Animal Classification with Keras

Mehdi ABOUZAID
Damien TOOMEY
Weihao ZHOU

INSA – National Institute of Applied Sciences

April 24, 2019

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

Introduction

Some information

- Article:

  Animal Recognition and Identification with Deep Convolutional Neural Networks for Automated Wildlife Monitoring

- Studied models:

				Validation Set	
	ILSVRC	**Parameters**	**Trainable Layers**	**Top-1 Accuracy**	**Top-5 Accuracy**
AlexNet	2012	62,378,344	8	0.633	0.846
VGG-16	2014	138,357,544	16	0.713	0.901
ResNet-50	2015	25,636,712	50	0.749	0.921
Xception	2017	22,910,480	71	0.790	0.945

- Vocabulary:

  kernel = filter = receptive field = mask

Introduction

Our dataset

Input	Output	Number of Images
	butterfly	1991
	cow	2039
	squirrel	2013

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

CNN: VGG-16 as example

Input

$$\overbrace{\underset{\mathcal{I}_{\color{red}{red}}}{\begin{pmatrix} R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} \\ R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} \\ R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} \\ R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} \end{pmatrix}} \quad \underset{\mathcal{I}_{\color{green}{green}}}{\begin{pmatrix} G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} \\ G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} \\ G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} \\ G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} \end{pmatrix}} \quad \underset{\mathcal{I}_{\color{blue}{blue}}}{\begin{pmatrix} B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} \\ B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} \\ B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} \\ B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} \end{pmatrix}}}^{\mathcal{I}=\begin{pmatrix} I_{1,1} & I_{1,2} & I_{1,3} & I_{1,4} \\ I_{2,1} & I_{2,2} & I_{2,3} & I_{2,4} \\ I_{3,1} & I_{3,2} & I_{3,3} & I_{3,4} \\ I_{4,1} & I_{4,2} & I_{4,3} & I_{4,4} \end{pmatrix}}$$

CNN: VGG-16 as example

2D Convolution : only **3x3 kernel**, **stride 1**, **zero padding of thickness 1**

$$\mathcal{I}_{red} \qquad \mathcal{I}_{green} \qquad \mathcal{I}_{blue}$$

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} & 0 \\ 0 & R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} & 0 \\ 0 & R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} & 0 \\ 0 & R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & 0 \\ 0 & G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & 0 \\ 0 & G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & 0 \\ 0 & G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} & 0 \\ 0 & B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} & 0 \\ 0 & B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} & 0 \\ 0 & B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Kernel[:,:,0] \qquad Kernel[:,:,1] \qquad Kernel[:,:,2]$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} \qquad \begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{13} & w_{14} & w_{15} \\ w_{16} & w_{17} & w_{18} \end{bmatrix} \qquad \begin{bmatrix} w_{19} & w_{20} & w_{21} \\ w_{22} & w_{23} & w_{24} \\ w_{25} & w_{26} & w_{27} \end{bmatrix}$$

Goal : learn the weights in the kernels

CNN: VGG-16 as example

2D Convolution : only **3x3 kernel**, **stride 1**, **zero padding of thickness 1**

$$\mathcal{I}_{red} \qquad \mathcal{I}_{green} \qquad \mathcal{I}_{blue}$$

$$\begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \mathbf{0} & R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} & 0 \\ \mathbf{0} & R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} & 0 \\ 0 & R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} & 0 \\ 0 & R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \mathbf{0} & G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & 0 \\ \mathbf{0} & G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & 0 \\ 0 & G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & 0 \\ 0 & G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 & 0 \\ \mathbf{0} & B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} & 0 \\ \mathbf{0} & B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} & 0 \\ 0 & B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} & 0 \\ 0 & B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\underset{\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}}{Kernel[:,:,0]} \qquad \underset{\begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{13} & w_{14} & w_{15} \\ w_{16} & w_{17} & w_{18} \end{bmatrix}}{Kernel[:,:,1]} \qquad \underset{\begin{bmatrix} w_{19} & w_{20} & w_{21} \\ w_{22} & w_{23} & w_{24} \\ w_{25} & w_{26} & w_{27} \end{bmatrix}}{Kernel[:,:,2]}$$

$$w = np.hstack((Kernel[:,:,0].flatten(), Kernel[:,:,1].flatten(), Kernel[:,:,2].flatten()))$$

$$x = np.hstack((Window.flatten(), Window.flatten(), Window.flatten()))$$

$$FeatureMap[0,0,0] = ReLU(w.dot(x) + bias) = ReLU(w^T x + bias)$$

CNN: VGG-16 as example

2D Convolution : only **3x3 kernel**, **stride 1**, **zero padding of thickness 1**

$$\mathcal{I}_{red} \qquad\qquad \mathcal{I}_{green} \qquad\qquad \mathcal{I}_{blue}$$

$$\begin{pmatrix} 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 \\ 0 & R_{1,1} & R_{1,2} & R_{1,3} & R_{1,4} & 0 \\ 0 & R_{2,1} & R_{2,2} & R_{2,3} & R_{2,4} & 0 \\ 0 & R_{3,1} & R_{3,2} & R_{3,3} & R_{3,4} & 0 \\ 0 & R_{4,1} & R_{4,2} & R_{4,3} & R_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 \\ 0 & G_{1,1} & G_{1,2} & G_{1,3} & G_{1,4} & 0 \\ 0 & G_{2,1} & G_{2,2} & G_{2,3} & G_{2,4} & 0 \\ 0 & G_{3,1} & G_{3,2} & G_{3,3} & G_{3,4} & 0 \\ 0 & G_{4,1} & G_{4,2} & G_{4,3} & G_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & 0 & 0 \\ 0 & B_{1,1} & B_{1,2} & B_{1,3} & B_{1,4} & 0 \\ 0 & B_{2,1} & B_{2,2} & B_{2,3} & B_{2,4} & 0 \\ 0 & B_{3,1} & B_{3,2} & B_{3,3} & B_{3,4} & 0 \\ 0 & B_{4,1} & B_{4,2} & B_{4,3} & B_{4,4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Kernel[:,:,0] \qquad\qquad Kernel[:,:,1] \qquad\qquad Kernel[:,:,2]$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} \qquad \begin{bmatrix} w_{10} & w_{11} & w_{12} \\ w_{13} & w_{14} & w_{15} \\ w_{16} & w_{17} & w_{18} \end{bmatrix} \qquad \begin{bmatrix} w_{19} & w_{20} & w_{21} \\ w_{22} & w_{23} & w_{24} \\ w_{25} & w_{26} & w_{27} \end{bmatrix}$$

$$w = np.hstack((Kernel[:,:,0].flatten(), Kernel[:,:,1].flatten(), Kernel[:,:,2].flatten()))$$

$$x = np.hstack((Window.flatten(), Window.flatten(), Window.flatten()))$$

$$FeatureMap[1,0,0] = ReLU(w.dot(x) + bias) = ReLU(w^T x + bias)$$

CNN: VGG-16 as example

Feature maps & Max pooling

- as many feature maps as there are kernels
- each kernel is detecting a particular feature (edges,texture,...)
- by adding more kernels, the model can learn to detect more complex features
- max pooling ⇒ shrinking of the feature maps
  - no learnable parameters
  - 2x2 kernel
  - stride 2
  - no zero padding

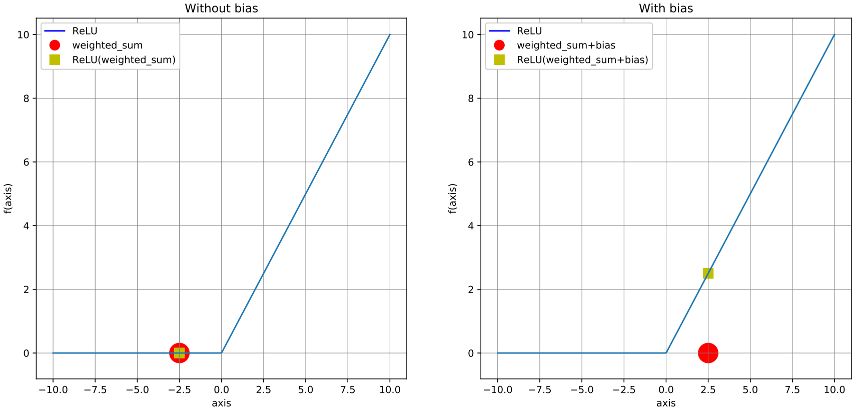
$$\begin{pmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{pmatrix} \underbrace{\rightarrow}_{\text{max pooling}} \begin{pmatrix} 6 & 8 \\ 3 & 4 \end{pmatrix}$$

CNN: VGG-16 as example

Bias

$$weighted\_sum = \sum_{i=1}^{n} w_i x_i = w^T x$$

- biases are learned parameters
- each neuron has a bias


Without bias

With bias

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

Xception

Architecture

71 trainable layers ; 22,910,480 parameters

Layer	Type	Activation function	Output Shape	Param
input_1	(InputLayer)	N/A	(299, 299, 3)	0
block1_conv1	(Conv2D)	N/A	(149, 149, 32)	864
block1_conv1_bn	(Batch Normalization)	N/A	(149, 149, 32)	128
block1_conv1_act	(Activation)	ReLU	(149, 149, 32)	0
block1_conv2	(Conv2D)	N/A	(147, 147, 64)	18,432
block1_conv2_bn	(Batch Normalization)	N/A	(147, 147, 64)	256
block1_conv2_act	(Activation)	ReLU	(147, 147, 64)	0
block2...	(...)	...	(147, 147, 128)	...
conv2d_45	(Conv2D)	N/A	(74,74,128)	8192
block2_pool	(MaxPooling2D)	N/A	(74,74,128)	0
bn_45	(Batch Normalization)	N/A	(74,74,128)	512
add	(Add)	N/A	(74,74,128)	0
block3...	(...)	...	(74, 74, 256)	...
...	(...)	...	(...)	...
block14...	(...)	...	(10, 10, 1536)	1,582,080
avg_pool	(GlobalAveragePooling2D)	N/A	(, 2048)	0
predictions	(Dense)	Softmax	(, 1000)	2,049,000

Xception

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(299, 299, 3)
	(Conv2D)	N/A	(149, 149, 32)
	(Batch Normalization)	N/A	(149, 149, 32)
	(Activation)	ReLU	(149, 149, 32)
	(...)	...	(...)
	(...)	...	(...)
	(GlobalAveragePooling2D)	N/A	(, 2048)
	(Dense)	ReLU	
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
train this layer	(Dense)	Softmax	(, nbClasses)

Trainable params: 6,147

Xception

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(299, 299, 3)
	(Conv2D)	N/A	(149, 149, 32)
	(Batch Normalization)	N/A	(149, 149, 32)
	(Activation)	ReLU	(149, 149, 32)
	(...)	...	(...)
	(...)	...	(...)
	(GlobalAveragePooling2D)	N/A	(, 2048)
	(Dense)	ReLU	
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
	(Dropout)	N/A	(, 2048)
train this layer	(Dense)	Softmax	(, nbClasses)

Trainable params: 6,147

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

VGG-16

Architecture (2014)

16 trainable layers ; 138,357,544 parameters

Layer	Type	Activation function	Output Shape	Param
input_1	(InputLayer)	N/A	(224, 224, 3)	0
block1_conv1	(Conv2D)	ReLU	(224, 224, 64)	1,792
block1_conv2	(Conv2D)	ReLU	(224, 224, 64)	36,928
block1_pool	(MaxPooling2D)	N/A	(112, 112, 64)	0
block2_conv1	(Conv2D)	ReLU	(112, 112, 128)	73,856
block2_conv2	(Conv2D)	ReLU	(112, 112, 128)	147,584
block2_pool	(MaxPooling2D)	N/A	(56, 56, 128)	0
block3_conv1	(Conv2D)	ReLU	(56, 56, 256)	295,168
block3_conv2	(Conv2D)	ReLU	(56, 56, 256)	590,080
block3_conv3	(Conv2D)	ReLU	(56, 56, 256)	590,080
block3_pool	(MaxPooling2D)	N/A	(28, 28, 256)	0
block4_conv1	(Conv2D)	ReLU	(28, 28, 512)	118,0160
block4_conv2	(Conv2D)	ReLU	(28, 28, 512)	2,359,808
block4_conv3	(Conv2D)	ReLU	(28, 28, 512)	2,35,9808
block4_pool	(MaxPooling2D)	N/A	(14, 14, 512)	0
block5_conv1	(Conv2D)	ReLU	(14, 14, 512)	2,359,808
block5_conv2	(Conv2D)	ReLU	(14, 14, 512)	2,359,808
block5_conv3	(Conv2D)	ReLU	(14, 14, 512)	2,359,808
block5_pool	(MaxPooling2D)	N/A	(7, 7, 512)	0
flatten	(Flatten)	N/A	(, 25088)	0
fc1	(Dense)	ReLU	(, 4096)	102,764,544
fc2	(Dense)	ReLU	(, 4096)	16,781,312
predictions	(Dense)	Softmax	(, 1000)	4,097,000

VGG-16

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(224, 224, 3)
	(Conv2D)	ReLU	(224, 224, 64)
	(Conv2D)	ReLU	(224, 224, 64)
	(MaxPooling2D)	N/A	(112, 112, 64)
	(Conv2D)	ReLU	(112, 112, 128)
	(Conv2D)	ReLU	(112, 112, 128)
	(MaxPooling2D)	N/A	(56, 56, 128)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(56, 56, 256)
	(MaxPooling2D)	N/A	(28, 28, 256)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(28, 28, 512)
	(MaxPooling2D)	N/A	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(MaxPooling2D)	N/A	(7, 7, 512)
	(Flatten)	N/A	(, 25088)
	(Dense)	ReLU	(, 4096)
	(Dense)	ReLU	(, 4096)
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
train this layer	(Dense)	Softmax	(, nbClasses)

VGG-16

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(224, 224, 3)
	(Conv2D)	ReLU	(224, 224, 64)
	(Conv2D)	ReLU	(224, 224, 64)
	(MaxPooling2D)	N/A	(112, 112, 64)
	(Conv2D)	ReLU	(112, 112, 128)
	(Conv2D)	ReLU	(112, 112, 128)
	(MaxPooling2D)	N/A	(56, 56, 128)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(56, 56, 256)
	(MaxPooling2D)	N/A	(28, 28, 256)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(28, 28, 512)
	(MaxPooling2D)	N/A	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(Conv2D)	ReLU	(14, 14, 512)
	(MaxPooling2D)	N/A	(7, 7, 512)
	(Flatten)	N/A	(, 25088)
	(Dense)	ReLU	(, 4096)
	(Dense)	ReLU	(, 4096)
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
	(Dropout)	N/A	(, 4096)
train this layer	(Dense)	Softmax	(, nbClasses)

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. **ResNet-50**
6. Conclusion
7. References

RESNET-50

Architecture (2015)

50 trainable layers ; 25,636,712 parameters

Layer	Type	Activation function	Output Shape	Param
input_1	(InputLayer)	N/A	(224, 224, 3)	0
res.._branch..	(Conv2D)	N/A	(112, 112, 64)	9,472
bn.._branch..	(Batch Normalization)	N/A	(112, 112, 64)	256
activation_..	(Activation)	ReLU	(112, 112, 64)	0
max_pooling2d_1	(MaxPooling2D)	N/A	(56, 56, 64)	0
res.._branch..	(Conv2D)	N/A	(56, 56, 64)	4,160
bn.._branch..	(Batch Normalization)	N/A	(56, 56, 64)	256
activation_..	(Activation)	ReLU	(56, 56, 64)	0
res.._branch..	(Conv2D)	N/A	(56, 56, 64)	36,928
bn.._branch..	(Batch Normalization)	N/A	(56, 56, 64)	256
activation_..	(Activation)	ReLU	(56, 56, 64)	0
res.._branch..	(Conv2D)	N/A	(56, 56, 256)	16,640
bn.._branch..	(Batch Normalization)	N/A	(56, 56, 256)	256
activation_..	(Activation)	ReLU	(56, 56, 64)	0
	...			
res.._branch..	(Conv2D)	ReLU	(28, 28, 128)	32,896
	...			
res.._branch..	(Conv2D)	ReLU	(28, 28, 128)	147,584
res.._branch..	(Conv2D)	ReLU	(28, 28, 512)	66,048
res.._branch..	(Conv2D)	ReLU	(14, 14, 256)	131,328
res.._branch..	(Conv2D)	ReLU	(14, 14, 256)	590,080
res.._branch..	(Conv2D)	ReLU	(14, 14, 1024)	263,168
res.._branch..	(Conv2D)	ReLU	(7, 7, 512)	524,800
res.._branch..	(Conv2D)	ReLU	(7, 7, 512)	2,359,808
res.._branch..	(Conv2D)	ReLU	(7, 7, 2048)	1,050,624
avg_pool	(GlobalAveragePooling2D)	N/A	(,2048)	0
fc1000	(Dense)	SoftMax	(, 1000)	2,049,000

RESNET-50

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(224,224,3)
	(Conv2D)	ReLU	(112, 112, 64)
	(MaxPooling2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(28, 28 , 128)
	(Conv2D)	ReLU	(28, 28, 128)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(14, 14, 256)
	(Conv2D)	ReLU	(14, 14, 256)
	(Conv2D)	ReLU	(14, 14, 1024)
	(Conv2D)	ReLU	(7, 7, 512)
	(Conv2D)	ReLU	(7, 7, 512)
	(Conv2D)	ReLU	(7, 7, 2048)
	(GlobalAveragePooling2D)	ReLU	(,2048)
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
train this layer	(Dense)	Softmax	(, nbClasses)

Trainable params: 6,147

RESNET-50

Transfer Learning

	Layer Type	Activation function	Output Shape
freeze weights learned on ImageNet	(InputLayer)	N/A	(224,224,3)
	(Conv2D)	ReLU	(112, 112, 64)
	(MaxPooling2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 64)
	(Conv2D)	ReLU	(56, 56, 256)
	(Conv2D)	ReLU	(28, 28 , 128)
	(Conv2D)	ReLU	(28, 28, 128)
	(Conv2D)	ReLU	(28, 28, 512)
	(Conv2D)	ReLU	(14, 14, 256)
	(Conv2D)	ReLU	(14, 14, 256)
	(Conv2D)	ReLU	(14, 14, 1024)
	(Conv2D)	ReLU	(7, 7, 512)
	(Conv2D)	ReLU	(7, 7, 512)
	(Conv2D)	ReLU	(7, 7, 2048)
	(GlobalAveragePooling2D)	ReLU	(,2048)
	~~(Dense)~~	~~Softmax~~	~~(, 1000)~~
	(Dropout)	N/A	(, 2048)
train this layer {	(Dense)	Softmax	(, nbClasses)

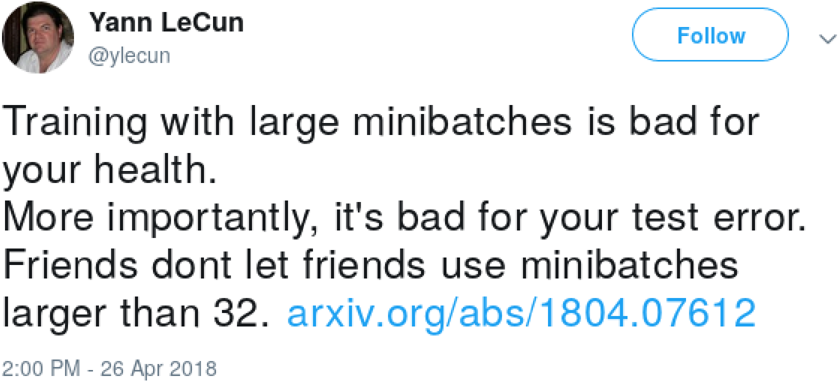
Trainable params: 6,147

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

Conclusion

Our results



Yann LeCun
@ylecun

Training with large minibatches is bad for your health.
More importantly, it's bad for your test error.
Friends dont let friends use minibatches larger than 32. arxiv.org/abs/1804.07612

2:00 PM - 26 Apr 2018

Conclusion

Our results

Training set (3867 images)

	Validation set (967 images) : Top 1 Accuracy											
	With 0.5 dropout						Without dropout					
Epochs	2			10			2			10		
Batch size	8	16	32	8	16	32	8	16	32	8	16	32
Xception	0.3588	0.3661	0.3609	0.3909	0.3433	0.3454	0.4116	0.4012	0.4012	0.4199	0.3899	0.3733
VGG-16	0.9741	0.9835	0.9824	0.9824	0.9855	0.9824	0.9762	0.9845	0.9866	0.9814	0.9814	0.9814
ResNet-50	0.9659	0.9700	0.9721	0.9731	0.9741	0.9731	0.9690	0.9690	0.9659	0.9710	0.9710	0.9710

	Test set (1209 images) : Top 1 Accuracy											
	With 0.5 dropout						Without dropout					
Epochs	2			10			2			10		
Batch size	8	16	32	8	16	32	8	16	32	8	16	32
Xception	0.3524	0.3677	0.3490	0.4030	0.3711	0.3708	0.3984	0.3764	0.3423	0.3667	0.3422	0.3598
VGG-16	0.9586	0.9702	0.9727	0.9644	0.9702	0.9744	0.9628	0.9686	0.9653	0.9669	0.9694	0.9686
ResNet-50	0.9661	0.9661	0.9711	0.9702	0.9735	0.9711	0.9686	0.9694	0.9694	0.9537	0.9644	0.9639

Plan

1. Introduction
2. CNN
3. Xception
4. VGG-16
5. ResNet-50
6. Conclusion
7. References

References

- Stanford University School of Engineering
  - Lecture 5 | Convolutional Neural Networks
    https://www.youtube.com/watch?v=bNb2fEVKeEo
  - Lecture 7 | Training Neural Networks II
    https://www.youtube.com/watch?v=_JB0AO7QxSA (1h12min : transfer learning)
  - Lecture 9 | CNN Architectures
    https://www.youtube.com/watch?v=DAOcjicFr1Y
  - CS231n Convolutional Neural Networks for Visual Recognition
    http://cs231n.github.io/convolutional-networks/
- MIT OpenCourseWare
  - 12a: Neural Nets
    https://www.youtube.com/watch?v=uXt8qF2Zzfo bias: 23min
  - 12b: Deep Neural Nets
    https://www.youtube.com/watch?v=VrMHA3yX_QI (kernels = neurone: 12min; softmax explanation: 34min)

References

- AlexNet Original Article:

  https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
- VGG-16 Original article : VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

  https://arxiv.org/pdf/1409.1556.pdf
- VGG-16: "The only preprocessing we do is subtracting the mean RGB value, computed on the training set, from each pixel."

  https://machinelearningmastery.com/use-pre-trained-vgg-model-classify-objects-photographs/)
- Batch Normalization : learnable parameters

  https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b
- Keras pretrained Models

  https://keras.io/applications/
- Pretrained Deep Neural Networks Matlab
- https://www.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html