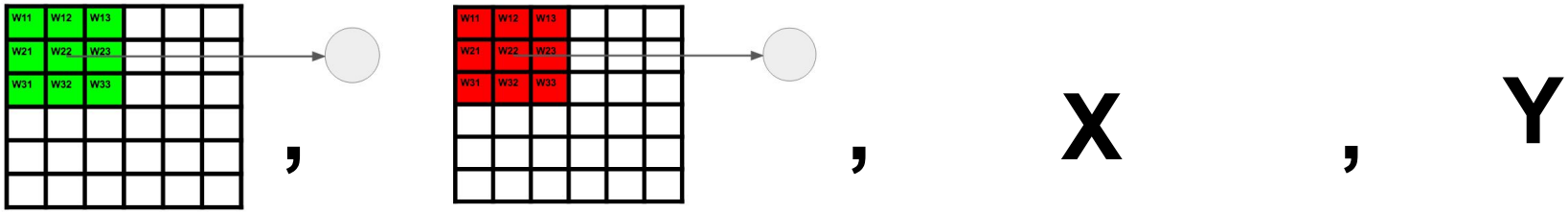


, X , Y

4D?

(1) Image (2) Feature map (3) x coordinate of pixel [center]
(4) y coordinate of pixel

4D Tensor for weights



(1) Feature maps out (2) Feature map in (3) x coordinate of pixel [center] (4) y coordinate of pixel

```
# Convolution layers.
w_1 =
theano.shared(np.asarray((np.random.randn(*(featureMapsL
ayer1, 1, patchWidth, patchHeight))*.01)))

w_2 =
theano.shared(np.asarray((np.random.randn(*(featureMapsL
ayer2, featureMapsLayer1, patchWidth, patchHeight))*.01)))

w_3 =
theano.shared(np.asarray((np.random.randn(*(featureMapsL
ayer3, featureMapsLayer2, patchWidth, patchHeight))*.01)))
...
```

```
def model(X, w_1, w_2, w_3, w_4, w_5, p_1, p_2):

    l1 = dropout(max_pool_2d(T.maximum(conv2d(X, w_1,
border_mode='full'),0.), (2, 2)), p_1)

    l2 = dropout(max_pool_2d(T.maximum(conv2d(l1, w_2),
0.), (2, 2)), p_1)

    l3 =
dropout(T.flatten(max_pool_2d(T.maximum(conv2d(l2, w_3),
0.), (2, 2)), outdim=2), p_1) # flatten to switch back to 1d
layers
    ...
```