Convolutional Neural Networks

Moisey Alaev, Naoya Kumagai

Table of Contents

Historical Background

In the 1950s-60s , two scientists, David Hubel and Torsten Wiesel, presented research predicated on observing the visual cortexes of cats and monkeys. They gathered that neurons individually correspond to small regions of the visual field. They described a “receptive field” that is the visual space from the broader visual stimuli which causes the firing of a single neutron.

In the 1980s Dr. Kunihiko Fukushima proposed the “neocognitron” model that was based off of Hubel and Wiesel’s work. Dr. Fukushima bridged the gap between the biological understanding of visual learning to the mathematical foundations researched over the next decades.

By the 1990s the first modern convolutional neural networks (CNNs) were created. Inspired by the neocognitron, Yann LeCun trained what is considered the first CNN on the MNIST dataset of handwritten digits.

Most recently, in 2012 a CNN model called AlexNet achieved incredible accuracy labeling pictures in the popular ImageNet challenge. In fact, AlexNet demonstrated almost half the error percentage of the human eye in visually identifying items. This breakthrough led to the current resurgence of interest into CNNs.

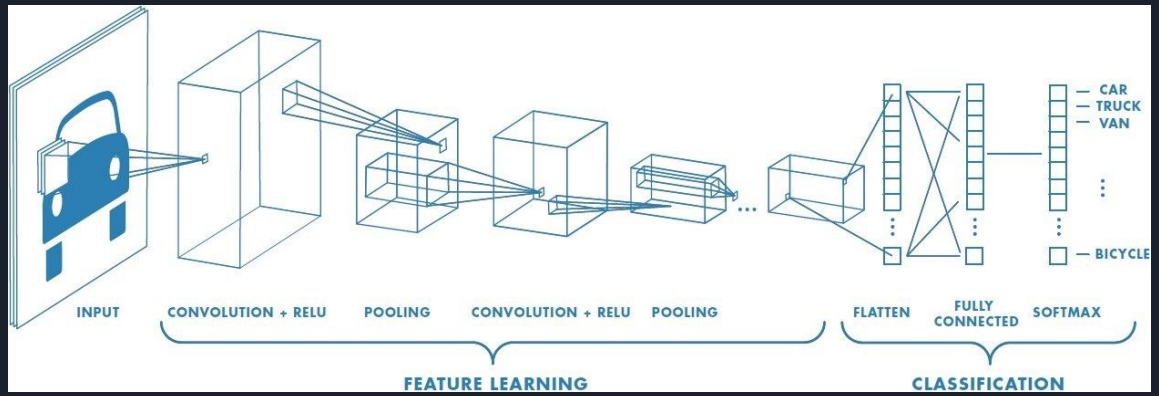
What is a Convolutional Neural Network?

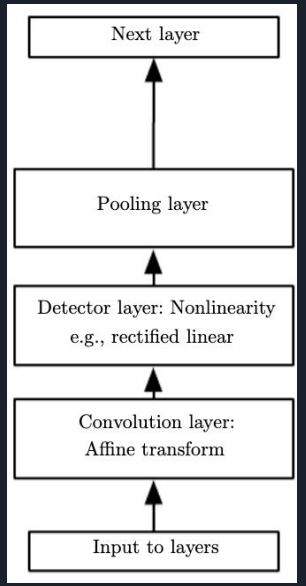
A Convolutional Neural Network (CNN) is a class of deep neural networks that are architected for the task of identifying and analyzing visual imagery. The primary difference in these models opposed to regular artificial neural networks is that they implement a “feature learning” tasks before being fed into a “classification” network.

The feature learning process involves two layers that our research described: “convolution with activation” and “pooling”

Implementations of CNNs in modern technology include

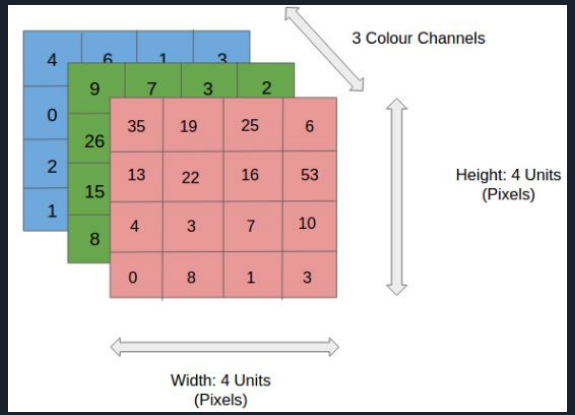
- recommender systems (Instagram, YouTube, Facebook, etc.)
- medical image analysis (identifying hemorrhages, cancers, etc.)
- face recognition (security)

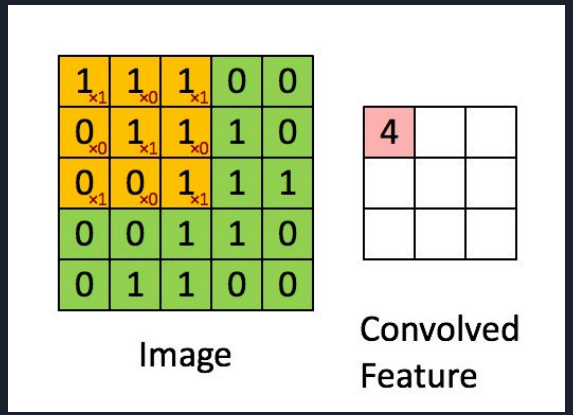


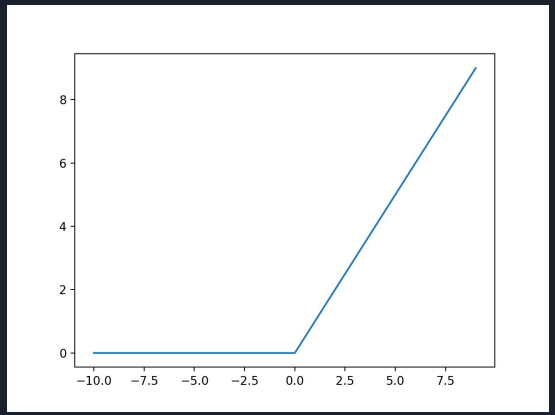


Layer type I: Convolution & Activation Layer

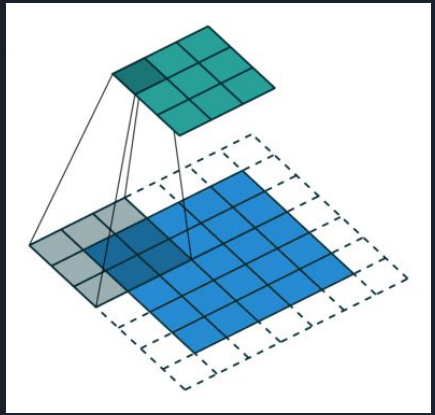
Convolution is the linear operation that takes an input image and applies an element-wise dot product with a “filter”. The filter is representative of weights that are trained to detect features. Mathematically, the input image is represented as a 3D array: height and width defined by the pixel size of the image and the 3rd dimension defined by the color channels. A filter is always of smaller height and width, but must be of the same depth. Therefore, the filter is applied to a section of the image at a time, called the “receptive field”. The filter then continues to take strides over the image to get the output layers, a.k.a “ feature maps”. Finally the feature maps are passed through an activation function, often times ReLU.

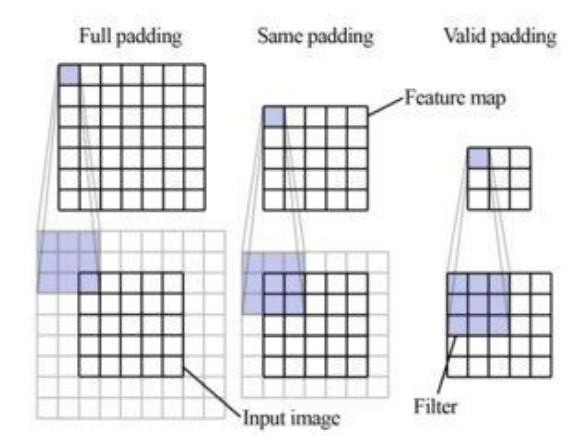






Padding Image During Convolution



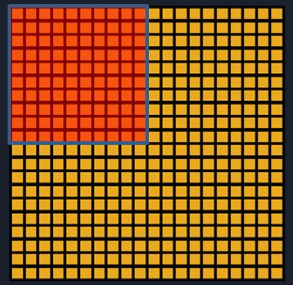


Often times since the pixels around the edges of the image are transversed fewer times by the kernel, the information or features around the edge is lost during convolution. Consequently, a method called padding is implemented to circumvent this problem. The most common procedure, “Same Padding”, places zeros around the perimeter of the image array. There also exists other padding methods such as “Full Padding” and “Valid Padding” (valid padding doesn’t actually change the input perimeter).

Layer type II: Pooling layer

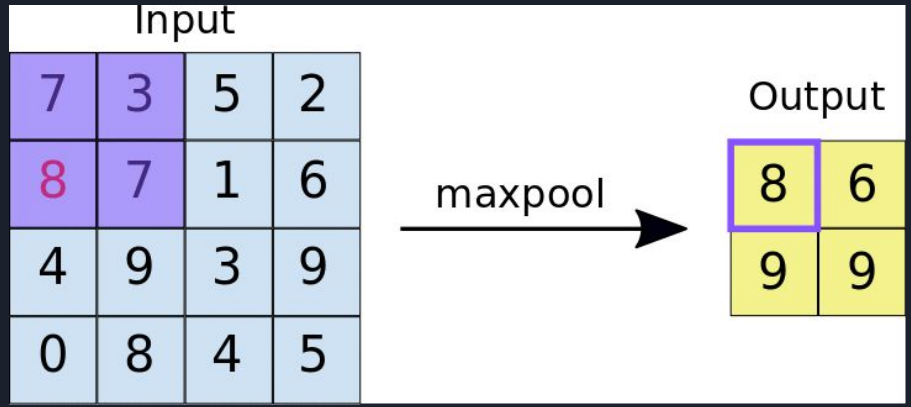
Since we apply convolution between every layer and often several times per layer, the computational intensity of maintaining the parameters from the feature maps progressively grows. Therefore, to counter dealing with large spatial size and to overall reduce the computation time between layers, we apply the pooling operation.

Pooling is applied during forward propagation by taking an input block of larger dimensional size corresponding to the receptive field and transforming it into a 1×1 output block in the pooled matrix. There are several different types of ways to choose this output block, the two most commonly implemented: average and max pooling. Max pooling chooses the largest value in the receptive field as the output block. While average pooling takes the sample average of the values in the receptive fields.



Convolved feature

Pooled feature

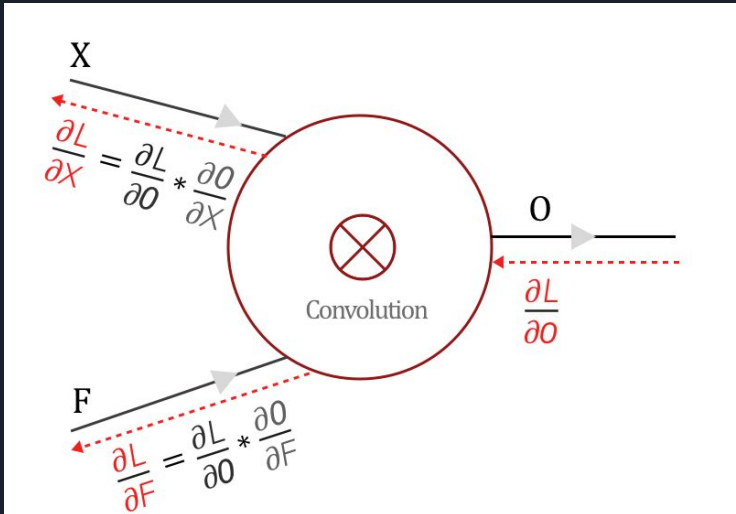


Backpropagation

CNNs use backpropagation to update its parameters (filters). Convolution is used during the backpropagation as well.

Each layer receives the loss gradient from the previous (forward propagation output) layer. It then,

1. Updates its filter
2. Passes on the loss gradient information to the next layer

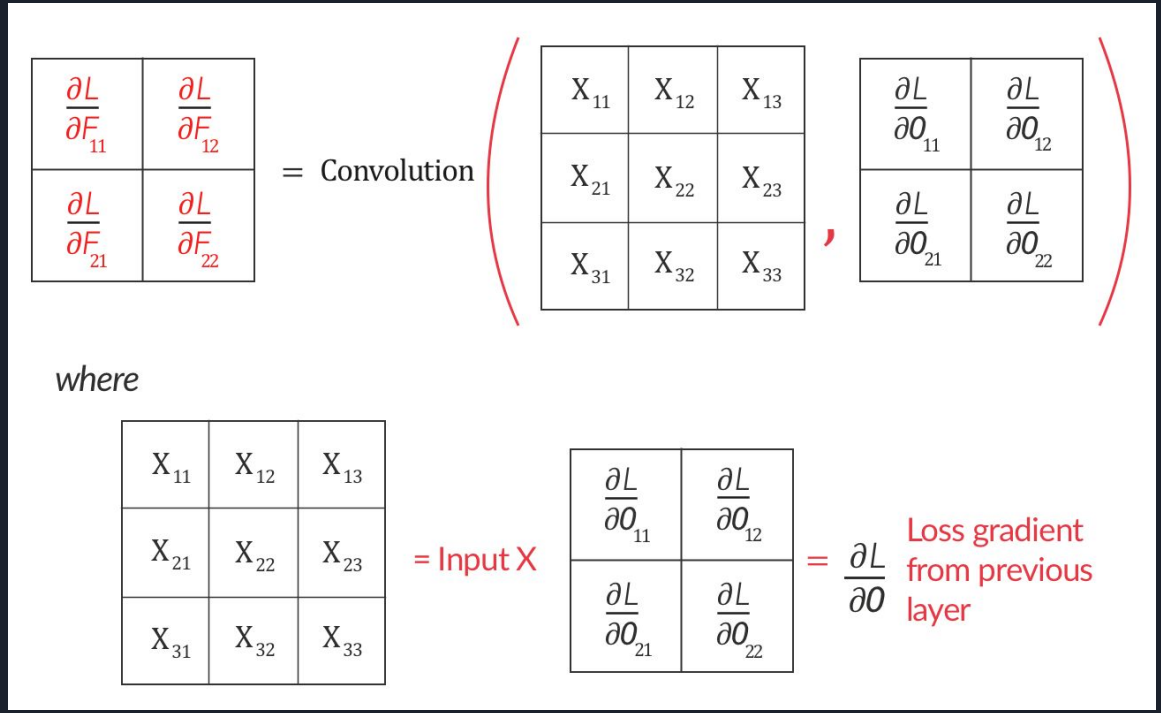


$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

Backpropagation

Finding the partial derivative of the loss function w.r.t. the filter.



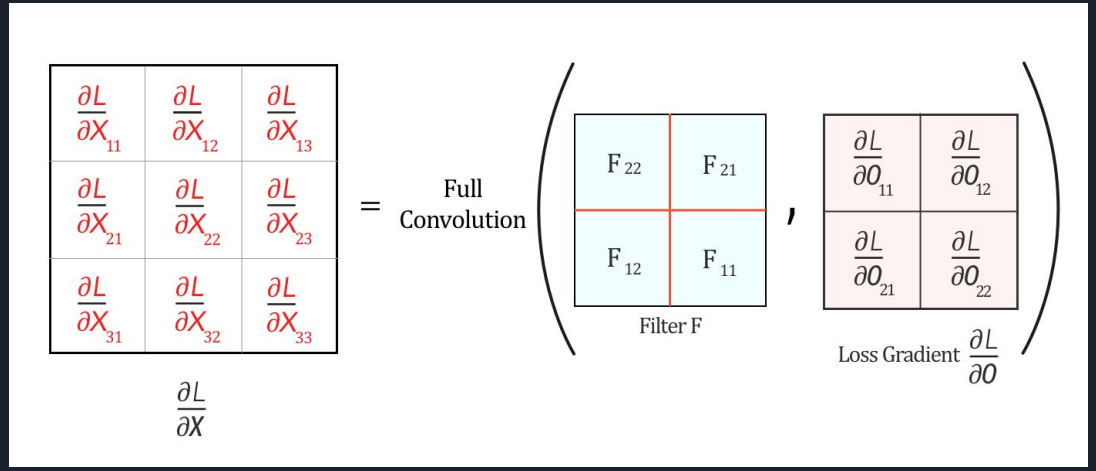
Backpropagation

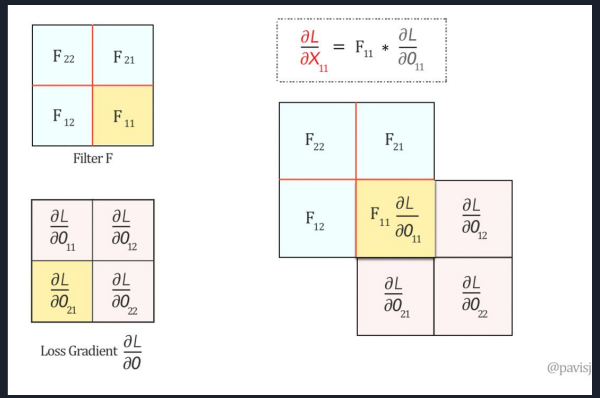
Passing the loss function information to the next layer.

*Full convolution can be interpreted as:

-flipping the filter horizontally and vertically

-taking the convolution with full padding





Experiment

-MNIST dataset of 60,000 handwritten digit images.

-We trained our model 10 times using the training set and tested its accuracy using a testing set of 10,000 more images.

-Model:

3 Convolution layers

2 Pooling layers

2 Fully Connected layers

2 Dropout layers to avoid overfitting

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	80
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 12, 12, 128)	18560
conv2d_2 (Conv2D)	(None, 10, 10, 256)	295168
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 256)	0
dropout (Dropout)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
dense (Dense)	(None, 128)	819328
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
activation (Activation)	(None, 10)	0

Experiment

Feature maps after first convolution and max pooling.

Usually, the first layer detects edges and deeper layers detect abstract characteristics of the image.

====>

Experiment

We trained the model using cross entropy as the error.

$$H(p,q) = -\sum_{x} p(x) ln(q(x)).$$

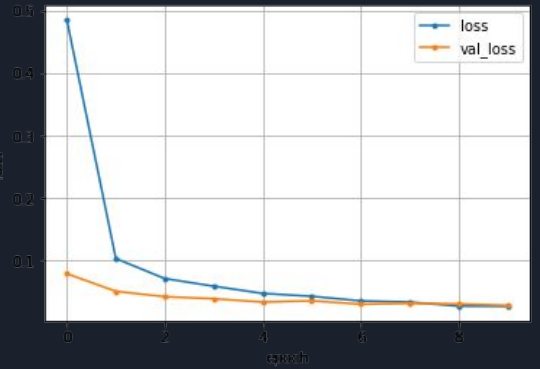
p(x): output of final layer

q(x): correct label (vector of size 10 with one entry equal to 1 and others equal to 0)

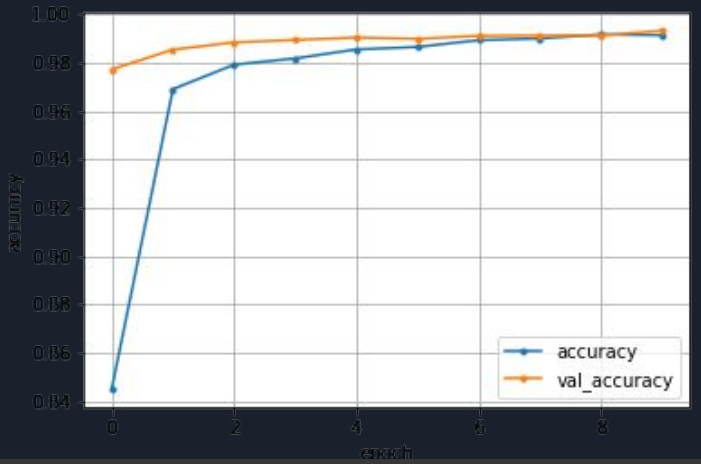
$$y_i = \frac{e^{x_i}}{\sum_{k=1}^{N} e^{x_k}} \text{ for i = 1,2,...,N}$$

The final layer uses softmax as the activation function.

-This function gives output of the same size with non-negative floats that sum up to 1 (for example, (0.2, 0, 0.1, 0.7))



Experiment Result



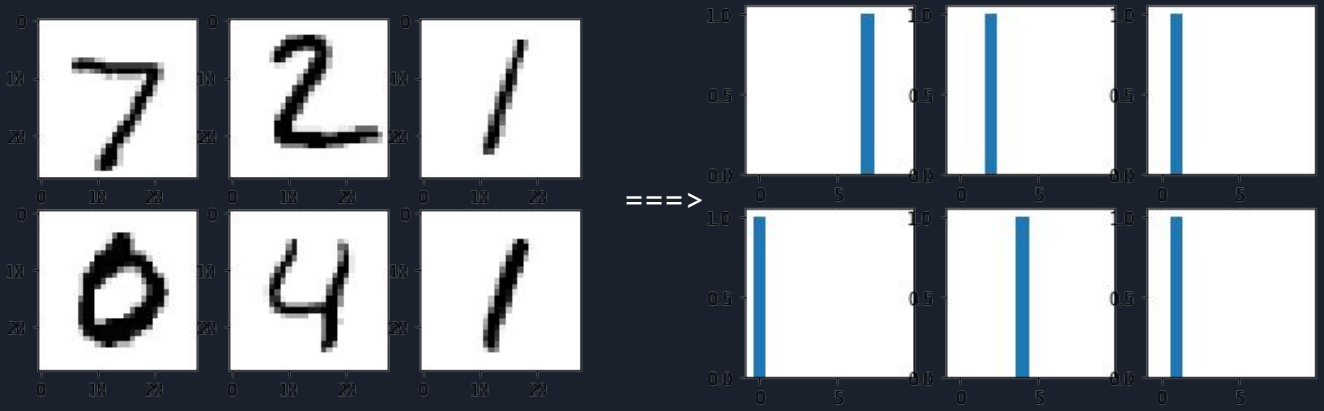
-Accuracy increased but plateaued after around 10 epochs.

-Achieved over 99% accuracy in the testing data.

-No apparent overfitting

-Below: outputs of the final layer for each image on the left.

```
Test loss: 0.020970428362488747
Test accuracy: 0.9934999942779541
```



Conclusion

Convolutional Neural Networks are powerful architectures geared towards meeting modern day visual learning tasks without requiring very large amounts of computational time. As a result of our investigation we have have uncovered the following features that make CNNs so useful:

- Sparse Interactions
  - By using filters that are smaller than the inputs we can identify low level features like edges in an image.
- Parameter Sharing
  - Opposed to traditional NN, CNNs do not use each weight in the traditional weight matrix once, but instead use any given element of the kernel in every position of the input (except for the boundary depending on padding implementation)
- Equivariant Representations
  - Object detection is invariant to the changes in illumination and position, thus, allowing our model to handle a degree of distortion and transformation in the image.

THE END


COMPUTER VISION PROBLEMS
ADD A CONVOLUTIONAL NEURAL NETWORK