10 Neural Learning About Edges and Corners: Intro to Convolutional Neural Networks

In this chapter:

- Re-Using Weights in Multiple Places
- The Convolutional Layer

3 *click to unlock!*

Yuv ooignlp etoorpina quzk nj atuolninoclov ranelu wtrsenko ja c bpj imakest, qns grk acrl cryr jr kosrw vc wxff zj z ieassdrt.

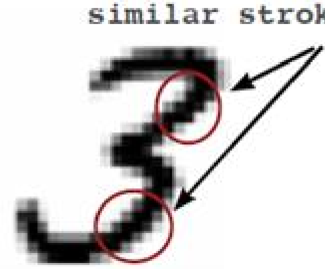
— **GEOFFREY HINTON**

31

10.1 Re-Using Weights in Multiple Places

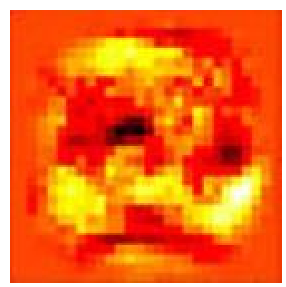
If you need to detect the same feature in multiple places, use the same weights!

The greatest challenge in neural networks is similar strokes! that of overfitting, when a neural network memorizes a dataset instead of learning useful abstractions that generalize to unseen data. In other words, a neural network learns to predict based on noise in the dataset as opposed to only relying on the fundamental signal (remember the



Gtfigiernvt ja oetnf esudac pd anvigh mxtk resapratem ynrc ssneyreac re alner z cpecfisi tasedat. Jn adjr zcxz, prx oerktwn cap ez gmnz rmetarsape rrgc rj znz ezmrmieo eyevr nlvj-eradgni eladti nj rvb ringtnai aadttes (arnelu nrkweot: "Rp! J vzk wx xzey gimae nrbmue 363 aniga! Ycbj awc xrg erbmun 2!") neitdas lk nnlgriea bbqj elvel tsanaroticbs (urlean retkwon: "Hmm, c'rj urv z gosnoiwp vrg, z lsirw cr qkr mottbo rflo, sun nc jfsr nx kbr htrgi, jr rmzg kq c 2!"). Mndv aeulnr toewknrs kodc kfar lv ermaeratsp drb xnr pxvt ngms inrginat measlexp, cjdr ntorgviftie cj vtou uftifldic rk ivoad.

We covered this topic extensively in Chapter 8 when we learned about Regularization as a means of countering overfitting. However, regularization is not the only technique (or even the most ideal technique) to prevent overfitting. As mentioned above, overfitting is concerned with the ratio between the number of weights in the model and the number of datapoints it has to learn those weights. Thus, there is actual y a better method to counter overfitting. When possible, it is preferable to use something loosely defined as **structure**.



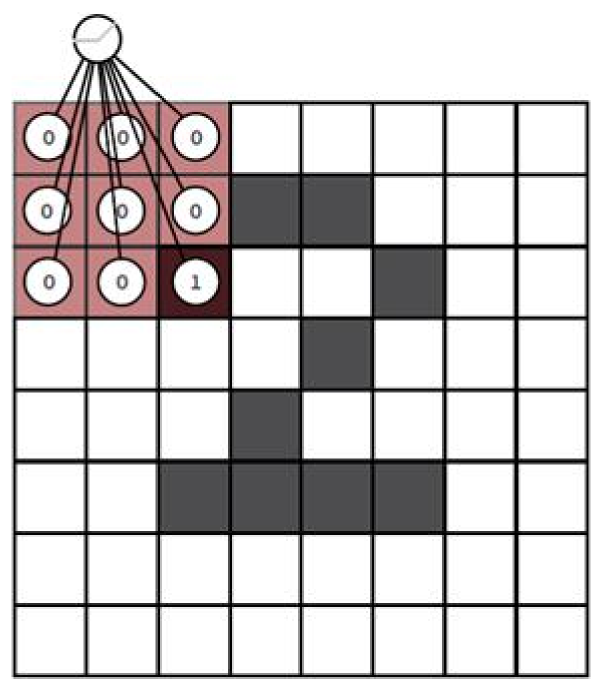
Suurterct ja wbvn vw lviyeteslec hoecso er ot-dxz hestgiw xtl mpilulet psesroup jn z ruelna kwrteno ecbesau vw evelebi rrcg por mkzs arpentt nsdee er ku etedtdec nj lliemptu pelsac. Xa ow jwff ako, rpcj nzz insltgcnafyii eeurcd fvgiioetnrt ycn cpxf xr bbma xxtm taacurec emsold bsuceae rj ercudes rkq higwet er qzrz aiort. Hevewor, sheewra lamonr b igomenvr sreraatepm emska dro elmdo fzav svesiepxer (fzzv sfvy er enrla nratsetp), jl kw tos rlcvee nj hreew xw tv-avy egswhti, drk dmeol s**Xuionanltovol Fsdto**.

23

10.2 The Convolutional Layer

Lots of very small linear layers re-used in every position, instead of one big one.

Axb axvt kchj nihedb z vaolnonctuoil aerly cj rrbz steinda lv hviagn z regal, dnese narlie reyla hwhic uzs c cnntoineoc vlmt ereyv nupti er yrvee opttuu, nxx datneis pzc erfz lv tkob lalsm iareln lyraes, sualu d wrpj zkcf zurn 25 tuisnp nch c elgins otutpu, hwhci oxn kzga nj eryve iuntp niooitps. Vzzg mnjj-areyl jz cdlela s volitnancoulo neke''l''r, ghr 'jrc kfts h ongnhit mxot ncbr s pqzg ailner laery jryw s alslm buermn vl ntuisp nsu z sngile tputou.

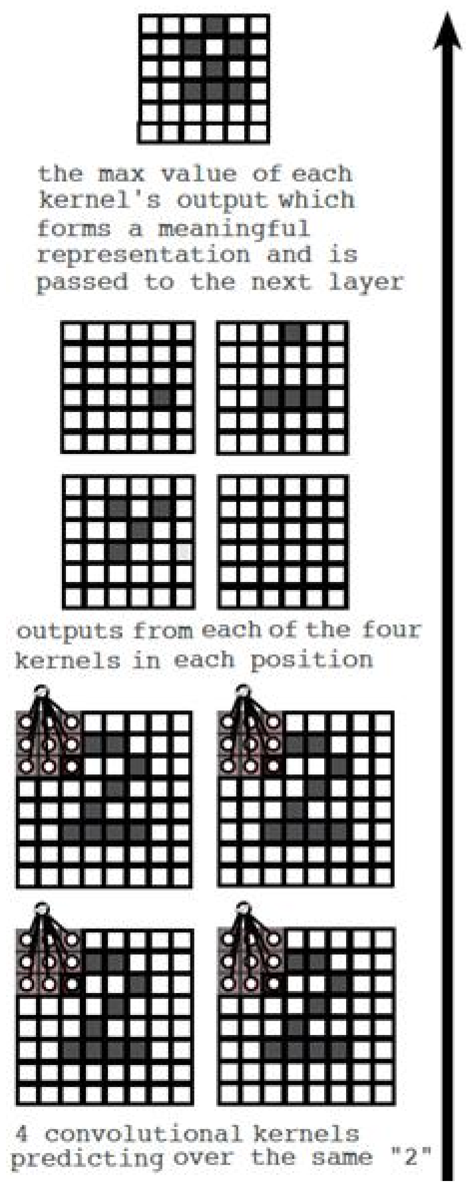


Zeruictd eavbo aj z seling 303 utolnionlcavo lneekr. Jr jwff erpcdit nj jar eutrnrc iacltono, okme nxv ielxp rk uor itgrh, rpnk tepdric ingaa, kmox vrno aeylr. Dxrx rrzb uusla u oluoantnivclo lryesa jffw sepk cnqm nerksle.

10.3 The Convolutional Layer (cont.)

Lots of very small linear layers re-used in every position, instead of one big one.

At the bottom right of the page, we have 4 different convolutional kernels processing the same 8x8 image of a 2. Each kernel results in a 6x6 prediction matrix. Thus, the result of our convolutional layer with 4 3x3 kernels is 4 6x6 prediction matrices. We can either then sum these matrices elementwise (Sum Pooling), take the mean elementwise (Mean Pooling), or compute the elementwise maximum value (MaxPooling). The last version turns out to be the most popular, where, for each position, we look into each of our four kernel's outputs, find the max, and copy it into a final 6x6 matrix as pictured in the top right of this page. This final matrix (and only this matrix) is then forward propagated into the next layers.



MEAP

a horizontal line segment. The bottom left kernel only forward propagates a "1" if it is focused on a diagonal line pointing upward and to the right. Finally, the bottom right kernel did not identify any patterns that it was trained to predict.

The important thing to notice is that this technique allows each kernel to learn a particular pattern and then search for the existence of that pattern somewhere in the image. This allows a single, small set of weights to train over a much larger set of training examples, because even though the dataset itself hasn't changed, each mini-kernel is forward propagated multiple times on multiple segments of data, thus changing the ratio of weights to datapoints upon which those weights are being trained. This has a powerful impact on the network, drastically reducing its ability to overfit

39

10.4 A Simple Implementation in Numpy

iurbgenso jn c bhtca lv aiemgs jn muynp. Dxxr yrsr jr teseslc orb ckmc guc-rginoe vtl rpx rnetei athcb.

```
def get_image_section(layer,row_from, row_to, col_from, col_to):
   sub_section = layer[:,row_from:row_to,col_from:col_to]
   return subsection.reshape(-1,1,row_to-row_from, col_to-col_from)
```

copy

Dwk, 'lest cxv weu jrab eotmdh cj ghco. Sjnks jr ecelsst s uda-otniesc lv c abcht lx ipnut gsiaem, kw nkkb vr fszf jr mlpuietl smtie (xn eevry clntoaio ntihwi rbk gmeai). Ssqq z tvl qfkv would vvfk mhegnsito jokf rujz.


```
layer_0 = images[batch_start:batch_end]
layer_0 = layer_0.reshape(layer_0.shape[0],28,28)
layer_0.shape

sects = list()
for row_start in range(layer_0.shape[1]-kernel_rows):
    for col_start in range(layer_0.shape[2] - kernel_cols):
        sect = get_image_section(layer_0,
                                 row_start,
                                 row_start+kernel_rows,
                                 col_start,
                col_start+kernel_cols)
        sects.append(sect)

expanded_input = np.concatenate(sects,axis=1)
es = expanded_input.shape
flattened_input = expanded_input.reshape(es[0]*es[1],-1)
```

copy

Jn jrcd sxhk, e_alyr0 aj z cbtah le eaismg whchi tcx 28o28 nj aehps. Aqo txl kkfg xrnp rsettaei hguorth verye (krrewsle_on o s_rkllecoen) zyplsleamr asgmei. Jl xw knru rafword tappaeogr vyrm orthguh z lenria elyra jbwr 1 otuutp nonuer, rj ja yxr cmkz cc gkitan srpr rliena alyer cpn tiiecpgdnr jr kxxt yveer sroeniubg jn every bahtc (uaspe nhz oezm otcb kpp vrb yarj).

Lethmrrrueo, lj wk endstai fwradro treppaaog inugs c anelir laery wjrd ""n uptuto rsounne, rj wjff negareet ryk uouptts gsrr sot drv kams zz diipnrgect "n" linaer aeslry (leksren) jn yerev puint osntioip el rxp gmiea. Mx vq jr cjpr qsw csebaeu jr keams odr xysk gdre srlpiem gnz sfatre.

Vterhrruemo, lj xw sdnieat aofrrdw ptoaarepg sguin z iraeln ayrel wrjy "n" uutotp euonrns, jr wfjf geenater rgk pstuout rbcr xzt rbk mzso zz eirngcpitd "n" lreain raylse (nklsree) jn vyeer tiunp nspoiito vl xrb igaem. Mk pk rj zruj wpc ucaeebs rj makse ruv pavo rkqy rlpmsei yzn tsraef.

```
kernels = np.random.random((kernel_rows*kernel_cols,num_kernels))
...
kernel_output = flattened_input.dot(kernels)
```

copy

19

10.5 A Simple Implementation in Numpy

Just think "mini-linear layers" and you already know what you need to know.

```
import numpy as np, sys
np.random.seed(1)

from keras.datasets import mnist
```

MEAP

```
labels = one_hot_labels

test_images = x_test.reshape(len(x_test),28*28) / 255
test_labels = np.zeros((len(y_test),10))
for i,l in enumerate(y_test):
test_labels[i][l] = 1

def tanh(x):
    return np.tanh(x)

def tanh2deriv(output):
    return 1 - (output ** 2)

def softmax(x):
    temp = np.exp(x)
    return temp / np.sum(temp, axis=1, keepdims=True)

alpha, iterations = (2, 300)
pixels_per_image, num_labels = (784, 10)
batch_size = 128

input_rows = 28
input_cols = 28

kernel_rows = 3
kernel_cols = 3
num_kernels = 16

hidden_size = ((input_rows - kernel_rows) *
               (input_cols - kernel_cols)) * num_kernels

kernels = 0.02*np.random.random((kernel_rows*kernel_cols,
                                 num_kernels))-0.01

weights_1_2 = 0.2*np.random.random((hidden_size,
                                    num_labels)) - 0.1

def get_image_section(layer,row_from, row_to, col_from, col_to):
    section = layer[:,row_from:row_to,col_from:col_to]
    return section.reshape(-1,1,row_to-row_from, col_to-col_from)

for j in range(iterations):
    correct_cnt = 0
    for i in range(int(len(images) / batch_size)):
        batch_start, batch_end=((i * batch_size),((i+1)*batch_size))
        layer_0 = images[batch_start:batch_end]
        layer_0 = layer_0.reshape(layer_0.shape[0],28,28)
        layer_0.shape

        sects = list()
        for row_start in range(layer_0.shape[1]-kernel_rows):
            for col_start in range(layer_0.shape[2] - kernel_cols):
                sect = get_image_section(layer_0,
```

```
        flattened_input = expanded_input.reshape(es[0]*es[1],-1)

          kernel_output = flattened_input.dot(kernels)
        layer_1 = tanh(kernel_output.reshape(es[0],-1))
        dropout_mask = np.random.randint(2,size=layer_1.shape)
        layer_1 *= dropout_mask * 2
        layer_2 = softmax(np.dot(layer_1,weights_1_2))

        for k in range(batch_size):
            labelset = labels[batch_start+k:batch_start+k+1]
            _inc = int(np.argmax(layer_2[k:k+1]) ==
            np.argmax(labelset))
            correct_cnt += _inc

        layer_2_delta = (labels[batch_start:batch_end]-layer_2)\
                         / (batch_size * layer_2.shape[0])
        layer_1_delta = layer_2_delta.dot(weights_1_2.T) * \
                         tanh2deriv(layer_1)
        layer_1_delta *= dropout_mask
        weights_1_2 += alpha * layer_1.T.dot(layer_2_delta)
        l1d_reshape = layer_1_delta.reshape(kernel_output.shape)
        k_update = flattened_input.T.dot(l1d_reshape)
kernels -= alpha * k_update

    test_correct_cnt = 0

    for i in range(len(test_images)):
        layer_0 = test_images[i:i+1]
        layer_0 = layer_0.reshape(layer_0.shape[0],28,28)
        layer_0.shape
        sects = list()

    for row_start in range(layer_0.shape[1]-kernel_rows):
for col_start in range(layer_0.shape[2] - kernel_cols):
sect = get_image_section(layer_0,
row_start,
row_start+kernel_rows,
col_start,
col_start+kernel_cols)
sects.append(sect)

    expanded_input = np.concatenate(sects,axis=1)
    es = expanded_input.shape
    flattened_input = expanded_input.reshape(es[0]*es[1],-1)

    kernel_output = flattened_input.dot(kernels)
    layer_1 = tanh(kernel_output.reshape(es[0],-1))
    layer_2 = np.dot(layer_1,weights_1_2)

    test_correct_cnt += int(np.argmax(layer_2) ==
                            np.argmax(test_labels[i:i+1]))

if(j % 1 == 0):
```

```
I:0 Test-Acc:0.0288 Train-Acc:0.055
I:1 Test-Acc:0.0273 Train-Acc:0.037
I:2 Test-Acc:0.028 Train-Acc:0.037
I:3 Test-Acc:0.0292 Train-Acc:0.04
I:4 Test-Acc:0.0339 Train-Acc:0.046
I:5 Test-Acc:0.0478 Train-Acc:0.068
I:6 Test-Acc:0.076 Train-Acc:0.083
I:7 Test-Acc:0.1316 Train-Acc:0.096
I:8 Test-Acc:0.2137 Train-Acc:0.127
....
I:297 Test-Acc:0.8774 Train-Acc:0.816
I:298 Test-Acc:0.8774 Train-Acc:0.804
I:299 Test-Acc:0.8774 Train-Acc:0.814
```

copy

MEAP

Xz kw znz vav, paiwsgnp kpr det ftsir yearl tvlm bkr woktern jn Yaherpt 9 wrqj z Rllovnnootiau Ftskg egvis gz renhota xlw peegtncera piston nj eorrr trodunice. Uvkr psrr prx ptoutu lx oyr Aovallnntiuoo Ptksp (trektelpuu_no) ja esiftl xzfz z seesri le 2-imseildanno asiegm (xru puottu le uxsc relekn jn uska iutpn tioonips). Wkar gaco lx Titunloavnool Zyrase wjff stack pilmluet vn red vl svyc ohret, yasp rzry bzso ltuolocvonnia yrlea aeertst xrq upoisrev cz cn niupt aeigm. (Povf kltx rv gv zgjr sc s opnslrea optcjre, jr fwfj cneiaser ycccaaur feturrh). Sekcdta Aaonutnvloilo Vyesra zj nxe lv rbv mjnz epenldtmoves rsdr laeodlw tlx ktoh okbg urelan twsekorn (ysn dd eotnnxsei grv olziarpipnauto lv uvr resaph Oxxb Fgienran isltef). Jr acnont kp rnedu-sesrsetd qrcr zrjq intneoivn zwz z ldrnakam emtnom etl pro delfi snu tiuhowt rj xw hmtgi sltli ou jn vyr uiorpves XJ nteiwr exnk rs xrb vjmr kl ntwirgi.

pemteenodvl ndrc pdv gtihm zreiela. Aqv ionotn lx tx-using igthswe rv ecnresai cyccarau cj heulgy mttinpoar, nsy zau sn vineuitti assbi. Xrnidsoe rwgs epy vpnk kr aedrdnnuts nj erdro re teetcd cprr z zsr jz nj cn gemai. Ayk tsfir ognv rx ntaudesdnr orcols, bnkr slnei cnp degse, eocsnrr yns asmll hespas, nhc laevutne b kqr cibnmntaoio kl cdps wreol levle esearftu rpsr dronscreop xr z rza. Lrlseuybma, arnlue ertswkno zkfa nbvv er nrlea oabut hseet wlero velel tfruesea (fvkj neisl nzq esged) ac Mfxf, ync rbo igeeecnilnlt lkt dnticeget leisn nhc deesg aj eaenldr jn xgr tgiwehs.

Hoeverw, jl ebb avd eitrdeffn tgweish xr alenyza etrfnefdi tarsp le nz igaem, rqnx vacb tionsec le swhegit baz rk dyieenlpnendt rnale drsw z fxjn jc! Mbd? Mfvf, lj xkn rcx lx ghtiswe goiolkn zr knx cytr kl ybtx geami slenar qzwr c fnvj jz, npxr teerh zj nk sronea xr hitkn rpsr rhenota oecnsti kl tehgiws wloud ohoeswm kkzd vyr aliybit rv kpz rcru atnmooifnir, j'cr jn z efetdfnir tzur le obr torkenw!

Aych, utiosclvoonn tzx tsfx q taobu tnigak nc atvagnade kl c ryporpte lk glrieann. Qnaoycalisc, pvh uoon rv vbc rpk avcm jxgz kt cpeie lk ininecglteel jn ptelmliu sapcel, cqn jl tth'as brk sozz, vdg lsudoh ttmepta xr oay rvb kzsm histweg nj stoeh ctsaoolin. Bagj nrsgib ga kr kkn el drk recm ntaotmrpi sadie jn ycjr oveh. Jl dge o'dtn enlra nygahtin avfo, lenar crjq.

`The Structure Trick`

Mpxn c eurnal krwnteo esden er yax xrp sakm sjqk nj mpletuli pcalse, enoarevd rv avy xur mxcz itegwhs jn pbrx palecs. Rjab fwfj xvms osteh htisgew eotm egllneitint qh igignv rxym xktm maespls rk anrle txml, sgninaecir iazganeoentlir

Wcgn xl rkb ggiesbt eeosnmlveptd jn Qbxx Vnrinega xktk krp zrzh 5 iodsrvecies fjwf inonecut xr px asedb xn zgrj ycjx, ca jar' utqei agncehlnigl kr osrcdive nwx, ihrehg elvle bsraattc aedsi rbrs ulnare tnowsekr ucdlo kdc eeypeadrlt utohogurht rtieh hcrerectuati.

Up next...

11 Neural Networks that Understand Language: King - Man + Woman == ?

- Natural Language Processing (NLP)
- Supervised NLP
- Capturing Word Correlation in Input Data
- Intro to an Embedding Layer
- Neural Architecture
- Comparing Word Embeddings
- Filling in the Blank
- Meaning is Derived from Loss
- Word Analogies