Niklas Lang Follow

Dec 4, 2021 · 9 min read · Listen

Using Convolutional Neural Network for Image Classification

Pho... ...ash

The Convolutional Neural Network (CNN or ConvNet) is a subtype of Neural Networks that is mainly used for applications in image and speech recognition. Its built-in convolutional layer reduces the high dimensionality of images without losing its information. That is why CNNs are especially suited for this use case.

Image Processing Problems

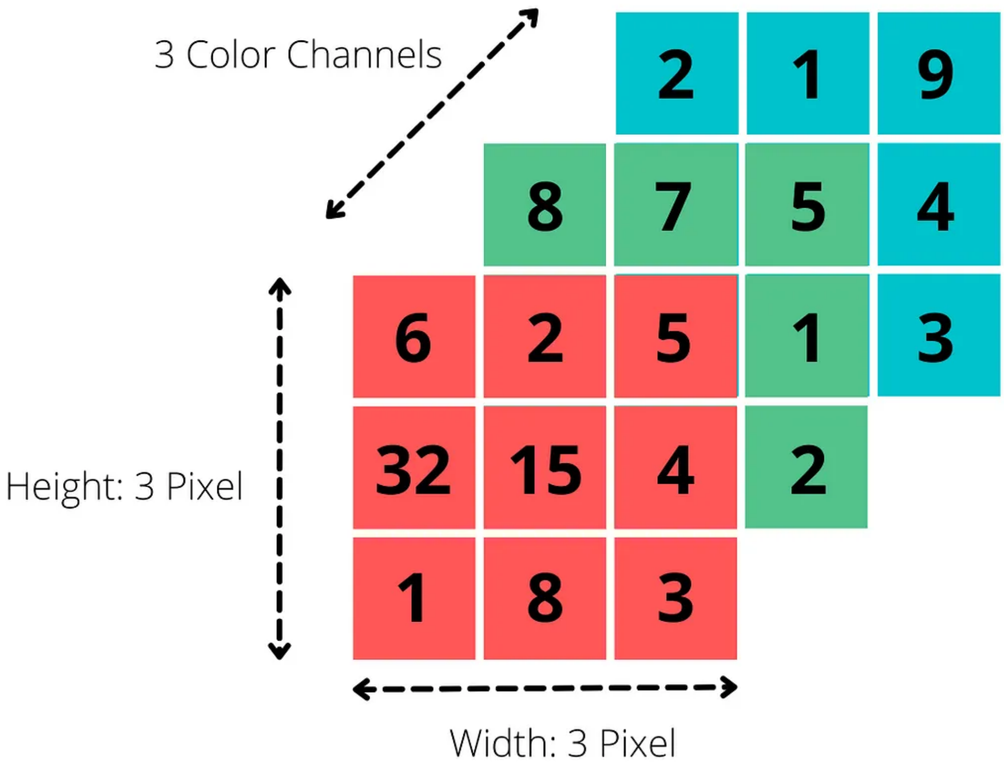
If we want to use a fully-connected neural network for image processing, we quickly discover that it does not scale very well.

Intuitive Guide to Artificial Neural Networks

Artificial Neural Networks (ANN) are the most commonly used buzzword in the context of Artificial Intelligence and…

medium.com

For the computer, an image in RGB notation is the summary of three different matrices. For each pixel of the image, it describes what color that pixel displays. We do this by defining the red component in the first matrix, the green component in the second, and then the blue component in the last. So for an image with the size 3 on 3 pixels, we get three different 3x3 matrices.



3×3×3 RGB Picture | Photo: Author

To process an image, we enter each pixel as input into the network. So for an image of size 200x200x3 (i.e. 200 pixels on 200 pixels with 3 color channels, e.g. red, green and blue) we have to provide 200 * 200 * 3= 120,000 input neurons. Then each matrix has a size of 200 by 200 pixels, so 200 * 200 entries in total. This matrix then finally exists three times, each for red, blue, and green. The problem then arises in the first hidden layer, because each of the neurons there would have 120,000 weights from the input layer. This means the number of parameters would increase very quickly as we increase the number of neurons in the Hidden Layer.

This challenge is exacerbated when we want to process larger images with more pixels and more color channels. Such a network with a huge number of parameters will most likely run into overfitting. This means that the model will give good predictions for the training set, but will not generalize well to new cases that it does not yet know. Additionally, due to a large number of parameters, the network would very likely stop attending to individual image details as they would be lost in sheer mass. However, if we want to classify an image, e.g. whether there is a dog in it or not, these details, such as the nose or the ears, can be the decisive factor for the correct result.

Convolutional Neural Network

For these reasons, the Convolutional Neural Network takes a different approach, mimicking the way we perceive our environment with our eyes. When we see an image, we automatically divide it into many small sub-images and analyze them one by one. By assembling these sub-images, we process and interpret the image. How can this principle be implemented in a Convolutional Neural Network?

The work happens in the so-called **convolution layer**. To do this, we define a filter that determines how large the partial images we are looking at should be, and a step length that decides how many pixels we continue between calculations, i.e. how close the partial images are to each other. By taking this step, we have greatly reduced the dimensionality of the image.

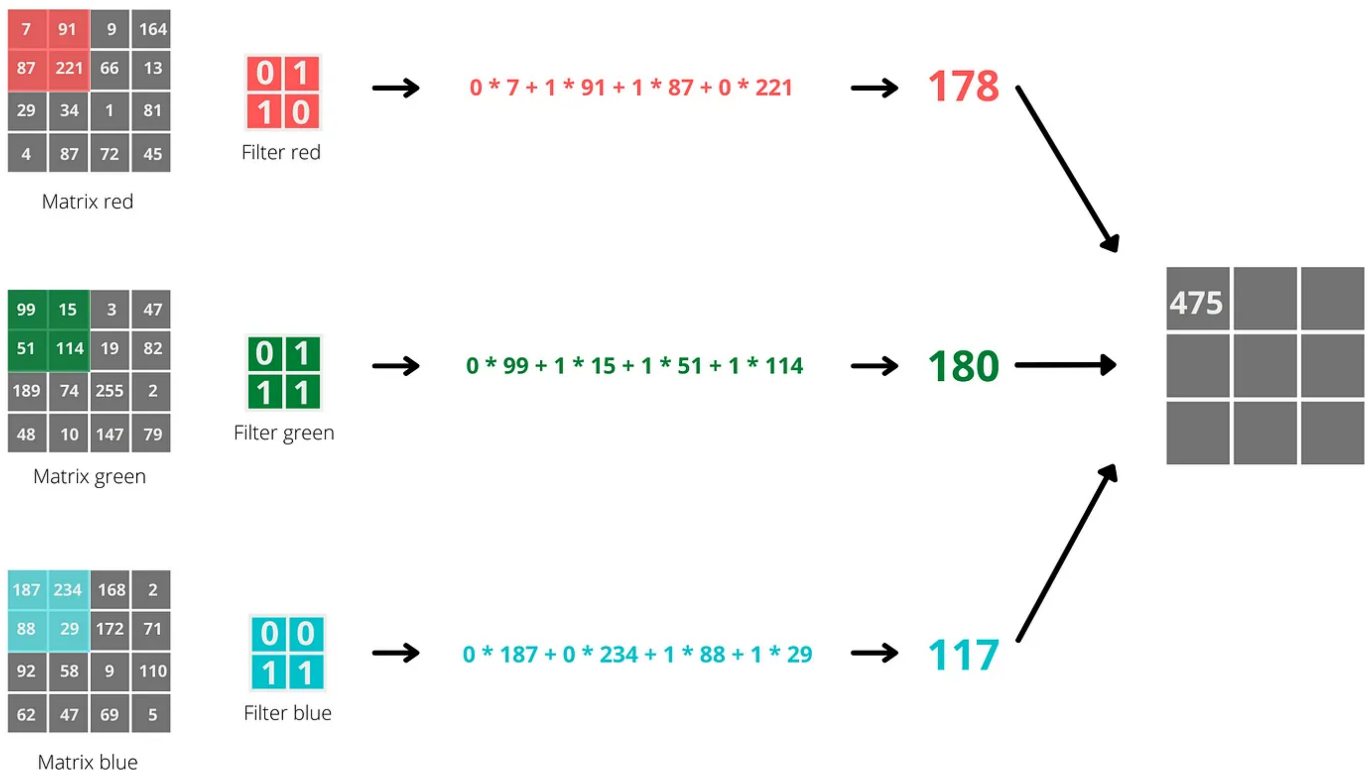
The next step is the **pooling layer**. From a purely computational point of view, the same thing happens here as in the convolution layer, with the difference that we only take either the average or maximum value from the result, depending on the application. This preserves small features in a few pixels that are crucial for the task solution.

Finally, there is a **fully-connected layer**, as we already know it from regular neural networks. Now that we have greatly reduced the dimensions of the image, we can use the tightly meshed layers. Here, the individual sub-images are linked again in order to recognize the connections and carry out the classification.

Now that we have a basic understanding of what the individual layers roughly do, we can look in detail at how an image becomes a classification. For this purpose, we try to recognize from a 4x4x3 image whether there is a dog in it.

Detail: Convolution Layer

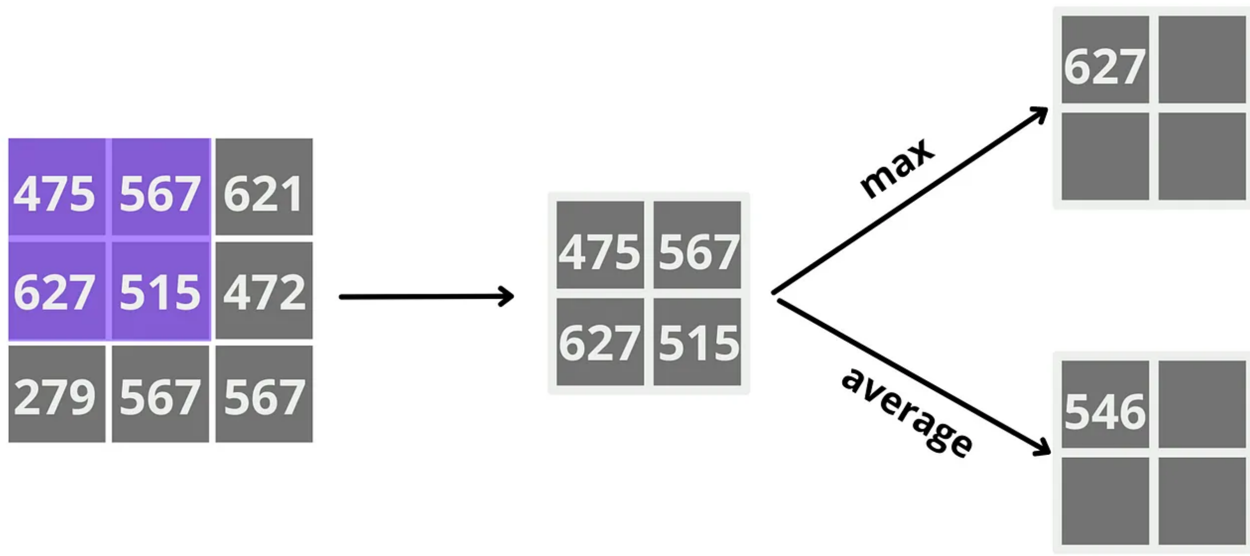
In the first step, we want to reduce the dimensions of the 4x4x3 image. For this purpose, we define a filter with the dimension 2x2 for each color. In addition, we want a step length of 1, i.e. after each calculation step, the filter should be moved forward by exactly one pixel. This will not reduce the dimension as much, but the details of the image will be preserved. If we migrate a 4x4 matrix with a 2x2 and advance one column or one row in each step, our Convolutional Layer will have a 3x3 matrix as output. The individual values of the matrix are calculated by taking the scalar product of the 2x2 matrices, as shown in the graphic.



Convolution Layer | Photo: Author

Detail: Pooling Layer

The (Max) Pooling Layer takes the 3x3 matrix of the convolution layer as input and tries to reduce the dimensionality further and additionally take the important features in the image. We want to generate a 2x2 matrix as the output of this layer, so we divide the input into all possible 2x2 partial matrices and search for the highest value in these fields. This will be the value in the field of the output matrix. If we were to use the average pooling layer instead of a max-pooling layer, we would calculate the average of the four fields instead.



Pooling Layer | Photo: Author

The pooling layer also filters out noise from the image, i.e. elements of the image that do not contribute to the classification. For example, whether the dog is standing in front of a house or in front of a forest is not important at first.

Detail: Fully-Connected Layer

The fully-connected layer now does exactly what we intended to do with the whole image at the beginning. We create a neuron for each entry in the smaller 2x2 matrix and connect them to all neurons in the next layer. This gives us significantly fewer dimensions and requires fewer resources in training.

This layer then finally learns which parts of the image are needed to make the classification dog or non-dog. If we have images that are much larger than our 5x5x3 example, it is of course also possible to set the convolution layer and pooling layer several times in a row before going into the fully-connected layer. This way you can reduce the dimensionality far enough to reduce the training effort.

Data Set

Tensorflow has a wide variety of datasets that we can download and use with just a few lines of code. This is especially helpful when you want to test new models and their implementation and therefore do not want to search for appropriate data for a long time. In addition, Google also offers a dataset search, with which one can find a suitable dataset within a few clicks.

An Introduction to TensorFlow

Get to know the Machine Learning Framework, its Architecture and the Comparison to PyTorch

towardsdatascience.com

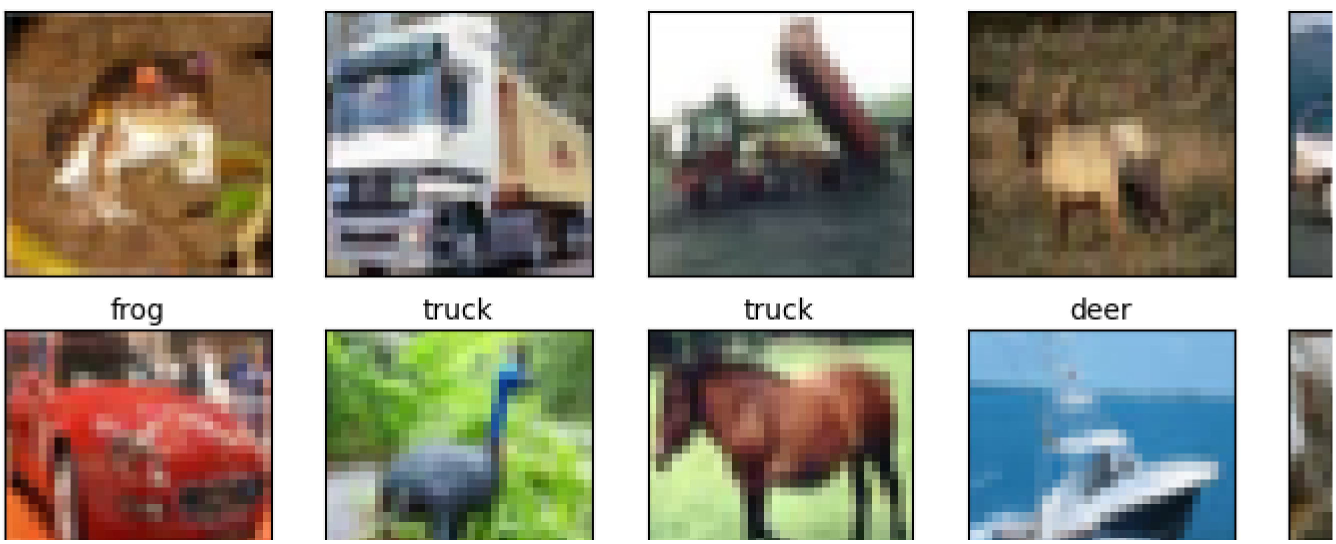
For our exemplar Convolutional Neural Network, we use the CIFAR10 dataset, which is available through Tensorflow. The dataset contains a total of 60,000 images in color, divided into ten different image classes, e.g. horse, duck, or truck. We note that this is a perfect training dataset as each class contains exactly 6,000 images. In classification models, we must always make sure that every class is included in the dataset an equal number of times, if possible. For the test dataset, we take a total of 10,000 images and thus 50,000 images for the training dataset.

Each of these images is 32×32 pixels in size. The pixels in turn have a value between 0 and 255, where each number represents a color code. Therefore, we divide each pixel value by 255 so that we normalize the pixel values to the range between 0 and 1.

```
# Library for plotting the images and the loss function
import matplotlib.pyplot as plt

# We import the data set from tensorflow and build the model there
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

# Download the data set
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
```

To check that all images are displayed correctly, we print the first ten images including the class they belong to. Since these are only 32×32 images, they are relatively blurry, but you can still tell which class they are part of.

```
# Define the 10 image classes
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

# Show the first 10 images
plt.figure(figsize=(10,10))
for i in range(10):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # Die CIFAR Labels sind Arrays, deshalb benötigen wir den extra Index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```



Build a Convolutional Neural Network

In Tensorflow we can now build the Convolutional Neural Network by defining the sequence of each layer. Since we are dealing with relatively small images we will use the stack of Convolutional Layer and Max Pooling Layer twice. The images have, as we already know, 32 height dimensions, 32 width dimensions, and 3 color channels (red, green, blue).

The Convolutional Layer uses first 32 and then 64 filters with a 3×3 kernel as a filter and the Max Pooling Layer searches for the maximum value within a 2×2 matrix.

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
```

After these two stacks, we have already reduced the dimensions of the images significantly, to 6 height pixels, 6 width pixels, and a total of 64 filters. With a third and final convolutional layer, we reduce these dimensions further to 4x4x64. Before we now build a fully meshed network from this, we replace the 3×3 matrix per image, with a vector of 1024 elements (4*4*64), without losing any information.

```
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 13, 13, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 6, 6, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (None, 4, 4, 64)          36928     
                                                                 
 flatten (Flatten)           (None, 1024)              0         
                                                                 
 dense (Dense)               (None, 64)                65600     
                                                                 
 dense_1 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

Now we have sufficiently reduced the dimensions of the images and can add one more hidden layer with a total of 64 neurons before the model ends in the output layer with the ten neurons for the ten different classes.

The model with a total of 122,570 parameters is now ready to be built and trained.

Compile and Train the Model

Before we can start training the Convolutional Neural Network, we have to compile the model. In it we define which loss function the model should be trained according to, the optimizer, i.e. according to which algorithm the parameters change, and which metric we want to be shown in order to be able to monitor the training process.

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
1563/1563 [==============================] - 113s 72ms/step - loss: 1.5383 - accuracy: 0.44
Epoch 2/10
1563/1563 [==============================] - 108s 69ms/step - loss: 1.1798 - accuracy: 0.57
Epoch 3/10
1563/1563 [==============================] - 106s 68ms/step - loss: 1.0295 - accuracy: 0.63
Epoch 4/10
1563/1563 [==============================] - 106s 68ms/step - loss: 0.9302 - accuracy: 0.67
Epoch 5/10
1563/1563 [==============================] - 106s 68ms/step - loss: 0.8551 - accuracy: 0.70
Epoch 6/10
1563/1563 [==============================] - 110s 70ms/step - loss: 0.7964 - accuracy: 0.72
Epoch 7/10
1563/1563 [==============================] - 110s 70ms/step - loss: 0.7510 - accuracy: 0.73
Epoch 8/10
1563/1563 [==============================] - 107s 69ms/step - loss: 0.7084 - accuracy: 0.75
Epoch 9/10
1563/1563 [==============================] - 105s 67ms/step - loss: 0.6737 - accuracy: 0.76
Epoch 10/10
1563/1563 [==============================] - 105s 67ms/step - loss: 0.6350 - accuracy: 0.77
```

Evaluate the Model

After training the Convolutional Neural Network for a total of 10 epochs, we can look at the progression of the model's accuracy to determine if we are satisfied with the training.

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
```

Our prediction of the image class is correct in about 80% of the cases. This is not a bad value, but not a particularly good one either. If we want to increase this even further, we could have the Convolutional Neural Network trained for more epochs or possibly configure the dense layers even differently.

This is what you should take with you

- Convolutional neural networks are used in image and speech processing and are based on the structure of the human visual cortex.

- They consist of a convolution layer, a pooling layer, and a fully connected layer.
- Convolutional neural networks divide the image into smaller areas in order to view them separately for the first time.
- Convolutional neural networks can be programmed in just a few steps using Tensorflow.
- It is important to adjust the arrangement of the convolutional and max-pooling layers to each different use case.

*If you like my work, please subscribe here or check out my website Data Basecamp! Also, medium permits you to read **3 articles** per month for free. If you wish to have **unlimited** access to my articles and thousands of great articles, don't hesitate to get a membership for $5 per month by clicking my referral link:* https://medium.com/@niklas_lang/membership

Why you should know Big Data

Defining Big Data and its potential threats

medium.com

Understanding MapReduce with the Help of Harry Potter

MapReduce is an algorithm that allows large data sets to be processed in parallel, i.e. on multiple computers...

medium.com

Learn Coding: 13 free sites to help you start

Once you've decided you want to learn coding, you'll be spoiled for choice with numerous online tools designed to help...

towardsdatascience.com

References

1. https://www.tensorflow.org/tutorials/images/cnn

Convolutional Neural Net Image Classification Cifar 10 Neural Networks

Artificial Intelligence