1 Dropout

Briefly speaking drop parts of the activations randomly with probability p.
Note that we need to correct for this during the training (multiply by $(\frac{1}{1-p})$).

Example:

$$\begin{bmatrix} \text{RELU}(f_1) \\ \text{RELU}(f_2) \\ \text{RELU}(f_3) \end{bmatrix} \rightarrow \begin{bmatrix} \frac{1}{1-p}\text{RELU}(f_1) \\ 0 \\ \frac{1}{1-p}\text{RELU}(f_3) \end{bmatrix}$$

2 BatchNorm

Let x be a tensor of shape [BATCH, HIDDEN_DIM].

2.1 Training mode

For each element of the second dimension we are going to calculate mean and variance across the batch dimension.

```
mean = torch.mean(x, dim=0)
var = torch.var(x, dim=0, unbiased=False)
return (x - mean) / (torch.sqrt(var + eps)) * gamma + beta
```

Where gamma and beta are trainable parameters of the BatchNorm layer.

2.2 Eval mode

We can keep moving averages of mean and variance to use them in evaluation.

```
with torch.no_grad():
  running_mean = (1-momentum) * running_mean + momentum * mean
  running_var =  (1-momentum) * running_var + momentum * unbiased_var
```

3 Convolutions

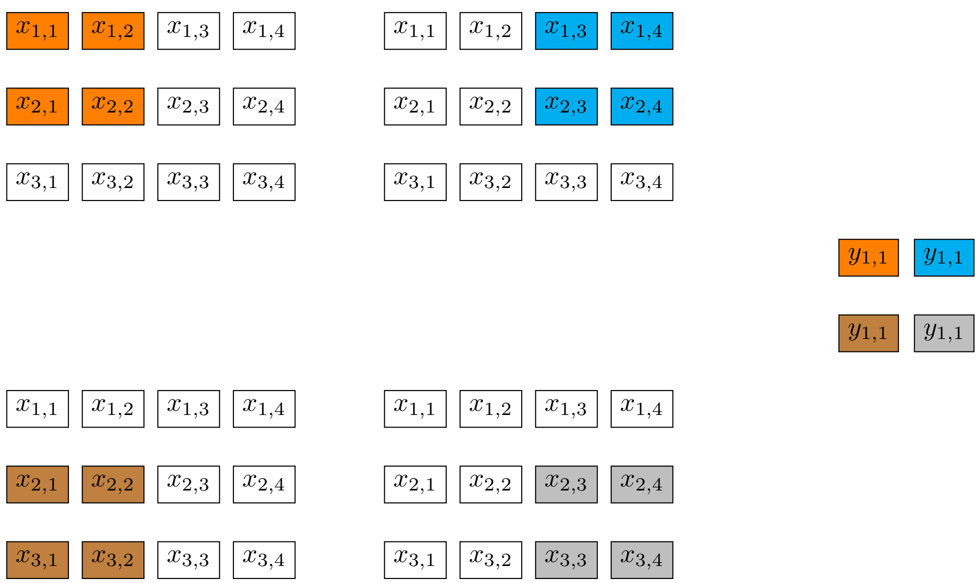
Consider the following example:

```
INPUT_SHAPE = 1x3x4
input = [
  [ 1,  2,  3,  4],
  [ 5,  6,  7,  8],
  [ 9, 10, 11, 12],
]

kernel = [
  [a, b],
  [c, d],
]

stride = (1, 2)
convolution(input, kernel, stride) = [
  [a*1 + b*2 + c*5 + d*6, a*3 + b*4 + c*7 + d*8],
  [a*5 + b*6 + c*9 + d*10, a*7 + b*8 + c*11 + d*12]
]
```

We basically slide throughout the image using a kernel and specified strides.



3.1 Torch

Lets look at

`torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0)`

- in_channels - number of channels/colors in the image (torch expects inputs of shape BATCH x CHANNELS x H x W)
- out_channels - number of output channels (number of kernels to use, each kernel will be of shape in_channels x (kernel_size, kernel_size) or in_channels x kernel_size)
- stride - int or tuple with stride for H and W dimension
- padding - int, tuple or string that specifies padding, when int then for both H and W, tuple for separate specification; pads both at the beginning and at the end

3.2 MaxPool

Similar to convolution but now we treat each input channel separately and instead of multyplying and adding we compute the max.

```
INPUT_SHAPE = 1x3x4
input = [
  [ 1,  2,  3,  4],
  [ 5,  6,  7,  8],
  [ 9, 10, 11, 12],
]
```

```
stride = (1, 2)
maxpool(input, kernel=(2,2), stride) = [
  [max(1, 2, 5, 6), max(3, 4, 7, 8)],
  [max(5, 6, 9, 10), max(7, 8, 11, 12)]
]
```