LECTURE 11
OBJECT RECOGNITION II

Punnarai Siricharoen, Ph.D.

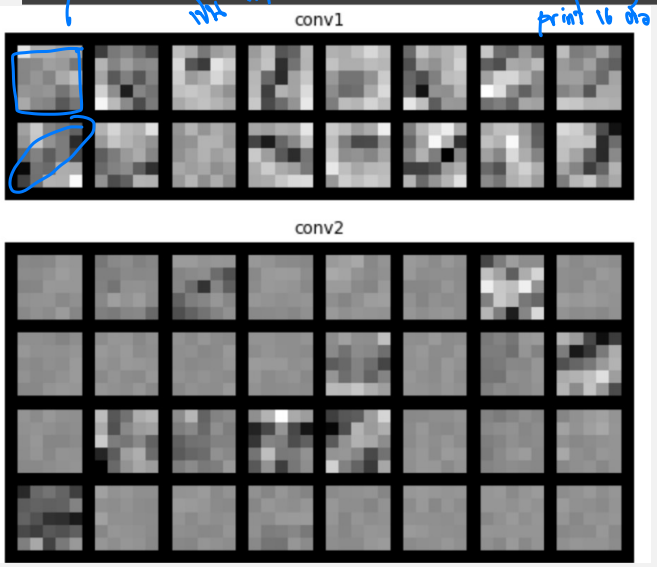
OBJECTIVES

- Learn how to improving your model by augmentation techniques
- Learn the evolution of deep learning and how can you reuse the pre-trained model.

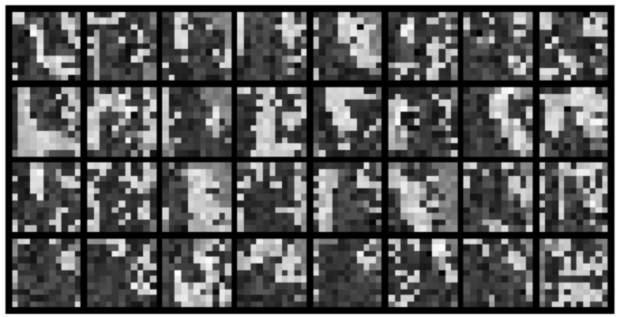
TODAY'S CONTENT

- Introduction
- Evaluation
- Recap - Image Transformation / Augmentation
- Visualizing filter and feature maps
- Evolution of Deep learning
- Transfer Learning

VISUALIZING FILTERS



Conv3 (if added)



```
Net(
  (conv1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (out): Linear(in_features=1568, out_features=10, bias=True)
)
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 28, 28]             416
              ReLU-2           [-1, 16, 28, 28]               0
         MaxPool2d-3           [-1, 16, 14, 14]               0
            Conv2d-4           [-1, 32, 14, 14]          12,832
              ReLU-5           [-1, 32, 14, 14]               0
         MaxPool2d-6             [-1, 32, 7, 7]               0
            Linear-7                   [-1, 10]          15,690
================================================================
Total params: 28,938
Trainable params: 28,938
Non-trainable params: 0
```

32 x 7 x 7

Try to change kernel size?

VISUALIZING FEATURE MAPS

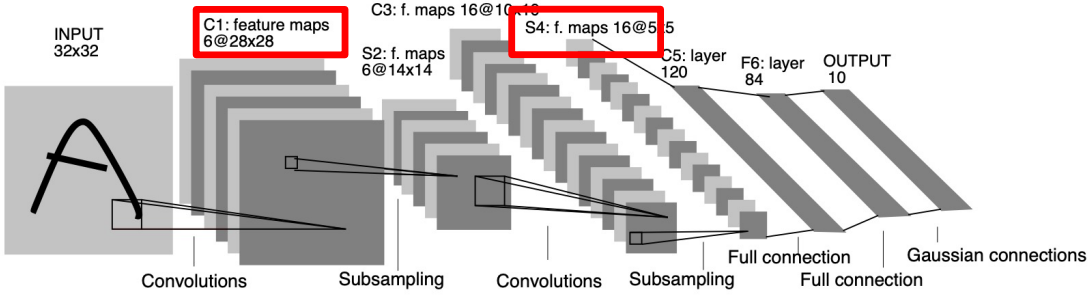
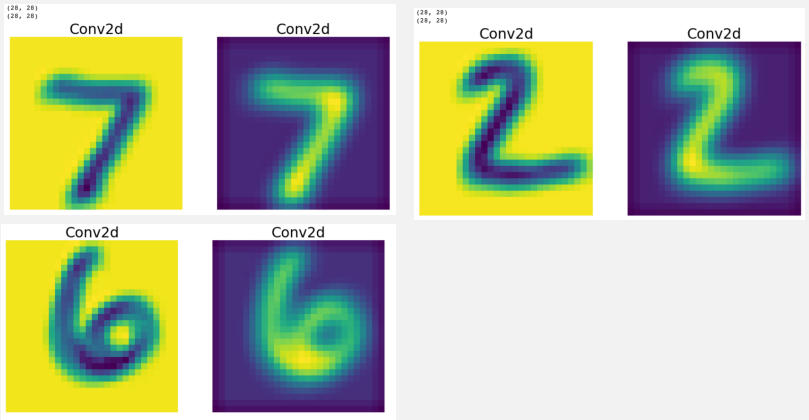
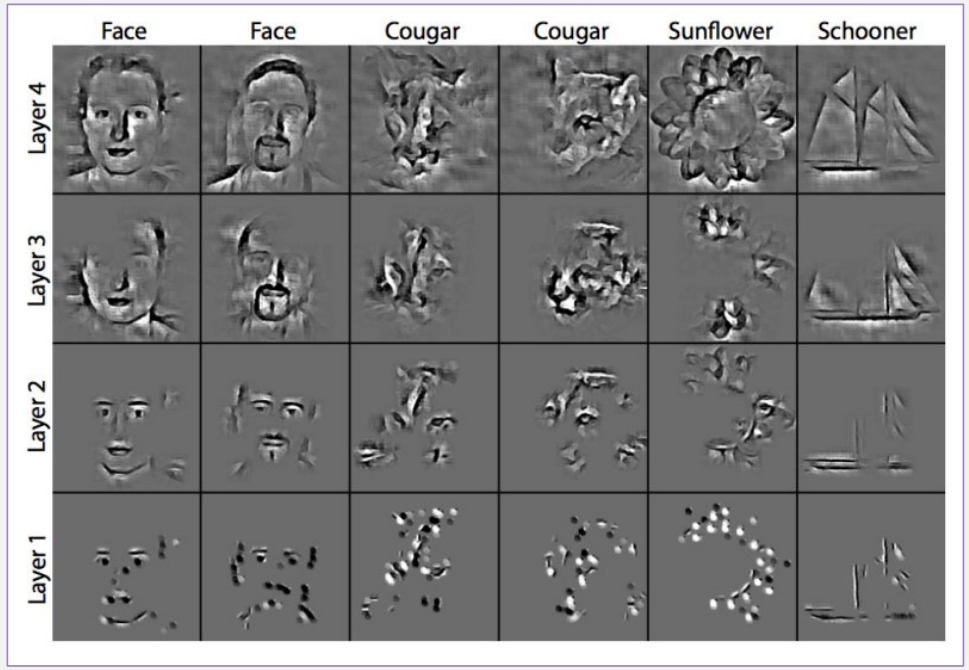


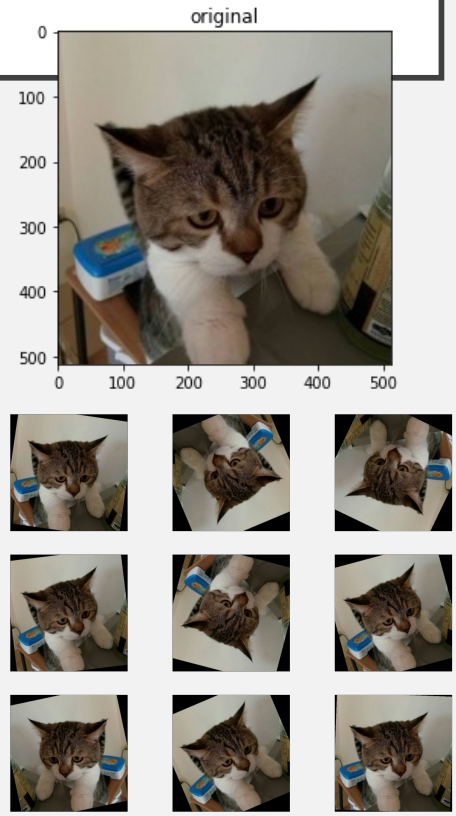
Fig. 1. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.





Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833). Springer International Publishing

TRANSFORMING AND AUGMENTING IMAGES



- **Pre-processing** – resize an image into the same size / convert an image into tensor type.
- **Image augmentation** - a technique that is used to artificially expand the dataset.
  - This is helpful when we are given a dataset with very few data samples.
  - In case of Deep Learning, this situation is bad as the model tends to over-fit when we train it on limited number of data samples.
  - To improve the accuracy of Image Classification models
  - Popular augmentation techniques - flip / scale / adding noise / rotation / crop / translation

เช่น CCTV มีน้ำเกาะหน้าเลนส์
แต่ train ภาพชัด หน้าเลนส์ add noise

https://pytorch.org/vision/stable/transforms.html

TRANSFORMING AND AUGMENTING IMAGES

- Augmentation / Trasnformation in pytorch
  - Accept both PIL images and tensor images, although some transformations are PIL-only and some are tensor-only.

```
from PIL import Image
import numpy as np
im = Image.open("hopper.jpg")
a = np.asarray(im)
```

`Resize`(size[, interpolation, max_size, ...])	Resize the input image to the given size.
`RandomRotation`(degrees[, interpolation, ...])	Rotate the image by angle.
`RandomVerticalFlip`([p])	Vertically flip the given image randomly with a g probability.

Conversion Transforms

`ToPILImage`([mode])	Convert a tensor or an ndarray to PIL Image.
`ToTensor`()	Convert a `PIL Image` or `numpy.ndarray` to tensor.
`PILToTensor`()	Convert a `PIL Image` to a tensor of the same type.

https://pytorch.org/vision/stable/transforms.html

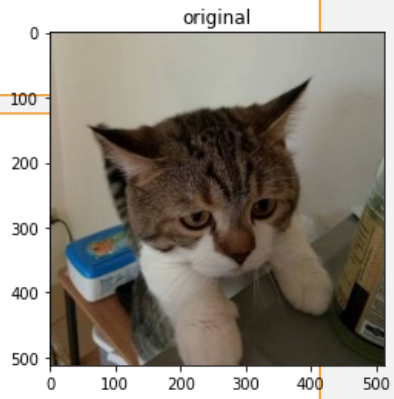
TRANSFORMING AND AUGMENTING IMAGES

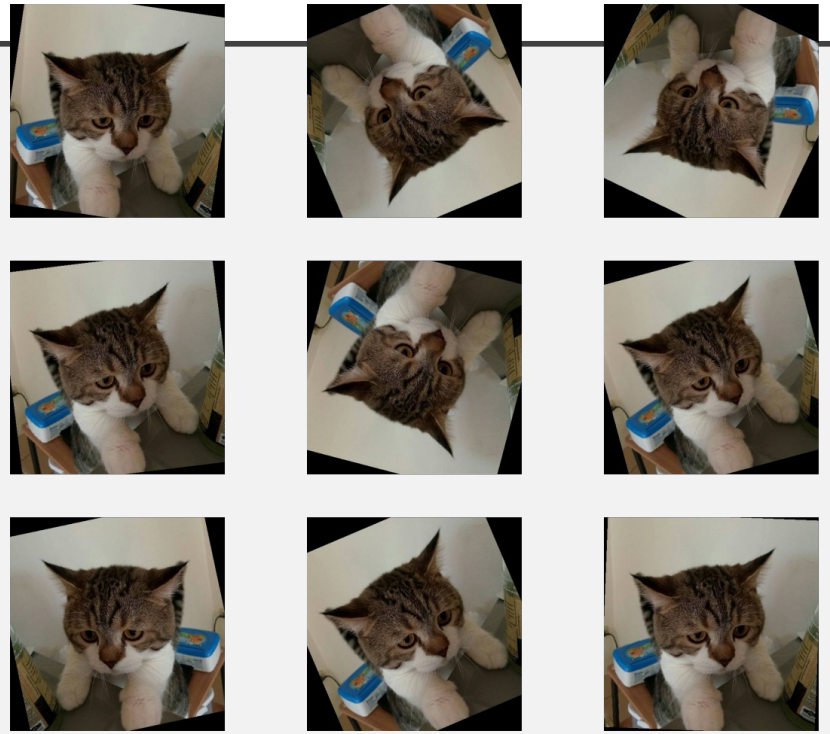
```
data_transform = transforms.Compose([
        transforms.RandomHorizontalFlip(0.3),
        transforms.RandomVerticalFlip(0.3),
        transforms.RandomRotation((-30,30)),
        transforms.ToTensor(),
])
```

```
from PIL import Image
import numpy as np
im = Image.open("kitty.jpg")

figure(figsize=(18, 16), dpi=80)
for i in range(1,10):
  img_tr=data_transform(im).numpy()
  plt.subplot(3,3,i)
  plt.imshow(img_tr.transpose(1,2,0));
  plt.axis('off')

Plt.show()
```





Use in the model

```
train_data = datasets.MNIST(
    root = 'data',
    train = True,
    transform = data_transform,
    download = True,
)
```

EXERCISE #1: AUGMENTATION WITH MNIST DATASET

- How are you going to augment the images for MNIST dataset?

EVOLUTION OF DEEP LEARNING

- The state-of-the-art **convolutional neural networks (CNNs or ConvNets)** is used for classification and regression.
- A pioneering 7-level convolutional network by LeCun et al in 1998, that classifies digits, was applied by several banks to recognize hand-written numbers on checks (cheques) digitized in 32x32 pixel greyscale input images.

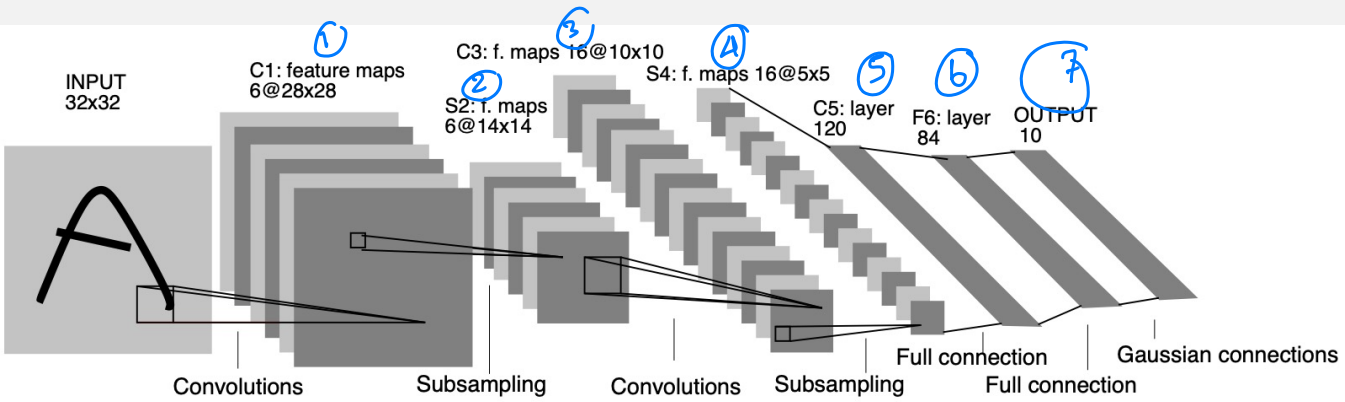
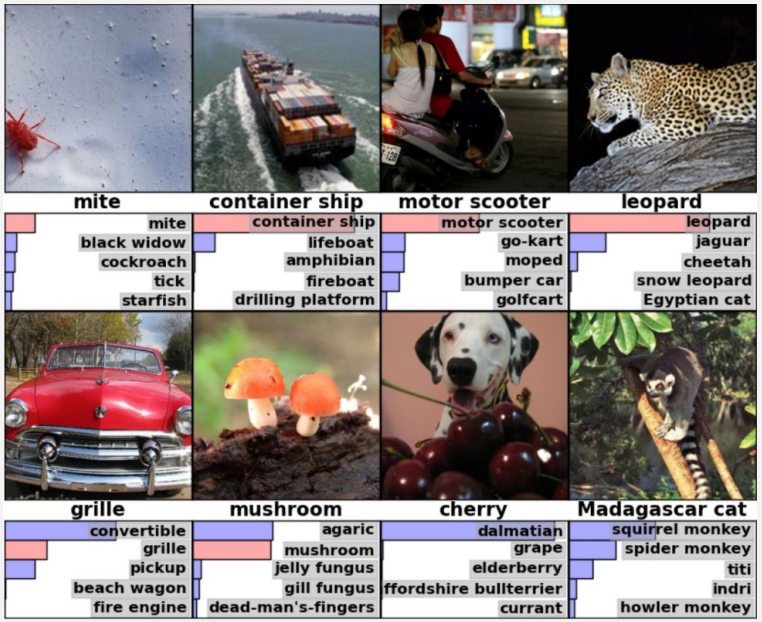


Fig. 1. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

1999 LeCun paper

EVOLUTION OF DEEP LEARNING

- **ImageNet** project is a large visual database for use in visual object recognition software research.
  - More than 14 million images have been hand-annotated by the project to indicate what objects are pictured
  - In at least one million of the images, bounding boxes are also provided.
  - ImageNet contains more than 20,000 categories
  - The ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) – popular from 2012-2017

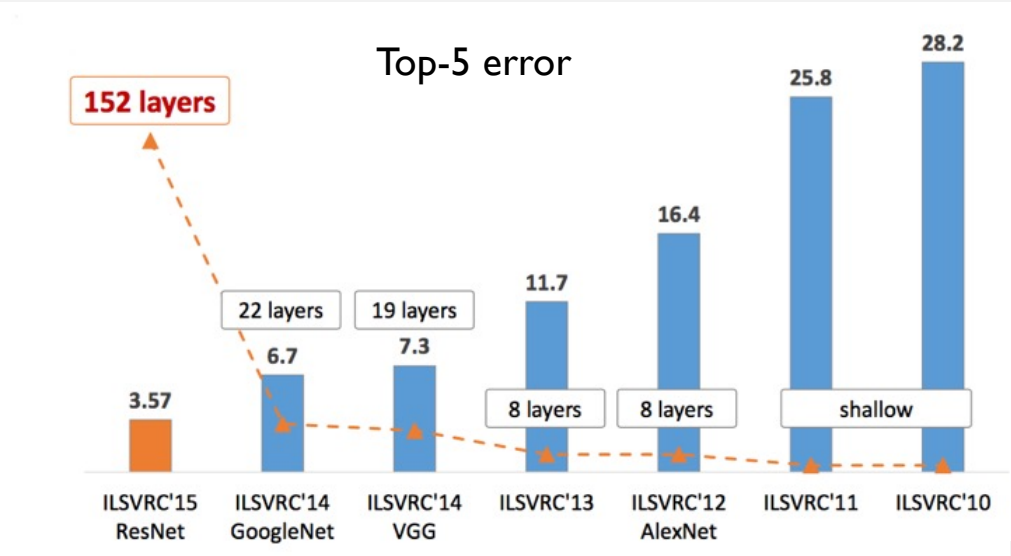


Challenge 2017

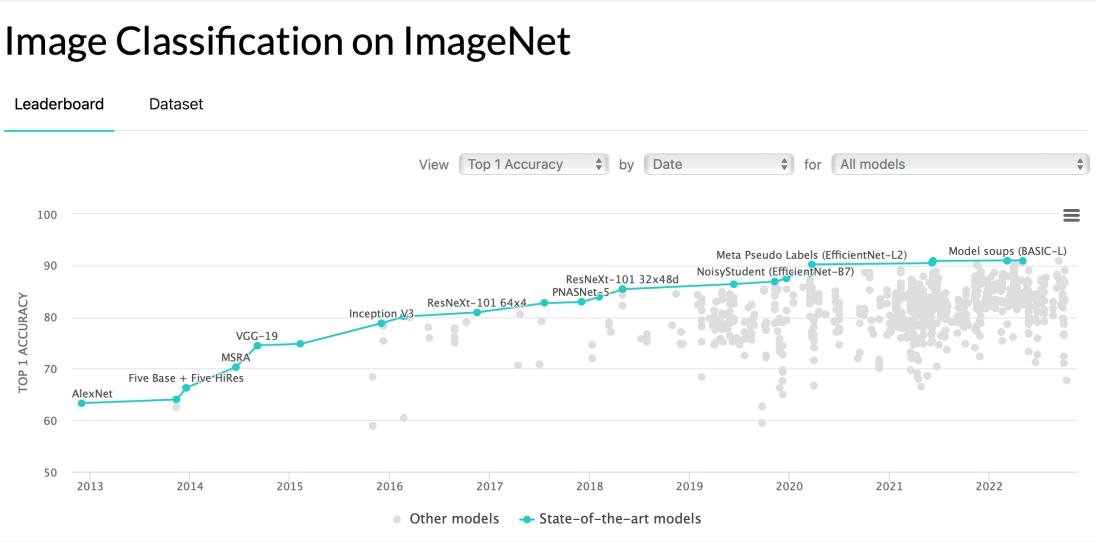
I. Object localization for 1000 categories.
II. Object detection for 200 fully labeled categories.
III. Object detection from video for 30 fully labeled categories.

Referecne: https://www.image-net.org

EVOLUTION OF DEEP LEARNING



ILSVRC

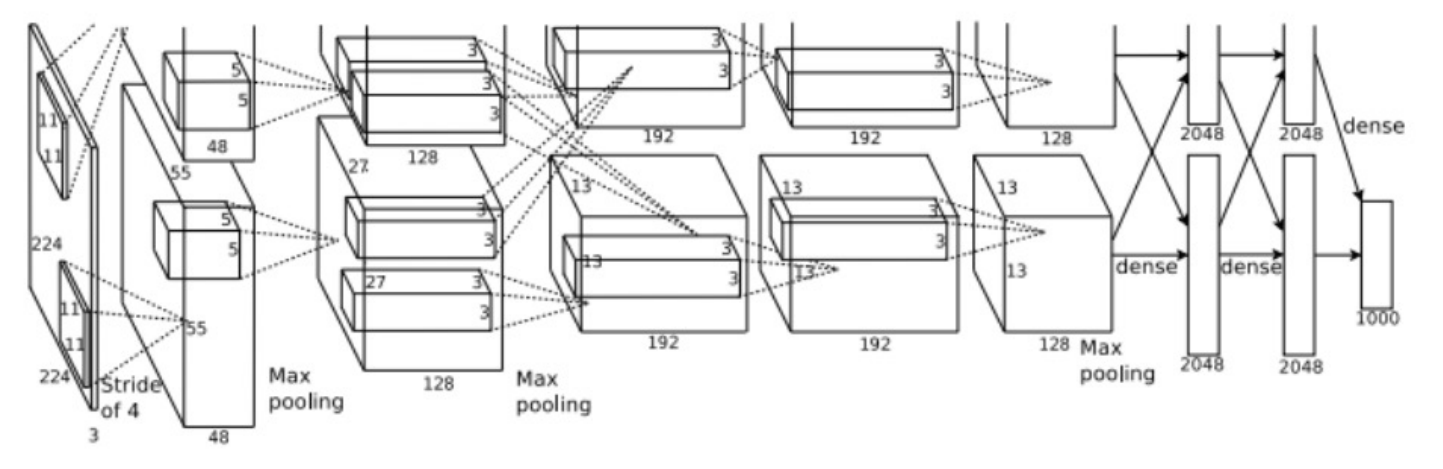


Paper with code

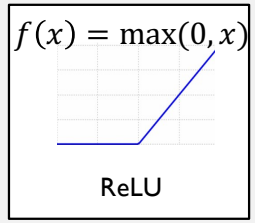
Referece - https://paperswithcode.com/sota/image-classification-on-imagenet

EVOLUTION OF DEEP LEARNING: ALEXNET (2012)

- In 2012, AlexNet, significantly outperformed all the prior competitors and won the challenge by reducing the top-5 error from 26% to 15.3%



Deeper than LeNet-5, with more filters per layer, and with stacked convolutional layers. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum.

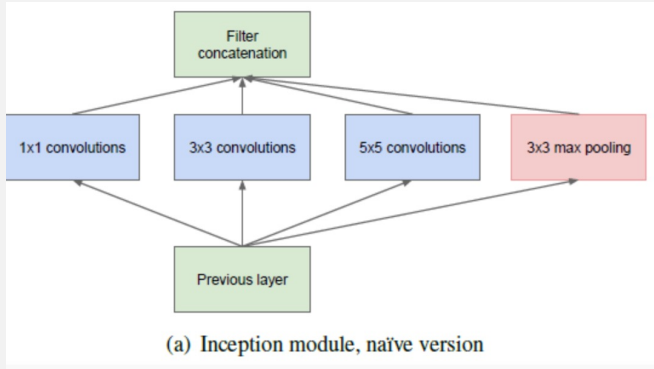


Quick to train!

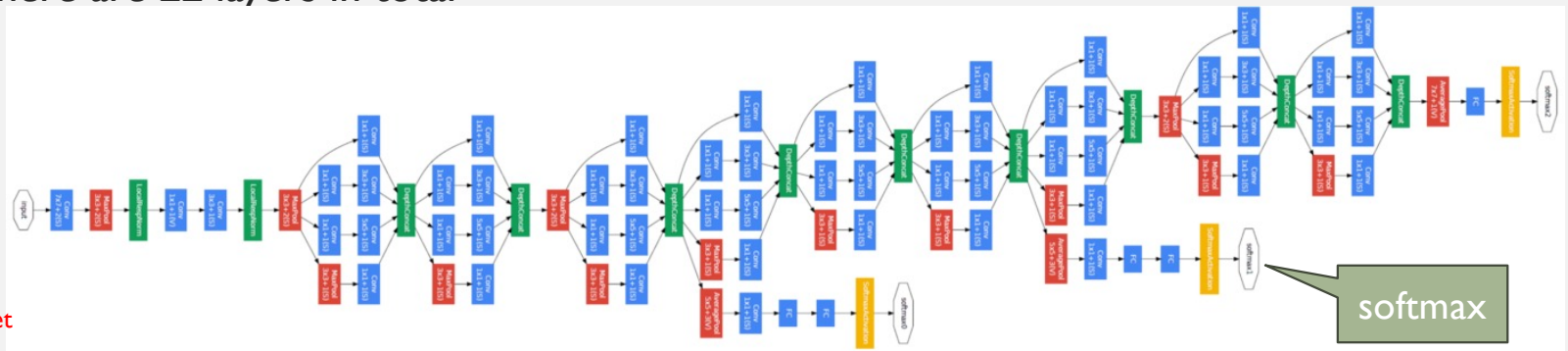
Alex net
https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

EVOLUTION OF DEEP LEARNING: GOOGLENET/INCEPTION(2014)

- **The winner of the ILSVRC 2014** competition was GoogleNet (a.k.a. Inception V1) from Google. It achieved a top-5 error rate of 6.67%!
- inception module, is to have different sizes/types of convolutions for the same input and stacking all the outputs.
- Batch normalization, image distortions and RMSprop
- **1×1 Convolution at the middle of the network** - reduced the number of parameters from 60 million (AlexNet) to 4 million - help to reduce model size which can also somehow help to reduce the overfitting problem!!
- There are 22 layers in total



(a) Inception module, naïve version



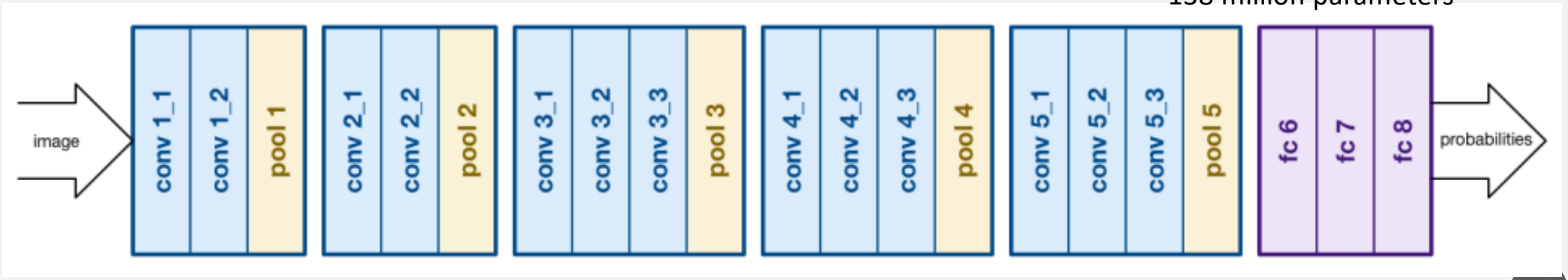
Convolution
Pooling
Softmax
Other

GoogleNet

EVOLUTION OF DEEP LEARNING: VGGNET (2014)

- The runner-up at the ILSVRC 2014 competition; developed by Simonyan and Zisserman (Oxford team)
- VGGNet consists of 16/19 convolutional layers and is very appealing because of its very uniform architecture.
- Similar to AlexNet, only 3x3 convolutions, but lots of filters. Trained on 4 GPUs for 2–3 weeks.

VGGNet consists of
138 million parameters



VGG

EVOLUTION OF DEEP LEARNING: RESNET (2015)

- Residual learning is adopt to every few stacked layers. The operation $\mathrm{F} + x$ is performed by a shortcut connection and element-wise addition.
- batch normalization (BN) is implemented right after each convolution and before activation.
- Such skip connections are also known as gated units or gated recurrent units and have a strong similarity to recent successful elements applied in RNNs.
- easier to optimize and achieves a top-5 error rate of 3.57% which beats human-level performance on this dataset.

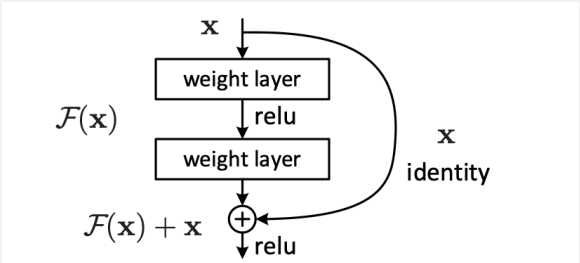


Figure 2. Residual learning: a building block.

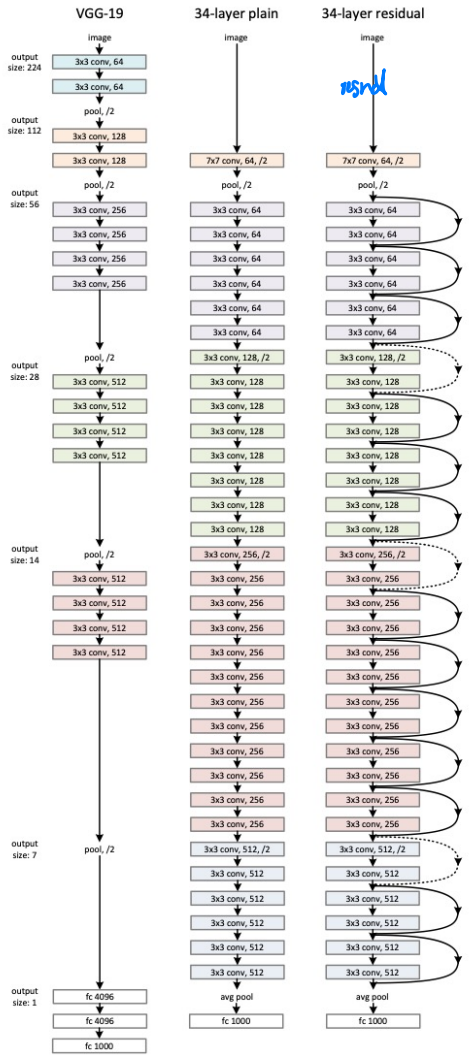


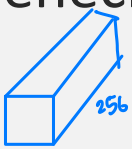
Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

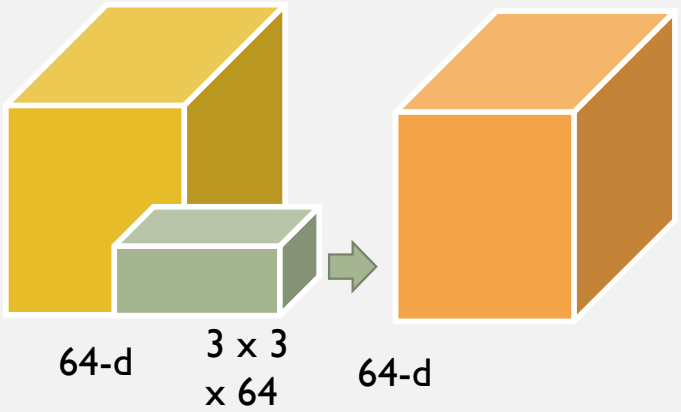
Resnet

actual feature คือ channel
ความหนาของ layer ด้านหน้า
จะ. param น้อยกว่า

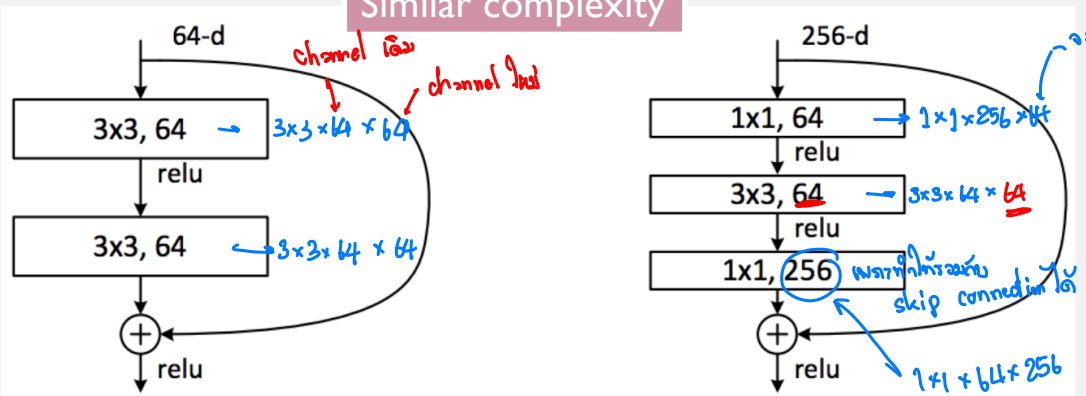
EVOLUTION OF DEEP LEARNING: RESNET (2015)

- A **BottleNeck** block is very similar to a BasicBlock in the figure below. Where the 1x1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3x3 layer a bottleneck with smaller input/output dimensions.





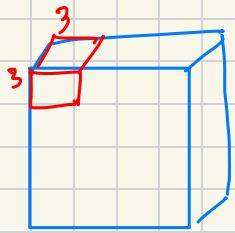
Similar complexity



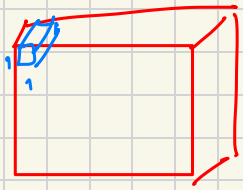
Convolution -> 64 x 3 x 3 x 64 x 2 = 73,728 params (excluding bias)

Convolution ->256 x 64 +64 x 3 x 3 x 64 + 64 x 256 = 69,632 params (excluding bias)

Figure 1: Basic block on the left. BottleNeck on the right.

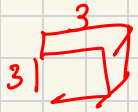


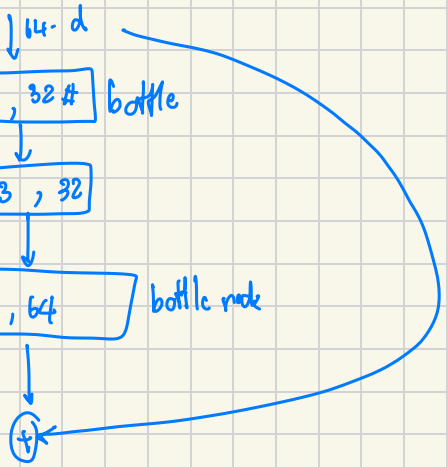
1 filter 3 x 3 x channel



filter 1 x 1 x C x 

เท่ากับจำนวน output





MCV #2

64 d

↓

3x3x64

↓

36 x64

64

↓

1x1x32 → 64x64, 4096

↓

3x3x32 → 3x3x32x64 = 18432

↓

1x1x64 → 1x1x64x64 = 4096

26 624

Input

↓ output

1x1x64x32

3x3x32x22

1x1x32x64

EVOLUTION OF DEEP LEARNING: RESNET (2015)

- Performance of the **Bottleneck**

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

a.u. Operation

2 in ~ ResNet 34

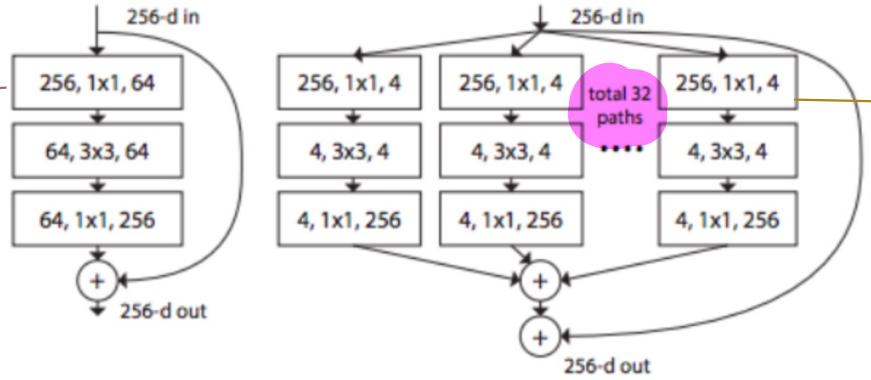
model	top-1 err.	top-5 err.
VGG-16 [40]	28.07	9.33
GoogLeNet [43]	-	9.15
PReLU-net [12]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validatio
VGG-16 is based on our test. ResNet-50/101/152 are of option
that only uses projections for increasing dimensions.

EVOLUTION OF DEEP LEARNING: RESNEXT (2017)

- **A combination of splitting, transforming, and aggregating.**
  - **Splitting**: the input is split into a few lower-dimensional embeddings
  - **Transforming**: the low-dimensional representation is transformed
  - **Aggregating**: the transformations in all embeddings are aggregated by summation.
- **Cardinality** is the size of the set of transformations. Refer to the following figure, the architecture includes 32 same topology blocks so the value of cardinality is 32. Because of using the same topology, fewer parameters are required while more layers are added into this architecture. **Increasing cardinality** lead to better result while keeping architecture complexity.
- Top 1 error = 20.4% and top 5 error = 5.3 %

Total params =
256 x 64
+ 64 x 3 x 3 x 64
+ 64 x 256
= 70,144 parameters
(excluding bias)



Total params =
32 x (256 x 4
+ 4 x 3 x 3 x 4
+ 4 x 256)
= 69,632 parameters
(excluding bias)

EVOLUTION OF DEEP LEARNING: RESNEXT (2017)

Total params =
256 x 64
+ 64 x 3 x 3 x 64
+ 64 x 256
= 70,144 parameters
(excluding bias)

983,040 parameters

3,932,160 parameters

stage	output	ResNet-50	**ResNeXt-50 (32×4d)**
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
conv2	56×56	3×3 max pool, stride 2	3×3 max pool, stride 2
		[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 128 3×3, 128, C=32 1×1, 256] ×3
conv3	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 256 3×3, 256, C=32 1×1, 512] ×4
conv4	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 512 3×3, 512, C=32 1×1, 1024] ×6
conv5	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 1024 3×3, 1024, C=32 1×1, 2048] ×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		**25.5**×10^6	**25.0**×10^6
FLOPs		**4.1**×10^9	**4.2**×10^9

Total params =
32 x (256 x 4
+ 4 x 3 x 3 x 4
+ 4 x 256)
= 69,632 parameters
(excluding bias)

614,400 parameters

3,440,640 parameters

EVOLUTION OF DEEP LEARNING: EFFICIENTNET (2020)

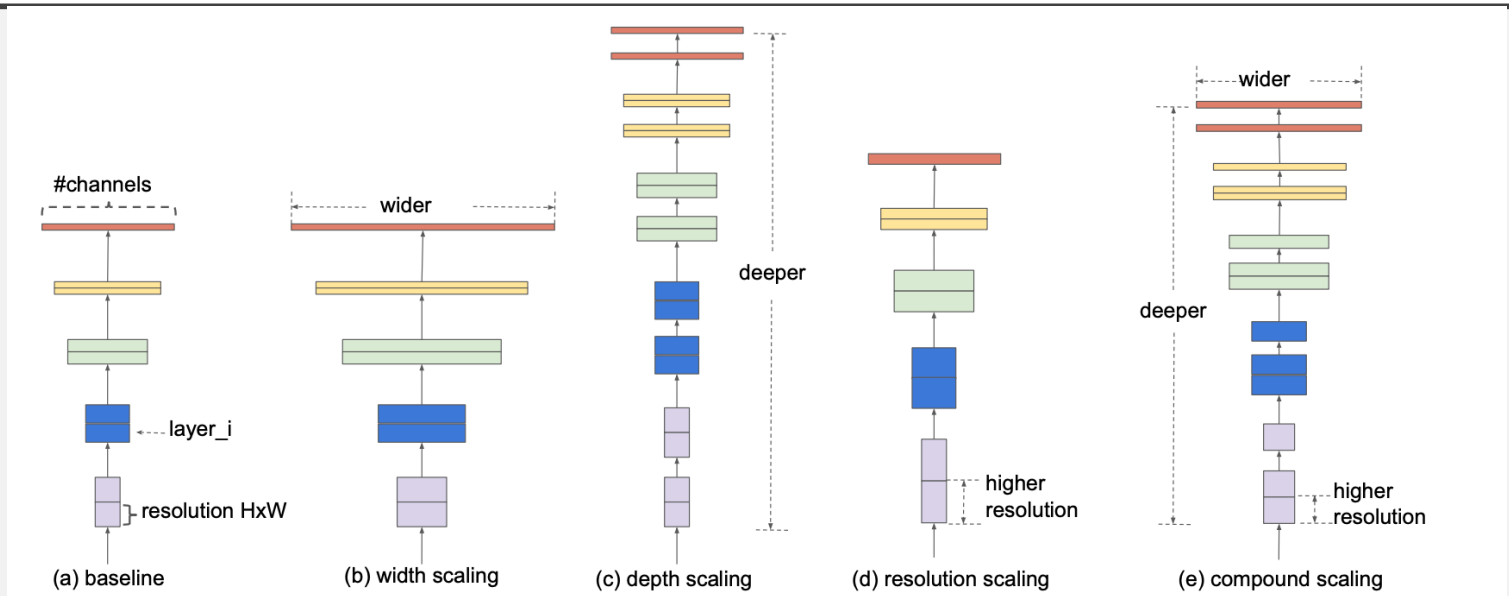


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

- A compound scaling method, which use a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way
- EfficientNet-B0 is the baseline scaled up to EfficientNet-B7

Table 8. **ImageNet Validation vs. Test Top-1/5 Accuracy.**

	B0	B1	B2	B3	B4	B5	B6	B7
Val top1	77.11	79.13	80.07	81.59	82.89	83.60	83.95	84.26
Test top1	77.23	79.17	80.16	81.72	82.94	83.69	84.04	84.33
Val top5	93.35	94.47	94.90	95.67	96.37	96.71	96.76	96.97
Test top5	93.45	94.43	94.98	95.70	96.27	96.64	96.86	96.94

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)

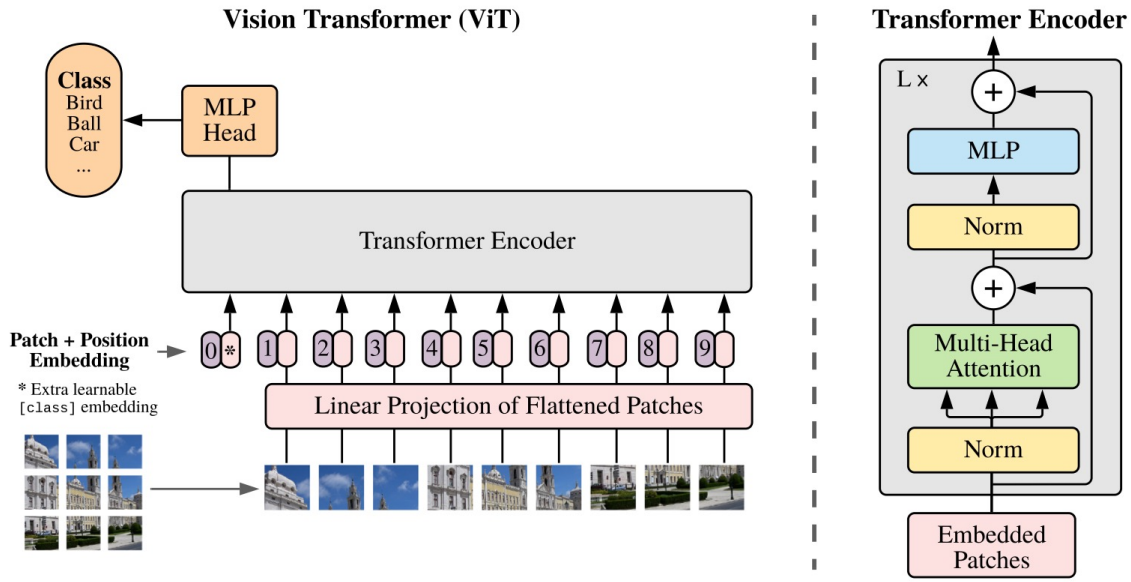
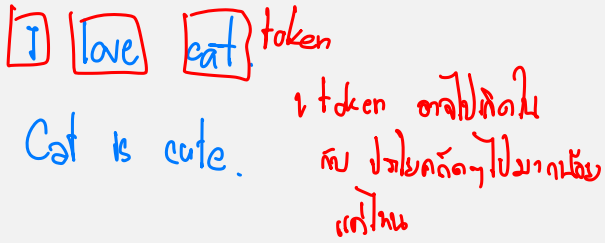
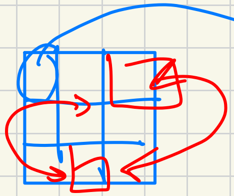


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

- For vision transformer, an input image is split into a set of image patches, called *visual tokens*.
- Position embeddings are added to the patch embeddings to retain positional information.

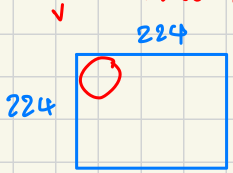


Attention is all you need, Vaswani A., et al. (2021)



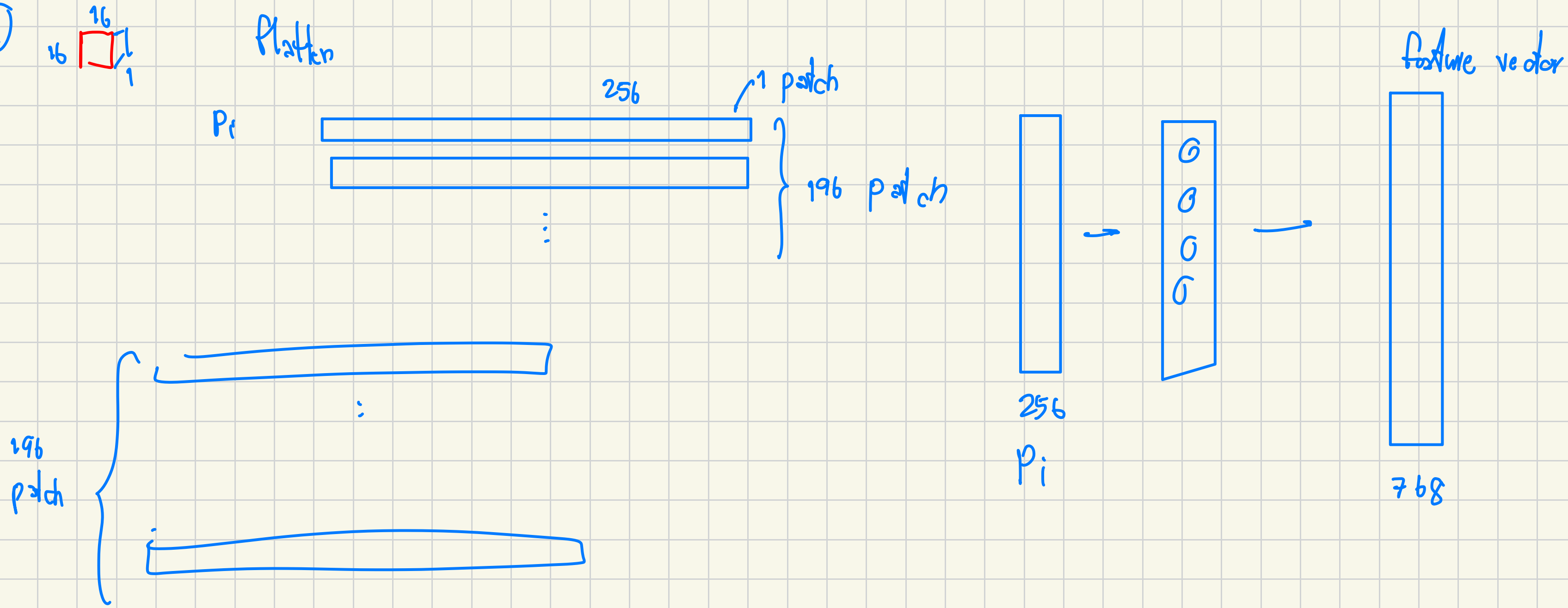
patch - token represent image (key Attention)

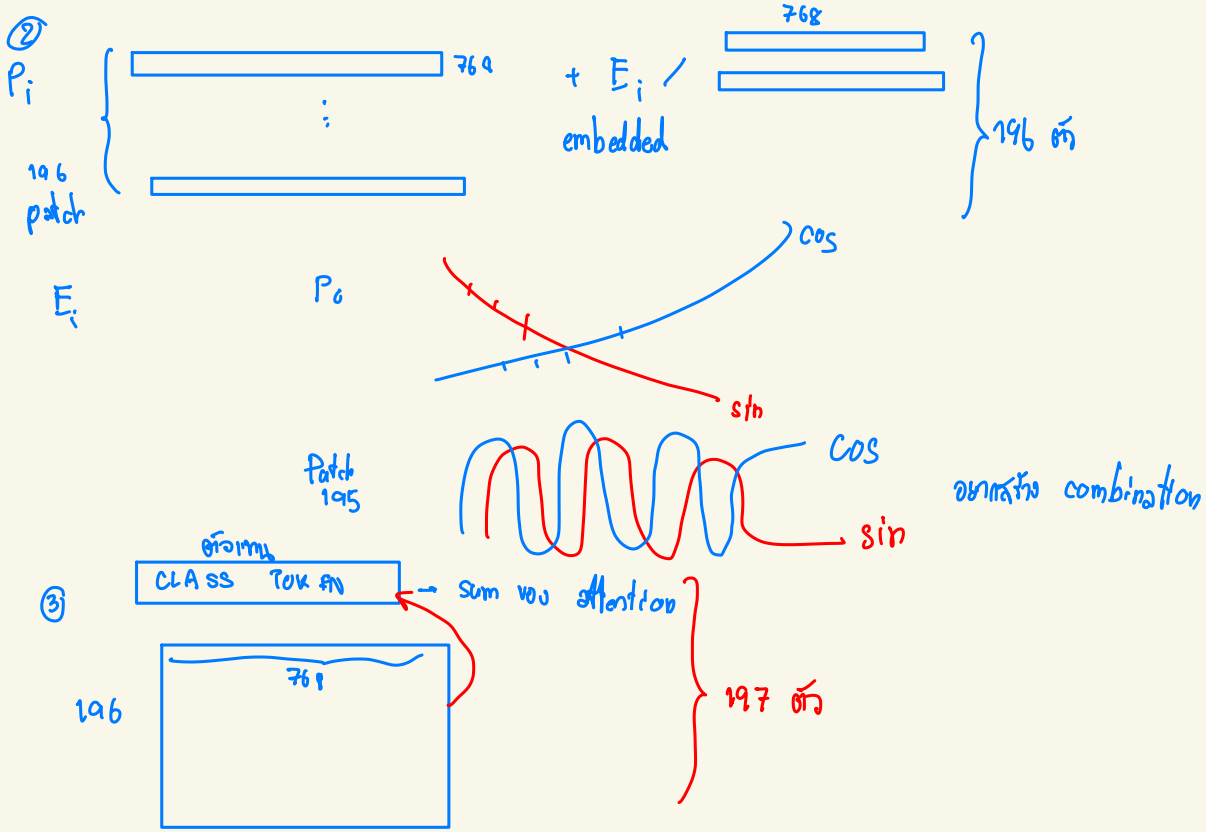
หูแมว - หนวดแมว Attention สูงเพราะมีโอกาสเกิดขึ้นพร้อมกัน



patch = 14 x 14 = 196

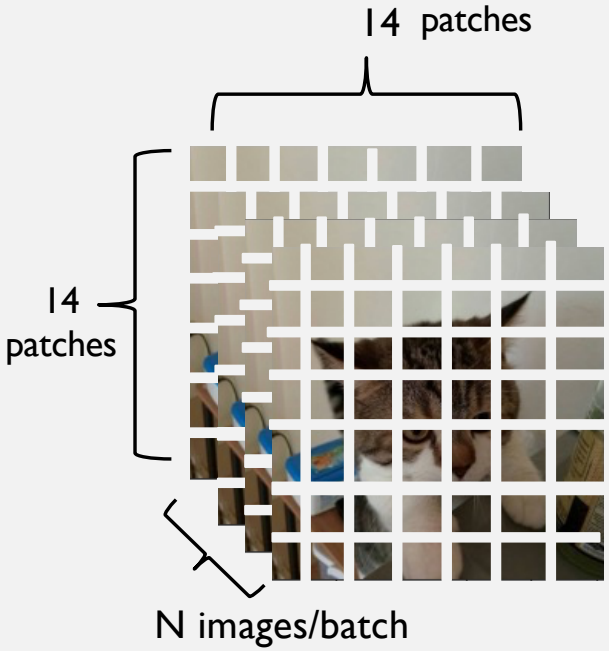
1 patch = 16 x 16 px

① 


②
P_i
768
196 patch
+ E_i /
embedded
768
196 ตัว
E_i
P_0
cos
sin
Patch 195
cos
sin
ผลลัพธ์ combination
ตัวแรก
CLASS Token
→ sum ของ attention
③
196
768
197 ตัว

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)

- Linear embedding of flattened patches and adding class token



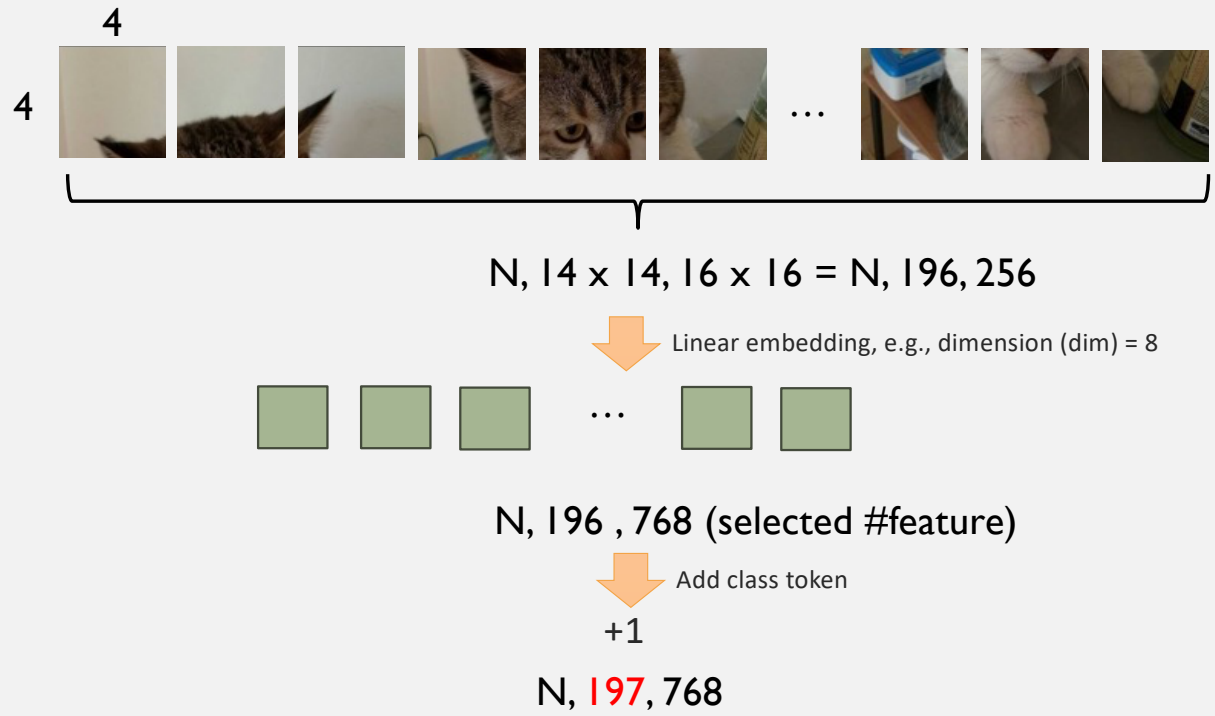


Image size : 224 x 224
Patch size : 16 x 16 px
#patch = 14 x 14 = 196 patches

Attention is all you need, Vaswani A., et al. (2021)

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)

- Positional embedding – model knows where each patched would be placed.

196, 768 (selected #feature)

196, 768

Positional Encoding

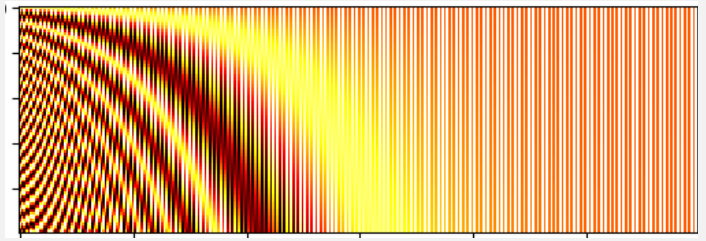
$$PE_{(pos,j)} = \begin{cases} \sin(pos/10000^{2*j/dim}), j\ even \\ \sin(pos/10000^{2*j-1/dim}), j\ odd \end{cases}$$

0, 196

196

บาง paper ใช้ cos

สำหรับคี่



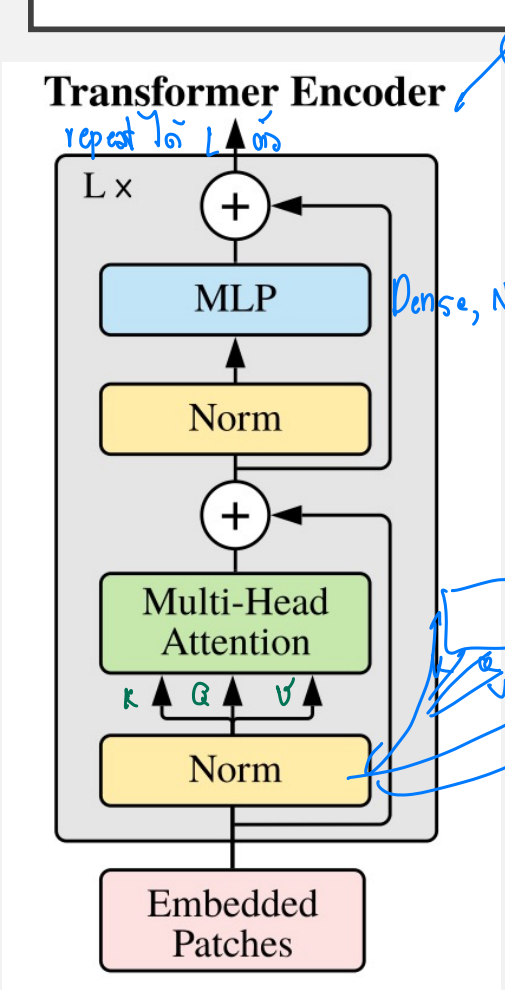
Pos - position (196)

Dimension (768)

e.g., our dim = 768, have 196 different positional encoding values for each sequence (patch). It would allow the model to easily learn to attend by relative positions.

Attention is all you need, Vaswani A., et al. (2021)

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)



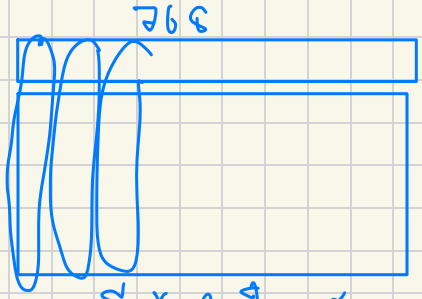
- **Transformer Encoder**
  - There are multiple blocks in the ViT encoder (L x)
  - Each block consists of three major processing elements:
    1. ***Layer Norm*** - Applies Layer Normalization over a mini-batch of inputs across all features and keeps the training process on track and let model adapt to the variations among the training images.

$$y = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta$$

The mean (μ) and standard-deviation (σ) are calculated over the last dimension. γ and β are learnable parameters.

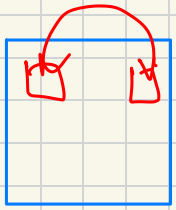
Attention is all you need, Vaswani A., et al. (2021)

④ LN

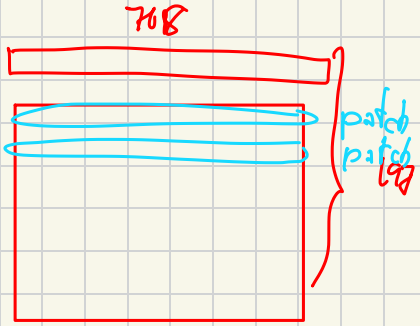


γ, β 768×2

มี γ, β เป็นของตัวเอง



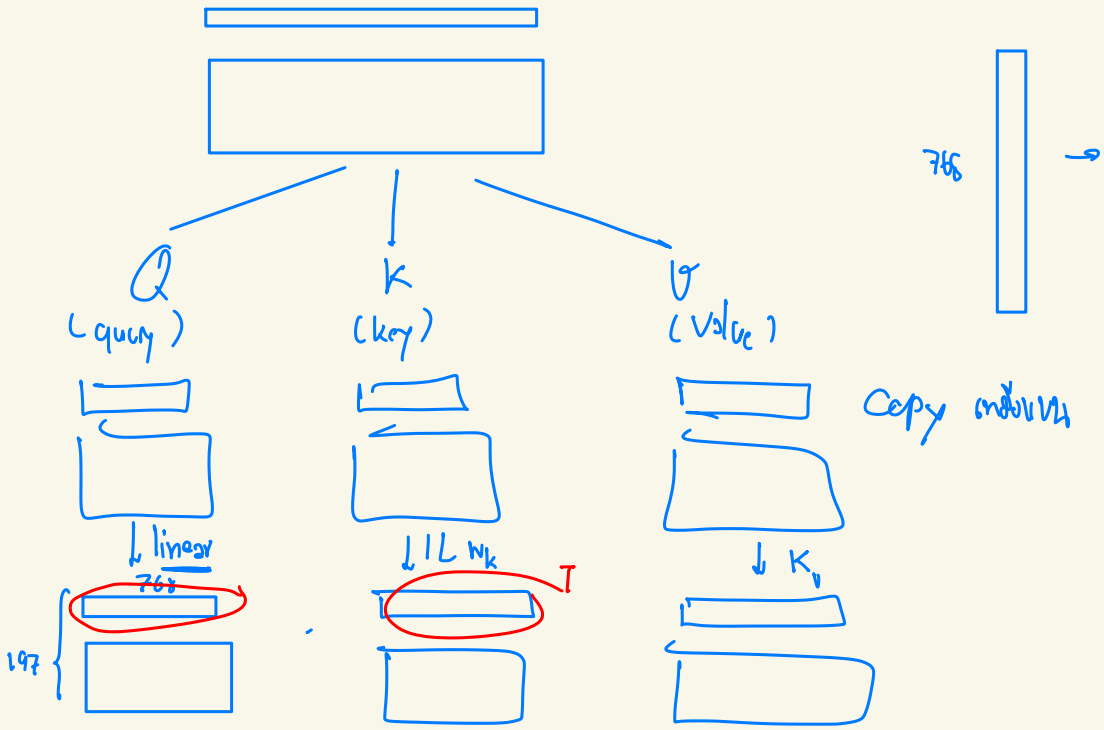
self-attention ของแต่ละ patch ในภาพเอง

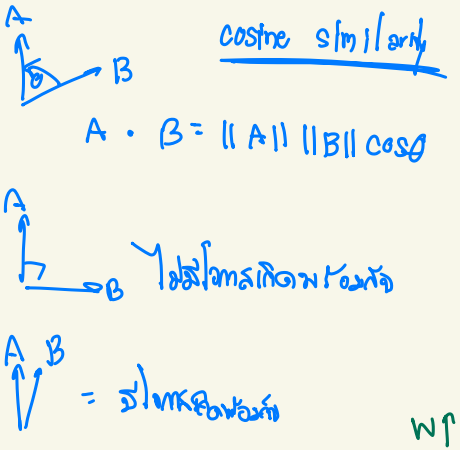


I love cat

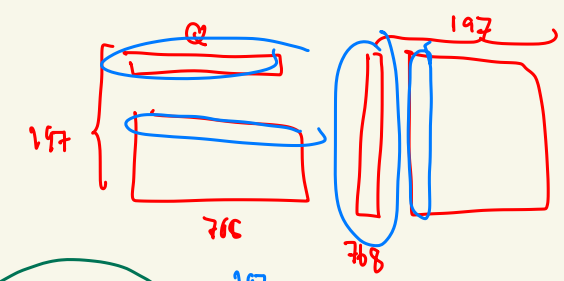
↳ ≠ self attention = cross attention

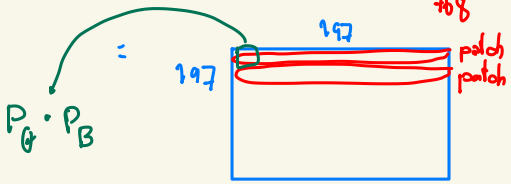
cat is cute

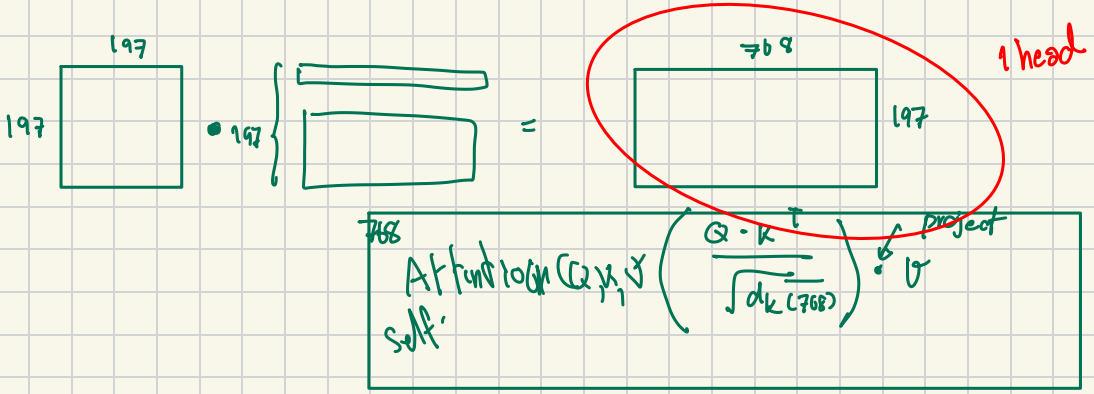


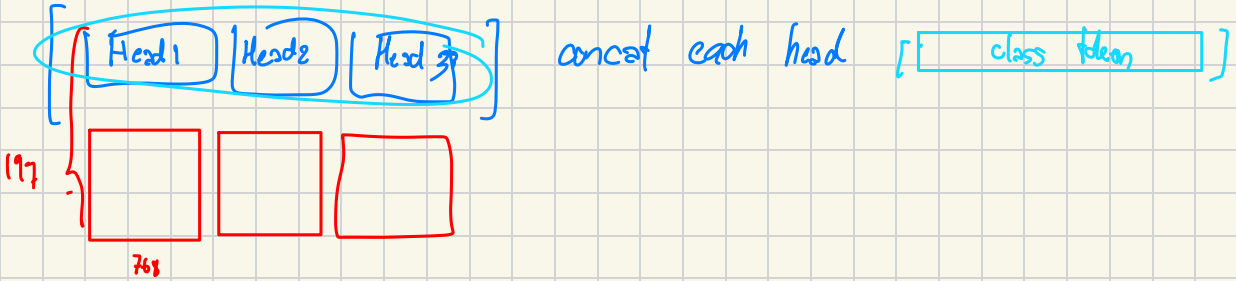


$$\text{Attention}(Q,K) = \frac{Q \cdot K^T}{\sqrt{d_k}}$$

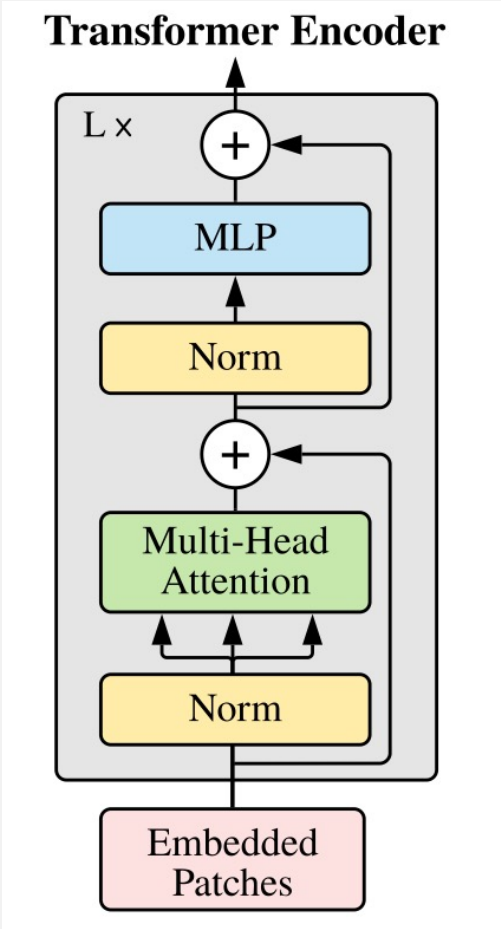




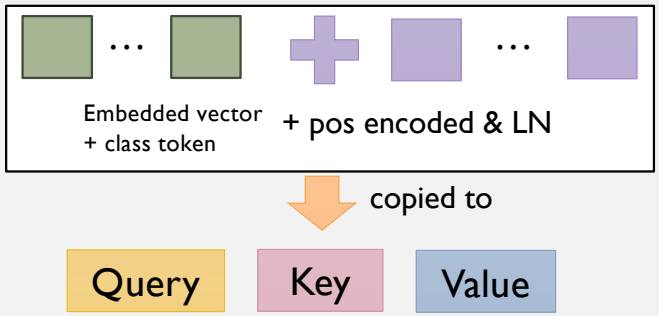

197
197
197
=
768
197
1 head
768
Attention(Q, K, V) = softmax(Q·K^T / √d_k(768)) · V
project

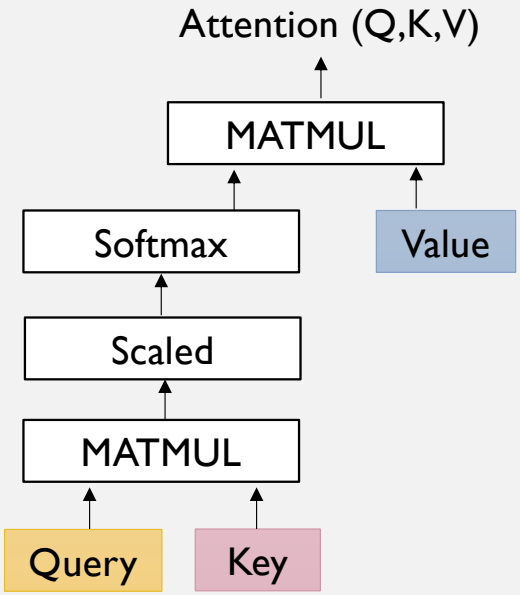

Head1
Head2
Head 3
concat each head
class token
197
768

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)



2. Multi-head Attention Network (MSA) - a network responsible for generation of attention maps from the given embedded visual tokens. These attention maps help network focus on most important regions in the image such as object(s).

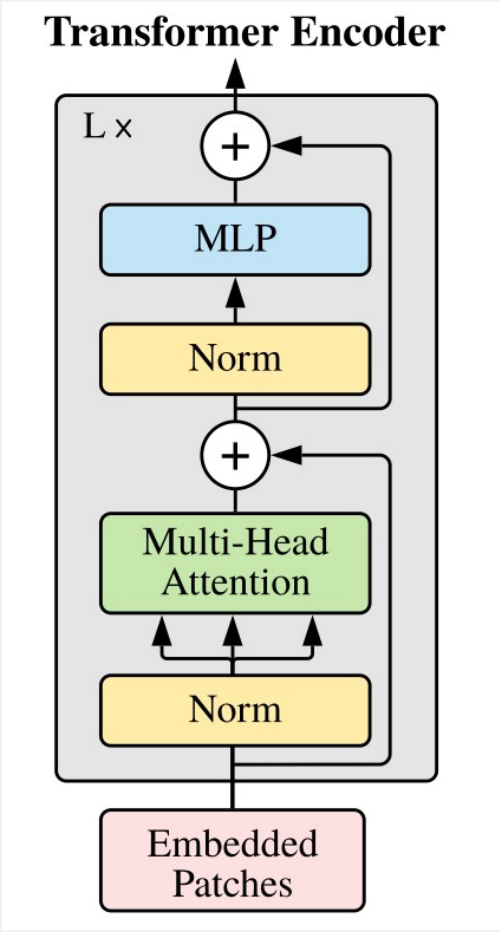




$$Attention\ (Q,K,V) = softmax(\frac{Q \cdot K^T)}{\sqrt{dim}} \cdot V$$

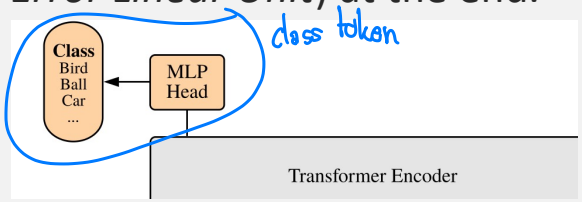
Attention is all you need, Vaswani A., et al. (2021)

EVOLUTION OF DEEP LEARNING: VISION TRANSFORMER (2021)



3. ***Residual Connection*** – add the original input to the result of some computation

- ***Multi-Layer Perceptrons (MLP)***
  - Encoder blocks– composed of A two-layer classification network with GELU (*Gaussian Error Linear Unit*) at the end.



  - Classification MLP block, also termed as *MLP head* (only the class token) is used as an output of the transformer. An application of *softmax* on this output can provide classification labels (i.e. if the application is Image Classification).

N, 768 x nHead → MLP → N, 10 for 10-class

Original use of ViT:

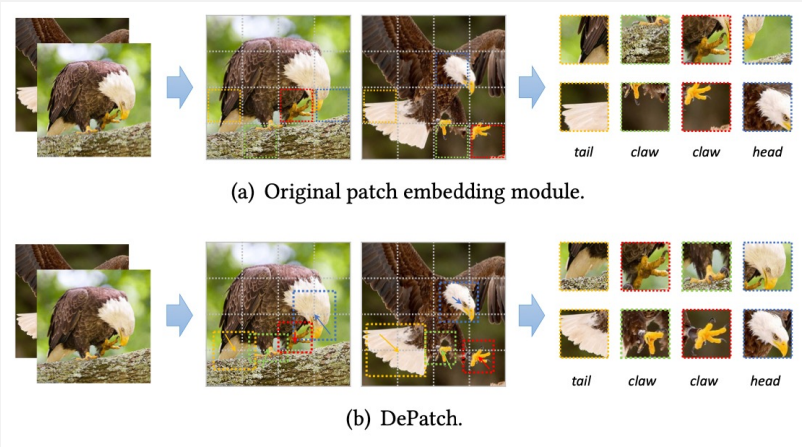
- https://github.com/lucidrains/vit-pytorch?tab=readme-ov-file#vision-transformer---pytorch

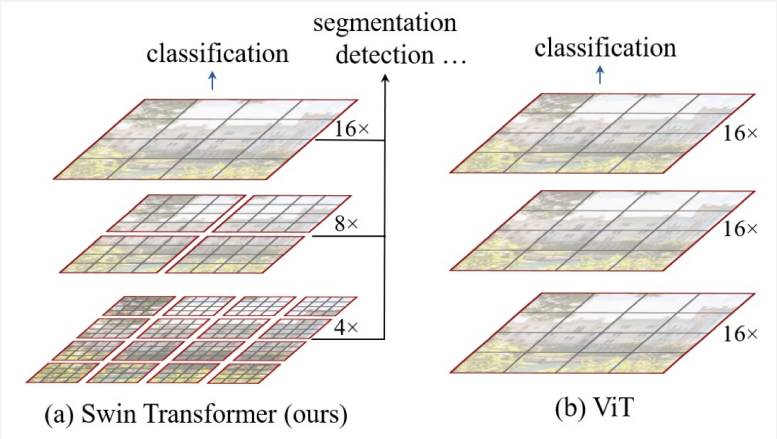
ViT from scratch

- Simplified version https://github.com/tintn/vision-transformer-from-scratch
  - https://colab.research.google.com/drive/1FrOmq3G-xLYkc08sYV43xBY6ZOOF-LaA
- https://medium.com/@brianpulfer/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c

VARIANTS OF VITS

- **Deformable Patch-based Transformer (DPT)** – instead of using fixed-size patch embedding, DPT learns to adaptively split the images into patches with different positions and scales.
- **Swin Transformer: Hierarchical Vision Transformer using Shifted Windows - t**he shifted windowing scheme brings greater efficiency by limiting self-attention computation to non-overlapping local windows while also allowing for **cross-window connection**.

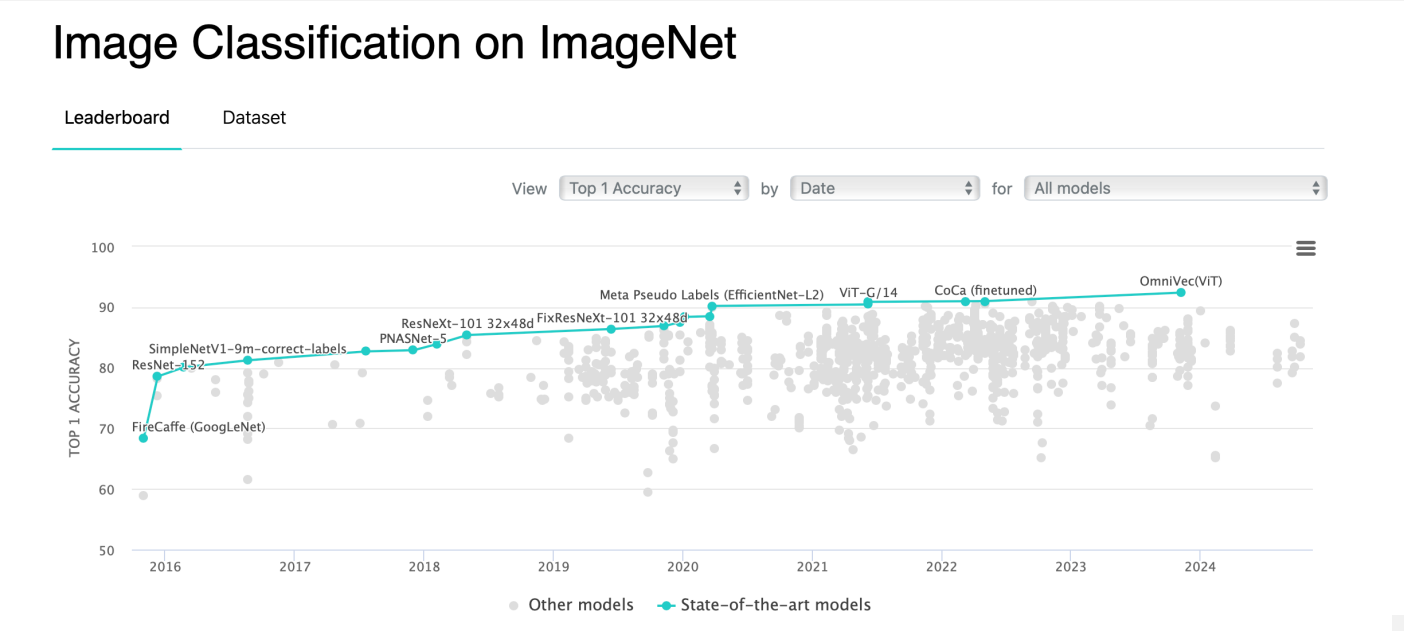




Chen Z., et al. (2021) DPT: Deformable Patch-based Transformer for Visual Recognition
Liu Z., et al. (2021) Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

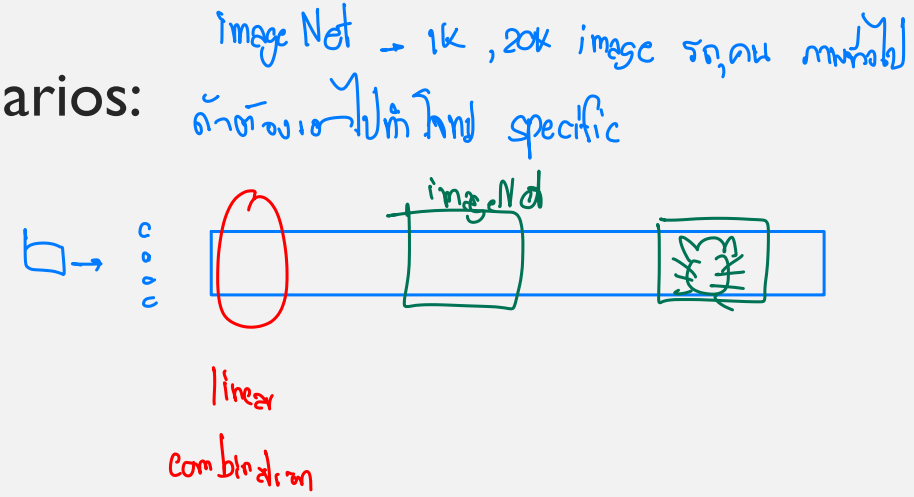
VARIANTS OF VITS

- https://paperswithcode.com/sota/image-classification-on-imagenet?tag_filter=171

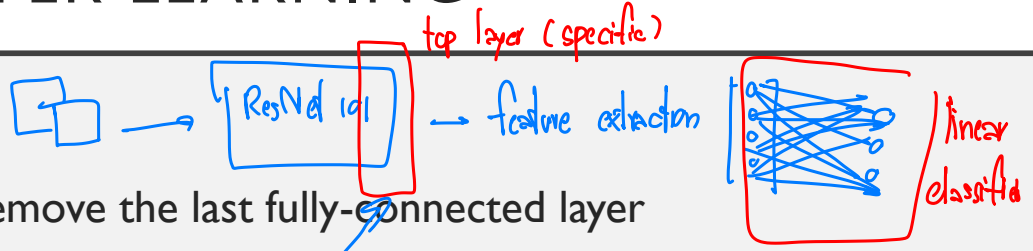


TRANSFER LEARNING

- In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size.
- Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.
- Three major Transfer Learning scenarios:
  1. ConvNet as fixed feature extractor
  2. Fine-tuning the ConvNet
  3. Pretrained models



TRANSFER LEARNING



1. ConvNet as fixed feature extractor
   - Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer
   - Train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.
2. Fine-tuning the ConvNet
   - not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network.
   - It is possible to fine-tune all the layers, or to keep some of the earlier layers fixed (overfitting concerns)
   - The observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors)
3. Pretrained models
   - Modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet. The release of the final ConvNet checkpoints of many researcher can be beneficial for fine-tuning your model.

References: https://cs231n.github.io/transfer-learning/

TRANSFER LEARNING

When and how to fine-tune?

- New dataset is small and similar to original dataset.
- New dataset is large and similar to the original dataset.
- New dataset is small but very different from the original dataset.
- New dataset is large and very different from the original dataset.

The best idea might be to train a linear classifier on the CNN

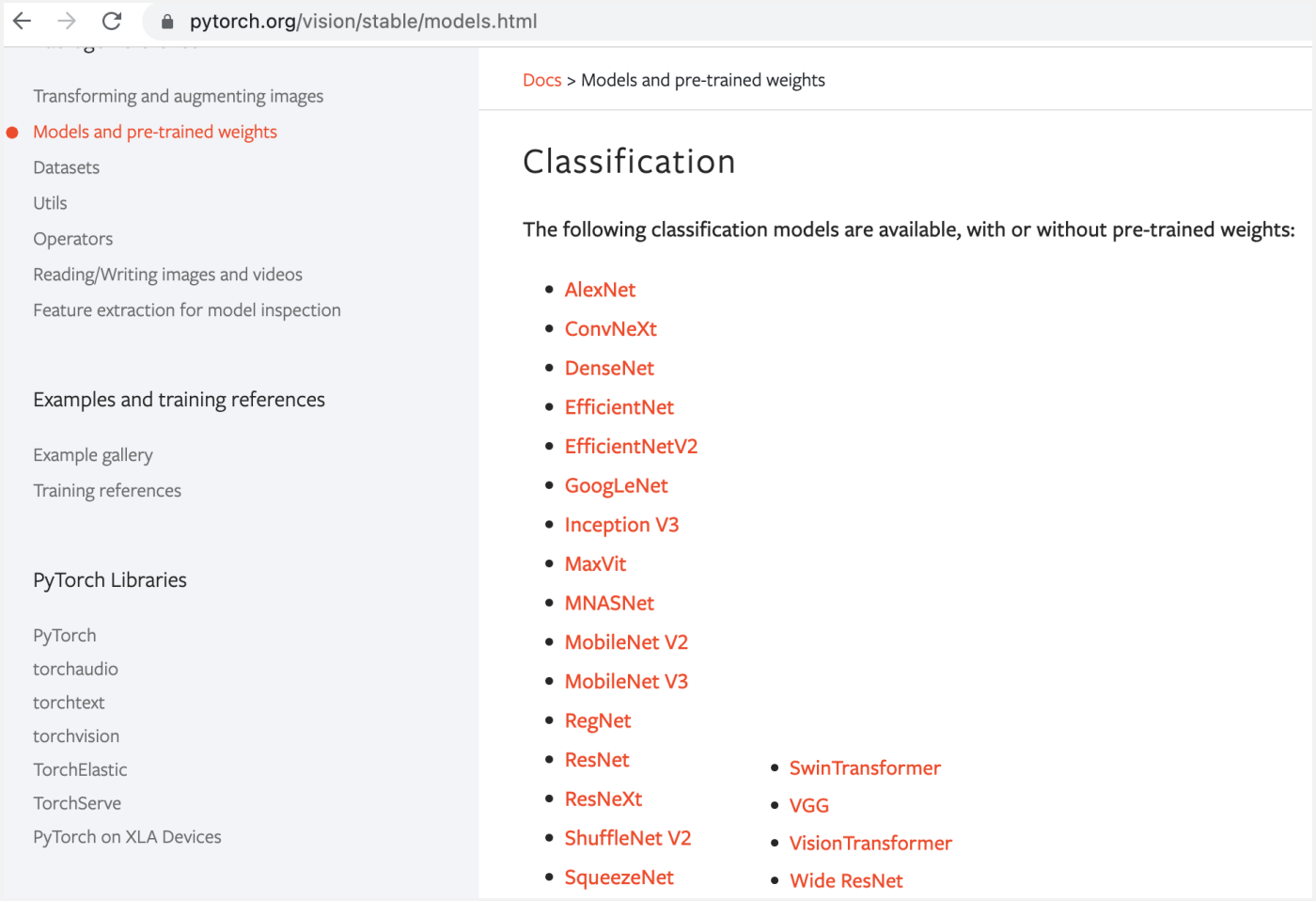
More confidence that wont be overfit. Try to fine-tune through the full network.

best to only train a linear classifier.
- The top networks contains dataset-specific features
- work better to train the SVM classifier from activations somewhere earlier in the network.

- To train a ConvNet from scratch.
- Still beneficial to initialize with weights from a pretrained mode

References: https://cs231n.github.io/transfer-learning/

EXALPLE #2: TRANSFER LEARNING



https://pytorch.org/vision/stable/models.html

EXALPLE #2: TRANSFER LEARNING

- Fine-tuning the model - Load a pretrained model and reset final fully connected layer.

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
# Here the size of each output sample is set to 2.
# Alternatively, it can be generalized to nn.Linear(num_ftrs, len(class_names)).
model_ft.fc = nn.Linear(num_ftrs, 2)

model_ft = model_ft.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)
```

Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

EXALPLE #2: TRANSFER LEARNING

- **ConvNet as fixed feature extractor** - freeze all the network except the final layer. We need to set requires_grad = False to freeze the parameters so that the gradients are not computed in backward().

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)

model_conv = model_conv.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that only parameters of final layer are being optimized as
# opposed to before.
optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
```

Reference: https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

REFERENCES

- Pytorch: https://pytorch.org/tutorials/
- DL Evolution: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5
- Transformer: https://medium.com/mlearning-ai/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c
- Additional reading
  - Basic ML - https://www.coursera.org/learn/machine-learning
  - Convolutional neural networks & Object detection - https://www.coursera.org/lecture/convolutional-neural-networks/object-detection-VgyWR

REFERENCES

- Krizhevsky, Alex, and Geoffrey E Hinton. n.d. “ImageNet Classification with Deep Convolutional Neural Networks,” 1–9.
- Liu, Chang, Yujie Zhong, Andrew Zisserman, and Weidi Xie. n.d. “ArXiv : 2208 . 13721v2 [Cs . CV] 10 Oct 2022 CounTR : Transformer-Based Generalised Visual Counting,” 1–16.
- Sangeetha, V., and K. J.Rajendra Prasad. 2006. “Syntheses of Novel Derivatives of 2-Acetylfuro[2,3-a]Carbazoles, Benzo[1,2-b]-1,4-Thiazepino[2,3-a]Carbazoles and 1-Acetyloxycarbazole-2- Carbaldehydes.” *Indian Journal of Chemistry - Section B Organic and Medicinal Chemistry* 45 (8): 1951–54. https://doi.org/10.1002/chin.200650130.
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. “Going Deeper with Convolutions.”
- Tan, Mingxing, and Quoc V. Le. 2019. “EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.” *36th International Conference on Machine Learning, ICML 2019* 2019-June: 10691–700. “VGGNet.Pdf.”
- Xie, Saining, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. “Aggregated Residual Transformations for Deep Neural Networks.” *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* 2017-Janua: 5987–95. https://doi.org/10.1109/CVPR.2017.634.
- Chen Z., et al. (2021) DPT: Deformable Patch-based Transformer for Visual Recognition
- Liu Z., et al. (2021) Swin Transformer: Hierarchical Vision Transformer using Shifted Windows