Lecture 1. Recap of DL Features

Mayur Hemani

Outline

1. CNN Features
   i) They are amazing
   ii) Change with the input domain
   iii) Change with the objective
   iv) Transfer learning works
   v) Multi-domain, multi-task training
   vi) Attention is all you need.

2. Transformers
   i) Born from text
   ii) Drawn into vision

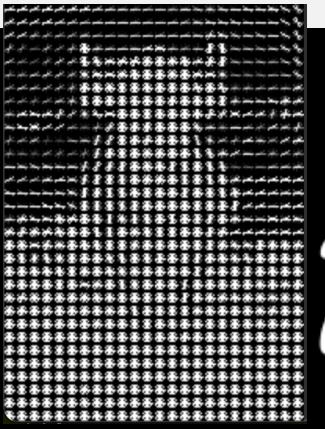
Visual Features Before Deep Learning

Image

Gradient Magnitude & Direction

```
In [1]: def gradients(gray, norm=12):
   ...:     kx = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]], dtype=np.float32)
   ...:     ky = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]], dtype=np.float32)
   ...:     ex = cv2.filter2D(gray, cv2.CV_32F, kx/norm)
   ...:     ey = cv2.filter2D(gray, cv2.CV_32F, ky/norm)
   ...:     mag = np.sqrt(ex*ex + ey*ey)
   ...:     ang = np.arctan2(ey, ex)
   ...:     return mag, ang
```

```
In [2]: def gradhist(mag, ang, cellsz=8, blocksz=3):
   ...:     bi = np.arange(0, np.pi + np.pi/8, np.pi/8)
   ...:     qang = bi[np.digitize(np.abs(ang), bins=bi)]
   ...:     hists = []
   ...:     for i in range(0, ang.shape[0], cellsz):
   ...:         hists.append([])
   ...:         for j in range(0, ang.shape[1], cellsz):
   ...:             freqs, bis = np.histogram(qang[i:(i+cellsz), j:(j+cellsz)], bi,
   ...:                 weights=mag[i:(i+cellsz),j:(j+cellsz)])
   ...:             hists[-1].append(freqs)
   ...:     hists = np.array(hists)
   ...:     blocks = []
   ...:     for br in range(0, hists.shape[0]-blocksz, 1):
   ...:         blocks.append([])
   ...:         for bc in range(0, hists.shape[1]-blocksz, 1):
   ...:             block = hists[br:(br+blocksz), bc:(bc+blocksz)]
   ...:             blocks[-1].append(block / np.sqrt(np.sum(block**2) + 1e-8))
   ...:     blocks = np.array(blocks)
   ...:     return blocks
```



Histogram of Oriented Gradients

Deep Learning Changed Everything

```
def image2tensor(image: np.ndarray):
    mean = np.array([[[0.485, 0.456, 0.406]]])
    std = np.array([[[0.229, 0.224, 0.225]]])
    image = (image/255. - mean)/std
    image = image.transpose((2, 0, 1))
    return torch.from_numpy(image).float()
```

```
(features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
```

Convolutional Neural Features

~57.1%

```
class AlexNet(nn.Module):
 def __init__(self, num_classes=1000):
    super().__init__()
    self.features = nn.Sequential(
      nn.Conv2d(3, 96, 11, stride=4), nn.ReLU(),
      nn.MaxPool2d(3, stride=2),
      nn.Conv2d(96, 256, 5, padding=2), nn.ReLU(),
      nn.MaxPool2d(3, stride=2),
      nn.Conv2d(256, 384, 3, padding=1), nn.ReLU(),
      nn.Conv2d(384, 384, 3, padding=1), nn.ReLU(),
      nn.Conv2d(384, 256, 3, padding=1), nn.ReLU(),
      nn.MaxPool2d(3, stride=2),
    )
    self.classifier = nn.Sequential(
       nn.Dropout(), nn.Linear(256*6*6, 4096), nn.ReLU(),
       nn.Dropout(), nn.Linear(4096, 4096), nn.ReLU(),
       nn.Linear(4096, num_classes)
    )
 def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    return self.classifier(x)
```

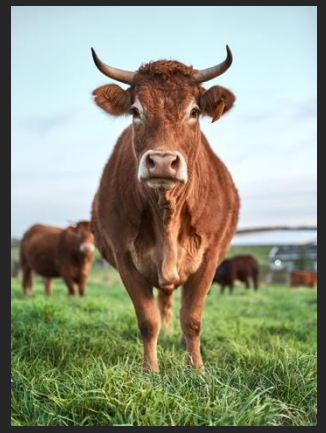
~71.5%

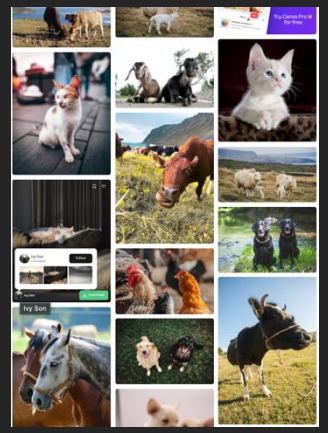
```
def make_layers(cfg):
   layers, in_channels = [], 3
   for v in cfg:
      if v == 'M':
        layers += [nn.MaxPool2d(2, 2)]
      else:
        layers += [
            nn.Conv2d(
               in_channels,
               v,
               kernel_size=3,
               padding=1
            ),
           nn.ReLU()
        ]
        in_channels = v
   return nn.Sequential(*layers)

cfg = [64, 64, 'M',
128, 128, 'M',
256, 256, 256, 'M',
512, 512, 512, 'M',
512, 512, 512, 'M']

VGG16 = nn.Sequential(
                  make_layers(cfg),
                   nn.Flatten(),
                   nn.Linear(512*7*7, 4096))
```

~76.2%

```
class ResidualBlock(nn.Module):
   def __init__(self, in_channels, out_channels, stride=1):
      super().__init__()
      self.conv1 = nn.Conv2d(in_channels, out_channels, 3, stride,
1)
      self.bn1 = nn.BatchNorm2d(out_channels)
      self.conv2 = nn.Conv2d(out_channels, out_channels, 3, 1, 1)
      self.bn2 = nn.BatchNorm2d(out_channels)
      self.shortcut = nn.Identity()
      if stride != 1 or in_channels != out_channels:
         self.shortcut = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 1, stride),
            nn.BatchNorm2d(out_channels)
         )
   def forward(self, x):
      out = nn.ReLU()(self.bn1(self.conv1(x)))
      out = self.bn2(self.conv2(out))
       out += self.shortcut(x)
      return nn.ReLU()(out)
```

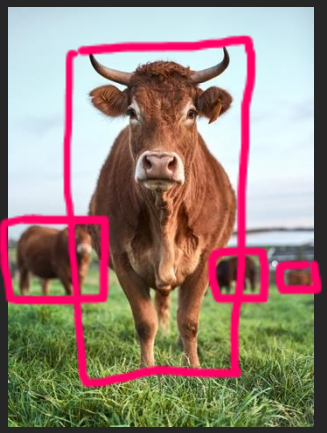
Now you can not only categorize your cows but also locate them and find them in a haystack of images





Deep Neural Network

Bbox pred.

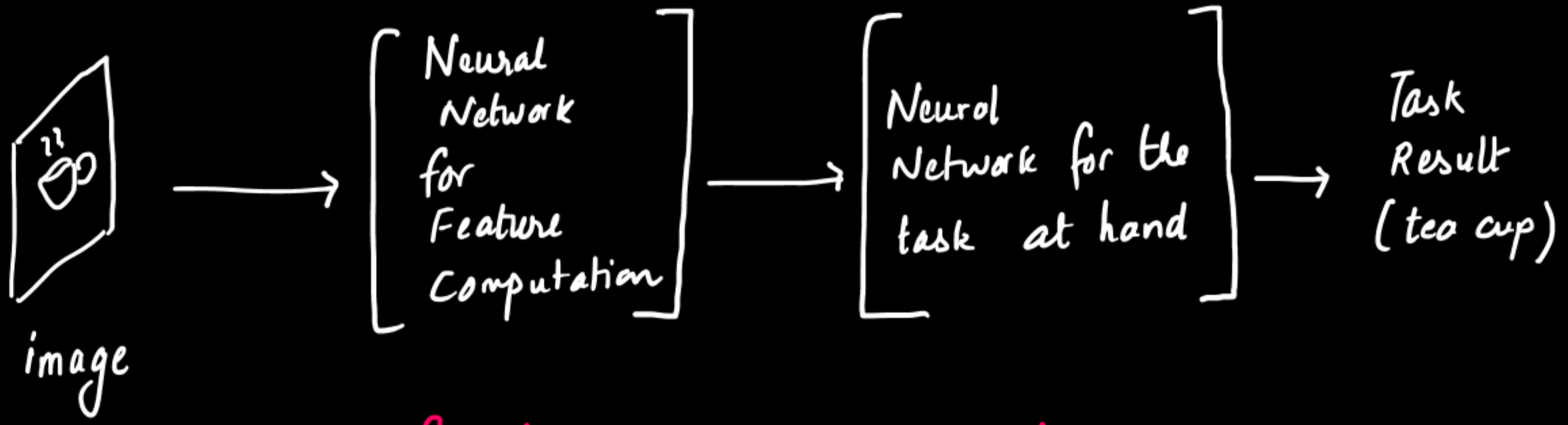


Bboxes

Query

Indexing

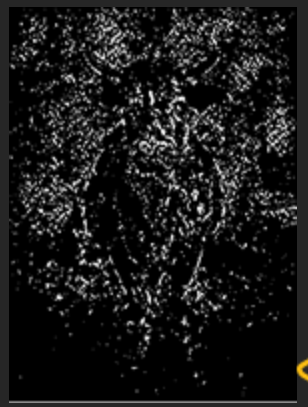
Retrieved images


Neural Network for Feature Computation
Neural Network for the task at hand
Task Result (tea cup)
image
Encoder
Decoder

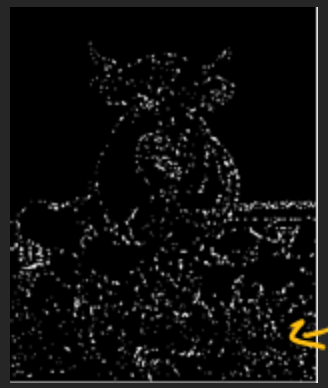
What happens if the INPUT changes?

Resnet 50

Layer 4 feature vis.



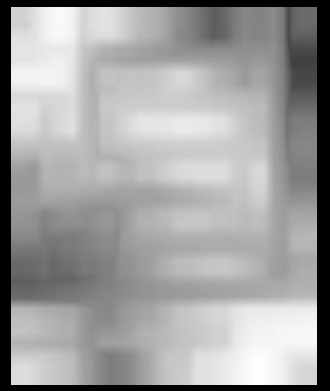
Nothing in the grass



A lot in the grass.

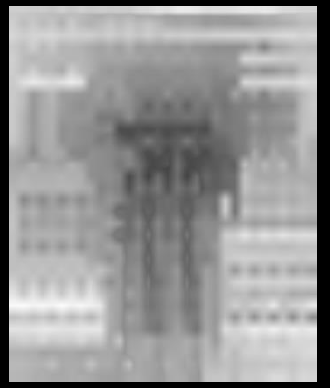
But things change as tasks change

ResNet50 →



Classification Features

Deeplab v3 (Resnet 50) →



Segmentation features.

So, the features produced can be different depending on the task.
This could be called Objective Dependence

So. DL features are sensitive to / dependent on both

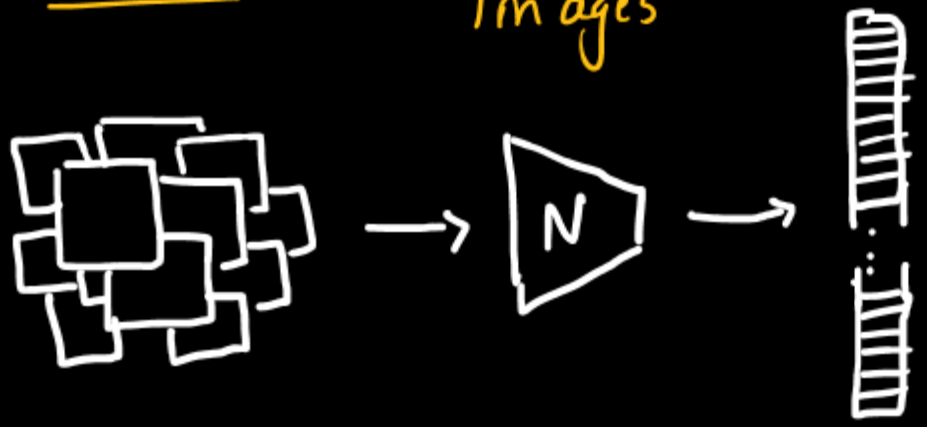
a. The input "domain"

b. The output objective

How then can we have features that work for other "domains" and objectives?

Perhaps by retraining for specific tasks

Step 1. Train on a ton of images



Step 2. Train on the specific classes / task you need ...

1. Why does this work?

2. Why "pre-train"?

Why might transfer learning work?

---

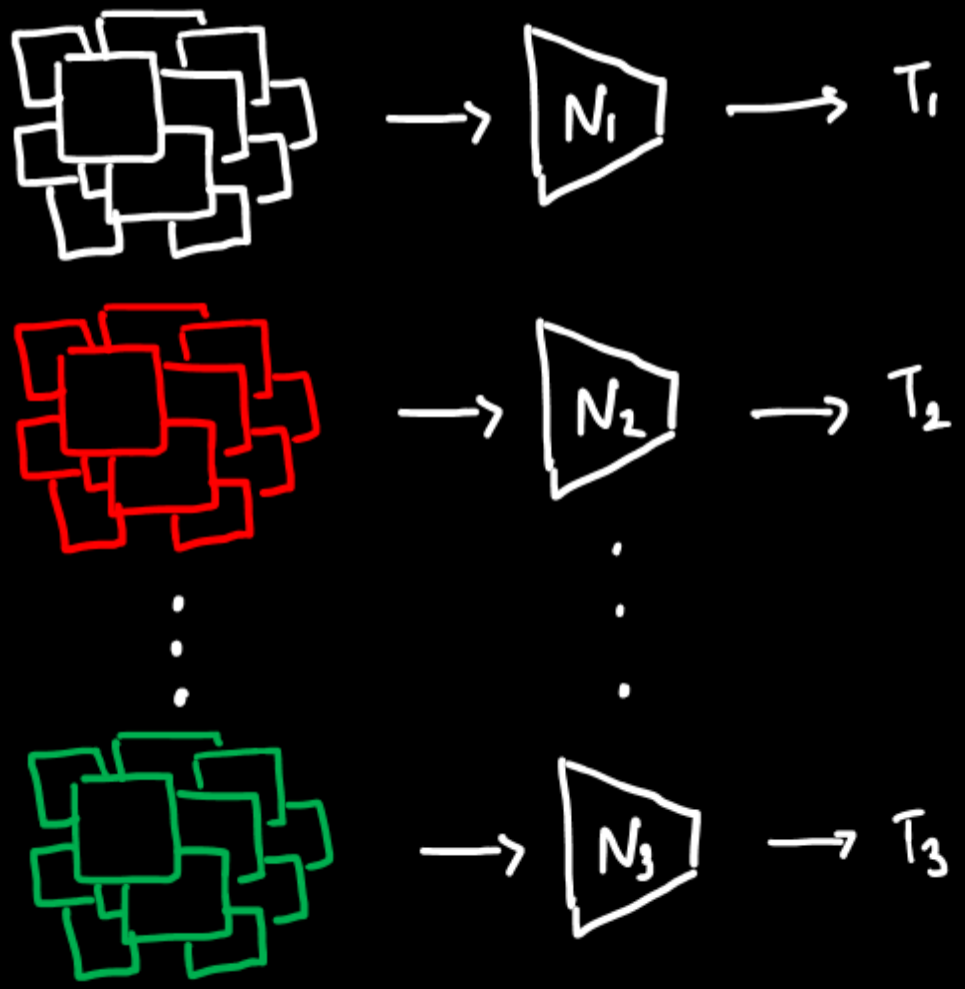
*Ref: Taskonomy – Know why segmentation networks don't work as well for Classification.

And why did we need to 'pre-train'?

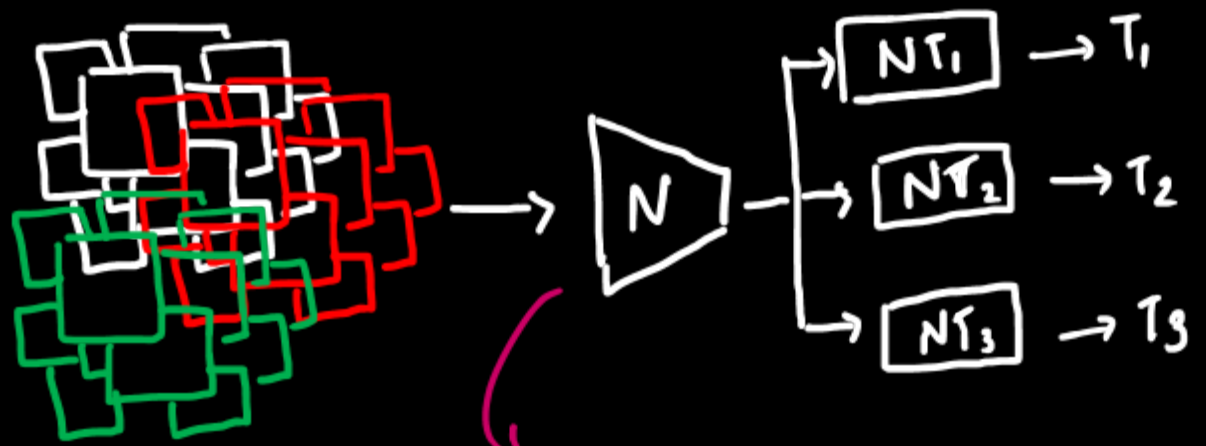
Practical Reason → Available Data for a few tasks ...

Not so obvious reasan → Developing "syntactical" ability

What else can we do?



Trained with a combined loss



Then this guy learns highly generic features

See UberNet (2017)

If CNNs are so Great...

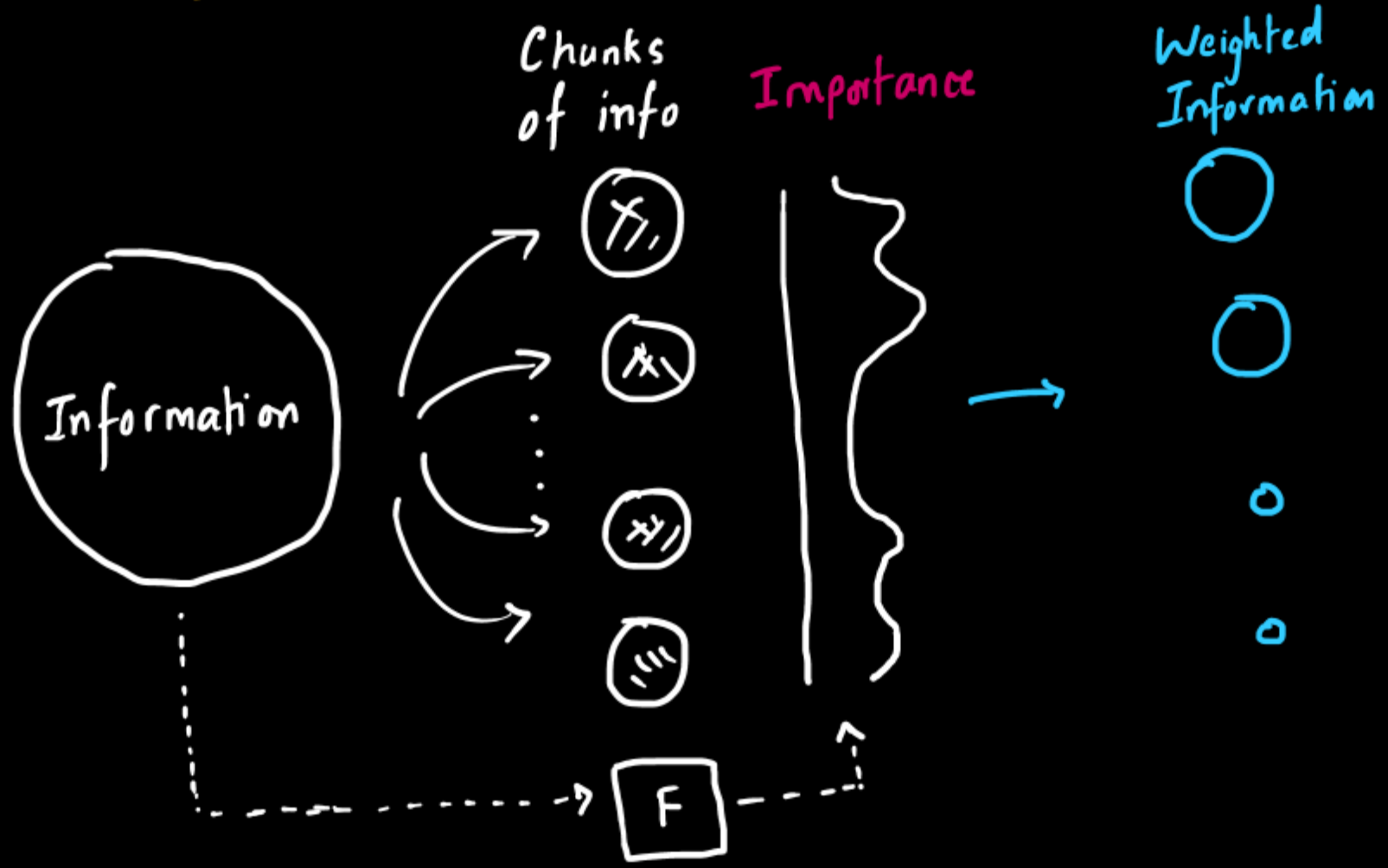
Why's everyone using Transformers ?

- Show, Attend, and Tell (SOTA in ~2016)
- Squeeze and Excite (SOTA in ~2019)

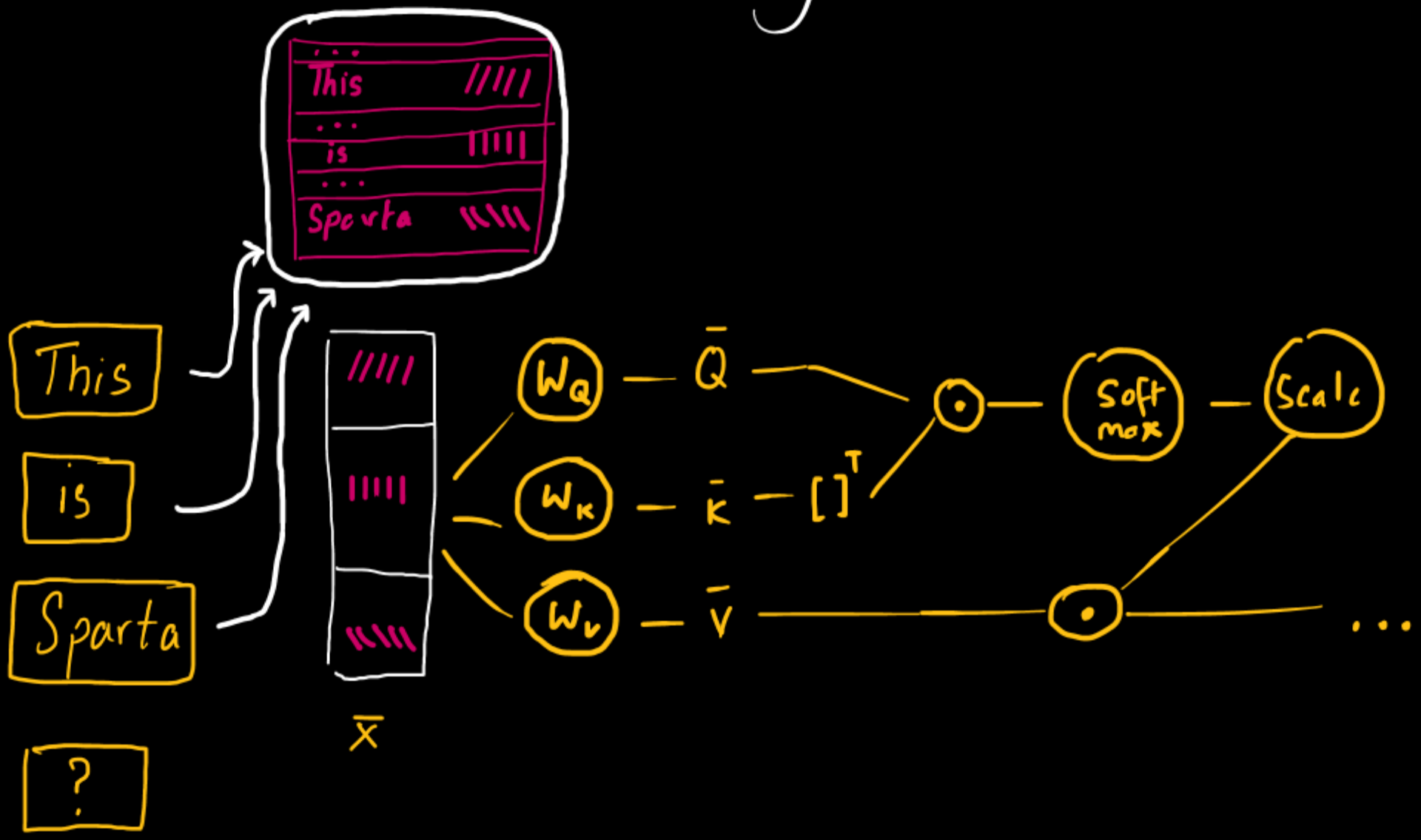
.
.
.

What's Common → Attention

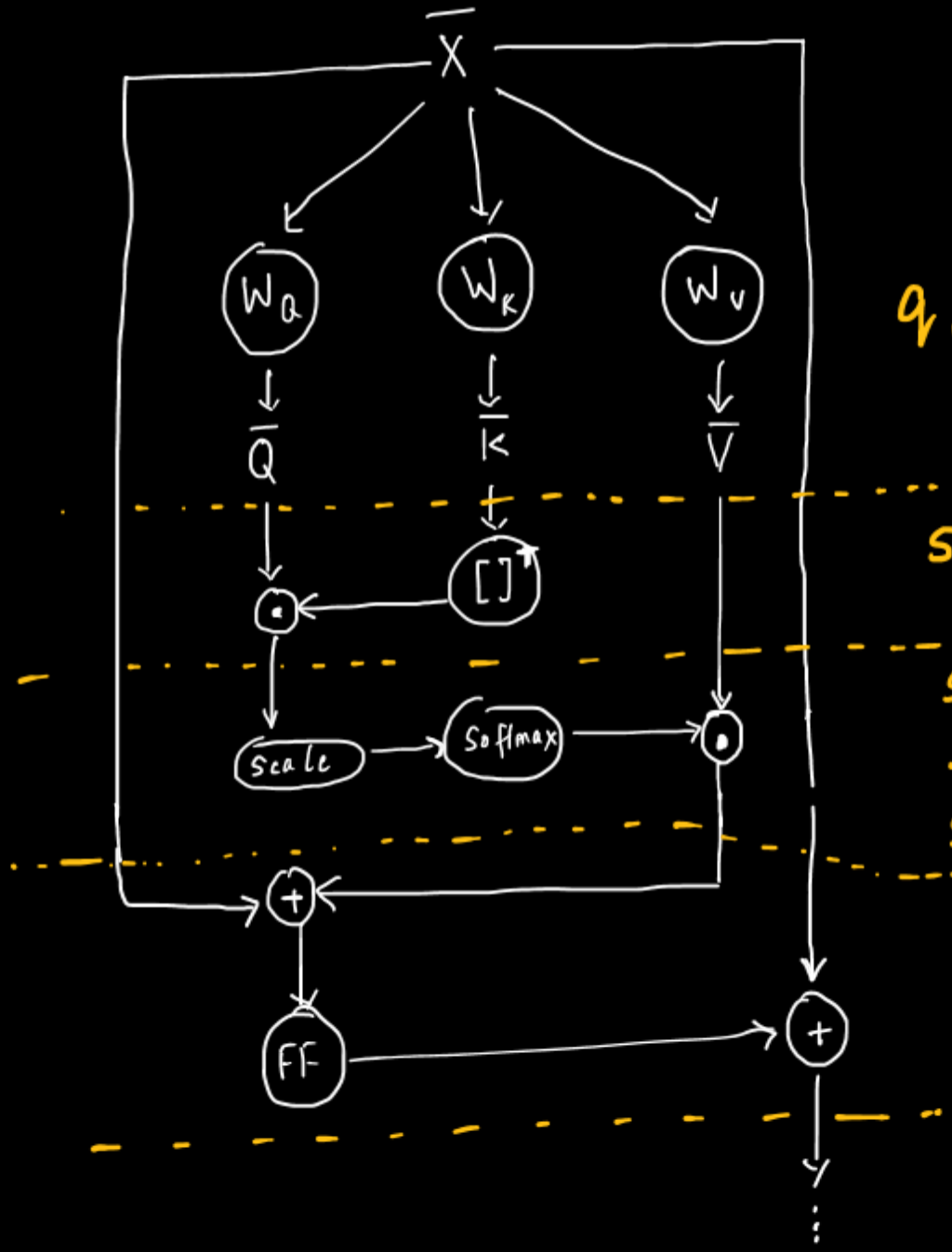
So what's Attention.



So when did this become a thing?



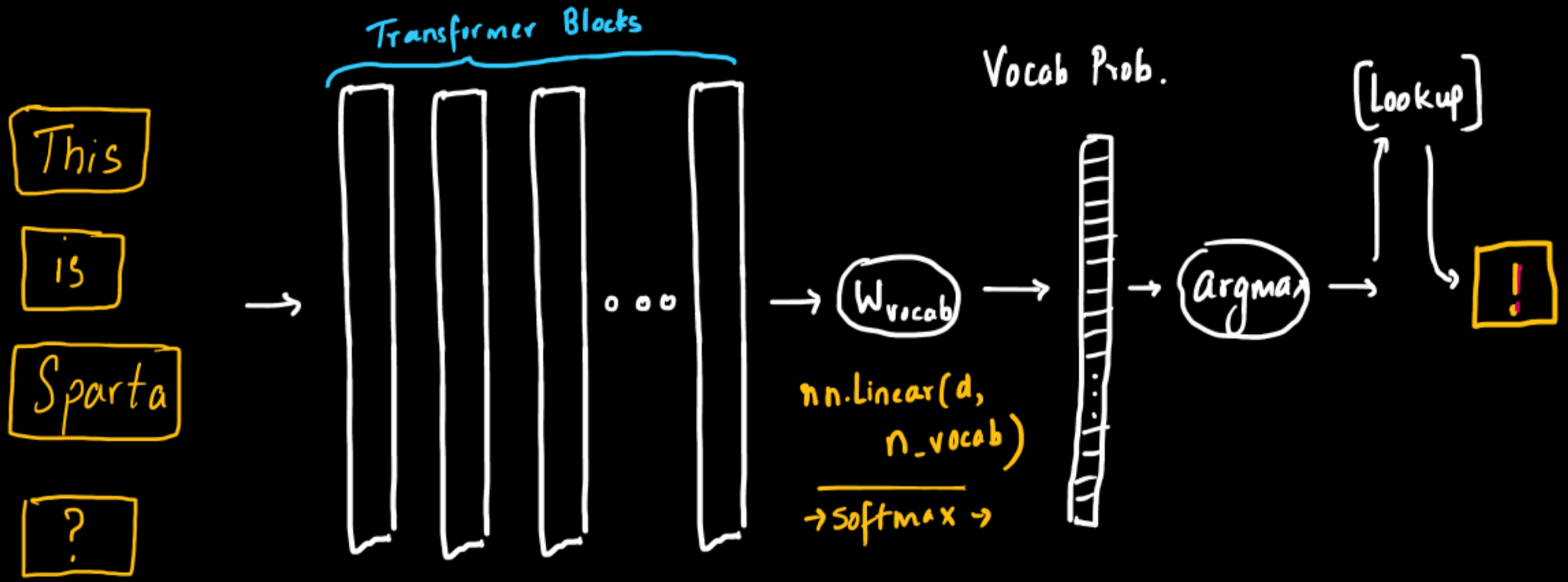
This is just the Attention bit!



This is one layer/block

```
q, k, v = nn.Linear(d, 3d)(x).split(-1)

s = q @ K.T

s = s / torch.math.sqrt(d)
s = torch.softmax(s)
s = s @ v
s = s + x
s = nn.linear(d, d)(s)   # actually multiple layers
s = s + x
```

So, to answer the question → What comes next...



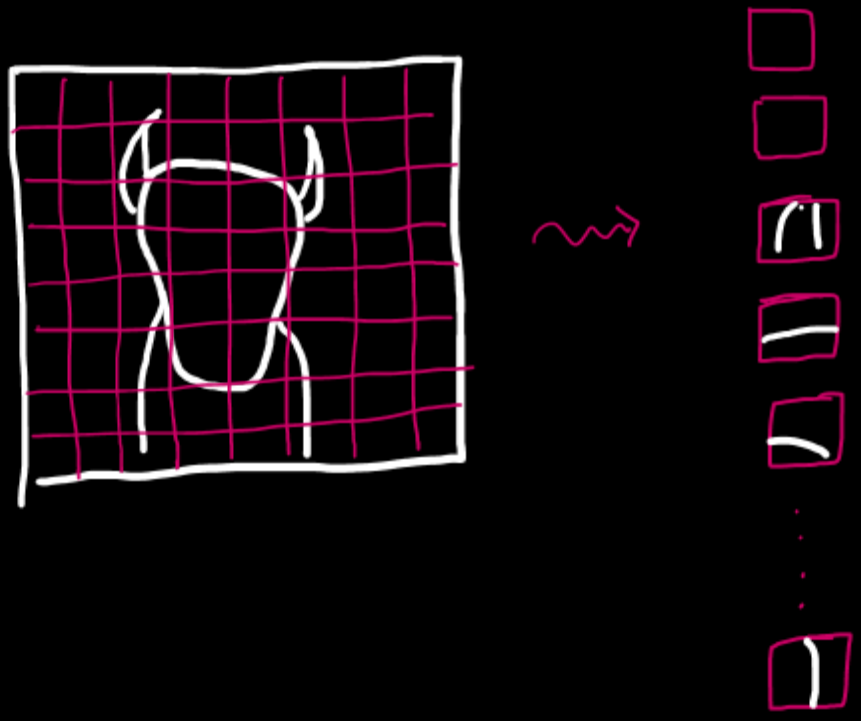
This is Sparta!

Transformers changed a lot of things

- → A race between Xformers and ConvNexts
- → Everyone agrees Attention is 'required.'
- → Efficient Xformers – the search is still on...
- → Vision folks said – let's go!...
- → ChatGPT became famous

Question - Images and Transformers!

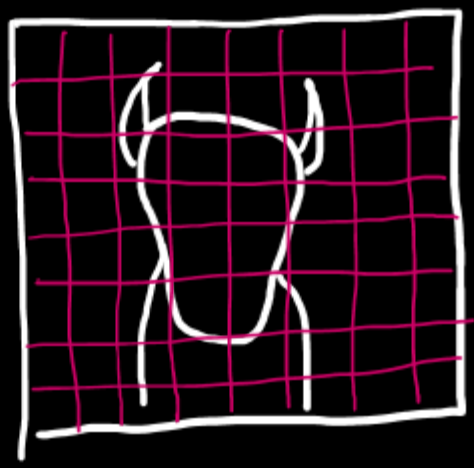
Simple! Break it up...



→ And the rest is the same... Almost

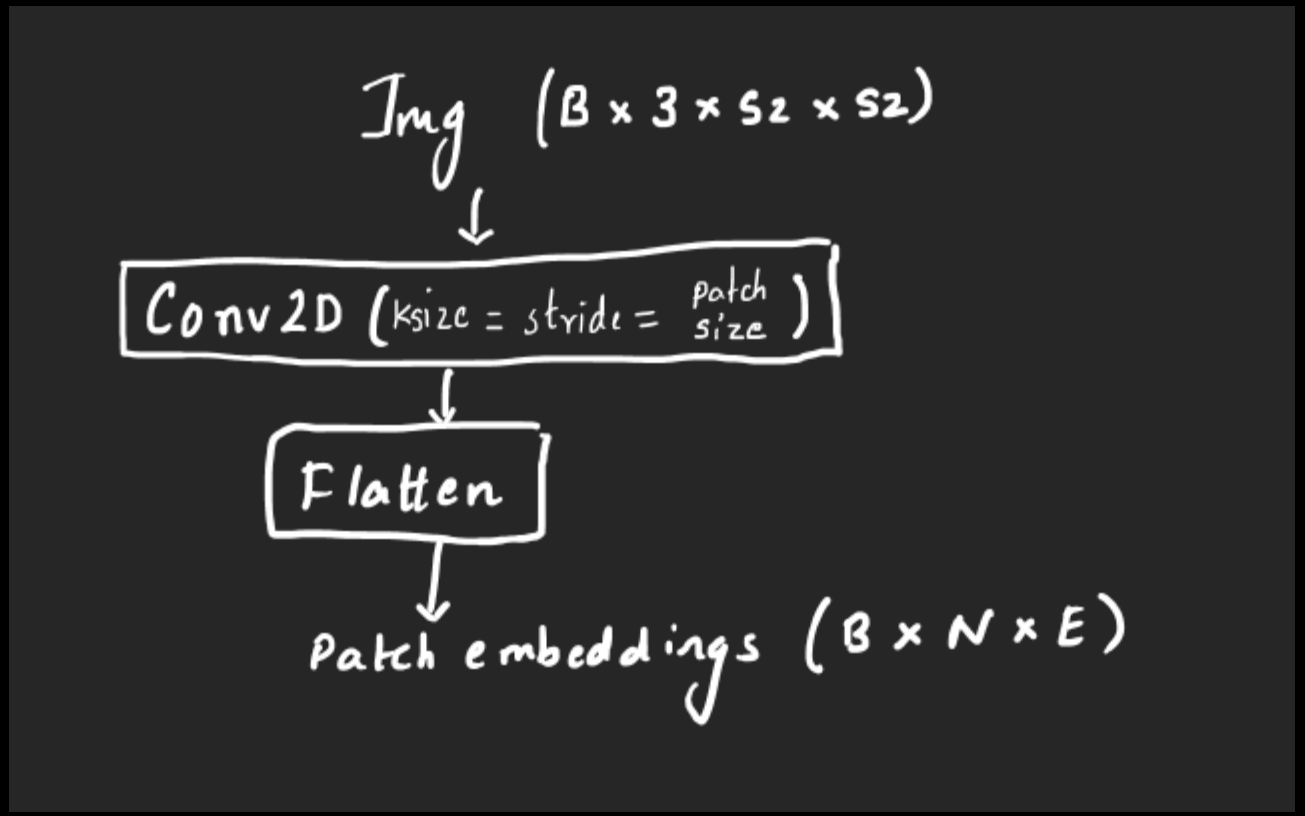
Question - Images and Transformers!

Simple! Break it up...



→ And the rest is the same... Almost

All in all a Vision Transformer



Here

- $N = \left[Sz \,//\, \text{patch size} \right]^2$
- E = Embedding dimensions

Time for Some Code