CNN – TUTORIAL – WITH KERAS

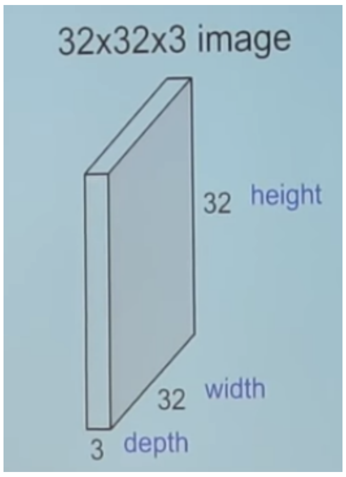
Table of Contents

1. Introduction to Convolutional Neural Network (CNN)

- In deep learning, a **convolutional neural network (CNN or ConvNet)** is a class of deep neural networks.
- CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output.
- The whole network has a loss function and all the basics for neural networks will apply on CNNs as well.
- A convolutional neural network (CNN) consists of an input and an output layer, as well as multiple hidden layers.
- The hidden layers of a CNN typically consist of a series of convolutional layers that convolve with a multiplication or other dot product.
- The activation function is commonly a RELU layer, and is subsequently followed by additional convolutions such as pooling layers, fully connected layers and normalization layers, referred to as hidden layers because their inputs and outputs are masked by the activation function and final convolution.
- The final convolution, in turn, often involves backpropagation in order to more accurately weight the end product.
- CNNs have wide variety of applications in image and video recognition, recommender systems and natural language processing.
-

CNNs operate over images

- Unlike neural networks, where the input is a vector, in CNNs the input is a multi-channeled image (3 channeled) as shown in the figure below:



2. Basic terminology of CNN

- We should be aware of some basic terminology regarding CNN. These are as follows:-

2.1 Filters

- CNN's make use of filters to detect what features, such as edges, are present throughout an image.
- A **filter** is just a matrix of values, called weights, that are trained to detect specific features.
- The filter moves over each part of the image to check if the feature it is meant to detect is present.
- To provide a value representing how confident it is that a specific feature is present, the filter carries out a convolution operation, which is an element-wise product and sum between two matrices.
- When the feature is present in part of an image, the convolution operation between the filter and that part of the image results in a real number with a high value. If the feature is not present, the resulting value is low.

2.2 Convolutional

- When programming a CNN, each convolutional layer within a neural network should have the following attributes:
  - Input is a tensor with shape (number of images) x (image width) x (image height) x (image depth).

    - Convolutional kernels whose width and height are hyper-parameters, and whose depth must be equal to that of the image.
- Convolutional layers convolve the input and pass its result to the next layer.

2.3 Pooling

- Convolutional networks may include local or global pooling layers to streamline the underlying computation.
- Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer.
- Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer.
- In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

2.4 Fully connected

- Fully connected layers connect every neuron in one layer to every neuron in another layer.
- It is in principle the same as the traditional multi-layer perceptron neural network (MLP).
- The flattened matrix goes through a fully connected layer to classify the images.

2.5 Receptive field

- In neural networks, each neuron receives input from some number of locations in the previous layer. In a fully connected layer, each neuron receives input from every element of the previous layer.
- In a convolutional layer, neurons receive input from only a restricted subarea of the previous layer. Typically the subarea is of a square shape (e.g., size 5 by 5). The input area of a neuron is called its **receptive field**.
- So, in a fully connected layer, the receptive field is the entire previous layer. In a convolutional layer, the receptive area is smaller than the entire previous layer.

2.6 Weights

- Each neuron in a neural network computes an output value by applying a specific function to the input values coming from the receptive field in the previous layer.
- The function that is applied to the input values is determined by a vector of weights and a bias (typically real numbers). Learning, in a neural network, progresses by making iterative adjustments to these biases and weights.
- The vector of weights and the bias are called filters and represent particular features of the input (e.g., a particular shape).

- A distinguishing feature of CNNs is that many neurons can share the same filter. This reduces memory footprint because a single bias and a single vector of weights are used across all receptive fields sharing that filter, as opposed to each receptive field having its own bias and vector weighting.

2.7 Strides

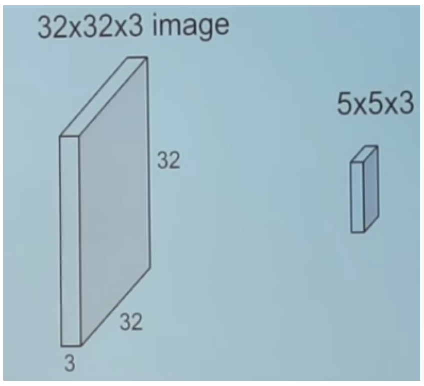
- **Strides** is the number of pixels shifts over the input matrix.
- When the stride is 1 then we move the filters to 1 pixel at a time.
- When the stride is 2 then we move the filters to 2 pixels at a time and so on.

2.8 Padding

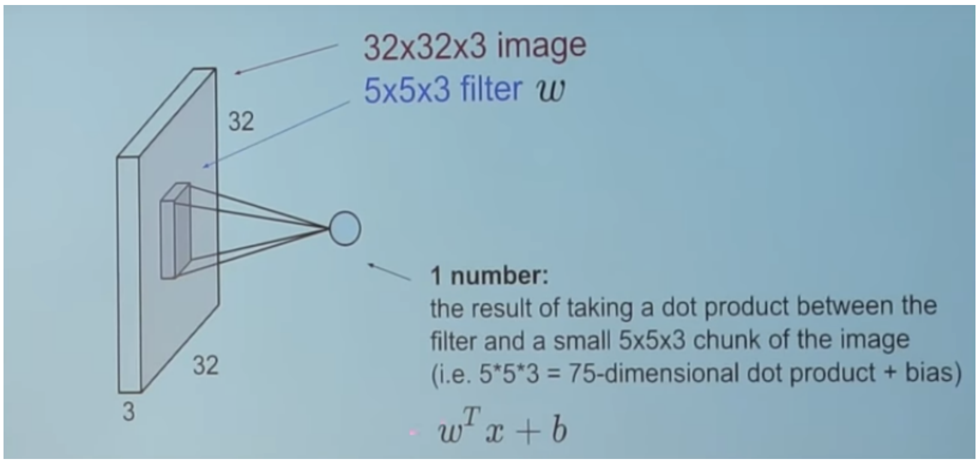
- In CNNs, we can also apply a technique of filling zeros around the margin of the image to improve the sweep that is done with the sliding window. The parameter to define this technique is called “padding”
- In CNNs, sometimes filter does not perfectly fit the input image. In this case, we have two options:
    - We can apply a technique of filling zeros around the margin of the image so that it fits the input image perfectly. This is called **zero-padding** where we pad the gaps with zeros.
    - Another option is to drop the part of the image where the filter does not fit the image perfectly. This is called **valid padding** which keeps only valid part of the image.

3. What is Convolution

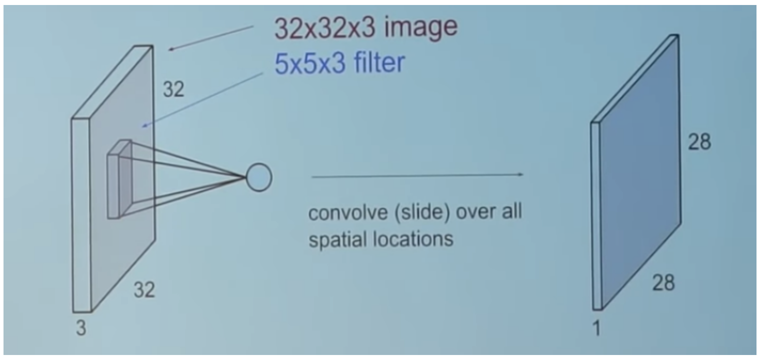
- Convolutional neural networks are simple neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- The name `convolutional neural network` indicates that the network employs a mathematical operation called convolution.
- Convolution is a specialized kind of linear operation. In mathematics, convolution is a mathematical operation on two functions (f and g) that produces a third function expressing how the shape of one is modified by the other.
- The term **convolution** refers to both the result function and to the process of computing it. It is defined as the integral of the product of the two functions after one is reversed and shifted.
- **Convolution** is explained visually as follows:



- In the above plot, we take the 5x5x3 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image.


32x32x3 image
5x5x3 filter w
32
32
3
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)
$w^T x + b$

- For every dot product taken, the result is a scalar.
- Now, we convolve the complete image with the filter as shown below.



4. Basic building blocks of CNN

- A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function.
- We use three main types of layers to build CNN architectures - **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer**. We will stack these layers to form a full CNN architecture.

- There are two other layers - **ReLU Layer** and **Loss Layer** that are also used to build a CNN. These layers are described below:

4.1 Convolutional layer

- The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume.
- During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter.
- As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input.
- Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer.
- Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

4.2 Pooling layer

- Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which `max pooling` is the most common. It partitions the input image into a set of non-overlapping rectangles and for each such sub-region, outputs the maximum.
- Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks.
- The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.
- The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.
- In addition to max pooling, pooling units can use other functions, such as average pooling or ℓ2-norm pooling.

4.3 ReLU layer

- **ReLU** is the abbreviation of **Rectified Linear Unit**, which applies the non-saturating activation function.

- It is given by the following formula f(x) = max(0,x).
- It effectively removes negative values from an activation map by setting them to zero.
- It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.
- ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

4.4 Fully connected layer

- Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers.
- Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks.
- Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

4.5 Loss layer

- The **loss layer** specifies how training penalizes the deviation between the predicted (output) and true labels and is normally the final layer of a neural network. Various loss functions are appropriate for different tasks may be used.
- **Softmax loss** is used for predicting a single class of K mutually exclusive classes.
- **Sigmoid cross-entropy loss** is used for predicting K independent probability values in [0,1].
- **Euclidean loss** is used for regressing to real-valued labels (-∞,∞).

5. Elements of a CNN model

- Enough of theory, now its time to see CNN in action.
- We will see how the convolutional neural network can be programmed with Keras using MNIST dataset.
- There are several values to be specified in order to parameterize the convolution and pooling stages.
- In this case, we will use a simplified model with a stride of 1 in each dimension (**stride** is the size of the step with which the window slides) and a padding of 0 (**padding** is the filling with zeros around the image). Both these hyperparameters will be presented below.
- The **pooling operation** will be a max-pooling operation with a 2×2 window.
- So, our CNN model will consist of a convolution followed by a max-pooling.

- In this case, we will have 32 filters using a 5×5 window for the convolutional layer and a 2×2 window for the pooling layer.
- We will use the ReLU activation function.
- In this case, we are configuring a convolutional neural network to process an input tensor of size (28, 28, 1), which is the size of the MNIST images (the third parameter is the color channel which in our case is depth 1), and we specify it by means of the value of the argument input_shape = (28, 28,1) in our first layer:

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (5,5), activation = 'relu', input_shape= (28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.summary()
```

```
Using TensorFlow backend.

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
=================================================================
Total params: 832
Trainable params: 832
Non-trainable params: 0
_________________________________________________________________
```

Description of parameters

- The number of parameters of the conv2D layer corresponds to the weight matrix W of 5×5 and a b bias for each of the filters is 832 parameters (32 × (25 + 1)).
- Max-pooling does not require parameters since it is a mathematical operation to find the maximum.

6. Basic CNN model

- Now, we will build a basic CNN model. We will stack several layers to build the model.
- We will create a second group of layers that will have 64 filters with a 5×5 window in the convolutional layer and a 2×2 window in the pooling layer.
- In this case, the number of input channels will take the value of the 32 features that we have obtained from the previous layer.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (5,5), activation = 'relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (5,5), activation = 'relu'))
model.add(layers.MaxPooling2D((2,2)))
model.summary()

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 64)          0
=================================================================
Total params: 52,096
Trainable params: 52,096
Non-trainable params: 0
_________________________________________________________________
```

Description of parameters

- In this case, the size of the resulting second convolution layer is 8×8 since we now start from an input space of 12×12×32 and a sliding window of 5×5, taking into account that it has a stride of 1.
- The number of parameters 51,264 corresponds to the fact that the second layer will have 64 filters (as we have specified in the argument), with 801 parameters each (1 corresponds to the bias, and a W matrix of 5×5 for each of the 32 entries). That means ((5 × 5 × 32) +1) × 64 = 51264.
- The next step, now that we have 64 4x4 filters, is to add a densely connected layer, which will serve to feed a final layer of softmax activation function with the following code snippet:

```
model.add(layers.Dense(10, activation='softmax'))
```

- In this example, we have to adjust the tensors to the input of the dense layer like the softmax, which is a 1D tensor, while the output of the previous one is a 3D tensor. So, we will first flatten the 3D tensor to one of 1D.
- Our output (4,4,64) must be flattened to a vector of (1024) before applying the Softmax with the following code snippet:

```
model.add(layers.Flatten())
```

- So, our final model becomes:

```
model = models.Sequential()

model.add(layers.Conv2D(32,(5,5),activation='relu', input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(10, activation='softmax'))
```

View the model summary

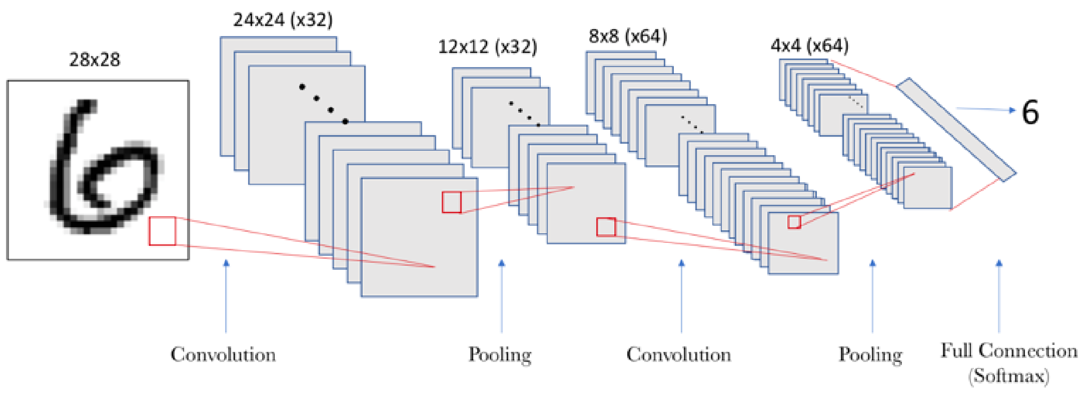
- We can view the summary of the model with the `summary()` method.
- With the `summary()` method, we can see this information about the parameters of each layer and shape of the output tensors of each layer.

```
model.summary()

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250
=================================================================
Total params: 62,346
Trainable params: 62,346
Non-trainable params: 0
_________________________________________________________________
```

Visual representation of the model

- A visual representation of the above information is shown in the following figure, where we see a graphic representation of the shape of the tensors that are passed between layers and their connections.



7. Training and evaluation of the model

- Now that we have built our CNN model, its time to train it.
- It means to adjust the parameters of all the convolutional layers.

- Training can be done as follows:

```
from keras.datasets import mnist
from keras.utils import to_categorical


(train_images, train_labels), (test_images, test_labels) = mnist.load_data()


train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255


test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255


train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)


batch_size = 100
epochs = 10


model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])



model.fit(train_images, train_labels,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1)

Downloading data from https://s3.amazonaws.com/img-datasets/mnist.npz
11493376/11490434 [==============================] - 1s 0us/step
Epoch 1/10
60000/60000 [==============================] - 31s 519us/step - loss: 0.9222 -
accuracy: 0.7754
Epoch 2/10
60000/60000 [==============================] - 31s 510us/step - loss: 0.2627 -
accuracy: 0.9231
Epoch 3/10
60000/60000 [==============================] - 31s 521us/step - loss: 0.1916 -
accuracy: 0.9444
Epoch 4/10
60000/60000 [==============================] - 31s 510us/step - loss: 0.1551 -
accuracy: 0.9542
Epoch 5/10
60000/60000 [==============================] - 31s 518us/step - loss: 0.1320 -
accuracy: 0.9618
Epoch 6/10
60000/60000 [==============================] - 30s 504us/step - loss: 0.1160 -
accuracy: 0.9663
Epoch 7/10
60000/60000 [==============================] - 31s 516us/step - loss: 0.1047 -
accuracy: 0.9695
Epoch 8/10
60000/60000 [==============================] - 30s 507us/step - loss: 0.0958 -
accuracy: 0.9720
Epoch 9/10
```

```
60000/60000 [==============================] - 31s 517us/step - loss: 0.0885 -
accuracy: 0.9742
Epoch 10/10
60000/60000 [==============================] - 32s 532us/step - loss: 0.0827 -
accuracy: 0.9761

<keras.callbacks.callbacks.History at 0x7f6cff175940>
```

Now, its time to evaluate the model. Model evaluation can be done as follows:

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test loss:', test_loss)
print('Test accuracy:', test_acc)

10000/10000 [==============================] - 2s 209us/step
Test loss: 0.06909727230984718
Test accuracy: 0.9805999994277954
```

8. Hyperparameters of the convolution layer

- There are three main hyperparameters of the CNN model. These are as follows:
    - the size and number of the filters window,
    - the padding, and
    - the stride.
- I will describe these hyperparameters below:

Size and number of filters

- The size of the window `(window_height × window_width)` that holds information from spatially close pixels is usually 3×3 or 5×5. The number of filters that tells us the number of characteristics that we want to handle (output_depth) is usually 32 or 64.
- In the Conv2D layers of Keras, these hyperparameters are what we pass as arguments in this order:

Conv2D(output_depth, (window_height, window_width))

Padding

- Padding is the technique of filling zeros around the margin of the image to improve the sweep that is done with the sliding window. The parameter to define this technique is called `padding`.

Stride

- **Strides** is the number of pixels shifts over the input matrix.

- When the stride is 1 then we move the filters to 1 pixel at a time.
- When the stride is 2 then we move the filters to 2 pixels at a time and so on.

9. Results and conclusion

- We can see that in the convolutional layers, where more memory is required and, therefore, more computation to store the data is needed.
- In contrast, in the densely connected layer of softmax, little memory space is needed. But, in comparison, the model requires numerous parameters which must be learned.
- It is important to be aware of the sizes of the data and the parameters because, when we have models based on convolutional neural networks, they have many layers, and these values can shoot exponentially.
- Our CNN model offers an accuracy of approximately 98%.