1. After each stride-2 conv, why do we double the number of filters?

A stride 2 conv with the default padding (1) and ks (3) will reduce the activation map dimension by half. Formula: (n + 2\*pad - ks)//stride + 1. As the activation map dimension reduces by half we double the number of filters. This results in no overall change in computation as the network gets deeper and deeper.

2. Why do we use a larger kernel with MNIST (with simple cnn) in the first conv?

A MLP program is useful to study multiple topics in Machine Learning [ML] on a basic level. However, MLPs with dense layers are certainly not at the forefront of ML technology – though they still are fundamental bricks in other more complicated architectures of “Artifical Neural Networks” [ANNs]. During my MLP experiments I became sufficiently acquainted with Python, Jupyter and matplotlib to make some curious first steps into another field of Machine Learning [ML] now: “Convolutional Neural Networks” [CNNs].

CNNs on my level as an interested IT-affine person are most of all fun. Nevertheless, I quickly found out that a somewhat systematic approach is helpful – especially if you later on want to use the Tensorflow’s API and not only Keras. When I now write about some experiments I did and do I summarize my own biased insights and sometimes surprises. Probably there are other hobbyists as me out there who also fight with elementary points in the literature and practical experiments. Books alone are not enough … I hope to deliver some practical hints for this audience. The present articles are, however, NOT intended for ML and CNN experts. Experts will almost certainly not find anything new here.

Although I address CNN-beginners I assume that people who stumble across this article and want to follow me through some experiments have read something about CNNs already. You should know fundamentals about filters, strides and the basic principles of convolution. I shall comment on all these points but I shall not repeat the very basics. I recommend to read relevant chapters in one of the books I recommend at the end of this article. You should in addition have some knowledge regarding the basic structure and functionality of a MLP as well as “gradient descent” as an optimization technique.

The objective of this introductory mini-series is to build a first simple CNN, to apply it to the MNIST dataset and to visualize some of the elementary “features” a CNN allegedly detects in the images of handwritten digits – at least according to many authors in the field of AI. We shall use Keras (with the Tensorflow 2.2 backend and CUDA 10.2) for this purpose. And, of course, a bit of matplotlib and Python/Numpy, too. We are working with MNIST images in the first place – although CNNs can be used to analyze other types of input data. After we have covered the simple standard MNIST image set, we shall also work a bit with the so called “MNIST fashion” set.

But in this article I start with some introductory words on the structure of CNNs and the task of its layers. We shall use the information later on as a reference. In the second article we shall set up and test a simple version of a CNN. Further articles will then concentrate on visualizing what a trained CNN reacts to and how it modifies and analyzes the input data on its layers.

Why CNNs?

When we studied an MLP in combination with the basic MNIST dataset of handwritten digits we found that we got an improvement in accuracy (for the same setup of dense layers) when we pre-processed the data to find “clusters” in the image data before training. Such a process corresponds to detecting parts of an MNIST image with certain gray-white pixel constellations. We used Scikit-Learn’s “MiniBatchKMeans” for this purpose.

We saw that the identification of 40 to 70 cluster areas in the images helped the MLP algorithm to analyze the MNIST data faster and better than before. Obviously, training the MLP with respect to combinations of characteristic sub-structures of the different images helped us to classify them as representations of digits. This leads directly to the following question:

What if we could combine the detection of sub-structures in an image with the training process of an ANN?

CNNs seem to the answer! According to teaching books they have the following abilities: They are designed to detect elementary structures or patterns in image data (and other data) systematically. In addition they are enabled to learn something about characteristic compositions of such elementary features during training. I.e., they detect more abstract and composite features specific for the appearance of certain objects within an image. We speak of a “feature hierarchy“, which a CNN can somehow grasp and use – e.g. for classification tasks.

While a MLP must learn about pixel constellations and their relations on the whole image area, CNNs are much more flexible and even reusable. They identify and remember elementary sub-structures independent of the exact position of such features within an image. They furthermore learn “abstract concepts” about depicted objects via identifying characteristic and complex composite features on a higher level.

This simplified description of the astonishing capabilities of a CNN indicates that its training and learning is basically a two-fold process:

Detecting elementary structures in an image (or other structured data sets) by filtering and extracting patterns within relatively small image areas. We shall call these areas “filter areas”.

Constructing abstract characteristic features out of the elementary filtered structural elements. This corresponds to building a “hierarchy” of significant features for the classification of images or of distinguished objects or of the positions of such objects within an image.

Now, if you think about the MNIST digit data we understand intuitively that written digits represent some abstract concepts like certain combinations of straight vertical and horizontal line elements, bows and line crossings. The recognition of certain feature combinations of such elementary structures would of course be helpful to recognize and classify written digits better – especially when the recognition of the combination of such features is independent of their exact position on an image.

So, CNNs seem to open up a world of wonders! Some authors of books on CNNs, GANs etc. praise the ability to react to “features” by describing them as humanly interpretable entities as e.g. “eyes”, “feathers”, “lips”, “line segments”, etc. – i.e. in the sense of entity conceptions. Well, we shall critically review this idea, which I think is a misleading over-interpretation of the capacities of CNNs.

3. What data is saved by ActivationStats for each layer?

A convolution converts all the pixels in its receptive field into a single value. For example, if you would apply a convolution to an image, you will be decreasing the image size as well as bringing all the information in the field together into a single pixel. The final output of the convolutional layer is a vector.

The class activation map is simply a weighted lin- ear sum of the presence of these visual patterns at different spatial locations. By simply upsampling the class activa- tion map to the size of the input image, we can identify the image regions most relevant to the particular category.

4. How do we get a learner's callback after they've completed training?

Events

Callbacks can occur at any of these times:: after\_create before\_fit before\_epoch before\_train before\_batch after\_pred after\_loss before\_backward after\_cancel\_backward after\_backward before\_step after\_cancel\_step after\_step after\_cancel\_batch after\_batch after\_cancel\_train after\_train before\_validate after\_cancel\_validate after\_validate after\_cancel\_epoch after\_epoch after\_cancel\_fit after\_fit.

event

event (\*args, \*\*kwargs)

All possible events as attributes to get tab-completion and typo-proofing

To ensure that you are referring to an event (that is, the name of one of the times when callbacks are called) that exists, and to get tab completion of event names, use event:

test\_eq(event.before\_step, 'before\_step')

[source](https://github.com/fastai/fastai/blob/master/fastai/callback/core.py#L46)

Callback

Callback (after\_create=None, before\_fit=None, before\_epoch=None,

before\_train=None, before\_batch=None, after\_pred=None,

after\_loss=None, before\_backward=None,

after\_cancel\_backward=None, after\_backward=None,

before\_step=None, after\_cancel\_step=None, after\_step=None,

after\_cancel\_batch=None, after\_batch=None,

after\_cancel\_train=None, after\_train=None,

before\_validate=None, after\_cancel\_validate=None,

after\_validate=None, after\_cancel\_epoch=None, after\_epoch=None,

after\_cancel\_fit=None, after\_fit=None)

Basic class handling tweaks of the training loop by changing a [Learner](https://docs.fast.ai/learner.html#learner) in various events

The training loop is defined in [Learner](https://docs.fast.ai/learner.html#learner) a bit below and consists in a minimal set of instructions: looping through the data we:

compute the output of the model from the input

calculate a loss between this output and the desired target

compute the gradients of this loss with respect to all the model parameters

update the parameters accordingly

zero all the gradients

Any tweak of this training loop is defined in a [Callback](https://docs.fast.ai/callback.core.html" \l "callback) to avoid over-complicating the code of the training loop, and to make it easy to mix and match different techniques (since they’ll be defined in different callbacks). A callback can implement actions on the following events:

after\_create: called after the [Learner](https://docs.fast.ai/learner.html#learner) is created

before\_fit: called before starting training or inference, ideal for initial setup.

before\_epoch: called at the beginning of each epoch, useful for any behavior you need to reset at each epoch.

before\_train: called at the beginning of the training part of an epoch.

before\_batch: called at the beginning of each batch, just after drawing said batch. It can be used to do any setup necessary for the batch (like hyper-parameter scheduling) or to change the input/target before it goes in the model (change of the input with techniques like mixup for instance).

after\_pred: called after computing the output of the model on the batch. It can be used to change that output before it’s fed to the loss.

after\_loss: called after the loss has been computed, but before the backward pass. It can be used to add any penalty to the loss (AR or TAR in RNN training for instance).

before\_backward: called after the loss has been computed, but only in training mode (i.e. when the backward pass will be used)

after\_backward: called after the backward pass, but before the update of the parameters. Generally before\_step should be used instead.

before\_step: called after the backward pass, but before the update of the parameters. It can be used to do any change to the gradients before said update (gradient clipping for instance).

after\_step: called after the step and before the gradients are zeroed.

after\_batch: called at the end of a batch, for any clean-up before the next one.

after\_train: called at the end of the training phase of an epoch.

before\_validate: called at the beginning of the validation phase of an epoch, useful for any setup needed specifically for validation.

after\_validate: called at the end of the validation part of an epoch.

after\_epoch: called at the end of an epoch, for any clean-up before the next one.

after\_fit: called at the end of training, for final clean-up.

5. What are the drawbacks of activations above zero?

Apart from that, the linear activation function has its set of disadvantages such as: We observe that the function's derivative is a constant. That means there is constant gradient descent occurring since there is no relation to the value of z.

6.Draw up the benefits and drawbacks of practicing in larger batches?

Batch production is common in more than just bakeries and small businesses that produce in controlled quantities. Regularly, you can find batch production being utilized by companies that make consistent, measured products such as paint, shoes, ingredients for pharmaceuticals and much more. So what are some of the advantages and disadvantages of this system that can help companies decide if it’s the right manufacturing process for them?

Pros of Batch Production

Products can be [produced in mass quantities](https://gesrepair.com/mass-production-advantages-and-disadvantages/), reducing the overall cost per unit

Companies only focus on a small group of products, leading to greater quality control and product expertise

Cost of labor is reduced, as workers only focus on a particular task or set of tasks

Cost of machinery is reduced, as one machine can handle several different product configurations

Lends itself to repeat orders, meaning a smoother, more consistent production flow over time

Machinery isn’t always on, saving on energy costs

Cons of Batch Production

Each batch must be tested for quality and uniformity before future batches can be produced, causing idle downtime

Machinery must be stopped and recalibrated between batches, also causing downtime

Storage costs are high for large batches of the same product

Fewer varieties of jobs can demotivate employees

Batch production clearly has its place, primarily with companies looking to make a consistent, on-demand product while keeping costs low and ensuring quality standards. Share your comments or experiences regarding batch production in the comments section below.  Be sure to visit us online at [gesrepair.com](https://gesrepair.com/) or call us at 1-877-249-1701 to learn more about our services. We’re proud to offer Surplus, Complete Repair and Maintenance on all types of Industrial Electronics, Servo Motors, AC and DC Motors, Hydraulics and Pneumatics. Please subscribe to our [YouTube](https://www.youtube.com/channel/UCJcn4Yzzzp7ileEWSua4mnQ) page and Like Us on [Facebook](https://www.facebook.com/gesrepair)! Thank you!

7. Why should we avoid starting training with a high learning rate?

If your learning rate is set too low, training will progress very slowly as you are making very tiny updates to the weights in your network. However, if your learning rate is set too high, it can cause undesirable divergent behavior in your loss function.

8. What are the pros of studying with a high rate of learning?

Career Preparation. ...

Broader Practical Benefits. ...

Personal Development. ...

Pursuing a Passion and Desired Field. ...

Cognitive and Communication Skills. ...

Social Experiences.

9. Why do we want to end the training with a low learning rate?

It iterates the process until it reaches a global minimum for the loss function. The number of data points to be learned is controlled by a parameter known as the learning rate. The lesser the learning rate, the more the data points included for the algorithm to learn.