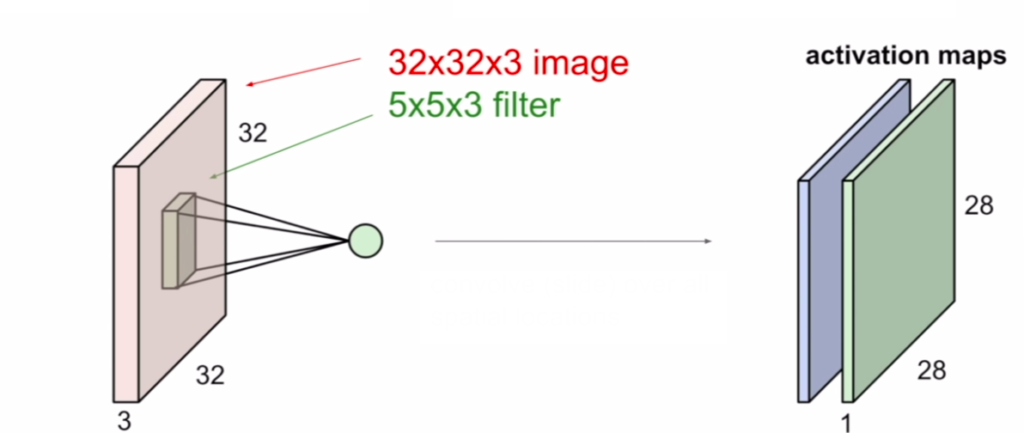
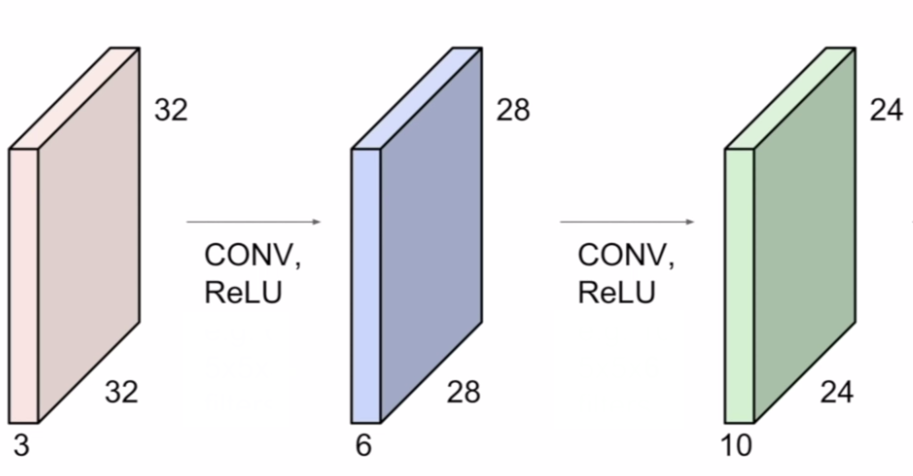
**How ReLU works in convolutional neural network**

Convolutional filters start at the upper left corner on top of every pixel in input image and at every position, it’s going to dot product and it will produce output which is called **activation map** and  fill it in activation function.

  
Each of the filters is producing an activation map.

Let’s take one filter, sliding it over all of the spatial locations in the image and then we’re going to get this activation map out which is the value of that filter at every spatial location.

  
The output of the layer is going to be the input to the next convolutional layer.

ConvNet has a sequence of convolutional layers stacked on top of each other. And then we’re going to intersperse these with activation functions, for example, a ReLU activation function.

In the convolutional layer, we have the data coming in.We multiply by  weight in the convolutional layer. Then we’ll pass this through an activation function for nonlinearity.

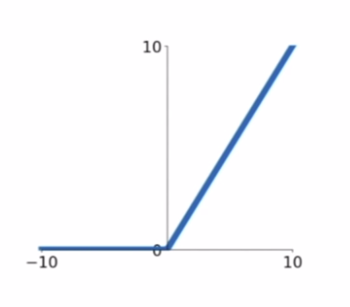
**ReLU Activation Functions**

ReLU was starting to be used a lot around 2012 when we had AlexNet, the first major convolutional neural network that was able to do well on ImageNet and large-scale data.

ReLU stands for **Re**ctified **L**inear **U**nit, and is represented by the function.

**ReLU(x)=*max*(0,X)**

It interspersed nonlinearity between many of the convolutional layers.  
In a nutshell, ReLU is used for filtering information that propagates forward through the network.



It takes an elementwise operation on your input and basically if your input is negative, it’s going to put it to zero.and then if it’s positive, it’s going to be just passed through its identity. This is one that’s pretty commonly used because it doesn’t saturate in the positive region.

The sigmoid was not zero-centered *tanh* fixed this and now ReLU has this problem again and that’s one of the issues of the ReLU. ReLU d**oesn’t activate for negative inputs**, it’s possible to end up with “dead neurons” that never fire.

**Advantages of ReLU over Sigmoid**

**1.Vanishing Gradient Problem**

In the Sigmoid activation function, the gradient is typically fraction between 0 and 1. In a multi-layer NN, these multiply and generate exponentially small gradients. So each step of gradient descent will make only a tiny change to the weights, leading to slow convergence. In contrast with ReLu activation, the gradient of the ReLu is either 0 or 1, so after many layers often the gradient will include the product of a bunch of 1’s, and thus the overall gradient is not too small or not too large.

**2.Performance**

**max(0,X)** runs much faster than sigmoid function **(1/(1+e^(-a))** which uses an exponent which is computational slow when done often. This is true for both feed forward and back propagation as the gradient of ReLU (if a<0, =0 else =1) is also very easy to compute compared to sigmoid.

**3.Sparsity**

The other benefit of ReLUs is sparsity. Sparse representations seem to be more beneficial than dense representations.Sparsity arises when X≤0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations.

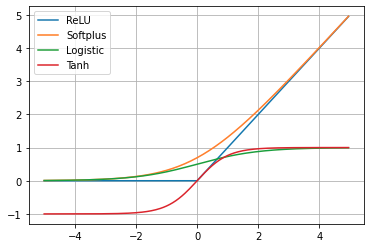
**Conclusion**

Sigmoid activation function was quite popular when train neural networks 10 years ago but it has vanishing gradients problem. The general recommendation is that you probably want to stick with ReLU for most cases or default choice because it tends to work well for a lot of different architectures.

**Why do we use ReLUs?** We use ReLUs for the same reason we use any other non-linear activation function: To achieve a non-linear transformation of the data.

**Why do we need non-linear transformations?** We apply non-linear transformations in the hope that the transformed data will be (close to) linear (for regression) or (close to) linearly separable (for classification). Drawing a linear function through non-linearly transformed data is equivalent to drawing a non-linear function through original data.

**Why are ReLUs better than other activation functions?** They are simple, fast to compute, and don't suffer from vanishing gradients, like sigmoid functions (logistic, tanh, erf, and similar). The simplicity of implementation makes them suitable for use on GPUs, which are very common today due to being optimised for matrix operations (which are also needed for 3D graphics).

[](https://i.stack.imgur.com/YKS2g.png)

**Why do we need matrix operations in neural networks?**: It's a compact and computationally efficient way of propagating the signals between the layers (multiplying the output of the previous layer with the weight matrix).

**Isn't softmax activation function for neural networks?** Softmax is not really an activation function of a single neuron, but a way of normalising outputs of multiple neurons. It is usually used in the output layer, to enforce the sum of outputs to be one, so that they can be interpreted as probabilities. You could also use it in hidden layers, to enforce the outputs to be in a limited range, but other approaches, like [batch normalisation](https://en.wikipedia.org/wiki/Batch_normalization), are better suited for that purpose.

**P.S. (1)** ReLU stands for "rectified linear unit", so, strictly speaking, it is a neuron with a [(half-wave) rectified-linear](https://en.wikipedia.org/wiki/Rectifier#Half-wave_rectification) activation function. But people usually mean the activation function when they talk about ReLUs.

**P.S. (2)** Passing the output of softmax to a ReLU doesn't have any effect because softmax produces only non-negative values, in range [0,1][0,1], where ReLU acts as identity function, i.e. doesn't change them

--------------------------------------------=--------------------------------------------

ReLU is the max function(x,0) with input x e.g. matrix from a convolved image. ReLU then sets all negative values in the matrix x to zero and all other values are kept constant.

ReLU is computed after the convolution and is a nonlinear activation function like tanh or sigmoid.

Softmax is a classifier at the end of the neural network. That is logistic regression to normalize outputs to values between 0 and 1. (Alternative here is a SVM classifier).

CNN Forward Pass e.g.: input->conv->ReLU->Pool->conv->ReLU->Pool->FC->softmax