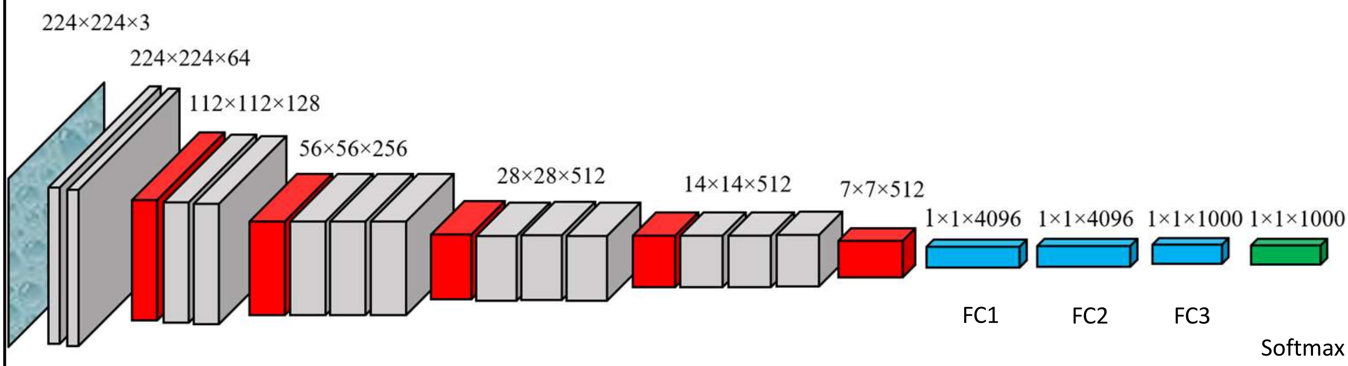
Deep Learning – VGG

Bernhard Kainz

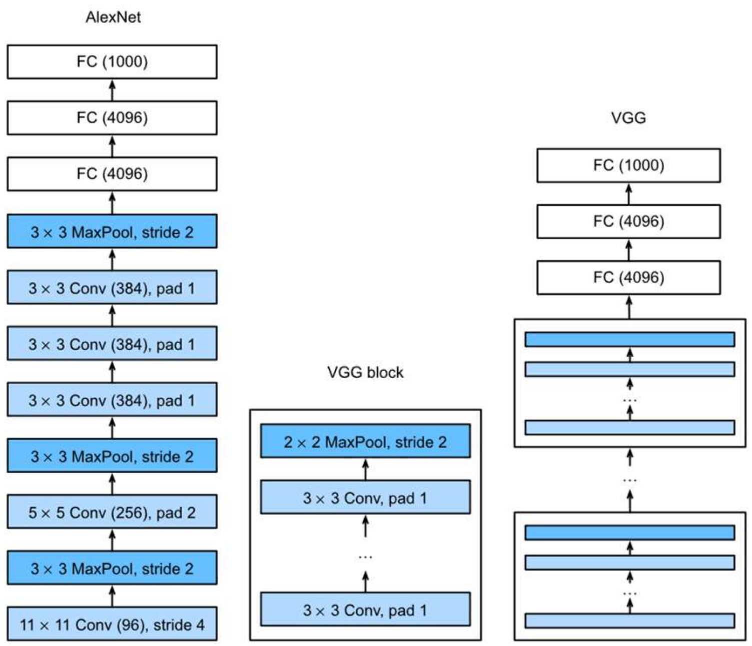
Lecture inspired by Alex Smola with add-ons

VGG



Let’s talk about a network architecture that you’ll probably use in practice as a backbone for feature extraction.
The next step in the evolution of design came in the form of VGG.
VGG stands for the visual geometry group in Oxford
They read the Alexnet paper and learned that apparently bigger is better so they decided to go even bigger.

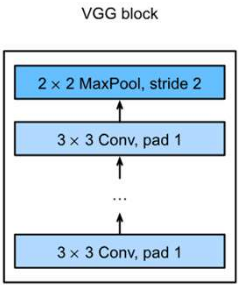
- AlexNet = bigger than LeNet
- Bigger = better?
- Options
  - **More** dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**



Deep Learning – Bernhard Kainz http://d2l.ai/chapter_convolutional-modern/vgg.html

let's have a look at how this looks like in detail
If we compare this to Alexnet and Lenet
To make it bigger you can add even more dense layers.
well maybe not really because that's too expensive.
or you add more convolutions
You can do that but then at some point you start getting tired of having to define every convolution separately so you might as well group them into blocks
once you go to 20 30 40 layers it gets quite annoying having to specify by hand
so the key innovation in VGG is actually this grouping into blocks which then turns into parametretisable repeated blocks that we can use for learning tasks.

VGG blocks



- Deeper vs. wider?
  - 13x13?
  - 5x5?
  - 3x3?
  - Deep and narrow = better
- VGG block
  - 3x3 convolutions (pad 1)
    (n layers, m channels)
  - 2x2 max-pooling
    (stride 2)

Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* & Andrew Zisserman+
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Let’s look at the VGG blocks

the first thing they had to solve is whether you should use fewer wide convolutions or more narrow ones

[paper] did a good comprehensive analysis and it showed that more layers of narrow convolutions were more powerful than a smaller number of whide convolutions

this tends to be a trend overall in the network designs that a larger number of compositions of simple functions turns out to be more expressive and more able to fit meaningful models than a small number of shallower and more complicated functions.

The VGG block has a bunch of 3x3 convolutions
if you padded them by one it didn't change the size of the input relative to the output and then in the end you have max pooling of two by two with a stride of two which halves the resolution.

now if you stack several of those things together and combine it with the same dense layers as we had in Alexnet, then you get VGG.

you get actually entire family of different such architectures simply by varying the number of such blocks that you will combine.

so you get eg VGG-16 or VGG-19 and so on.

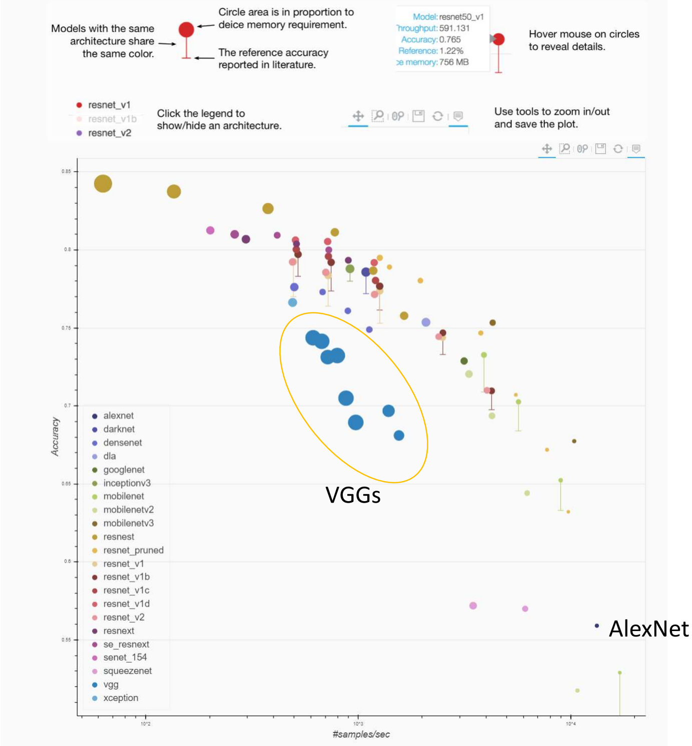
If you think about the overall progress so far it basically boils down to bigger and deeper.

progress

Wouldn’t have been possible without compute power progress (GPUs)

- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)

In LeNet you had 2 convolution and pooling layers
In AlexNet everything became bigger
Followed by VGG



https://cv.gluon.ai/model_zoo/classification.html

Here is a block of throughput vs. accuracy.
VGG is a lot slower but also more accurate than AlexNet

following one people move back to smaller networks but with higher accuracy

```python
def vgg16(pretrained: bool = False, progress: bool = True, **kwargs: Any) -> VGG:
    r"""VGG 16-layer model (configuration "D")
    `"Very Deep Convolutional Networks For Large-Scale Image Recognition" <https://arxiv.org/pdf/1409.1556.pdf>`_

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
        progress (bool): If True, displays a progress bar of the download to stderr
    """
    return _vgg('vgg16', 'D', False, pretrained, progress, **kwargs)
```

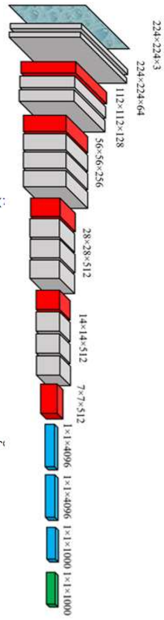
```python
def _vgg(arch: str, cfg: str, batch_norm: bool, pretrained: bool, progress: bool, **kwargs: Any) -> VGG:
    if pretrained:
        kwargs['init_weights'] = False
    model = VGG(make_layers(cfgs[cfg], batch_norm=batch_norm), **kwargs)
    if pretrained:
        state_dict = load_state_dict_from_url(model_urls[arch],
                                              progress=progress)
        model.load_state_dict(state_dict)
    return model
```

```python
def make_layers(cfg: List[Union[str, int]], batch_norm: bool = False) -> nn.Sequential:
    layers: List[nn.Module] = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            v = cast(int, v)
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

```python
class VGG(nn.Module):

    def __init__(
        self,
        features: nn.Module,
        num_classes: int = 1000,
        init_weights: bool = True
    ) -> None:
        super(VGG, self).__init__()
        self.features = features
        self.avgpool = nn.AdaptiveAvgPool2d((
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
        if init_weights:
            self._initialize_weights()

    def forward(self, x: torch.Tensor) -> tor
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```



Bernhard Kainz

https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py