Image Classifier Using Convolutional Neural Networks

Submitted in partial fulfillment of the requirements of

Of

BACHELORS OF ENGINEERING

IN

Electronics and Telecommunication Engineering

SUBMITTED BY

EKAGRA GUPTA	UE215030
GUNJAN SARODE	UE215032
HARKARAN PRATAP SINGH	UE215037

Under the Supervision of

DR. ARVIND KUMAR
(Professor ECE , UIET)

UNIVERSITY INSTITUTE IF ENGINEERING AND TECHNOLOGY
PANJAB UNIVERSITY CHANDIGARH 160014

CERTIFICATE

This is to certify that this project entitled" **Image Classifier Using Convolutional Neural Networks"** being submitted by the above mentioned students to UNIVERSITY INSTITUTE OF ENGINEERING AND TECHNOLOGY ,PANJAB UNIVERSITY ,CHANDIGARH is a a bonafide project carried out under supervision.The above submitted project is found satisfactory and hereby approved for submission.

SUPERVISOR	**COORDINATOR**	**DIRECTOR**
Dr. Arvind kumar	Mr. Jaget Singh	Prof. Sanjeev Puri
Professor,(ECE)	Professor,(ECE)	
UIET,Panjab University	UIET,Panjab University	UIET,Panjab University
Chandigarh	Chandigarh	Chandigarh

DECLARATION

WE hereby certify that the work presented in this Project entitled ―Image classifier using convolutional neural networks ‖ in partial fulfillment of the requirement for the third year of engineering, submitted in UIET, Panjab University, Chandigarh. This work has been carried out under the guidance of Dr. Arvind Kumar,Professor in Panjab University, Chandigarh. We have not submitted the matter embodied in this project for the award of any other degree.

EKAGRA GUPTA	GUNJAN RAJESH SARODE	HARKARAN SINGH
Roll No.UE215030	Roll No. UE215032	Roll No.UE215037

Date: ...../...../.....

ACKNOWLEDGEMENT

This project was not a solo endeavor, but rather the culmination of several people's efforts. I express my deepest appreciation to these individuals who assisted me in several ways throughout this endeavor.

We offer our heartfelt thanks and indebtedness to our supervisor Dr. Arvind Kumar (Professor) in UIET, Panjab University Chandigarh for providing their precious supervision and invaluable help in bringing this project from concept to reality. We are very grateful for their generosity and for providing their valuable time in entire procedure. They are the source of constant motivation and enthusiasm throughout the course of this project.

We would like to thank Prof. Sanjeev Puri (Director, UIET, Panjab University, Chandigarh) for providing the opportunity to work on this project and Mr.Jaget Singh (Coordinator ECE) for providing necessary information and sufficient facilities to complete this work.

We would also like to take the opportunity to thank our subject professors for their help and motivation.

Lasr but not the least; We would like to thank our Parents .Their constant faith and support always motivates us to work hard. We can’t express our gratitude in words towards their unconditional love. At the end, We express our acknowledgement to everyone who helped us in direct or indirect ways.

Abstract

In machine learning, Convolutional Neural Networks (CNN or ConvNet) are complex feed forward neural networks.

CNNs are used for image classification and recognition because of its high accuracy. This project aims to classify the input image as either a dog or a cat image. The image input which you give to the system will be analysed and the predicted result will be given as output. Machine learning algorithm [Convolutional Neural Networks] is used to classify the image. The model thus implemented can be extended to a mobile device or any website as per the developer's need.

The dataset contains a lot of images of cats and dogs. Our aim is to make the model learn the distinguishing features between the cat and dog. Once the model has learned, i.e once the model got trained, it will be able to classify the input image as either cat or a dog.

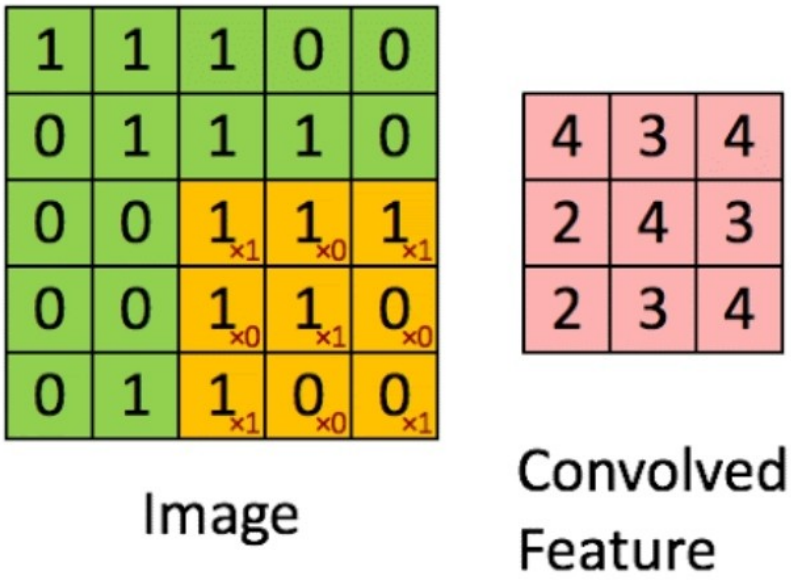
Table of Contents

1. Introduction

An Image classification refers to a process in computer vision that can classify an image according to its visual content. For example, an image classification algorithm may be designed to tell if an image contains a human figure or not. While detecting an object is trivial for humans, robust image classification is still a challenge in computer vision applications. The BoW model is a commonly used method in document classification and natural language processing. In the BoW model, the frequency of the occurrence of each word in the document is used as a parameter for training a machine learning algorithm. In addition to document classification, the BoW model can also be applied to image classification. To apply the BoW model to classify images, we need to extract a set of *words* (just like in document classification) from the image and count their occurrence. In computer vision, the words extracted from an image are commonly known as *features*. A feature generation algorithm reduces an image to a set of features that can serve as a signature for an image.

2. Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. ... This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values. Our model is a sequential model and we use 2D Convolution.



3. Edge detection and Padding

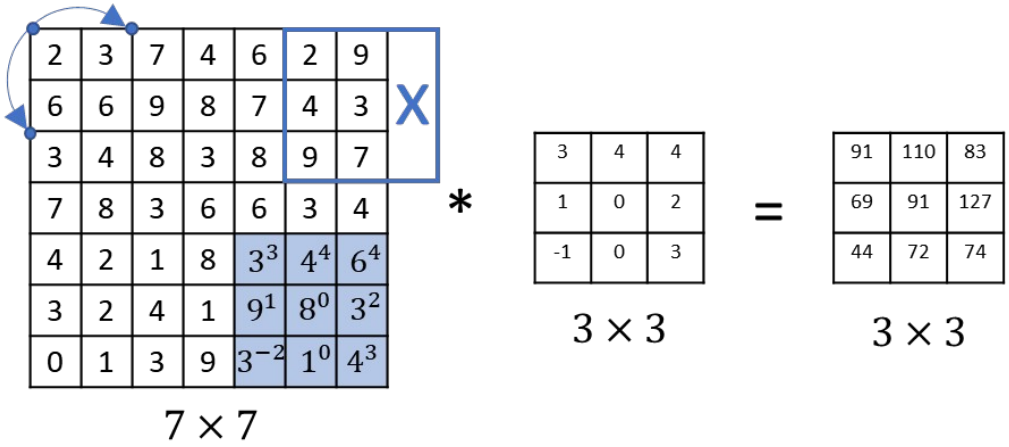
Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detection is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision. Here we use Relu activation and apply padding . For a gray scale (n x n) image and (f x f) filter/kernel, the dimensions of the image resulting from a convolution operation is (n-f+1)*(n-f+1).

For example, for an (8 x 8) image and (3 x 3) filter, the output resulting after convolution operation would be of size (6 x 6). Thus, the image shrinks every time a convolution operation is performed. This places an upper limit to the number of times such an operation could be performed before the image reduces to nothing thereby precluding us from building deeper networks. Also, the pixels on the corners and the edges are used much less than those in the middle.

After padding we get (n+2p – f + 1)*(n+2p – f +1). Where p = (f – 1) / 2 .

4. Strided Convolution and RGB Images

Let's say we want to convolve this 7×7 image with this 3×3 filter, except, that instead of doing it the usual way, we're going to do it with a stride of 2.



Convolutions with a stride of two

This means that we take the element-wise product as usual in this upper left 3×3 region, and then multiply and sum elements. That gives us 91. But then instead of stepping the blue box over by one step, we're going to step it over by two steps. It's illustrated how the upper left corner has gone from one dot to another jumping over one position. Then we do the usual element-wise product and summing, and that gives us 100. Next, we're going to do that again and make the

blue box jump over by two steps. We obtain the value 83. Then, when we go to the next row, again we take two steps instead of one step. We will move filter by 2 steps and we'll obtain 69 .

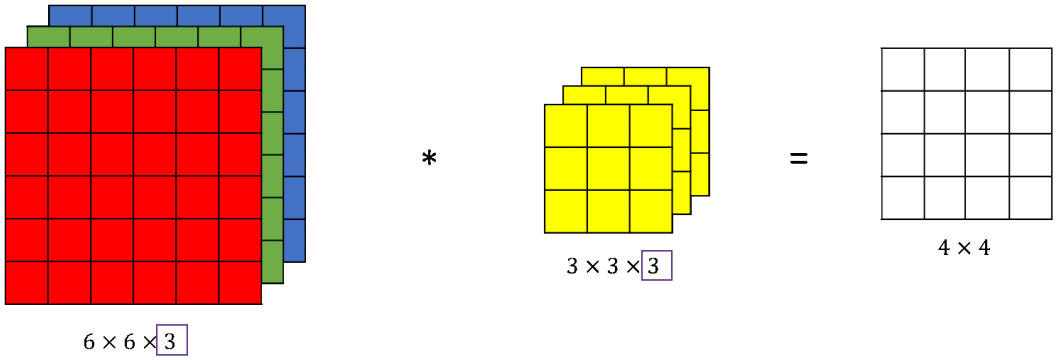
In this example we convolve 7×7 matrix with a 3×3 matrix and we get a 3×3 output. The input and output dimensions turns out to be governed by the following formula:

$$\lfloor n-f+2p/s \rfloor+1 \times \lfloor n-f+2p/s \rfloor+1$$

If we have $n\times n$ image convolved with an $f\times f$ filter and if we use a padding p and a stride s, in this example $s=2$, then we end up with an output that is $n-f+2p$. Because we're stepping s steps at the time instead of just one step at a time, we now divide by s and add 1.

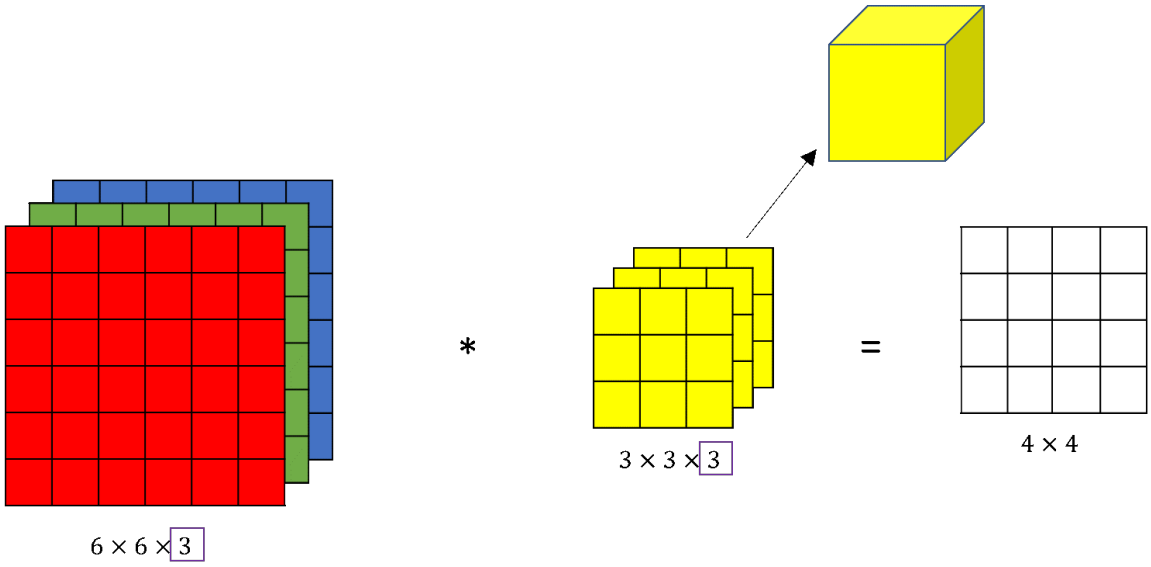
In our example, we have $(7+0-3)/2+1=4/2+1=3$, that is why we end up with this 3×3 output. Notice that in this formula above, we round the value of this fraction, which generally might not be an integer value, down to the nearest integer.

Convolutions on RGB image



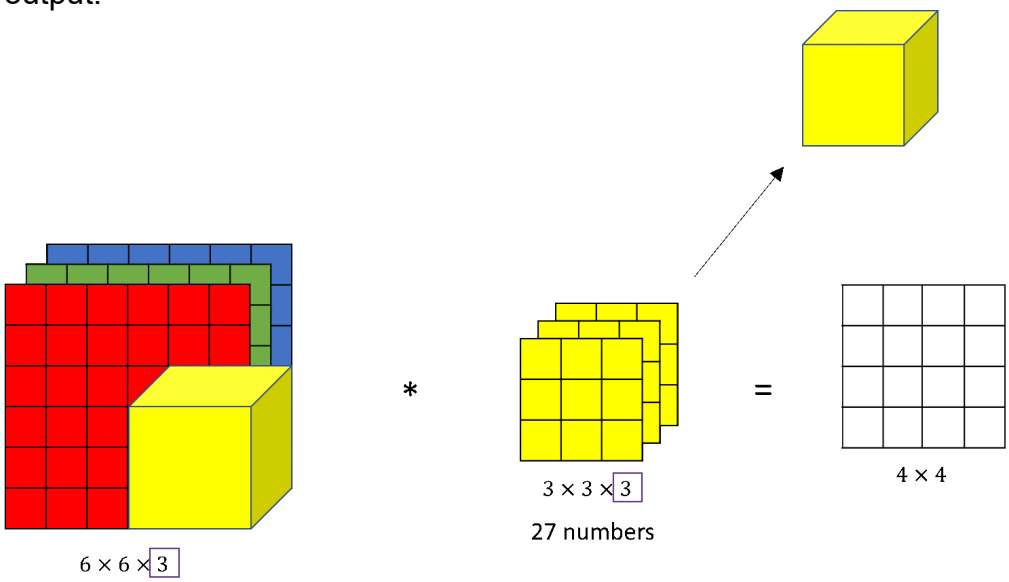
RGB image, corresponding filter for convolution and the result of a convolution

Here we can see the $6\times6\times3$ image and the $3\times3\times3$ filter. The last number is the number of channels and it matches between the image and the filter. To simplify the drawing the $3\times3\times3$ filter, we can draw it as a stack of three matrices. Sometimes, the filter is drawn as a three-dimensional cube as we can see in the image below.



The filter we use we can consider as a volume

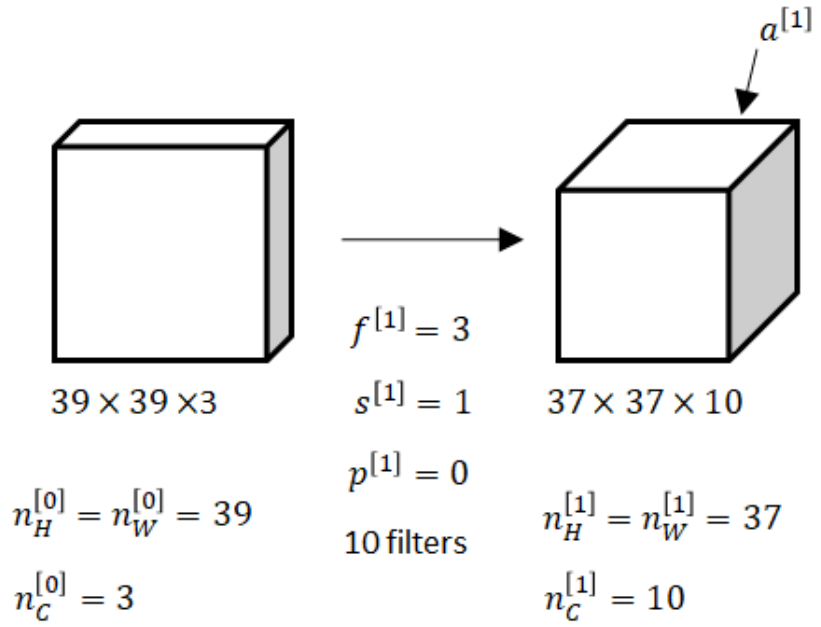
To compute the output of this convolution operation, we take the $3\times3\times3$ filter and first place it in that most upper left position. Notice that $3\times3\times3$ filter has 27 numbers. We take each of these 27 numbers and multiply them with the corresponding numbers from the red, green and blue channel. So, take the first nine numbers from red channel, then the three beneath it for the green channel, then three beneath it from the blue channel and multiply them with the corresponding 27 numbers covered by this yellow cube. Then, we add up all those numbers and this gives us the first number in the output. To compute the next output we take this cube and slide it over by one. Again we do the twenty-seven multiplications sum up 27 numbers and that gives us the next output.



When we apply $3\times3\times3$ filter on the RGB image it is as we implement the volume

5 . Convolutional networks using neural networks

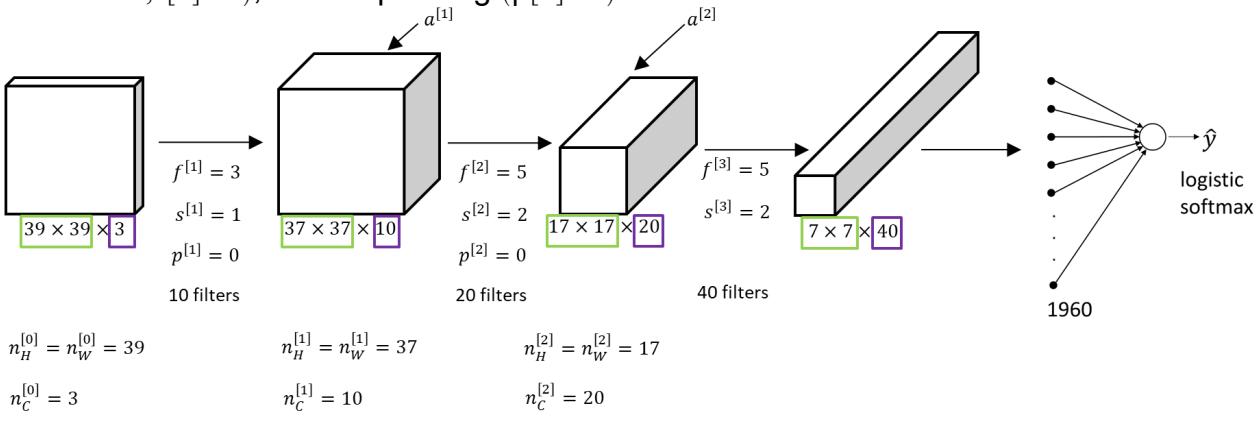
Let's say we have an image and we want to do an image classification or image recognition. We want to take an image X as input and decide if this is a cat or not a cat. Let's build an example of a ConvNet we could use. For this example, we're going to use a fairly small image. Let's say this image is $39 \times 39 \times 3$. So, $n[0]H = n[0]W = 39$, and $n[0]C = 3$ (*H* – height, *W* – weight, *C* – channel).
Next, the first layer uses a set of 3×3 filters to detect features, so $f[1] = 3$ because we're using 3×3 filters. Let's say we're using a stride of 1. We are also using a valid convolution. Let's say we have 10 filters, and they will produce 10 channels in the output volume.



One layer of convolution using 3×3 with a stride of 1 and no padding

The activations in this next layer of the neural network will be $37\times37\times10$. This 10 comes from the fact that we use 10 filters and 37 comes from this formula $(n+2p-f)/s+1$. Then, we have a $(39+0-3)/1+1=37$. That's why the output is 37×37, it's a valid convolution and that's the output size. In our notation we would have $n[1]H=n[1]W=37$, and $n[1]C=10$, is also equal to the number of filters from the first layer. So, this becomes the dimension of the activation at the first layer.

Let's now say we have another convolutional layer, and we use 5×5 filters. So, in our notation $f[2]$ at the next layer of network is equal to $5(f[2]=5)$, and let's say we use the stride of $2, s[2]=2)$, and no padding $(p[2]=0)$ and 20 filters.



An example of a convolution

Then the output of this will be another volume, this time, it will be $17\times17\times20$. Notice that because we're now using a stride of 2, so $(s[2]=2)$ the dimension has shrunk much faster and 37×37 has gone down in size by slightly more than 2, to 17×17.

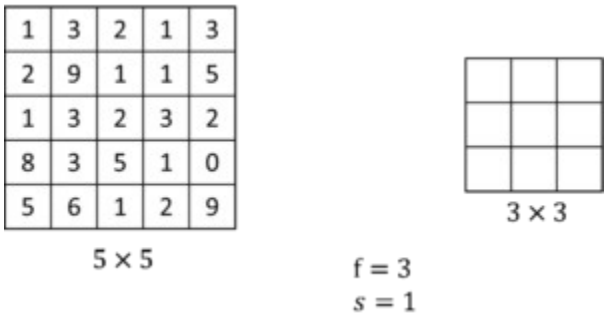
Because we're using 20 filters, the number of channels is now 20, so this activation $a[2]$ would be that dimension. And $n[2]H=n[2]W=17$ and $n[2]C=20$.
Let's apply one last convolutional layer. Let's say that we use a 5×5 filter again and, again a stride of 2. Eventually we end up with a 7×7. Finally, if we use 40 filters and no padding we end up with $7\times7\times40$.
Now our $39\times39\times3$ input image is processed and $7\times7\times40$ features are computed for this image. Finally, if we take the $7\times7\times40(=1960)$, and we flatten this volume or unroll it into 1960 units. By unrolling them into a very long vector we can feed into a softmax or into a logistic regression in order to make a prediction for the final output. This would be a typical design of a ConvNet. All of the work and designing a convolutional neural net is selecting its hyperparameters: deciding what's the filter size, what's the stride, what's the padding and how many filters to use. We will give some suggestions later and some guidelines for how to make these choices. At the moment, one thing to take away from this is that as we go deeper in the neural network, typically we start off with larger images 39×39, and then the height and width will stay the same for a while and gradually trend down as we go deeper in the neural network. That is, the size has gone from 39 to 37 to 17 to 7 , whereas the number of channels generally increases (from 3 to 10 to 20 to 40).
And it turns out that in a typical ConvNet there are usually three types of layers: one is the convolutional layer and often we'll denote that as a ConvNet. It turns out that there are two other common types of layers that you haven't seen yet, but we'll talk about them in our next posts. One is called a Pooling layer, from now on called Pool, and then the last is a Fullyconnectedlayer, called FC. Although it's possible to design a pretty good neural network using just convolutional layers, most neural network architectures will also have a few pooling layers and a few fully connected layers. Fortunately, pooling layers and fully connected layers are a bit simpler than convolutional layers to define.

5. Pooling and VGG16

There are two main reasons that people use Maxpooling:

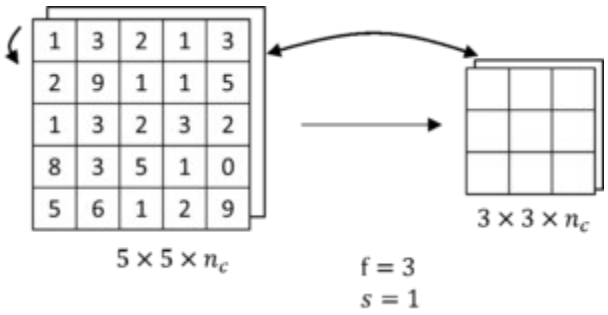
1. It's been found in a lot of experiments to work well.
2. It has no parameters to learn. There's actually nothing for the gradient descent to learn. Once we've fixed f and *s,* it's just a fixed computation and gradient descent doesn't change anything.

Here, we're going to use a 5×5 input and we're going to apply max pooling with a filter size 3×3. So, $f=3$ and let's use a stride of 1 ($s=1$). In this case the output size is going to be 3×3, and the formulas we have developed previously for all the conv layer outputs can be applied for Maxpooling as well.
The formula for calculating a dimension of the output of conv layer is: $n+2p-fs+1$ and it also works for calculating the output size of Maxpooling .
Here, we can compute every element of the 3×3 output. Note that the filter size is $f=3$ and that the stride is $s=1$. With this set of parameters we have obtained the following output (this 3×3):



Maxpooling *with the* 3×3 *filter and with a stride of 1*

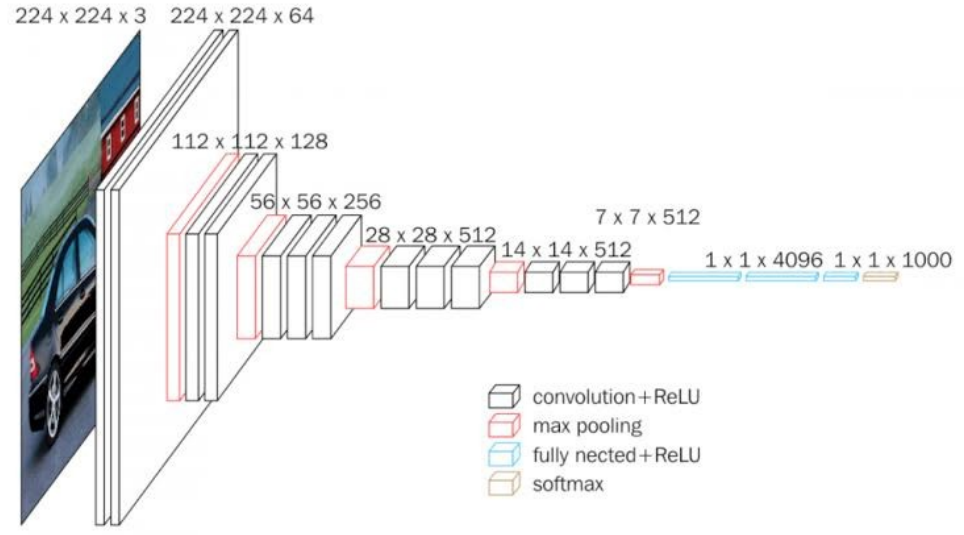
So far, we've seen Maxpooling on a 2D input. In case of a 3D input the output will have the same dimension as we can see in the picture below. For example, if we have $5\times5\times2$ then the output would be $3\times3\times2$. The Max pooling calculation is performed on each channel independently. More generally, if we have $5\times5\times n_c$, the output would be $3\times3\times n_c$. The Max pooling computation is done independently on each of these n_c channels.

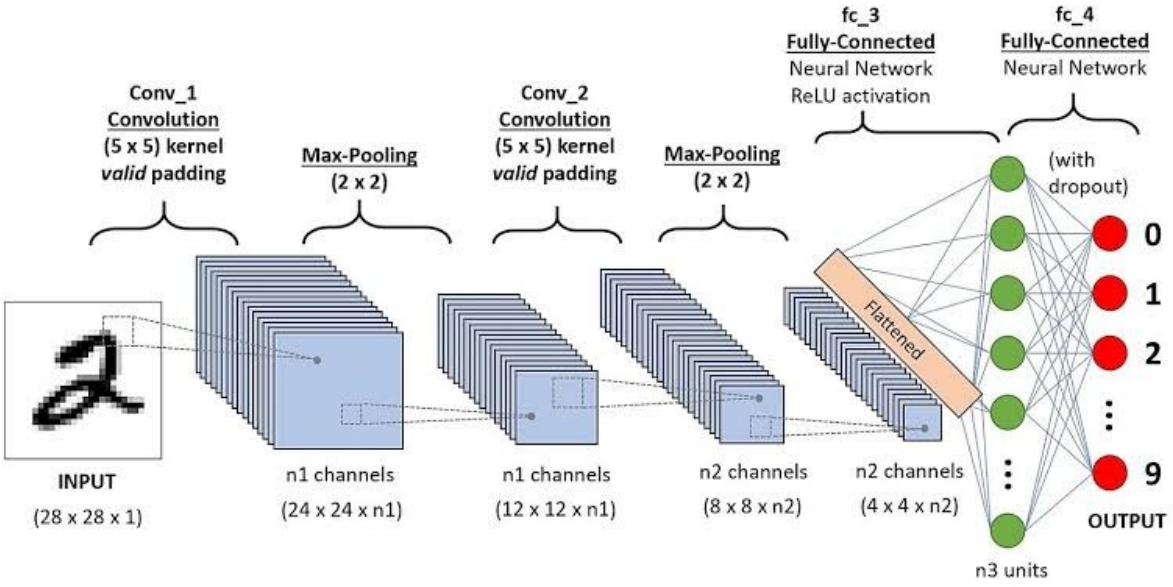


Result of the **Maxpooling** applied on a 3D volume

VGG16

VGG is an innovative object-recognition model that supports up to 19 layers. Built as a deep CNN, VGG also outperforms baselines on many tasks and datasets outside of ImageNet. VGG is now still one of the most used image-recognition architectures. Input. VGG takes in a 224x224 pixel RGB image. The convolutional layers in VGG use a very small receptive field (3x3, the smallest possible size that still captures left/right and up/down). There are also 1x1 convolution filters which act as a linear transformation of the input, which is followed by a ReLU unit. The convolution stride is fixed to 1 pixel so that the spatial resolution is preserved after convolution.





Applications

This project gives a general idea of how image classification can be done efficiently. The scope of the project can be extended to the various industries where there is a huge scope for automation, by just altering the dataset which is relevant to the problem.

6. Example Code and Output

Software used :- Python 3

```
import cv2
import os
import numpy as npimport cv2
import os
import numpy as np



'''Setting up the env'''
TRAIN_DIR = 'E:/dataset/PlantDisease/train'
TEST_DIR = 'E:/dataset/PlantDisease/test'
IMG_SIZE = 50
LR = 1e-3
'''Setting up the model which will help with TensorFlow models'''
MODEL_NAME = 'plantdisease-{}-{}.model'.format(LR, '6conv-basic')
```

```
'''Labelling the dataset'''
def label_img(img):
    word_label = img.split('.')[-3]
    # DIY One hot encoder
                                                    if word_label == 'healthy': return [1, 0]
    elif word_label == 'infected': return [0, 1]
'''Creating the training data'''
def create_train_data():
    # Creating an empty list where we should store the training data
                                                    # after a little preprocessing of the data
    training_data = []
    # tqdm is only used for interactive loading
    # loading the training data
    for img in tqdm(os.listdir(TRAIN_DIR)):
        # labeling the images
        label = label_img(img)
        path = os.path.join(TRAIN_DIR, img)
        # loading the image from the path and then converting them into
        # greyscale for easier covnet prob
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        # resizing the image for processing them in the covnet
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        # final step - forming the training data list with a numpy array of the images
        training_data.append([np.array(img), np.array(label)])
    # shuffling of the training data to preserve the random state of our data
    shuffle(training_data)
    # saving our trained data for further uses if required (not a necessary step)
                                                    np.save('train_data.npy', training_data)
    return training_data
'''Processing the given test data'''
# Almost the same as processing the training data but
# we don't have to label it.
def process_test_data():
    testing_data = []
    for img in tqdm(os.listdir(TEST_DIR)):
        path = os.path.join(TEST_DIR, img)
        img_num = img.split('.')[0]
        img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
        testing_data.append([np.array(img), img_num])
    shuffle(testing_data)
                                                    np.save('test_data.npy', testing_data)
    return testing_data
```

```
'''Running the training and the testing on the train and test dataset for our model'''
# train_data = create_train_data()
# test_data = process_test_data()
train_data = np.load('train_data.npy')
test_data = np.load('test_data.npy')
'''Creating the neural network using TensorFlow'''
# Importing the required libraries
import tflearn
                                                from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
                                                                        import tensorflow as tf
tf.reset_default_graph()
convnet = input_data(shape=[None, IMG_SIZE, IMG_SIZE, 1], name='input')
convnet = conv_2d(convnet, 32, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 128, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 64, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = conv_2d(convnet, 32, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)
convnet = fully_connected(convnet, 1024, activation='relu')
convnet = dropout(convnet, 0.8)
convnet = fully_connected(convnet, 2, activation='softmax')
                                    convnet = regression(convnet, optimizer='adam', learning_rate=LR,
                                                            loss='categorical_crossentropy', name='targets')
model = tflearn.DNN(convnet, tensorboard_dir='log')
# Loading the saved model
 if os.path.exists('C:/Users/knapseck/Desktop/Dev/Cov_Net{}.meta'.format(MODEL_NAME)):
   model.load(MODEL_NAME)
   print('model loaded!')
                                                        # Splitting the testing data and training data
train = train_data[:-500]
test = train_data[-500:]
'''Setting up the features and labels'''
# X-Features & Y-Labels
X = np.array([i[0] for i in train]).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
                                                                        Y = [i[1] for i in train]
test_x = np.array([i[0] for i in test]).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
test_y = [i[1] for i in test]
```

```
'''Fitting the data into our model'''
# epoch = 5 taken
      model.fit({'input': X}, {'targets': Y}, n_epoch=5, validation_set=({'input': test_x}, {'targets':
                                                                                                  test_y}),
          snapshot_step=500, show_metric=True, run_id=MODEL_NAME)
model.save(MODEL_NAME)
'''Testing the data'''
import matplotlib.pyplot as plt
                                                                        # if you need to create the data:
# test_data = process_test_data()
                                                                    # if you already have some saved:
test_data = np.load('test_data.npy')
fig = plt.figure()
for num, data in enumerate(test_data[:20]):
    # healthy: [1, 0]
    # infected: [0, 1]
    img_num = data[1]
    img_data = data[0]
    y = fig.add_subplot(4, 5, num + 1)
    orig = img_data
    data = img_data.reshape(IMG_SIZE, IMG_SIZE, 1)
    model_out = model.predict([data])[0]
    if np.argmax(model_out) == 1:
        str_label = 'Infected'
    else:
        str_label = 'Healthy'
```

7. Conclusion

We started with an input image and applied multiple different features to create a feature map. We applied the ReLU to increase non-linearity and applied a pooling layer to each feature map. (We did that to make sure we have spatial variance in our images, to reduce the size of the images, and to avoid overfitting of the model to the data while still preserving the features we're after.) We flattened all the pooled images into one long vector. We input the vector into our fully connected artificial neural network. This is where all the features were processed through the network. This gave us the final fully connected layer which provided the "voting" of the classes that we're after. All of this was trained through forward propagation and backpropagation until we wound up with a well defined neural network where the weights and feature detectors were trained.

References

[1] https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb
[2] http://datahacker.rs/architectures-convolutional-neural-networks
[3] https://www.robosync.in
[4] https://github.com/Subhajit135/Image-Classifier-ConvNet-in-Cats-Dog-dataset?files=1
[5] https://www.geeksforgeeks.org/project-idea-cat-vs-dog-image-classifier-using-cnn-implemented-using-keras/amp/?ref=rp