Divyanshu Soni Follow

Nov 14, 2019 · 10 min read · Listen

Save

Translation Invariance in Convolutional Neural Networks

An article on how translation invariance is achieved by convolutional NNs and its importance.

Image by instagram.com/eugene_chystiakov

INTRODUCTION:

— — — — — — — — — — — —

There are many advantages of employing convolutional neural networks for solving computer vision related tasks.

They are the default neural network architecture when it comes to image classifications, object detection,image style transfer etc.So the question is why are they so popular in the machine learning community?
well ,there are many reasons, Here I will list a few:

1. **Weight sharing :** Unlike the run-of-the-mill neural network which is characterized by dense layers , convolutional NN is made up of sequential arrangement of convolutional layers. A single convolutional layer in turn comprises of multiple filters.

The thing is , these filters are usually small in size(especially when dealing with simple images), like a grid of weights of size (5*5).
So if you have 40 filters in the first convolutional layer , the total number of weights that you have to train is 5*5*40=1000 . Compare this to a standard dense layer,having 100 neurons ,applied on an mnist image of size(28,28).
Total number of parameters here is 28*28*100=78400!!.This is a gigantic difference.
Having less weights allows Convolutional NN to converge faster to a minima, while reducing the probability of overfitting and using less memory and taking less training time.
(If you are having trouble understanding the above argument,
(http://cs231n.github.io/convolutional-networks/#overview) has a better , more in depth explanation)

2. Transfer Learning: Transfer learning is a technique that gives your NN a head start in training.
It has been observed that generally earlier layers of NN usually learn simple features and later layers work on features of previous layers to generate complex abstractions.
This is especially true for convolutional NNs. Researchers have found that most of the time the early layers of cnns learn simple and generic features like edge detection ,gabon filters,etc,which remain more or less the same irrespective of the task or image dataset used.
So under transfer learning , we copy the weights of the early layers so that the Convolutional NNs don't have to start from scratch each time . After this we tune the network with the task dataset . By using transfer learning , we reduce training time and we make the early layers more robust as they have essentially learned from multiple datasets rather than one, and more data means less overfitting to train dataset.
Interestingly, the no of layers that you can copy from an existing convolutional NN to a new one is directly proportional to the similarity between the dataset used to train the initial network and the dataset of the new task. There is a riveting and easy to read research paper

on it(https://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf) ,if anyone's interested.

3.Translation Invariance: Translation invariance ,in essence, is the ability to ignore postional shifts ,or translations , of the target in the image. A cat is still a cat regardless of whether it appears in the top half or the bottom half of the image.
Convolutional NNs have inbuilt translation invariance,and are thus better suited to dealing with image datasets than their ordinary counterparts.

"But How...", you ask?

Let us delve deeper into the depths of convolutional NNs to find out.

Achieving translation invariance in Convolutional NNs:

First ,let me give a more formal definition of translation invariance:

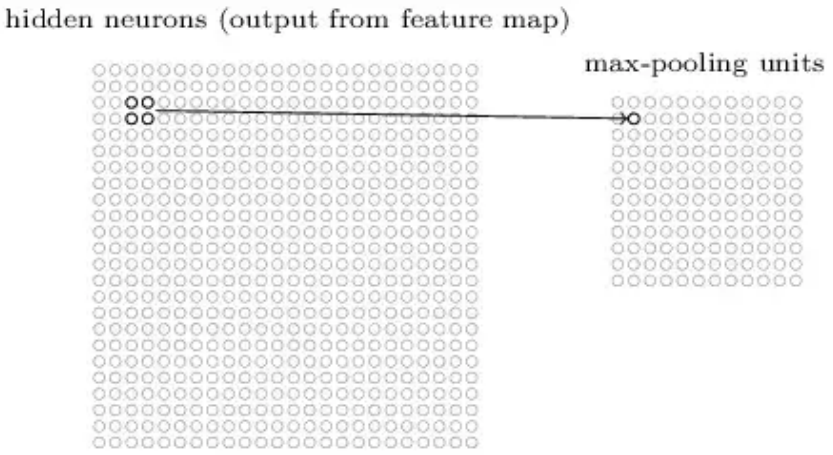
```
Assume a transformation T, which shifts the position of the target in the
input image x by some amount ,

then a NN having translation invariance would satisfy the equation

 NN(T(x))=NN(x)
```

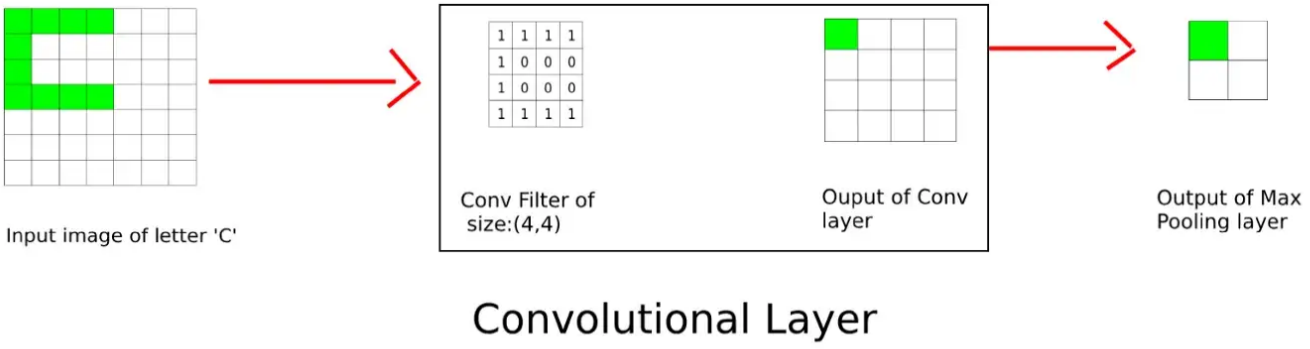
which means , if the NN has translation invariance, the output of the NN will not change when the transformation T is applied.

This translation invariance in the convolutional NN is achieved by a combination of convolutional layers and max pooling layers.
Firstly, the convolutional layer reduces the image to a set of features and their respective positions. Then the max pooling layer takes the output from the convolutional layer and reduces its resolution and complexity.
It does so by outputting only the max value from a grid.So the information about the **exact position of the max value in the grid is discarded**.
This is what imbues the CNNs with translation invariance.

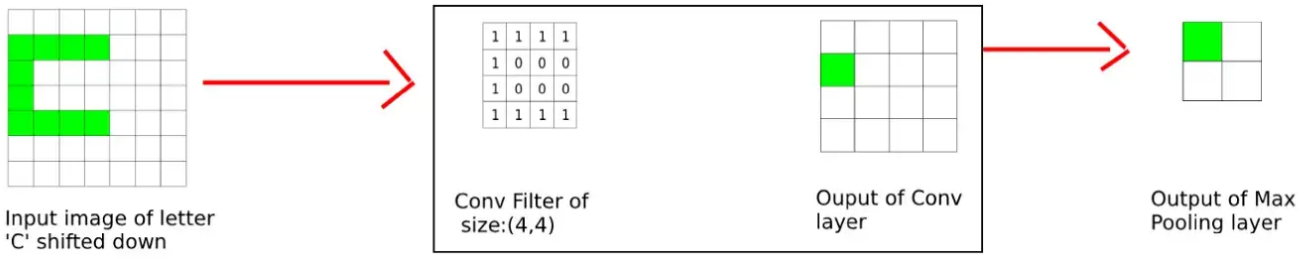


2×2 Max pooling layer at work

In the below figure,the CNN filter is trained to detect the letter ‘C’ in the input image. So lets input an image of letter ‘C’.



Now lets shift the letter ‘C’ down in the image by 1 pixel length.



Convolutional Layer

Comparing the output in the 2 cases, you can see that the max pooling layer gives the same result. The local positional information is lost. **This is translation invariance in action**.This means that if we train a Convolutional NN on images of a target, the cnn will automatically work for shifted images of that target as well.

But if u think about it … if we had shifted the letter 'C' more severely, the output would have been different.
To make the translation invariance more effective, we could use more convolutional + max pooling layers, as successive max pooling layers compund the effect of translation invariance.(try to think why this is the case).

Now this all has been just theory. To see it in action , we will compare the translation invariance of convolutional neural network and standard NN by training both on an image classification task.

Experimenting with Translation Invariance:

We will pick the classic MNIST image classification task to compare the performance of 3 NN architectures.
1.Standard NN
2.Convolutional NN with 1 (convolutional + max pooling) layer
3.Convolutional NN with 2 (convolutional + max pooling)layers

Firstly, lets create the datasets required :

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_testShift = shiftImageDataset(4,3,x_test)
x_testShiftMin = shiftImageDataset(1,1,x_test)
```

we create 2 new datasets from the base test dataset that has shifted images of handwritten digits.

x_testShift is the x_test dataset with each image randomly shifted by a value in [4,-4] along x-axis and by [3,-3] along y axis.

x_testShiftMin is the x_test dataset with each image randomly shifted by a value in [1,-1] along x-axis and by [1,-1] along y axis.(less severe image translation).

we will use these 2 shift datasets to measure how the networks perform against small translation distortion (x_testShiftMin) in data vs large translation distortion(x_testShift).

The function below was used to create the shifted datasets by shifting images:

```
def shiftImageDataset(maxXShift,maxYShift,x_dataset):
    x_Shift=[]
    for i in range(y_test.shape[0]):
        randShiftx=np.random.rand()*maxXShift
        randShifty=np.random.rand()*maxYShift
        ra=np.random.rand(2)
        if ra[0]>0.5:
            randShiftx=randShiftx*-1
        if ra[1]>0.5:
            randShifty=randShifty*-1
        M = np.float32([[1, 0, randShiftx], [0, 1, randShifty]])
        (rows, cols) = x_dataset[i].shape[:2]
        shiftedImage = cv2.warpAffine(x_dataset[i], M, (cols, rows))
        x_Shift.append(shiftedImage)
    x_Shift=np.array(x_Shift)
    x_Shift=x_Shift.reshape(10000,28,28,1)
    return x_Shift
```

Basically, the function shifts the images in the dataset by some random value in [maxXshift,-maxXshift] and [maxYshift,-maxYshift] along x and y axis respectively.

Now, lets create the 3 models using Keras:

```
input_shape = (28, 28, 1)

# Standard NN with 2 hidden layers
stdmodel = Sequential()
stdmodel.add(Dense(100,activation='relu' ,input_shape=input_shape))
stdmodel.add(Dropout(0.5))
stdmodel.add(Dense(100,activation='relu',input_shape=input_shape))
stdmodel.add(Dropout(0.25))
stdmodel.add(Flatten())
stdmodel.add(Dense(10, activation = "softmax"))

#Convolutional NN with 1 (conv + max pool)layer
modelcnn1 = Sequential()
modelcnn1.add(Conv2D(filters = 20,kernel_size = (5,5),padding =
'valid',strides=1,activation ='relu', input_shape = input_shape))

modelcnn1.add(MaxPooling2D(pool_size=(2,2)))
modelcnn1.add(Flatten())
modelcnn1.add(Dense(100,activation='relu'))
modelcnn1.add(Dense(10, activation = "softmax"))

#Convolutional NN with 2(conv + max pool)layer
modelcnn2 = Sequential()
modelcnn2.add(Conv2D(filters = 20,kernel_size = (5,5),padding =
'valid',strides=1,activation ='relu', input_shape = input_shape))
modelcnn2.add(MaxPooling2D(pool_size=(2,2)))

modelcnn2.add(Conv2D(filters = 40, kernel_size = (5,5),strides=1,padding =
'valid',activation ='relu'))
modelcnn2.add(MaxPooling2D(pool_size=(2,2)))

modelcnn2.add(Flatten())
modelcnn2.add(Dense(100,activation='relu'))
modelcnn2.add(Dense(10, activation = "softmax"))
```

Now its time to train the 3 models on MNIST train dataset and see what kind of accuracy each gets on our shifted image datasets.

```
batch_size = 128
lrn_rate=0.0001
epochs=100  #use early stopping if models start overfitting

stdmodel.compile(optimizer=Adam(lr=lrn_rate),loss=tf.keras.losses.categori
cal_crossentropy,metrics=['accuracy'])

stdmodel.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = stdmodel.evaluate(x_test, y_test, verbose=0)
```

```
scoreShiftMin = stdmodel.evaluate(x_testShiftMin, y_test, verbose=0)
scoreShift=stdmodel.evaluate(x_testShift, y_test, verbose=0)
```

Similar to above we evaluate the 2 cnn models on the 3 datasets(x_test, x_testShiftMin, x_testShift) as well.

Following is the accuracy of the models against the 3 test datasets:

Model Name	Accuracy : mnist image testset	Accuracy : testShiftMin testset	Accuracy : testShift testset
Standard NN	0.93	0.861	0.449
Convolutional NN model1	0.986	0.919	0.718
Convolutional NN model2	0.989	0.982	0.831

Just as our theory predicted.

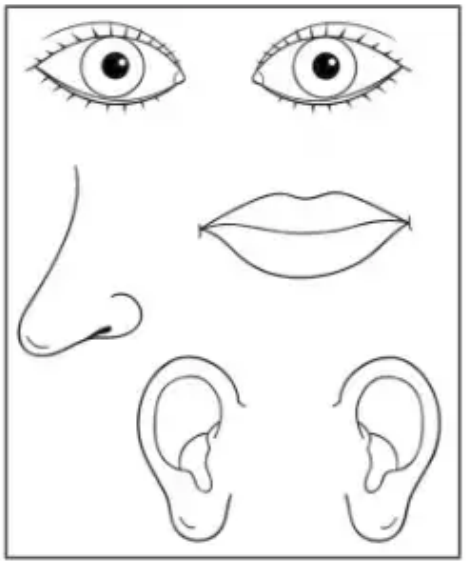
The standard NN performs poorly when compared to Convolutional NNs on shifted images . Its performance plummets as the distortion increases from testShiftMin to testShift. This is because standard NN **lacks inbuilt translation invariance.**

The ‘modelcnn1' which had 1 (convolutional +max pooling) layer is more stable in dealing with shifted images. But still it falls short on the testShift testset, which had larger shifts in images. This is because the effect of a single max pooling layer is not potent enough to deal with large positional shifts of target in image.

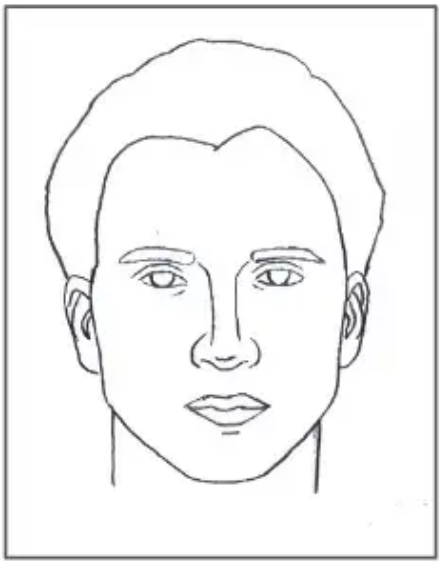
The ‘modelcnn2' which had 2 (convolutional +max pooling) layer is the best of the bunch. It performs reasonably well against larger image shifts in the testShift dataset. This is due to the compunding effect of using 2 max pooling layers .

Disadvantage of Max Pooling:

The major issue with max pooling is that the network fails to learn the spatial relation between different features, and thus will give a false positive if all features are present in the wrong position with respect to one another.
The following image illustrates the point.



Not Face



Face

Max pool layer is unable to distinguish between these images

This happens because the max pooling layer is applied to the outputs of each filter from previous convolution layer separately .Due to separately applying max pooling on each filter, much of the relative positional information between features is lost. So next layer can only give output based on the presence of features.
As we go to subsequent layers, more and more of the spatial information is lost , as max pooling layer compunds the effect of translation invariance. In the end it detects a face even when the eye is present in the place of the mouth. Pretty embarrassing moment for an AI.

This is why some people are not fond of max pooling layers.Pooling is inherently a lossy process. But we can't just remove it because it provides translation invariance and dimentionality reduction (reduces the spatial size of the representation exponentially to reduce the amount of parameters and computation in the network, and also ameliorates overfitting).

So are there other ways to make your convolutional NN translation invariant besides using max pooling layers?

Data Augmentation:
Data augmentation is a form of regularisation that is used to make the Model more robust against distortion in data .We do this by creating the distortion in our training data and training the model on this "Augmented" data.

In our case , we are expecting heavy translation distortion in test data, so we will train our model on images having similar distortions.

What it means here is that we will train our network on large shifted images to make it more translation invariant to large shifts.This will allow us to get better performance on x_testShift data while maintaining the same no. of layers.
we will use the keras ImageDataGenerator ,which is a python generator that outputs distorted images that the network can train on.

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 in data
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=0,  # randomly rotate images by degree
        zoom_range = 0, # Randomly zoom image
        width_shift_range=0.1,  # randomly shift images horizontally
        height_shift_range=0.1,  # randomly shift images vertically
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

Although the ImageDataGenerator provides various distortions , we will only use ‘width_shift_range’ and ‘ height_shift_range’. we use value ‘0.1’ as the max fraction of total width (or height) by which the image will be shifted.

Time to train our CNN using ImageDataGenerator.

```
datagen.fit(x_train)
batch_size = 128
lrn_rate=0.0001
epochs=100

modelcnn2.compile(optimizer=Adam(lr=lrn_rate),loss=tf.keras.losses.categor
ical_crossentropy,metrics=['accuracy'])

modelcnn2.fit_generator(datagen.flow(x_train,y_train,
batch_size=batch_size),
epochs = epochs,
validation_data = (x_test,y_test),
 verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)

score=modelcnn2.evaluate(x_test, y_test, verbose=0)

scoreShiftMin = modelcnn2.evaluate(x_testShiftMin, y_test,verbose=0)
```

```
scoreShift=modelcnn2.evaluate(x_testShift, y_test, verbose=0)
```

Our new setup gives us an accuracy of **0.9933,0.9918,.9839** on x_test, x_testShiftMin and x_testShift respectively, which is far better than what we got previously on modelcnn2. Its obvious that training our network on augmented data has made it more translation invariant.

Besides data augmentation, there are other alternatives to using max pooling.
The authors of Striving for Simplicity: The All Convolutional Net(https://arxiv.org/abs/1412.6806) recommend replacing the max pooling layers with conv layers having filters of size 3*3 and stride=2.

Others have suggested more complicated architectures to overcome the disadvantages of max pooling, like Detail-Preserving Pooling in Deep Networks(https://arxiv.org/pdf/1804.04076.pdf) or capsule networks (https://arxiv.org/abs/1805.04424).

invariance by max pooling. We learned how this property can be acquired by using successive max pooling layers or by data augmentation.
Hopefully, this post helped you in better understanding the convolutional NNs in general.

Resources:

CS231n Convolutional Neural Networks for Visual Recognition: (http://cs231n.github.io/convolutional-networks/#overview)

How transferable are features in deep neural networks?: https://papers.nips.cc/paper/5347-how-transferable-are-features-in-deep-neural-networks.pdf

The All Convolutional Net(https://arxiv.org/abs/1412.6806)

Machine Learning Convolutional Network Data Science Image Classification