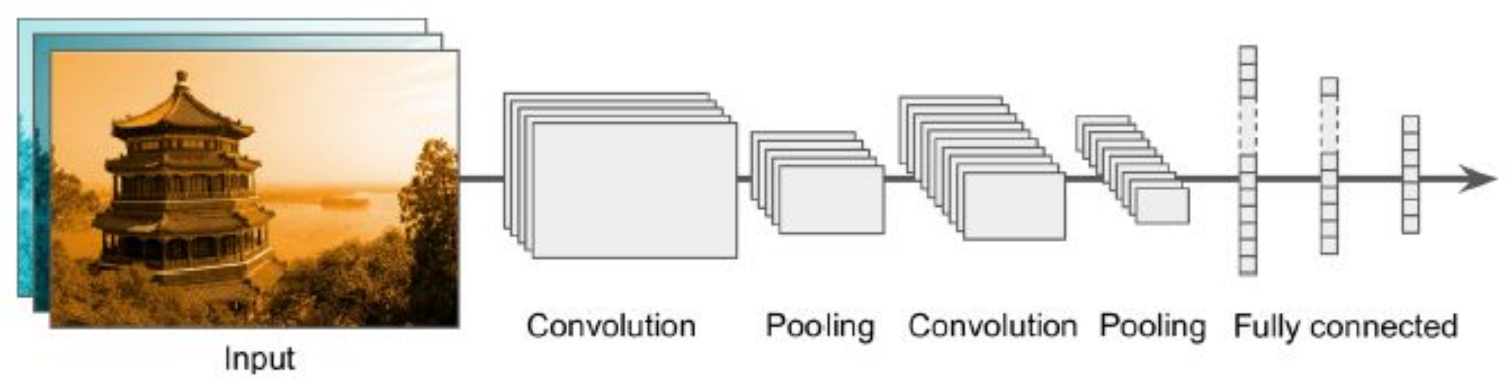
Convolutional Neural Networks (CNN) Architectures

Lesson #12

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Typical CNN Architecture




```
INPUT => [[CONV => RELU]*N => POOL?]*M => [FC => RELU]*K => FC
```

Best Practices

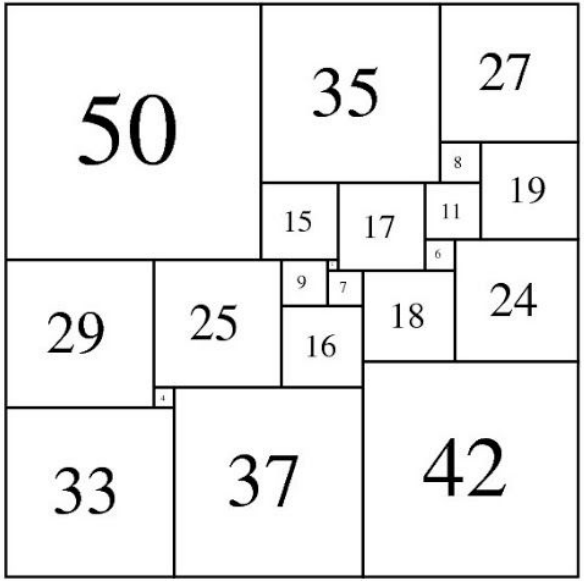
Where do the Batch Normalization layers go?

```
INPUT => [[CONV => RELU]*N => POOL?]*M => [FC => RELU]*K => FC
```

BN? BN? BN?

- Batch normalization has been shown to be extremely effective at **reducing the number of epochs** it takes to train a neural network.
- Batch normalization also has the added benefit of helping **“stabilize” training**, allowing for a larger variety of learning rates and regularization strengths.
- Using batch normalization doesn’t alleviate the need to tune these parameters of course, but it will make your life easier by making **learning rate and regularization less volatile** and more **straightforward to tune**.
- The **biggest drawback** of batch normalization is that it can actually slow down the wall time it takes to train your network (even though you’ll need fewer epochs to obtain reasonable accuracy) by **2-3x due to the computation** of per-batch statistics and normalization.

Best Practices



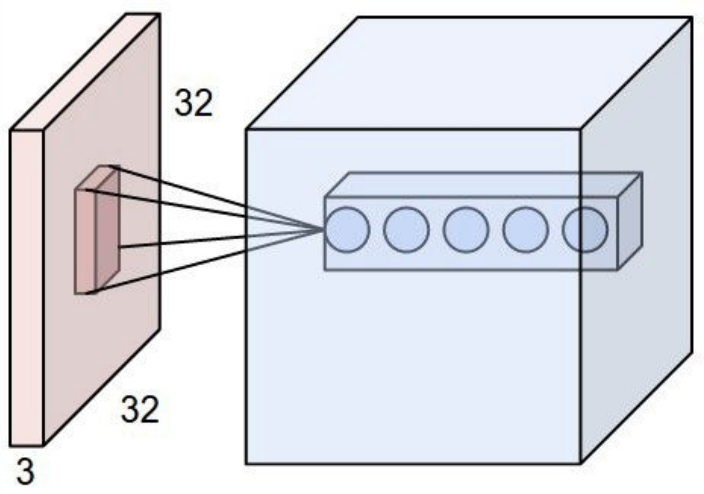
- To start, the images presented to the input layer should be **square**. Using square inputs allows us to take **advantage of linear algebra optimization libraries**.
- Common input layer sizes include 32 x 32, 64 x 64, 96 x 96, 224 x 224, 227 x 227 and 229 x 229 (leaving out the number of channels for notational convenience).

Best Practices

Layer (type)	Output Shape
conv2d (Conv2D)	(None, 28, 28, 6)
average_pooling2d (AveragePo	(None, 14, 14, 6)
conv2d_1 (Conv2D)	(None, 10, 10, 16)
average_pooling2d_1 (Average	(None, 5, 5, 16)
flatten (Flatten)	(None, 400)
dense (Dense)	(None, 120)
dense_1 (Dense)	(None, 84)
dense_2 (Dense)	(None, 10)

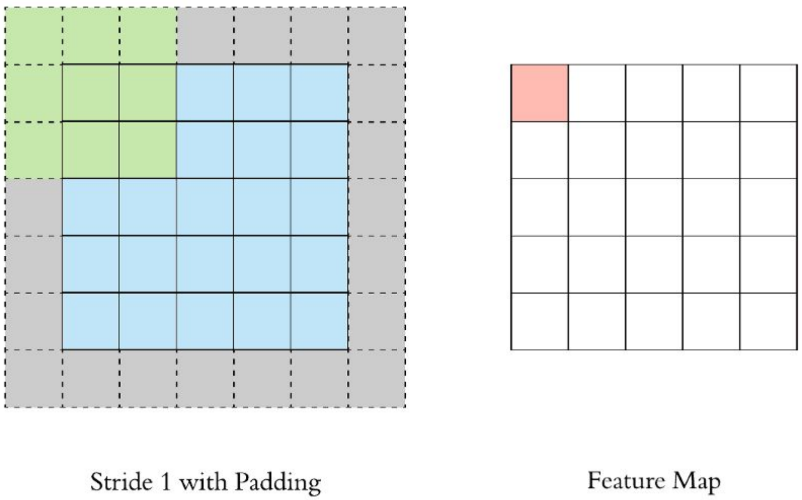
- The **"divisible by two rule"** enables the spatial inputs in our network to be conveniently down sampled via POOL operation in an efficient manner.
- The input layer should also be **divisible by two** multiple times after the first CONV operation is applied.
- You can do this by **tweaking your filter size and stride**.

Best Practices



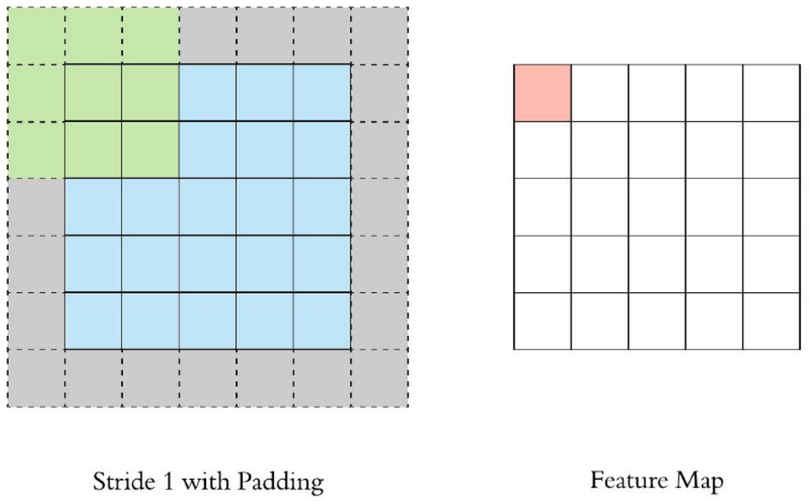
- In general, your CONV layers should use smaller **filter sizes such as 3x3 and 5x5**.
- Tiny **1x1 filters** are used to learn local features, but only in your more **advanced network architectures.**
- Larger filter sizes such as **7x7 and 11x11** may be used as the first CONV layer in the network (> 200x200 pixels). However, after this initial CONV layer the filter size should drop dramatically, otherwise you will reduce the spatial dimensions of your volume too quickly.

Best Practices



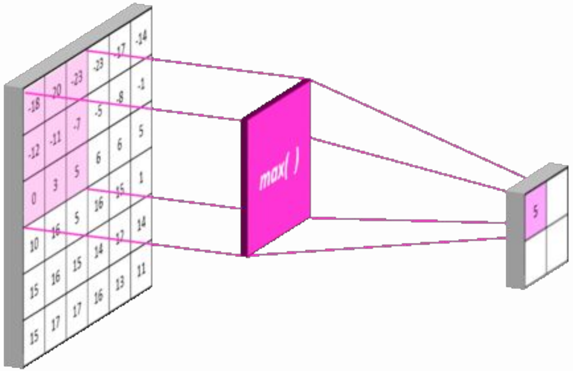
- Use a stride of **S = 1 for CONV layers**, at least for smaller spatial input volumes
- Networks that accept **larger input volumes** use a stride **S >= 2** in the first CONV layer to help reduce spatial dimensions.
- Using a stride of **S = 1 enables our CONV layers to learn filters while the POOL layer is responsible for downsampling**. However, keep in mind that not all network architectures follow this pattern – some architectures skip max pooling altogether and rely on the CONV stride to reduce volume size.

Best Practices

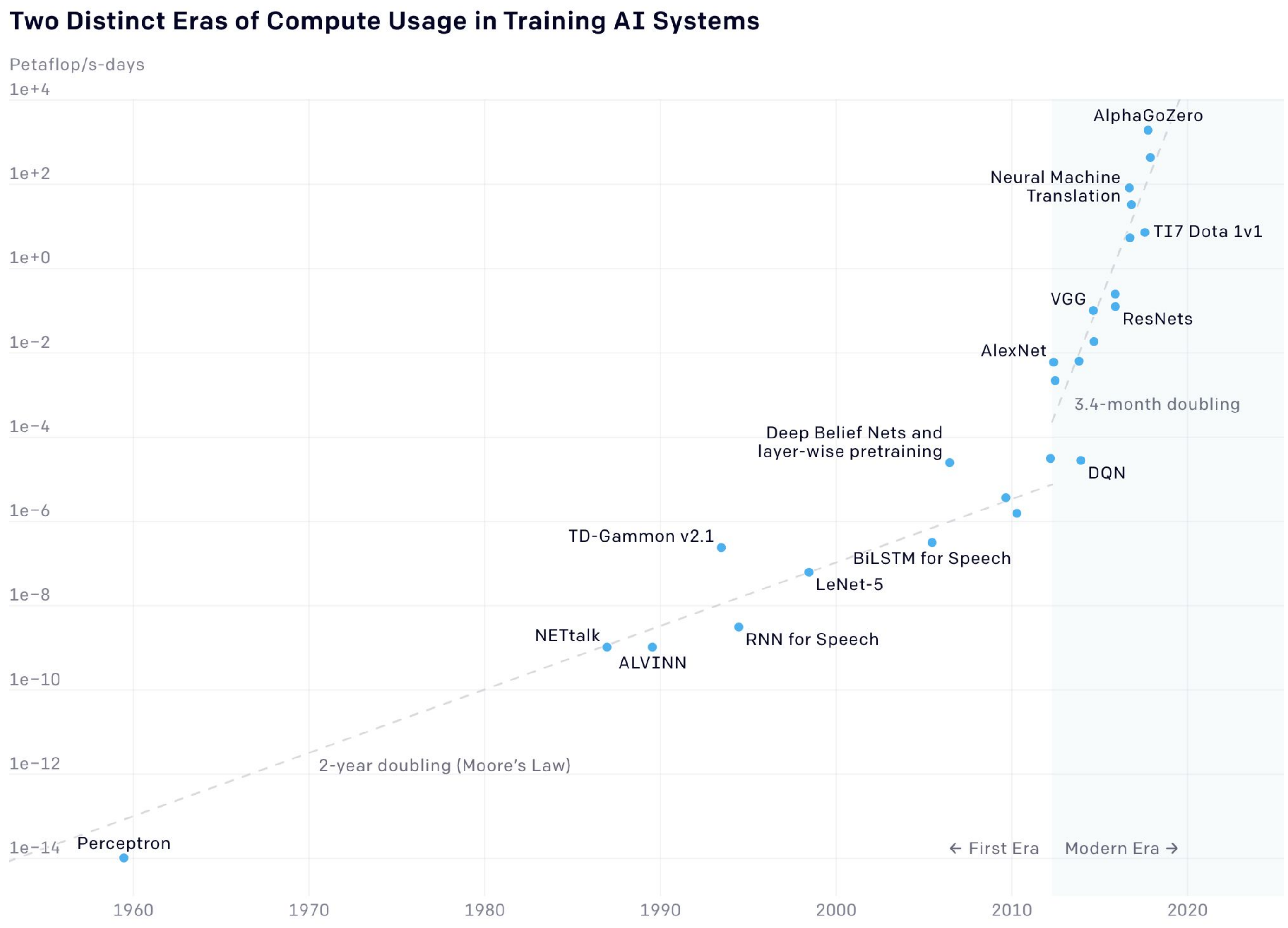


- Apply **zero-padding (same)** to my CONV layers to ensure the output dimension size matches the input dimension size – the only exception to this rule is if I want to purposely reduce spatial dimensions via convolution.
- Applying zero-padding when stacking multiple CONV layers on top of each other has also demonstrated to increase classification accuracy in practice.

Best Practices

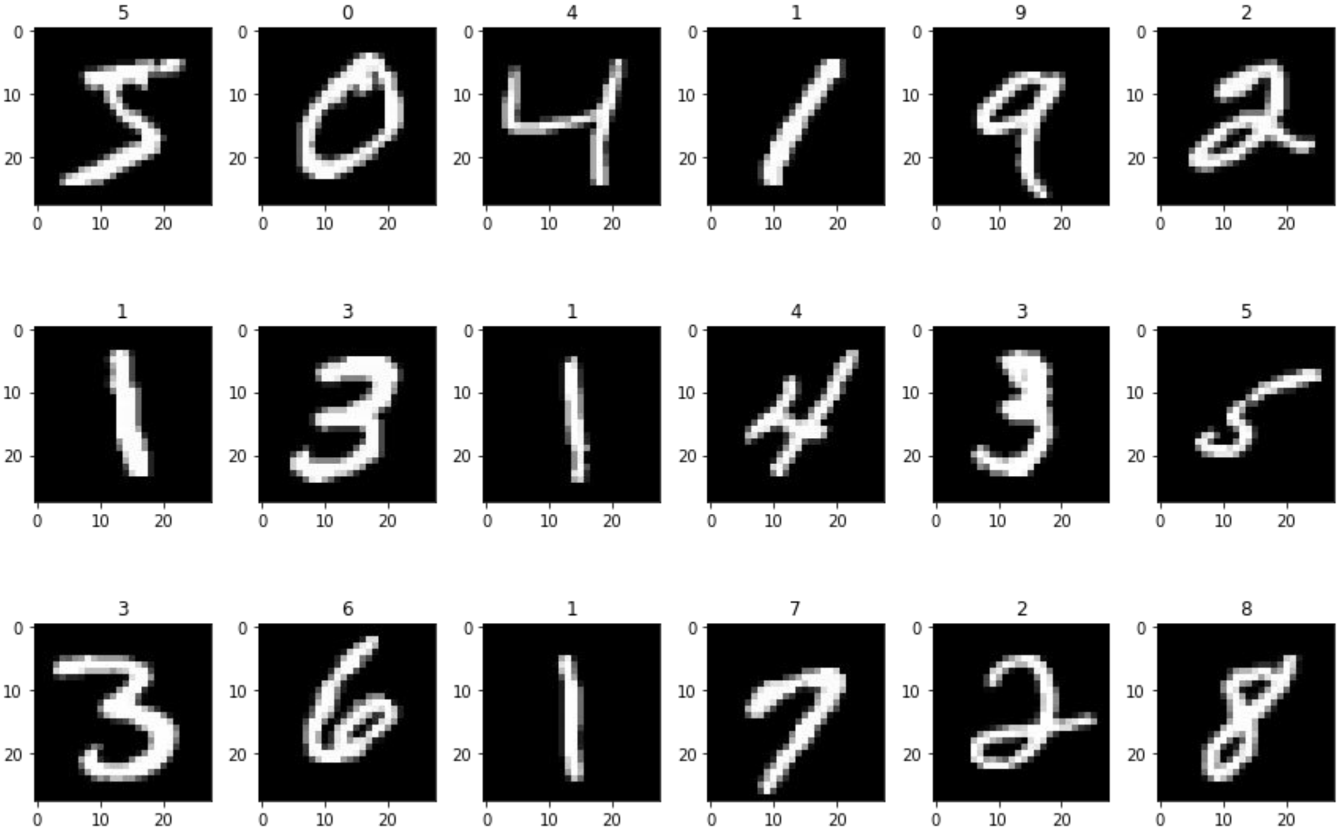


- It is recommended to **use POOL layers** (rather than CONV layers) **to reduce the spatial dimensions** of your input, at least until you become more experienced constructing your own CNN architectures. Once you reach that point, you should start experimenting with using CONV layers to reduce spatial input size and try removing max pooling layers from your architecture.
- Most commonly, you'll see **max pooling** applied over a **2x2 receptive field size** and a stride of **S = 2**.
- You might also see a **3x3 receptive field** early in the network architecture to help reduce image size.

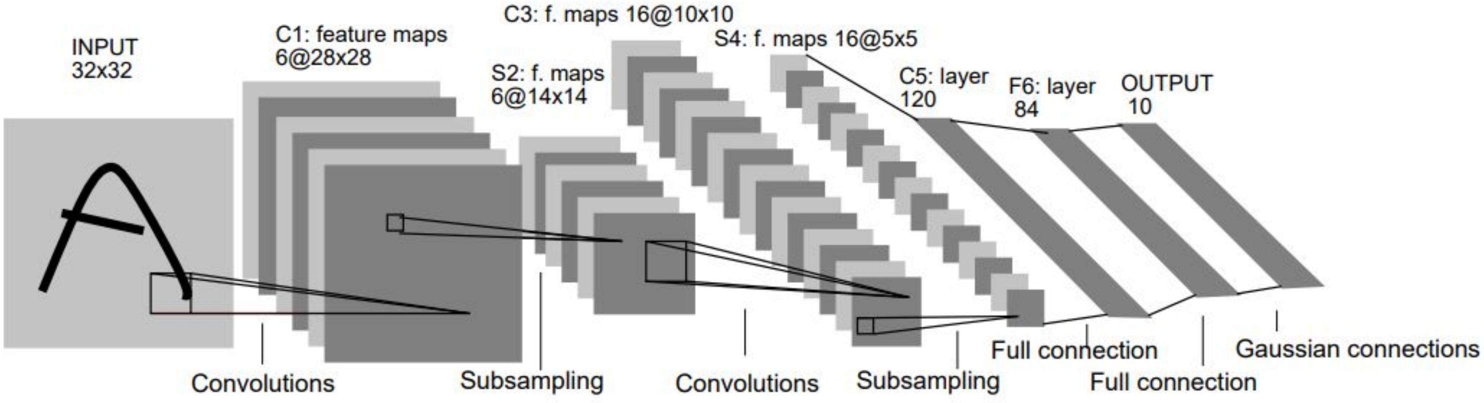


http://bit.do/twoeras

LeNet5



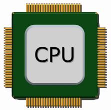
LeNet5




```python
# create model
lenet5 = Sequential()

lenet5.add(Conv2D(6, (5,5), strides=1,  activation='tanh',
input_shape=(28,28,1), padding='same')) #C1
lenet5.add(AveragePooling2D()) #S2
lenet5.add(Conv2D(16, (5,5), strides=1, activation='tanh', padding='valid')) #C3
lenet5.add(AveragePooling2D()) #S4
lenet5.add(Flatten()) #Flatten
lenet5.add(Dense(120, activation='tanh')) #C5
lenet5.add(Dense(84, activation='tanh')) #F6
lenet5.add(Dense(10, activation='softmax')) #Output layer
```

LeNet5


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 120)               48120
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```



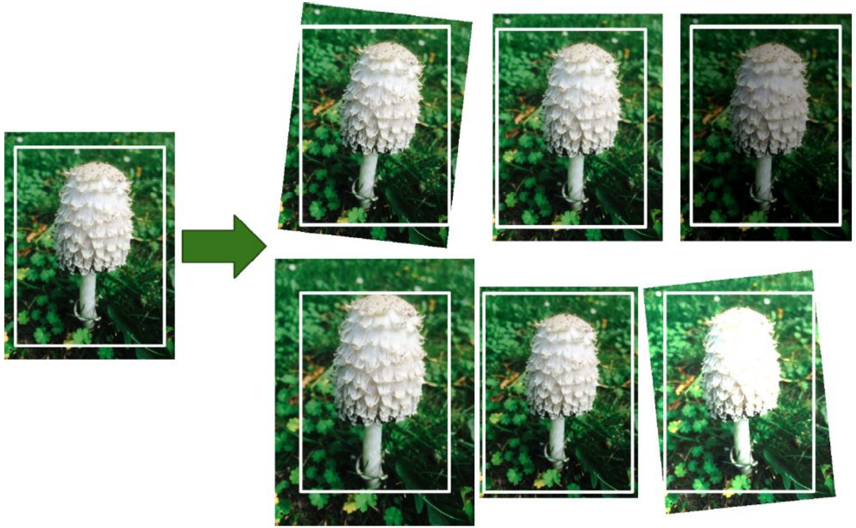
212 sec

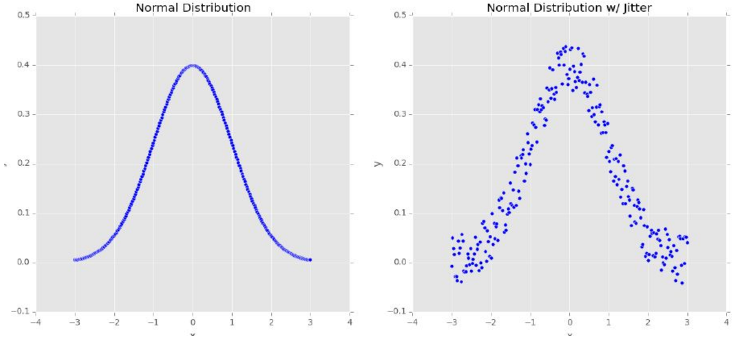
30 sec

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.99      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.99      0.99      0.99       982
           5       0.99      0.98      0.99       892
           6       0.98      0.99      0.98       958
           7       0.98      0.97      0.98      1028
           8       0.98      0.99      0.98       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

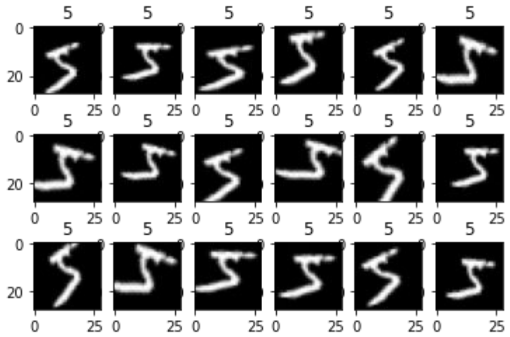
Data Augmentation

- When applying data augmentation is to increase the generalizability of the model
- In most cases, you'll see an increase in testing accuracy, perhaps at the expense at a slight dip in training accuracy



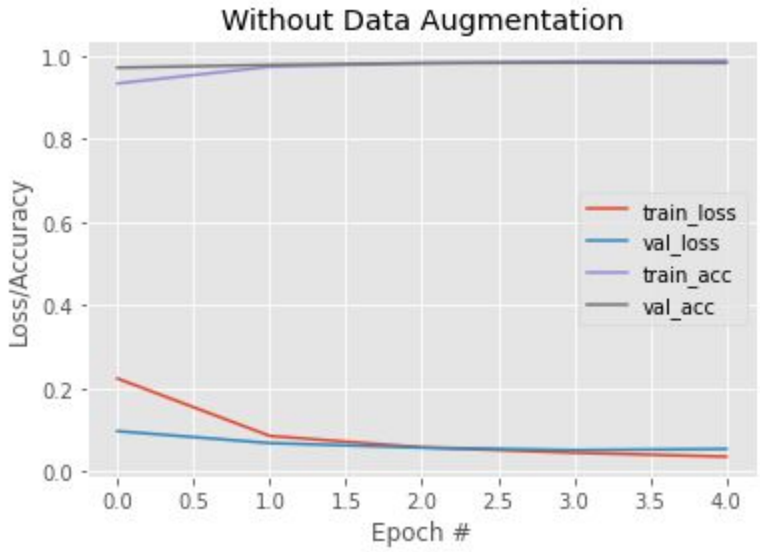


```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# construct the image generator for data augmentation then
# initialize the total number of images generated thus far
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
                         height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                         horizontal_flip=False, fill_mode="nearest")
total = 0
image = train_x[0:1,:,:,:]

# construct the actual Python generator
print("[INFO] generating images...")
imageGen = aug.flow(image, batch_size=1)

# loop over examples from our image data augmentation generator
for img in imageGen:

  show_image(img, train_y[0], total)

  # increment our counter
  total += 1

  # if we have reached 10 examples, break from the loop
  if total == 18:
    break
```

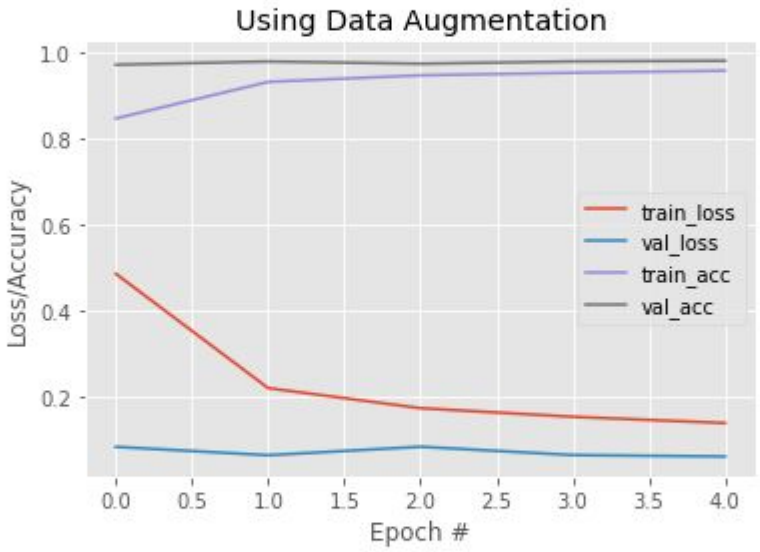


What change in training step when using data augmentation?

```
# construct the image generator for data augmentation then
# initialize the total number of images generated thus far
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
                         height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
                         horizontal_flip=False, fill_mode="nearest")

print("[INFO] training network...")
history = lenet5.fit(aug.flow(train_x, train_y, batch_size=32),
                     validation_data=(test_x, test_y),
                     #steps_per_epoch=len(train_x) // 32,
                     epochs=5, verbose=1,
                     callbacks=[MyCustomCallback()])
```



	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.99	0.99	0.99	1032
3	0.97	0.99	0.98	1010
4	0.97	0.99	0.98	982
5	0.97	0.99	0.98	892
6	0.97	0.99	0.98	958
7	0.98	0.99	0.98	1028
8	0.98	0.94	0.96	974
9	0.99	0.94	0.96	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000



	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.99	0.98	0.99	1032
3	0.97	0.99	0.98	1010
4	0.98	0.99	0.99	982
5	0.99	0.97	0.98	892
6	0.98	0.99	0.98	958
7	0.98	0.98	0.98	1028
8	0.97	0.98	0.98	974
9	0.99	0.96	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

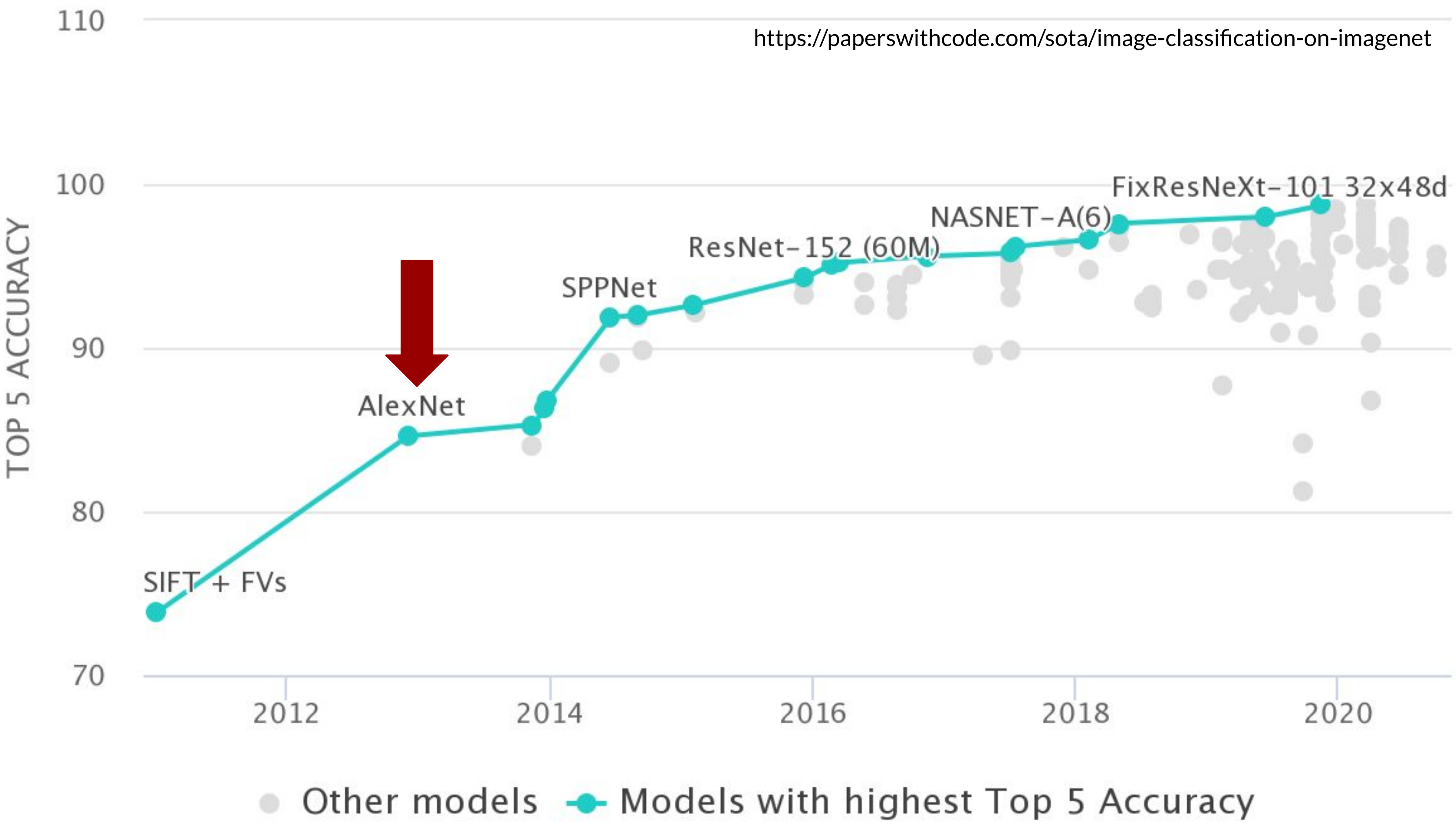
14,197,122 images, 21841 synsets indexed

http://www.image-net.org

1,000 image categories

Models are trained on ≈ 1.2M training images
50k images for validations (50 images per synset)
100k images for testing (100 images per synset)
rank-1 and rank-5 accuracies

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



http://papers.nips.cc/paper/4824-imagenet-classification-with-deepconvolutional-neural-networks.pdf

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

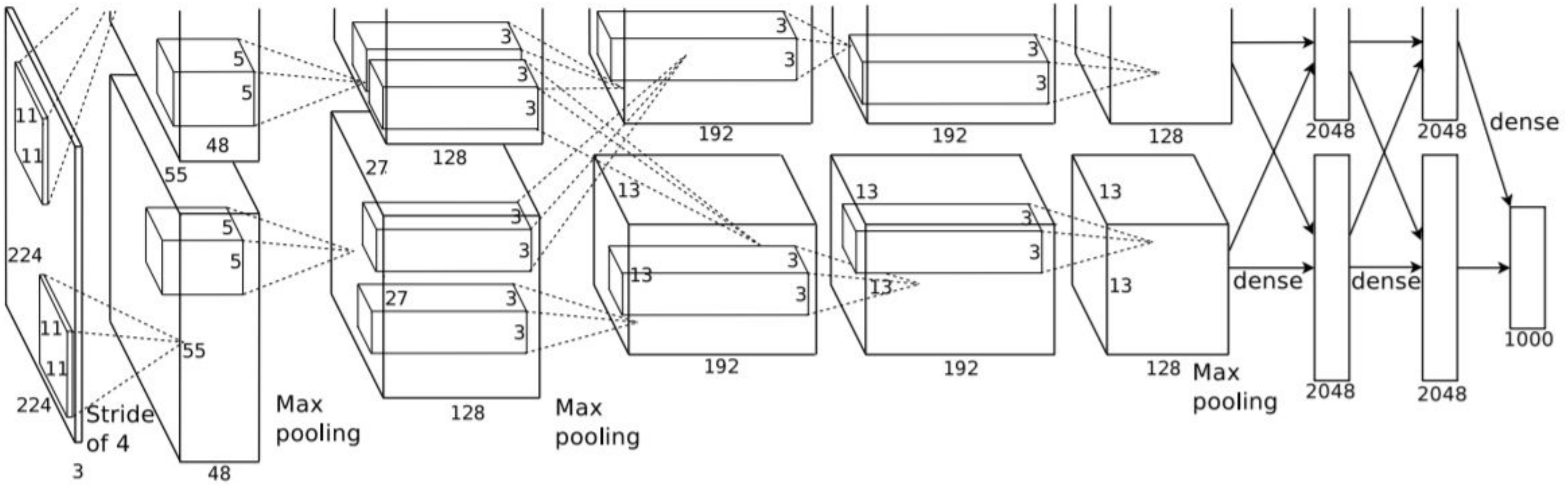
We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

1 Introduction

Current approaches to object recognition make essential use of machine learning methods. To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting. Until recently, datasets of labeled images were relatively small — on the order of tens of thousands of images (e.g., NORB [16], Caltech-101/256 [8, 9], and CIFAR-10/100 [12]). Simple recognition tasks can be solved quite well with datasets of this size, especially if they are augmented with label-preserving transformations. For example, the current-best error rate on the MNIST digit-recognition task (<0.3%) approaches human performance [4]. But objects in realistic settings exhibit considerable variability, so to learn to recognize them it is necessary to use much larger training sets. And indeed, the shortcomings of small image datasets have been widely recognized (e.g., Pinto et al. [21]), but it has only recently become possible to collect labeled datasets with millions of images. The new larger datasets include LabelMe [23], which consists of hundreds of thousands of fully-segmented images, and ImageNet [6], which consists of over 15 million labeled high-resolution images in over 22,000 categories.

To learn about thousands of objects from millions of images, we need a model with a large learning capacity. However, the immense complexity of the object recognition task means that this problem cannot be specified even by a dataset as large as ImageNet, so our model should also have lots of prior knowledge to compensate for all the data we don't have. Convolutional neural networks (CNNs) constitute one such class of models [16, 11, 13, 18, 15, 22, 26]. Their capacity can be con-

In: Advances in Neural Information Processing Systems 25. Edited by F. Pereira et al. Curran Associates, Inc., 2012, pages 1097–1105.



Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$227 \times 227 \times 3$	
CONV	$55 \times 55 \times 96$	$11 \times 11/4 \times 4, K = 96$
ACT	$55 \times 55 \times 96$	
BN	$55 \times 55 \times 96$	
POOL	$27 \times 27 \times 96$	$3 \times 3/2 \times 2$
DROPOUT	$27 \times 27 \times 96$	
CONV	$27 \times 27 \times 256$	$5 \times 5, K = 256$
ACT	$27 \times 27 \times 256$	
BN	$27 \times 27 \times 256$	
POOL	$13 \times 13 \times 256$	$3 \times 3/2 \times 2$
DROPOUT	$13 \times 13 \times 256$	
CONV	$13 \times 13 \times 384$	$3 \times 3, K = 384$
ACT	$13 \times 13 \times 384$	
BN	$13 \times 13 \times 384$	
CONV	$13 \times 13 \times 384$	$3 \times 3, K = 384$
ACT	$13 \times 13 \times 384$	
BN	$13 \times 13 \times 384$	
CONV	$13 \times 13 \times 256$	$3 \times 3, K = 256$
ACT	$13 \times 13 \times 256$	
BN	$13 \times 13 \times 256$	
POOL	$13 \times 13 \times 256$	$3 \times 3/2 \times 2$
DROPOUT	$6 \times 6 \times 256$	
FC	4096	
ACT	4096	
BN	4096	
DROPOUT	4096	
FC	4096	
ACT	4096	
BN	4096	
DROPOUT	4096	
FC	1000	
SOFTMAX	1000	

AlexNet
Blocks #01 and #02

```
# create a model
model = Sequential()

# Block #1: first CONV => RELU => POOL layer set
model.add(Conv2D(96, (11, 11), strides=(4, 4),
                 input_shape=(227,227,3), padding="valid",
                 kernel_regularizer=l2(0.0002),activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Dropout(0.25))

# Block #2: second CONV => RELU => POOL layer set
model.add(Conv2D(256, (5, 5), padding="same",
                 kernel_regularizer=l2(0.0002),activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Dropout(0.25))
```

Layer Type	Output Size	Filter Size / Stride
INPUT IMAGE	$227 \times 227 \times 3$	
CONV	$55 \times 55 \times 96$	$11 \times 11/4 \times 4, K = 96$
ACT	$55 \times 55 \times 96$	
BN	$55 \times 55 \times 96$	
POOL	$27 \times 27 \times 96$	$3 \times 3/2 \times 2$
DROPOUT	$27 \times 27 \times 96$	
CONV	$27 \times 27 \times 256$	$5 \times 5, K = 256$
ACT	$27 \times 27 \times 256$	
BN	$27 \times 27 \times 256$	
POOL	$13 \times 13 \times 256$	$3 \times 3/2 \times 2$
DROPOUT	$13 \times 13 \times 256$	

AlexNet
Block #03

```
# Block #3: CONV => RELU => CONV => RELU => CONV => RELU
model.add(Conv2D(384, (3, 3), padding="same",
                 kernel_regularizer=l2(0.0002),activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(384, (3, 3), padding="same",
                 kernel_regularizer=l2(0.002),activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(256, (3, 3), padding="same",
                 kernel_regularizer=l2(0.002),activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(Dropout(0.25))
```

CONV	$13 \times 13 \times 384$	$3 \times 3, K = 384$
ACT	$13 \times 13 \times 384$	
BN	$13 \times 13 \times 384$	
CONV	$13 \times 13 \times 384$	$3 \times 3, K = 384$
ACT	$13 \times 13 \times 384$	
BN	$13 \times 13 \times 384$	
CONV	$13 \times 13 \times 256$	$3 \times 3, K = 256$
ACT	$13 \times 13 \times 256$	
BN	$13 \times 13 \times 256$	
POOL	$13 \times 13 \times 256$	$3 \times 3/2 \times 2$
DROPOUT	$6 \times 6 \times 256$	

AlexNet
Blocks #04, #05

```
# Block #4: first set of FC => RELU layers
model.add(Flatten())
model.add(Dense(4096,kernel_regularizer=l2(0.0002),activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# Block #5: second set of FC => RELU layers
model.add(Dense(4096, kernel_regularizer=l2(0.0002),activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))

# softmax classifier
model.add(Dense(2, kernel_regularizer=l2(0.0002)))
model.add(Activation("softmax"))
```

FC	4096
ACT	4096
BN	4096
DROPOUT	4096
FC	4096
ACT	4096
BN	4096
DROPOUT	4096
FC	1000
SOFTMAX	1000

AlexNet

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 55, 55, 96)        34944
_________________________________________________________________
batch_normalization_21 (Batc (None, 55, 55, 96)        384
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 27, 27, 96)        0
_________________________________________________________________
dropout_15 (Dropout)         (None, 27, 27, 96)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 27, 27, 256)       614656
_________________________________________________________________
batch_normalization_22 (Batc (None, 27, 27, 256)       1024
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 256)       0
_________________________________________________________________
dropout_16 (Dropout)         (None, 13, 13, 256)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 13, 13, 384)       885120
_________________________________________________________________
batch_normalization_23 (Batc (None, 13, 13, 384)       1536
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 13, 13, 384)       1327488
_________________________________________________________________
batch_normalization_24 (Batc (None, 13, 13, 384)       1536
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 13, 13, 256)       884992
_________________________________________________________________
batch_normalization_25 (Batc (None, 13, 13, 256)       1024
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 6, 6, 256)         0
_________________________________________________________________
dropout_17 (Dropout)         (None, 6, 6, 256)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_7 (Dense)              (None, 4096)              37752832
_________________________________________________________________
batch_normalization_26 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_18 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_8 (Dense)              (None, 4096)              16781312
_________________________________________________________________
batch_normalization_27 (Batc (None, 4096)              16384
_________________________________________________________________
dropout_19 (Dropout)         (None, 4096)              0
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 8194
_________________________________________________________________
activation_1 (Activation)    (None, 2)                 0
=================================================================
Total params: 58,327,810
Trainable params: 58,308,674
Non-trainable params: 19,136
_________________________________________________________________
```

https://www.kaggle.com/c/dogs-vs-cats/overview

Dogs vs. Cats

Create an algorithm to distinguish dogs from cats

Kaggle · 213 teams · 7 years ago

Overview Data Notebooks Discussion Leaderboard Rules Team

Overview

- Description
- Prizes
- Evaluation
- Winners

In this competition, you'll write an algorithm to classify whether images contain either a dog or a cat. This is easy for humans, dogs, and cats. Your computer will find it a bit more difficult.

Lesson 12.ipynb

Train AlexNet to solve the Dogs & Cats challenge

Describe your solution on Medium