Lecture Notes for

Machine Learning in Python

⟦👨‍🏫, 👨‍💻, 🐍, 👨‍🔬⟧

Professor Eric Larson

Practical Introductory CNNs

Class logistics and Agenda

- Wide/Deep Lab (late turn in)
- Agenda:
  - History of CNNs
    - with Modern CNN Architectures
    - and some built-in demos
- Next Time:
  - Finish this lecture
  - and move on to Transformers

Class Overview, by topic

Table Data Visualization

Numpy, Pandas, Seaborn
Overviews with some in-depth discussion

Dimension Reduction and Image Processing

Scikit-learn, Scikit Image,
Intuition only, Some mathematics

Linear and Logistic Regression

Numpy, Recreate API for Scikit-learn
Detailed mathematics for simple optimization
intuition for advanced optimization

Neural Networks and Back Prop.

Numpy
Detailed mathematics for NN operations

Wide and Deep Networks

Convolutional Networks

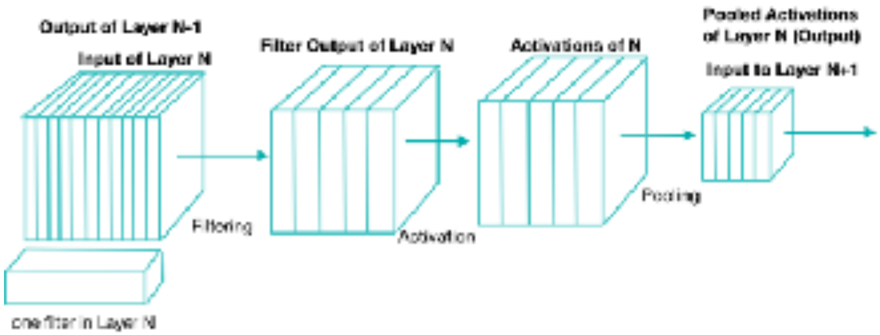
Recurrent Networks

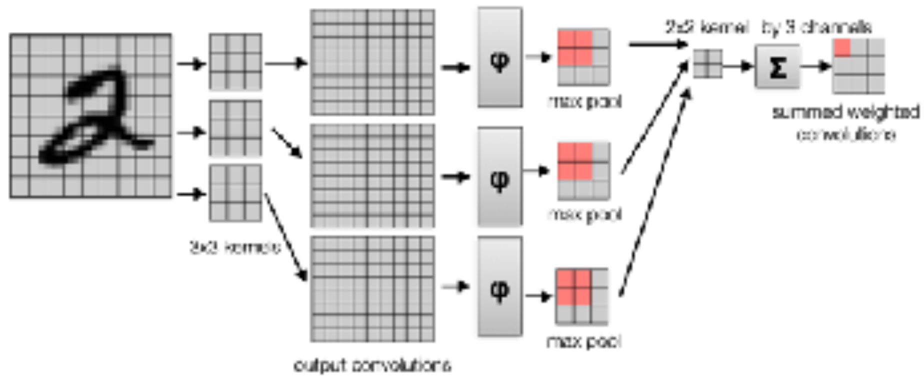
Keras, Tensorflow
Intuition, Detailed implement.

Ethics in Language Models

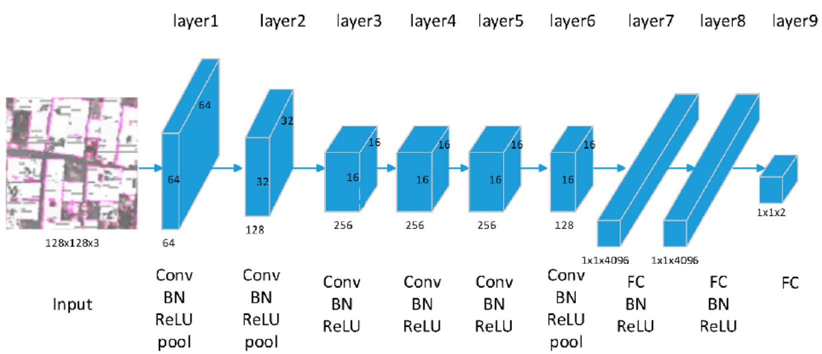
ConceptNet
Case studies

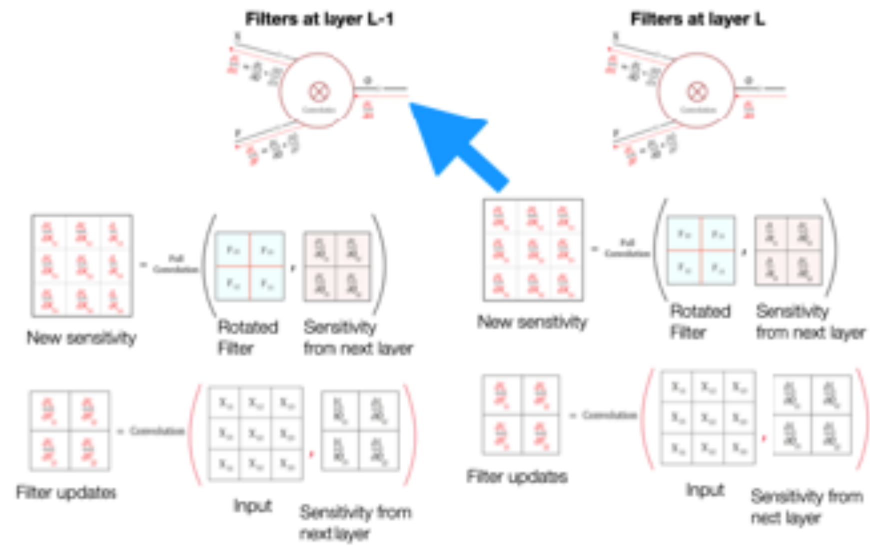
Last Time:





Structure of Each Tensor: Channels x Rows x Columns





TensorFlow and Basic CNNs

Last Time!

Convolutional Neural Networks
in TensorFlow
with Keras

with Sequential API!

If needed:

Finish Demo

`11. Convolutional Neural Networks.ipynb`

History of Convolutional Neural Networks

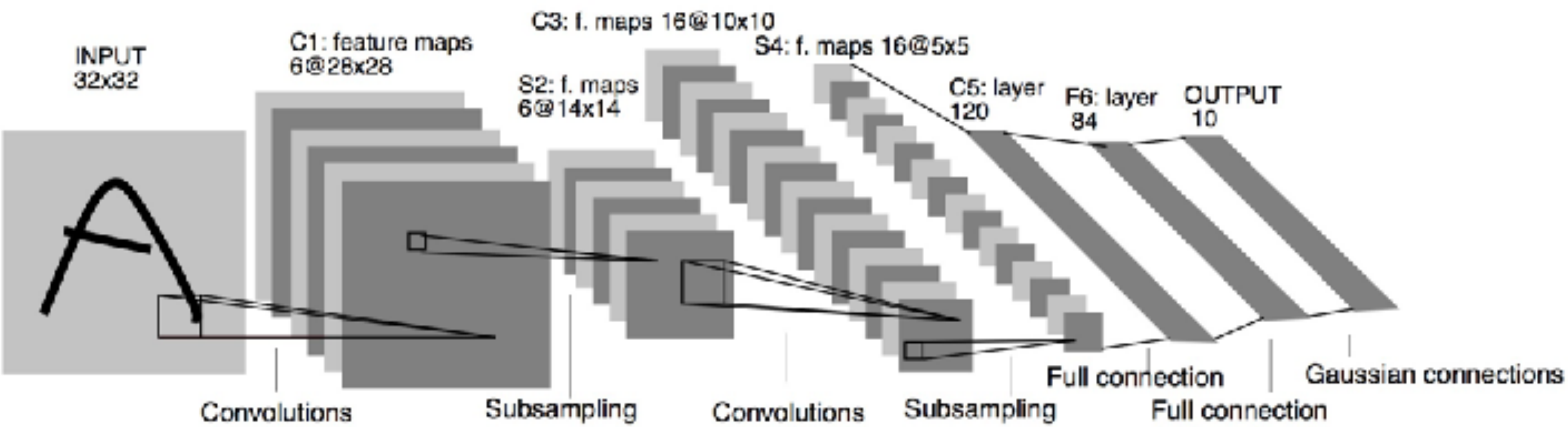
Machine Learning 101

Types of CNN, 1988-1998

Heads Facebook AI Team

- **LeNet-1** (1988)
  - ~2600 params, not many layers
- **LeNet-5** (1998)
  - 7 layers, gets excellent MNIST performance
- Major contribution, general structure:
  - conv=>pool=>non-linearity=> …=>MLP

avg **tanh or sigmoid**



https://culurciello.github.io/tech/2016/06/04/nets.html

CNN History

- List of major breakthroughs from 1998 through 2010 for convolutional networks:

¯_(ツ)_/¯

- 2010

Types of CNN, 2010

AI Researcher
IDSA, Switzerland

- **Ciresan Net**
- Publishes code for running CNN via GPU
    - Subsequently wins 5 international competitions
    - from stop signs => cancer detection
- Major contribution: NVIDIA parallelized training algorithms

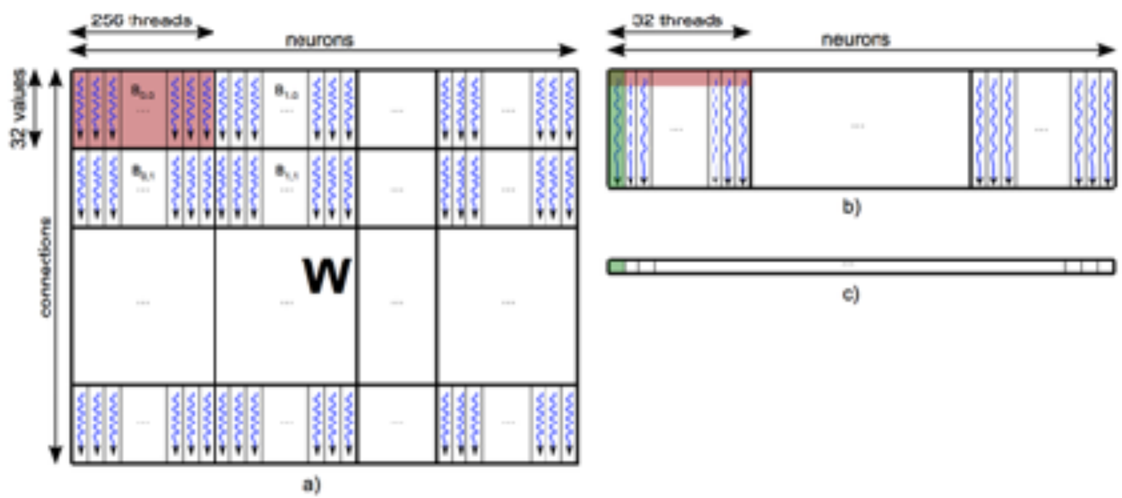
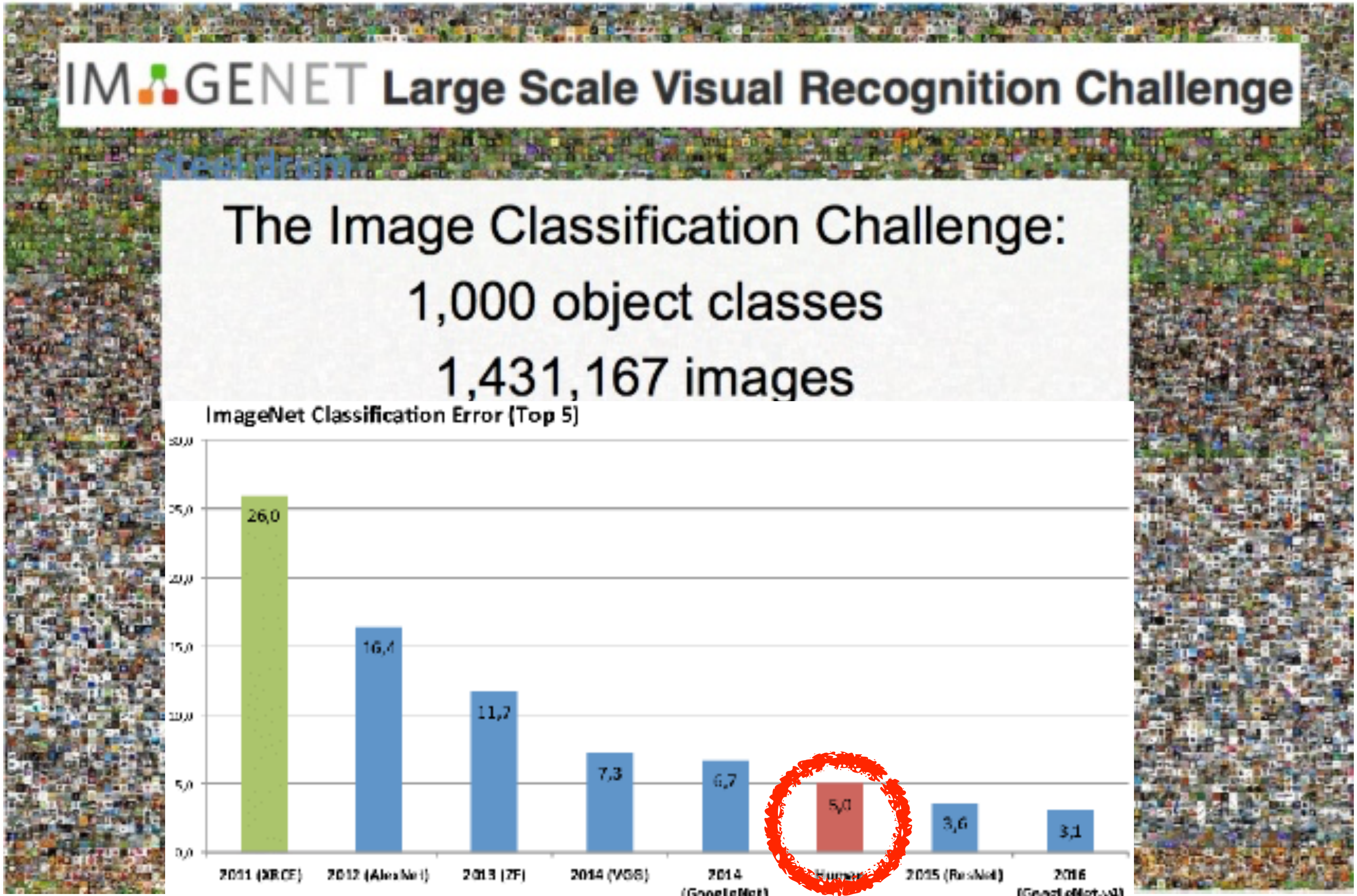


Figure 2: Forward propagation: a) mapping of kernel 1 grid onto the padded weight matrix; b) mapping the kernel 2 grid onto the partial dot products matrix; c) output of forward propagation.

https://arxiv.org/pdf/1003.0358.pdf

ImageNet Competition (2010-2016)



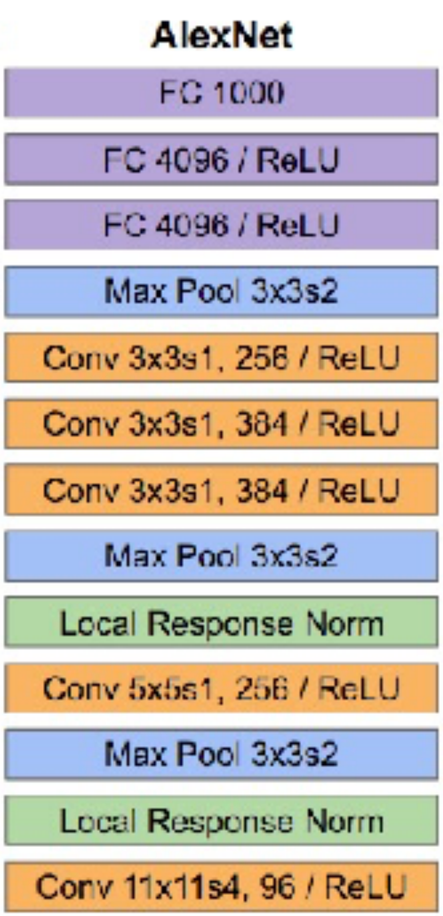
https://www.researchgate.net/figure/Winner-results-of-the-ImageNet-large-scale-visual-recognition-challenge-LSVRC-of-the_fig7_324476862

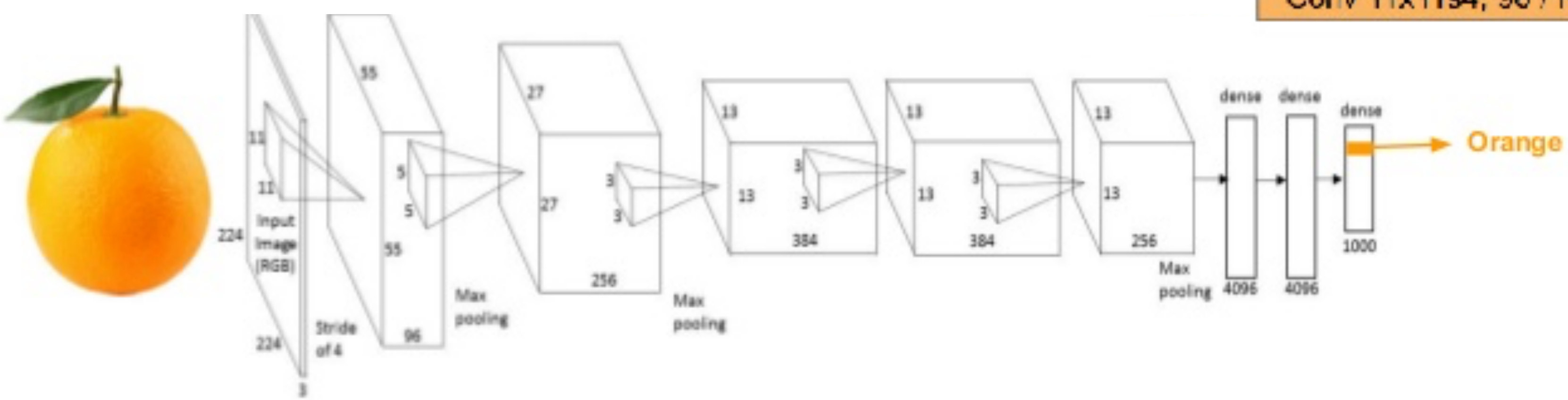
https://www.slideshare.net/nmhkahn/case-study-of-convolutional-neural-network-61556303

Types of CNN, 2012

Google

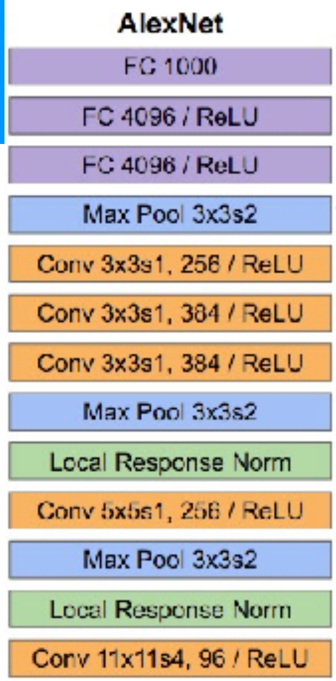
- **AlexNet**, Hinton is mentor
  - wins ImageNet competition
- Major contributions:
  - dropout for regularization
  - systematic use of ReLU
  - data expansion
  - ***overlapping max pool***





https://culurciello.github.io/tech/2016/06/04/nets.html

AlexNet, in Code




```
cnn2 = Sequential(name='AlexNet_Aug')

cnn2.add( Input([64,64,3]) )

# add in augmentations directly
cnn2.add( RandomFlip("horizontal") ) # flip horizontally
cnn2.add( RandomRotation(0.05) ) # rotate by 5%
cnn2.add( RandomTranslation(height_factor=0.1, width_factor=0.1) )
cnn2.add( RandomBrightness(factor=0.1, value_range=(0.0, 1.0)) ) #
cnn2.add( RandomContrast(0.1) ) # add or decrease contrast

cnn2.add(Conv2D(filters=96,
                kernel_size=(11,11),
                strides=(2,2), # add some initial downsampling
                padding='same',
                kernel_initializer='he_uniform',
                kernel_regularizer=l2(1e-6),
                activation='relu')) # more compact syntax
```

...

```
cnn2.add(Conv2D(filters=256,
                kernel_size=(5,5),
                padding='same',
                kernel_initializer='he_uniform',
                kernel_regularizer=l2(1e-5),
                activation='relu')) # more compact syntax
cnn2.add(MaxPooling2D(pool_size=(2, 2)))

cnn2.add(Conv2D(filters=256,
                kernel_size=(3,3),
                padding='same',
                kernel_initializer='he_uniform',
                kernel_regularizer=l2(1e-5),
                activation='relu')) # more compact syntax
cnn2.add(MaxPooling2D(pool_size=(3, 3)))
```

...

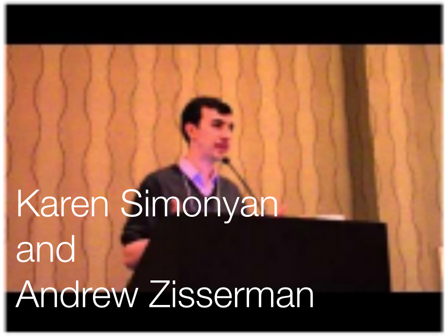
```
# add one layer on flattened output
cnn2.add(Flatten())
cnn2.add(Dense(1024, activation='relu',
               kernel_initializer='he_uniform',
               kernel_regularizer=l2(1e-4)))
cnn2.add(Dense(512, activation='relu',
               kernel_initializer='he_uniform',
               kernel_regularizer=l2(1e-4)))
cnn2.add(Dropout(0.5)) # add some dropout for regu
cnn2.add(Dense(NUM_CLASSES, activation='softmax',
               kernel_initializer='glorot_uniform'
               ))

opt = keras.optimizers.Adam(beta_1=.9,beta_2=.999,

# Let's train the model
cnn2.compile(loss='sparse_categorical_crossentropy
               optimizer=opt,
               metrics=['accuracy'])
```

12a. More Advanced CNN Techniques as TFData.ipynb

Warning


DON'T TRY WHAT YOU'RE ABOUT TO SEE AT HOME
WE'RE WHAT YOU CALL
EXPERTS
WeKnowMemes

Types of CNN, 2013



UNIVERSITY OF OXFORD

Google DeepMind

- Oxford **VGG Net** (Visual Geometry Group)
- Major contributions:
  - small cascaded kernels
  - way more layers (19 versus ~7)
  - *"emulates" biology "better"* 😒
  - trained on NVIDIA GPUs for 2-3 weeks

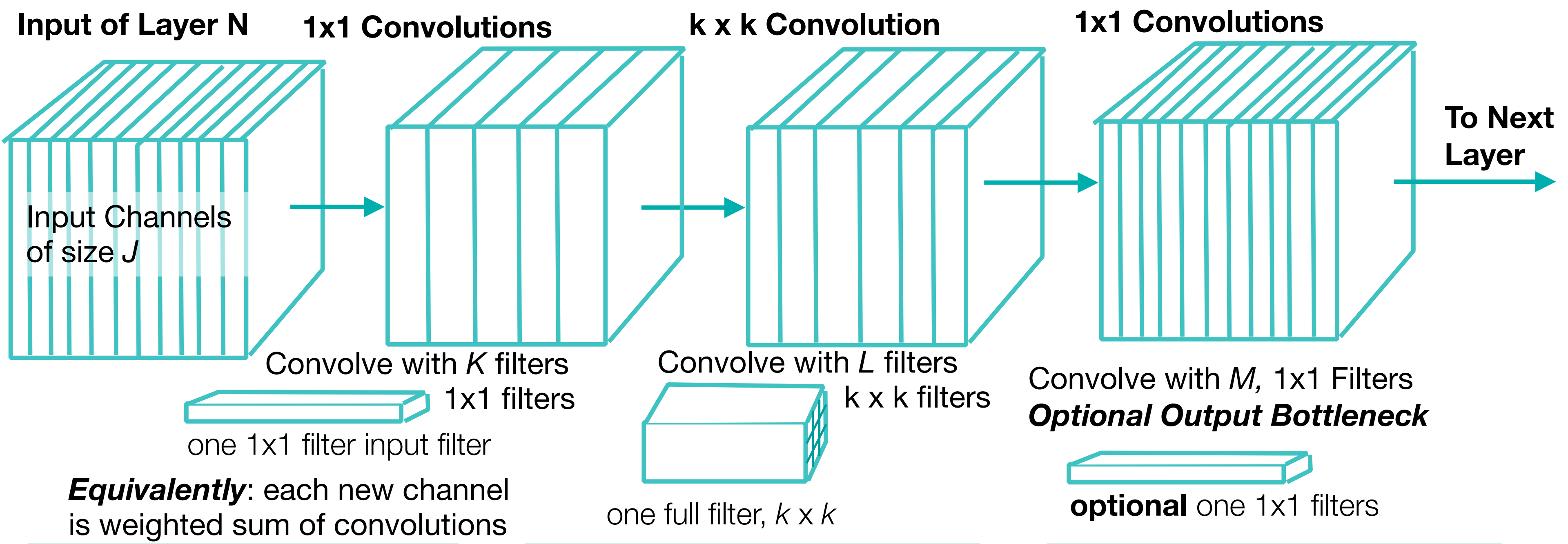
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

http://www.robots.ox.ac.uk/~vgg/practicals/cnn/

2014 Network in Network, expanded view



Reduce Channels using Lightweight 1x1 Filters

Larger k x k Filters on Reduced Channels

Expand with lightweight Filters, *(if desired)*

Example: Train 32 filters with kernel size of 3x3, input channels of 256 (outputs size of 32)

- No reduction: (32 x 3 x 3 x 256) = **73,728** (no bottleneck)
- Reduction to 128: (128 x 1 x 1 x 256) + (32 x 3 x 3 x 128) = **69,632** (reduce 128)
- Reduction to 64: (64 x 1 x 1 x 256) + (32 x 3 x 3 x 64) = **34,816** (reduce 64)

Bottleneck, in Code

```
# we can also use sequential like a function builder,
parallel_3x3a = keras.Sequential([
        Conv2D(filters=32, kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=128,kernel_size=(3,3),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5))
    ])
```

...

```
# no max pool before next conv layer!!
x = Conv2D(filters=96,
                kernel_size=(11,11),
                strides=(2,2), # add some initial downsampling
                padding='same',
                kernel_initializer='he_uniform',
                kernel_regularizer=l2(1e-6),
                activation='relu')(x)

# now save this tensor to use in mutliple branches
input_conv = MaxPooling2D(pool_size=(2, 2),  name='branch_point1')(x)
```

...

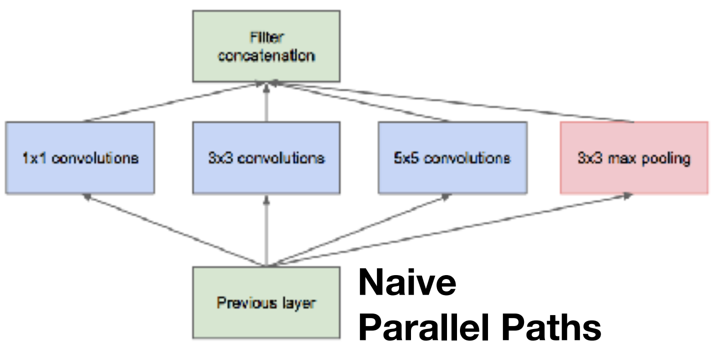
```
x = parallel_3x3a(input_conv)
```

...

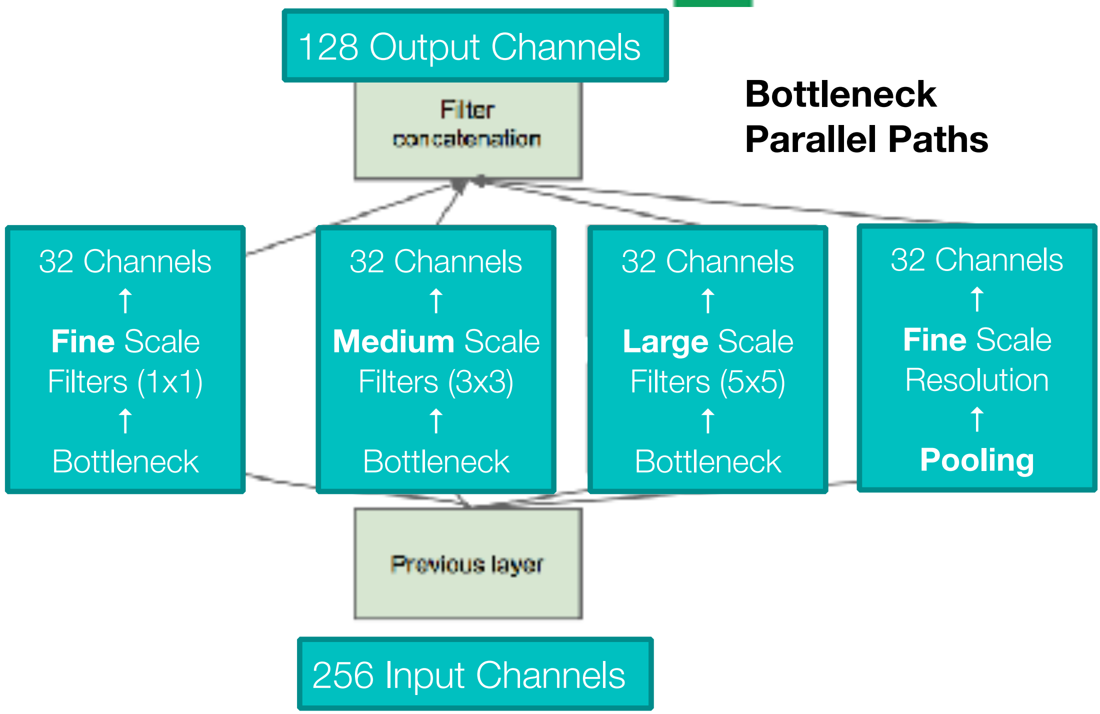
12a. More Advanced CNN Techniques as TFData.ipynb

Types of CNN, 2014 (parallel pathways)

- **GoogLeNet**
  - or **Inception V1**
- Major contribution:
  - multi-scale filtering
  - parallel NiN



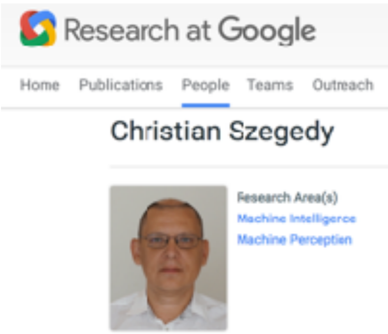
Naive Parallel Paths



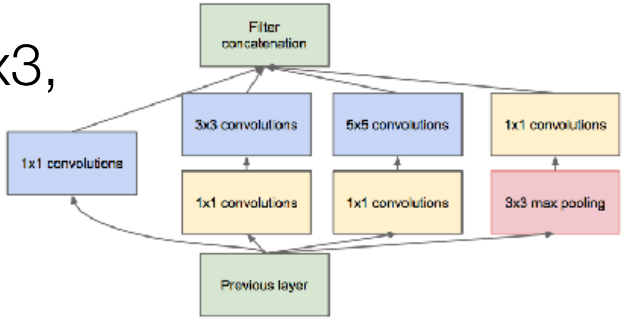
Bottleneck Parallel Paths

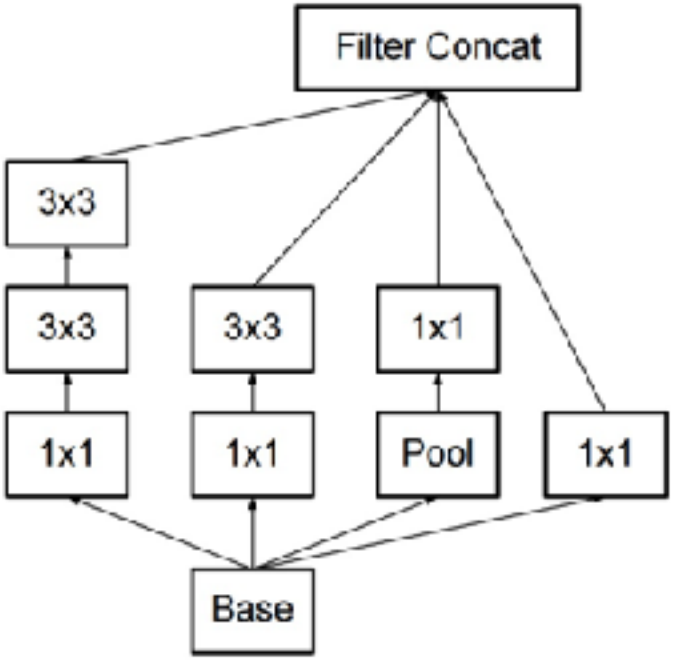
https://arxiv.org/pdf/1409.4842.pdf

Types of CNN, 2015 February and December

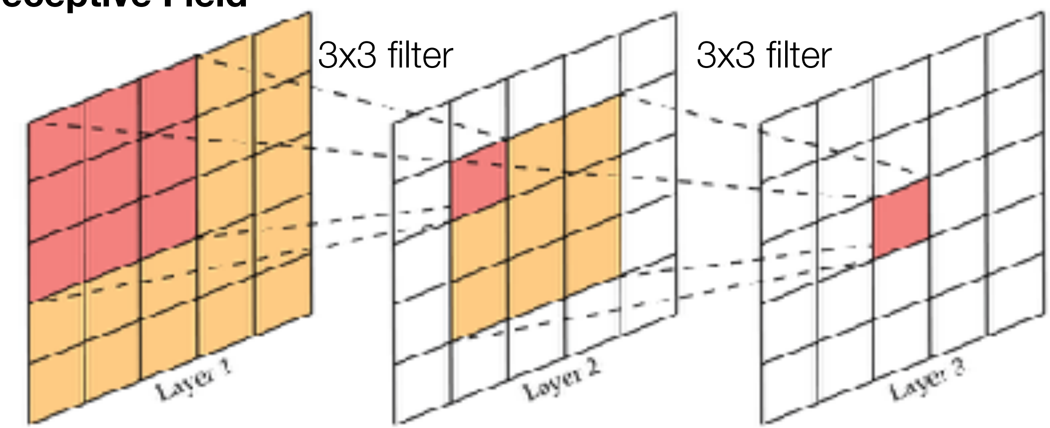


- **Inception V2**,
  - Inception V1 with different normalization
- **Inception V3**:
  - replace 5x5 with multiple 3x3, maintains receptive field





Receptive Field



in turn, influenced by these 5x5 pixels

influenced by these 3x3 pixels

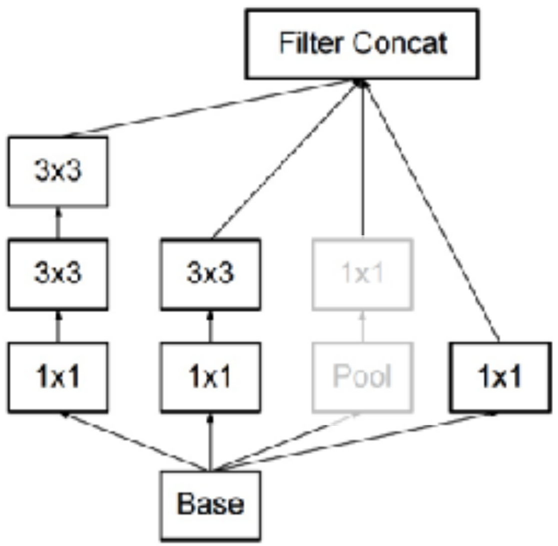
output

https://arxiv.org/pdf/1512.00567.pdf

https://medium.com/@rekalantar/receptive-fields-in-deep-convolutional-networks-43871d2ef2e9

Parallel Paths, in Code

```
# we can also use sequential like a function builder,
parallel_3x3a = keras.Sequential([
        Conv2D(filters=32, kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=128,kernel_size=(3,3),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5))
    ])

parallel_3x3multa = keras.Sequential([
        Conv2D(filters=32, kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=64,kernel_size=(3,3),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=64,kernel_size=(3,3),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5))
    ])

conv_1x1a = Conv2D(filters=64,kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-6))
```



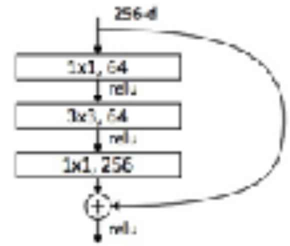
...

```
# now save this tensor to use in mutliple branches
input_conv = MaxPooling2D(pool_size=(2, 2),  name='branch_point1')(x)

# now place into parallel input branches
branches = []
x = conv_1x1a(input_conv)
branches.append(x)
x = parallel_3x3a(input_conv)
branches.append(x)
x = parallel_3x3multa(input_conv)
branches.append(x)
# that's it, we just need to average the results
x = Concatenate(axis=-1,name='ens_concat1')(branches)
```

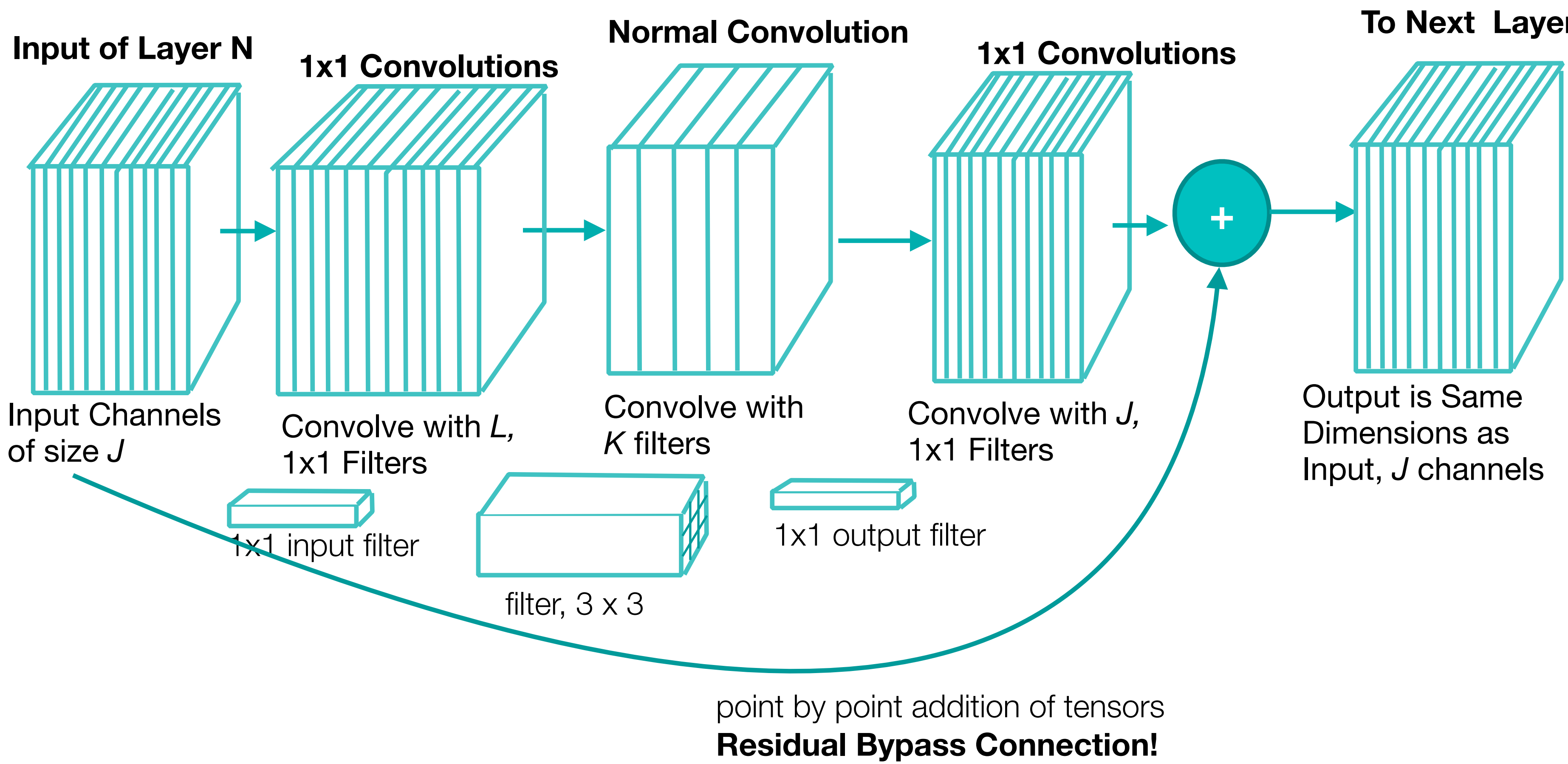
12a. More Advanced CNN Techniques as TFData.ipynb

Types of CNN, 2015 December

- **ResNet,** Major Contributions:
  - bypass pathways
  - “bio-plausible” with feedback 😒



Back Propagation: Two paths, including one without ANY operations that cause the gradient to vanish…

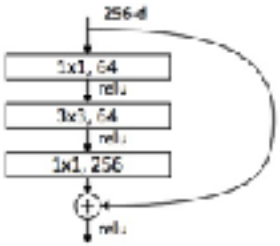


Residual Paths, in Code

```
concat_filt_size = 128

residual_3x3a = keras.Sequential([
        Conv2D(filters=32, kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=32,kernel_size=(3,3),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5)),
        Conv2D(filters=concat_filt_size,kernel_size=(1,1),padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5))
    ])

residual_3x3b = keras.Sequential([
```

...

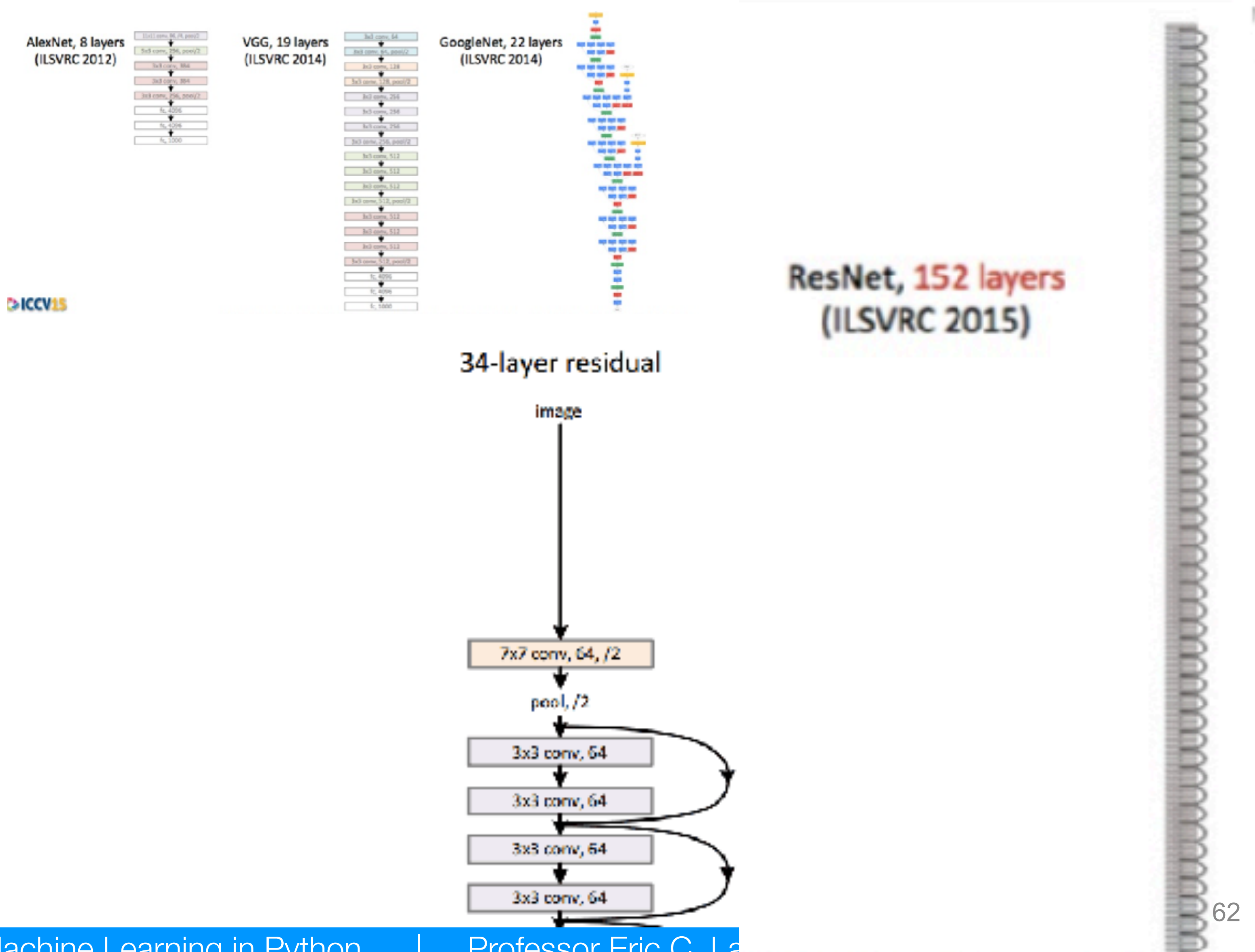
```
x = MaxPooling2D(pool_size=(2, 2),  name='branch_point1')(x)

x_split = Conv2D(filters=concat_filt_size, kernel_size=(1,1), padding='same',
                kernel_initializer='he_uniform', kernel_regularizer=l2(1e-5))(x)

x = residual_3x3a(x_split)
# now add back in the split layer, x_split (residual added in)
x = Add()([x, x_split])
x = Activation("relu")(x)

x_split = MaxPooling2D(pool_size=(2, 2))(x)

x = residual_3x3b(x_split)
# now add back in the split layer, x_split (residual added in)
x = Add()([x, x_split])
x = Activation("relu")(x)

x = MaxPooling2D(pool_size=(2, 2))(x)
```



`12a. More Advanced CNN Techniques as TFData.ipynb`

How big are these networks?

How big are these networks?



Transition Period in Convolutional Networks

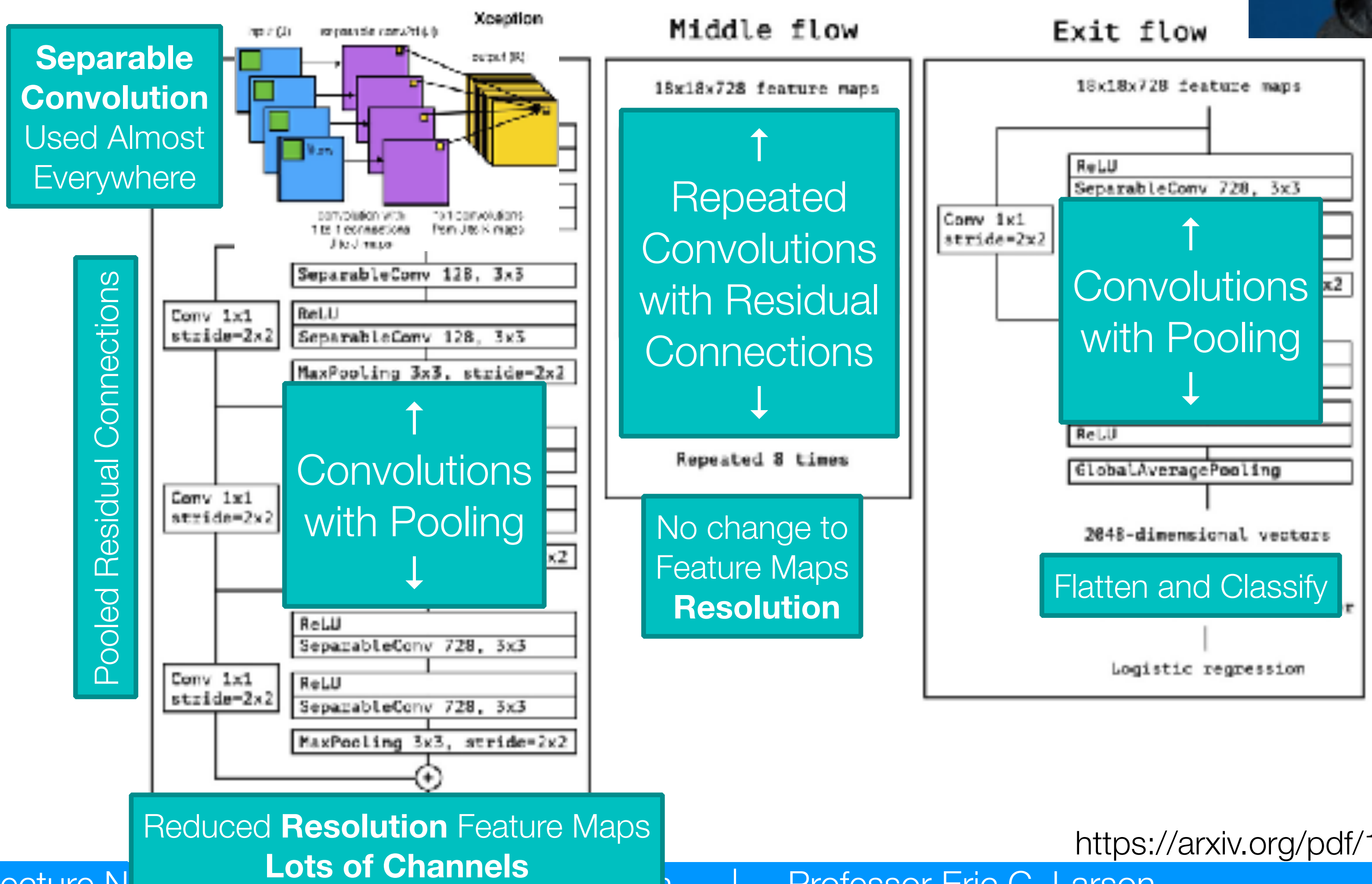
- 2012 - 2017:
    - Add more layers! 🤪
    - How can we train it even deeper? 🤓
    - Can we run out of memory? Let’s try! 🙈
- 2017-2021:
    - How can we get similar performance with reduced parameters? 🧐
    - How should the number of parameters scale for competing resource? Is there an optimum scaling for a given set of resources? 📉
- 2021 - present: Transformers…

Types of CNN, 2017

Francois Chollet
Google

Xception · Major Contributions:

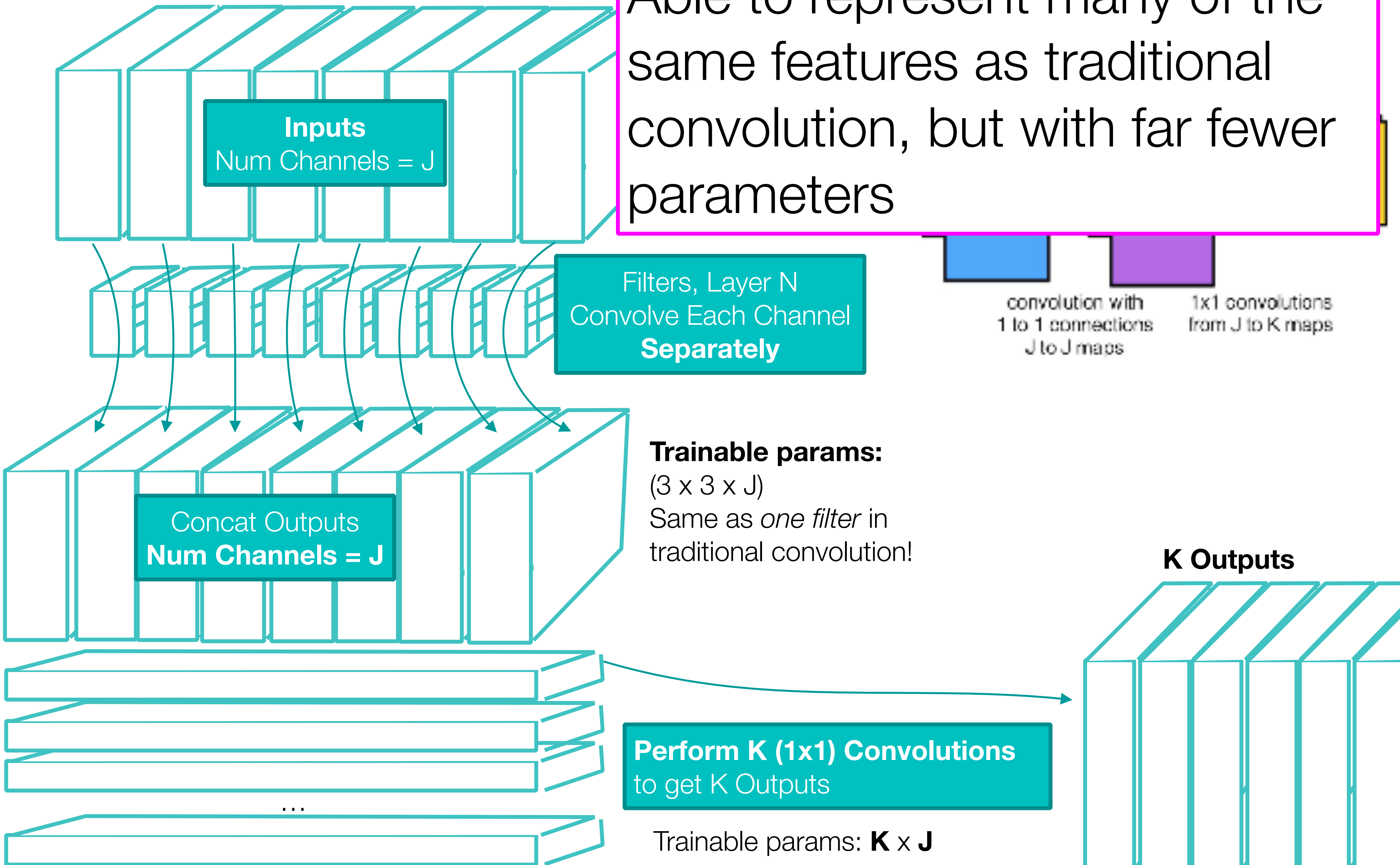
- combining branching / residual blocks
- separable convolutions (fewer trainable params)



https://arxiv.org/pdf/1610.02357.pdf

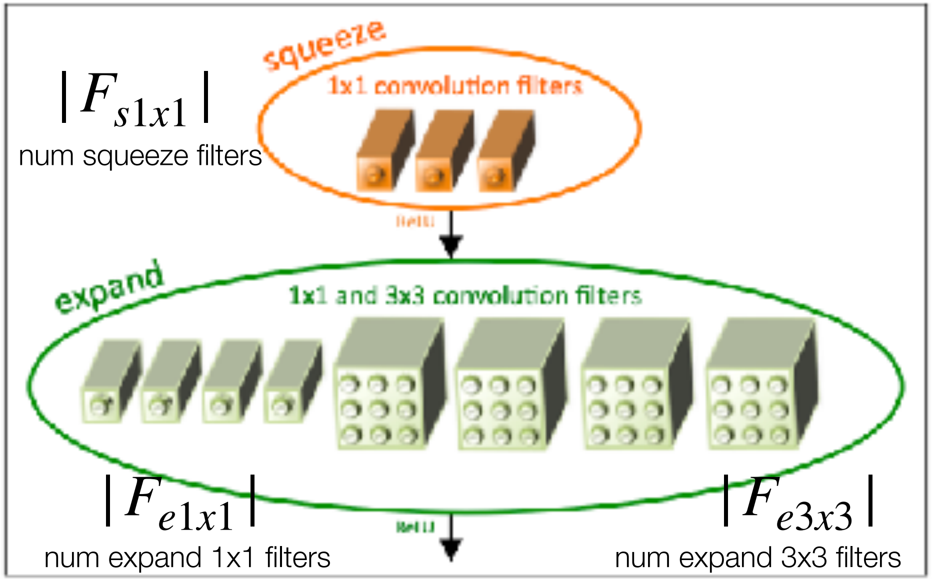
Separable Convolut

Separable Convolution: Able to represent many of the same features as traditional convolution, but with far fewer parameters



SqueezeNet (2018)

- How to squeeze and expand in each layer, with residual pathways?
  - Decide bottleneck size from a ratio, *what ratio is best?*
  - Use mostly 1x1 filters, *how many versus 3x3?*

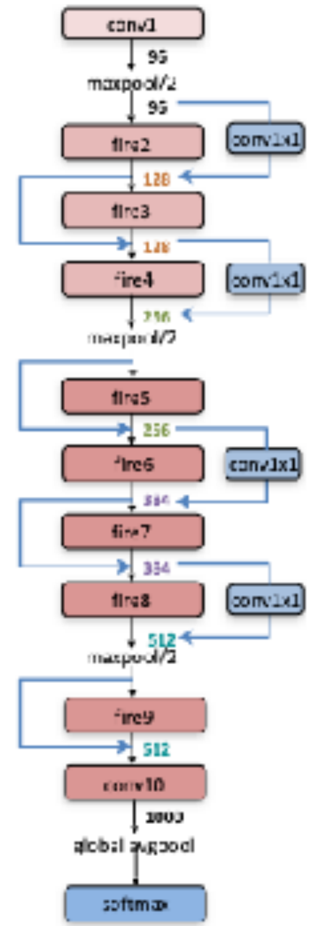


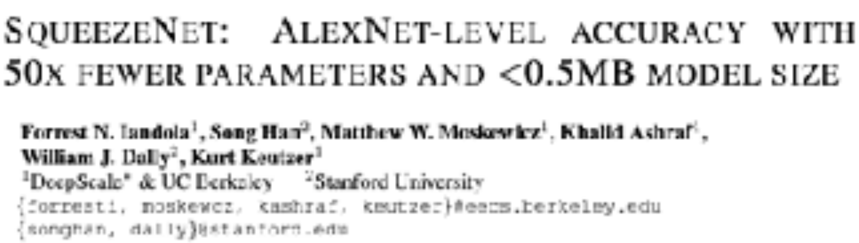
$$SR = \frac{|F_{s1x1}|}{|F_{e1x1}| + |F_{e3x3}|}$$

Controls how much to bottleneck

$$PCT_{3x3} = \frac{|F_{e3x3}|}{|F_{e1x1}| + |F_{e3x3}|}$$

Controls num filter params
(how many 1x1 versus 3x3)




SQUEEZENET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola[1], Song Han[2], Matthew W. Moskewicz[1], Khalid Ashraf[1], William J. Dally[2], Kurt Keutzer[1]
[1]DeepScale* & UC Berkeley [2]Stanford University
{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu

In paper, Good Performance when:
- SR is 12.5% up to 100%
- PCT_{3x3} is 25% up to 100%

Efficient Net (2019)

Start with so

Observation 1 – Scaling up any
width, depth, or resolution improve
racy gain diminishes for bigger mo

Observation 2 – In order to purs
efficiency, it is critical to balance al
width, depth, and resolution during

Filtering

Depth Scalin

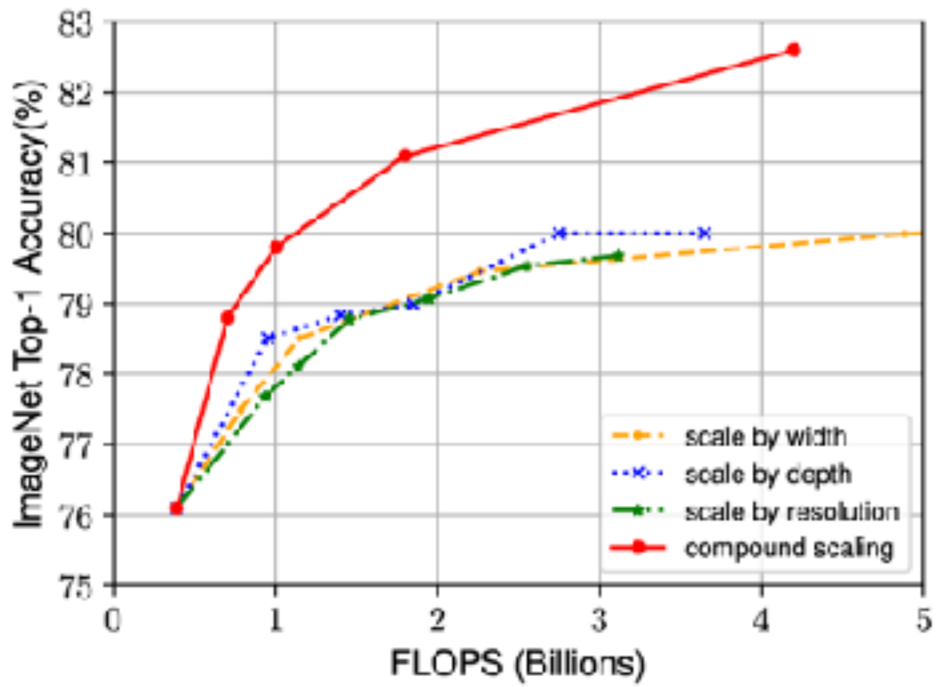


Figure 8. **Scaling Up EfficientNet-B0 with Different Methods.**

Resolution Scaling: For input different resol

depth: $d = \alpha^{\phi}$

width: $w = \beta^{\phi}$

res.: $r = \gamma^{\phi}$

s.t. $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$

$\alpha, \beta, \gamma \geq 1$

ϕ user specified scaling coefficient

$$\alpha = 1.2$$
$$\beta = 1.1$$
$$\gamma = 1.15$$

where α, β, γ are constants that can be determined by a small grid search. Intuitively, ϕ is a user-specified coefficient that controls how many more resources are available for model scaling, while α, β, γ specify how to assign these extra resources to network width, depth, and resolution re-

optimal values found in paper!

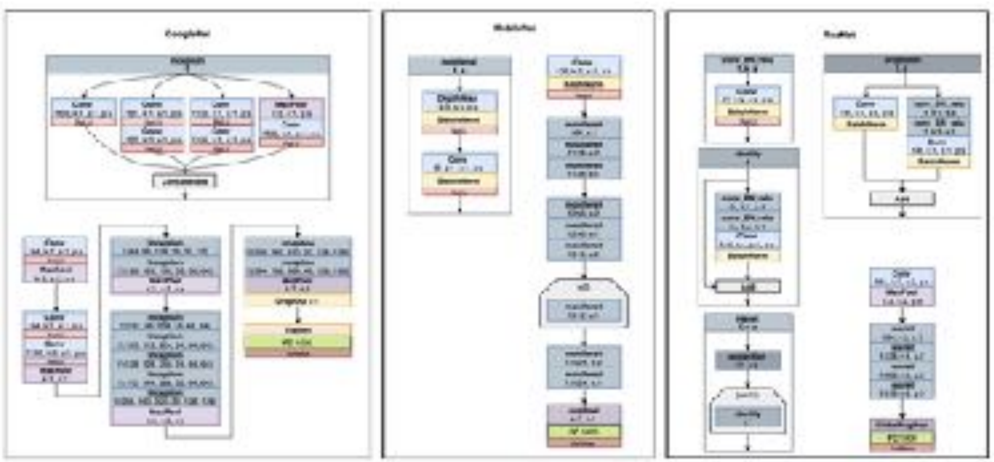
https://arxiv.org/pdf/1905.11946v5.pdf

Even more Convolutional Neural Networks

…in TensorFlow

…with Keras

Mostly Self Guided Demo



Transfer Learning Covered Here

`12. More Advanced CNN Techniques as TFData.ipynb`

Next Time:

- Intro to Sequential Neural Network Architectures
  - Word Embeddings, 1D CNNs, Transformers