INTERNSHIP

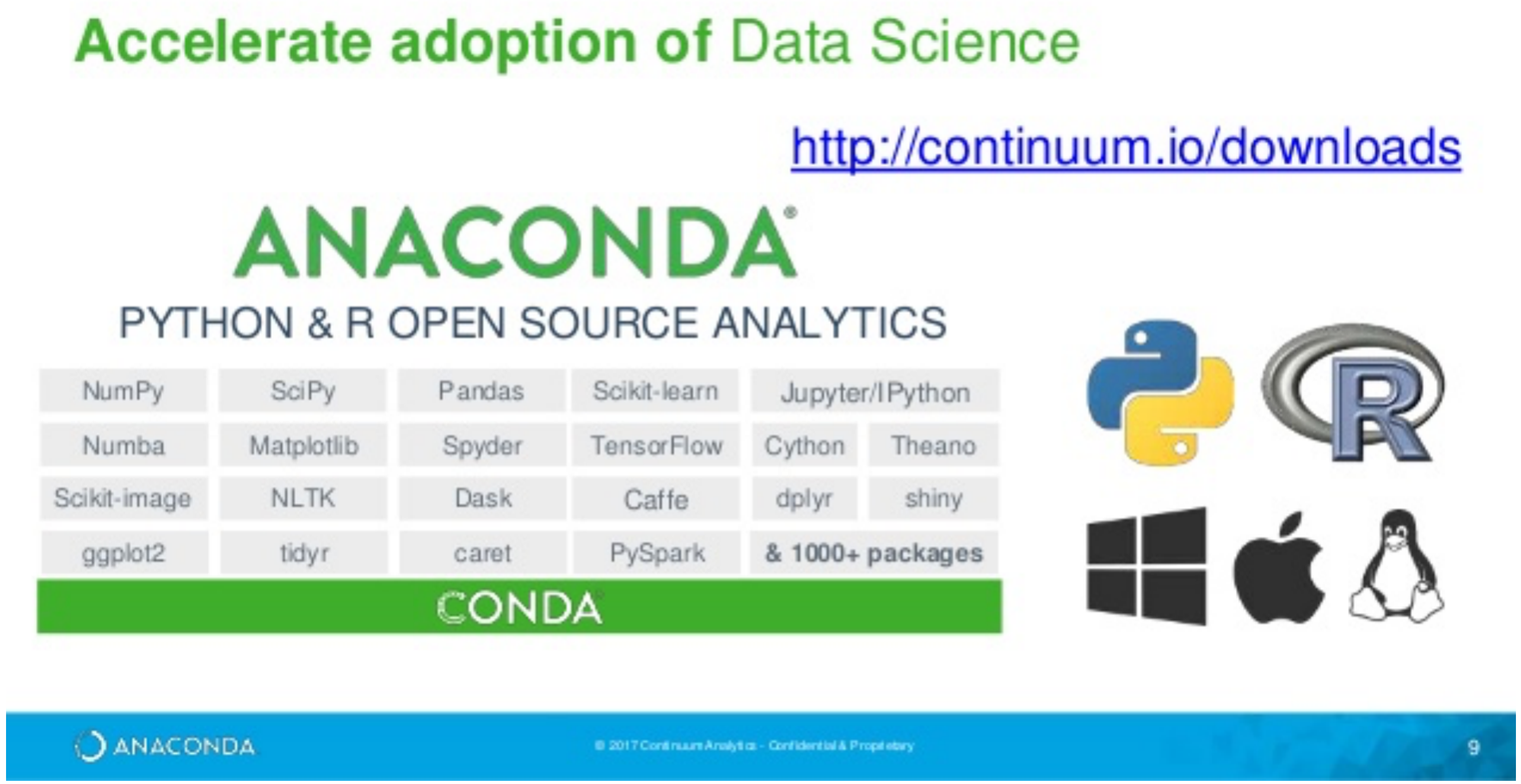
DEEP LEARNING

YTU ISL LAB.

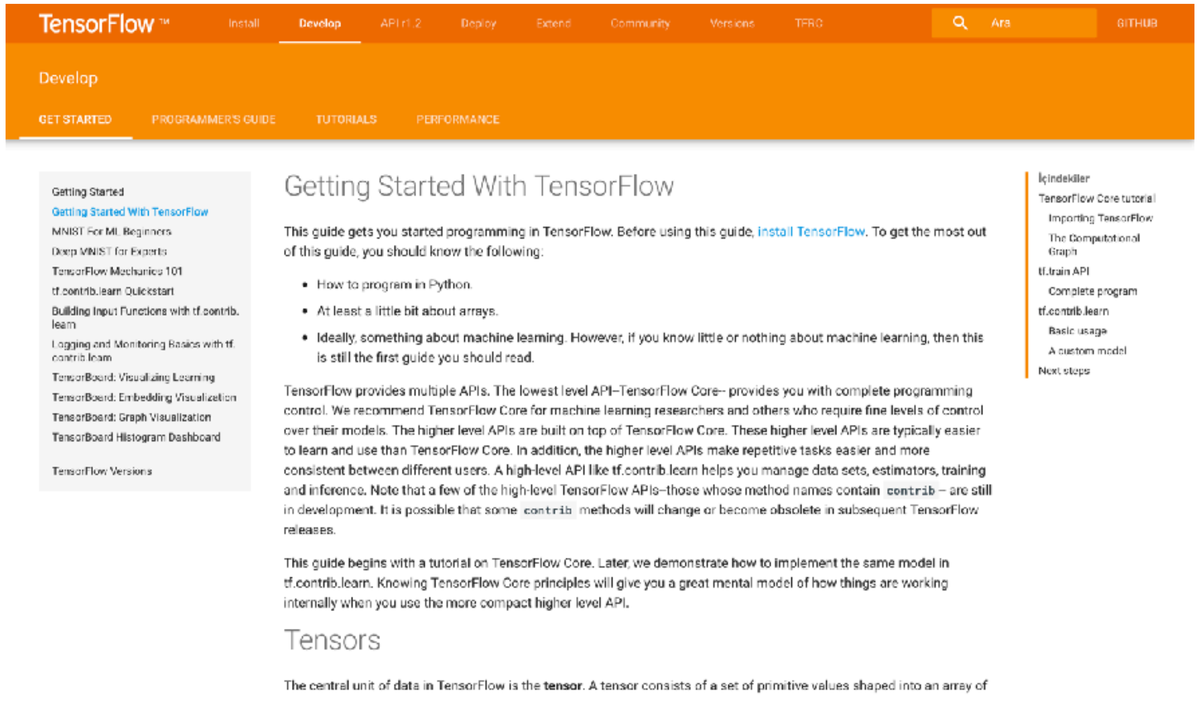
M.Yasin SAGLAM

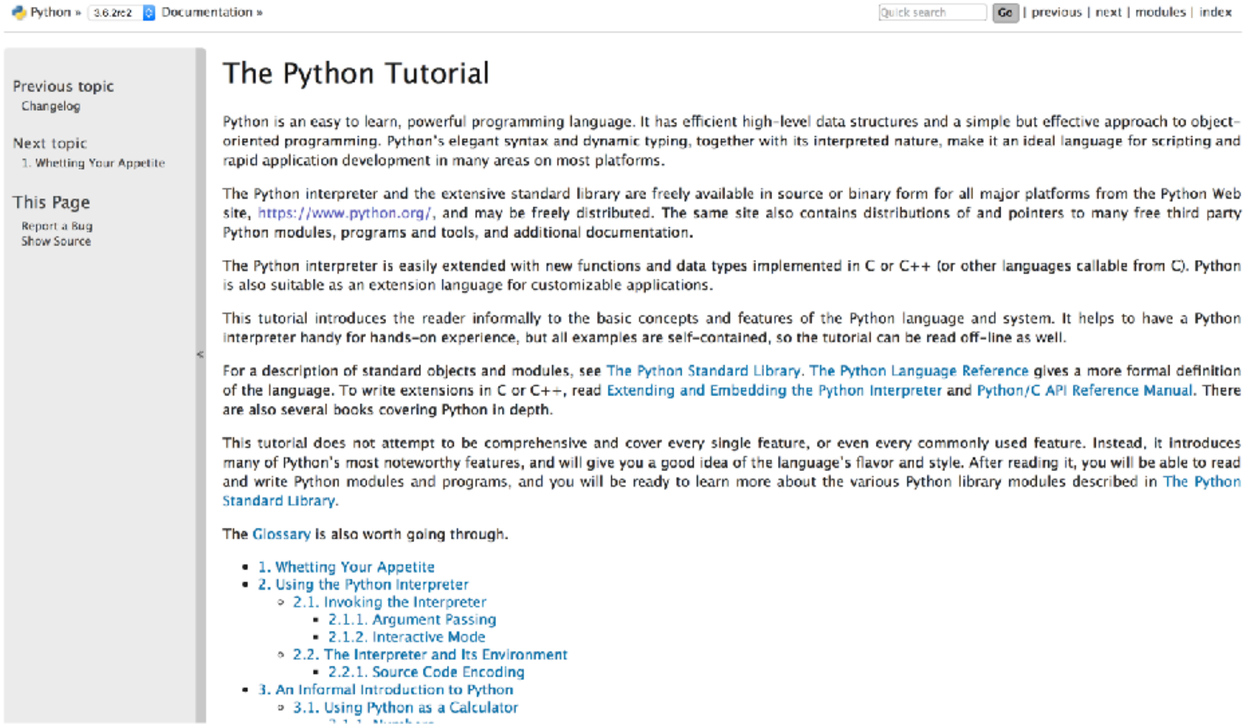
FRAMEWORK INSTALLATION

The installation of Python 3.6, Tensorflow 1.2 and other frameworks (numpy, opencv, matlib, pandas..) by ANACONDA



Python & Tensorflow Basics





https://www.tensorflow.org/get_started/get_started

https://docs.python.org/3/tutorial/index.html

CS 20SI: Tensorflow for Deep Learning Research

Assignment 3 is out. Due 3/17. See syllabus.

Course Description

Tensorflow is a powerful open-source software library for machine learning developed by researchers at Google Brain. It has many pre-built functions to ease the task of building different neural networks. Tensorflow allows distribution of computation across different computers, as well as multiple CPUs and GPUs within a single machine. TensorFlow provides a Python API, as well as a less documented C++ API. For this course, we will be using Python.

This course will cover the fundamentals and contemporary usage of the Tensorflow library for deep learning research. We aim to help students understand the graphical computational model of Tensorflow, explore the functions it has to offer, and learn how to build and structure models best suited for a deep learning project. Through the course, students will use Tensorflow to build models of different complexity, from simple linear/logistic regression to convolutional neural network and recurrent neural networks with LSTM to solve tasks such as word embeddings, translation, optical character recognition. Students will also learn best practices to structure a model and manage research experiments.

Detailed syllabus | Piazza forum | GitHub repo

Class Time and Location

Winter quarter (January - June, 2017)
Lecture: Wednesday, Friday 3:30-4:20
Location: 200-219

Instructor

Chip Huyen
huyenn [at] stanford [dot] edu
Office Hours by appointment

Grading Policy

There're three assignments, each graded on the check/check minus/check plus scale like in the CS106 series. You pass the course if:
+ You average a check in all three assignments
+ and you're reasonably active in class. I won't be taking attendance because it's really boring. But I expect to see you often in class.

Introduction to Artificial Neural Networks

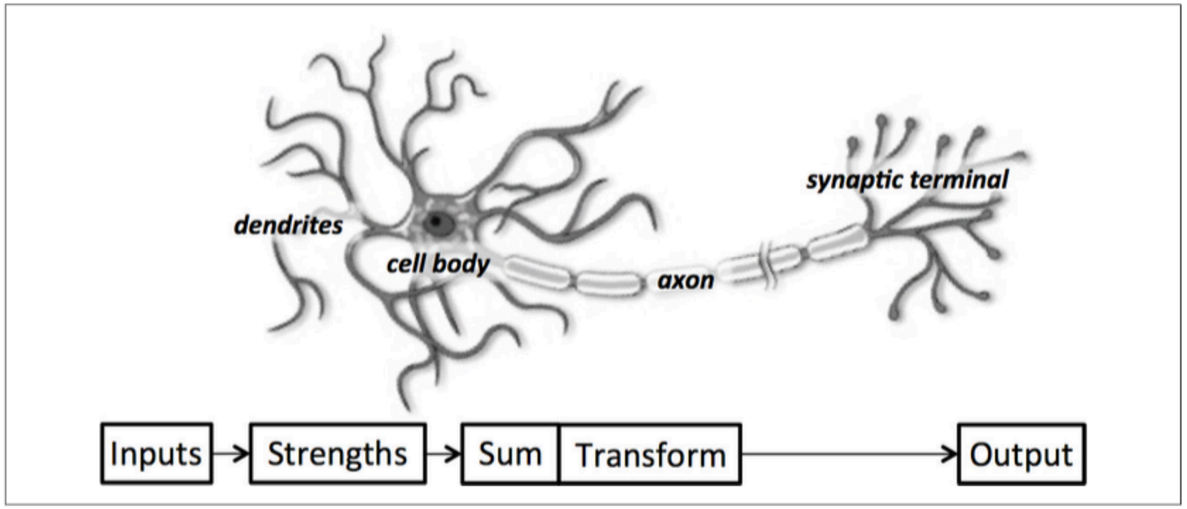


Figure 1-6. A functional description of a biological neuron's structure

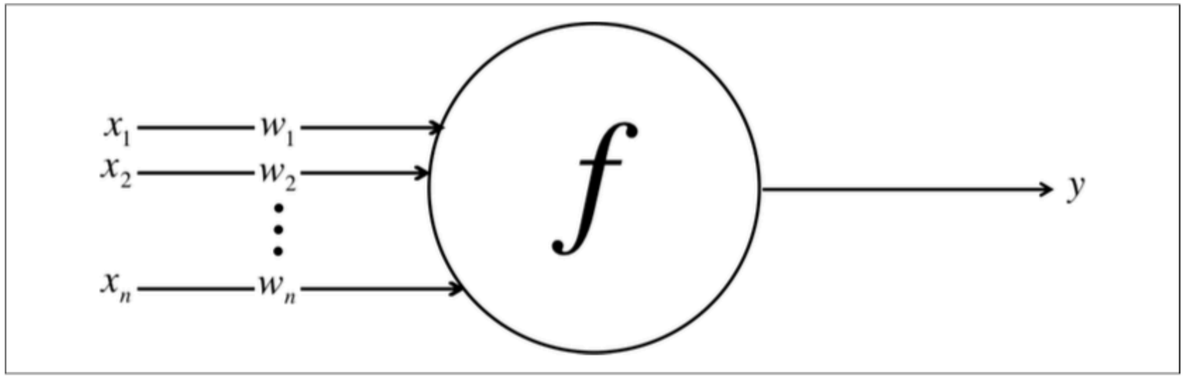


Figure 1-7. Schematic for a neuron in an artificial neural net

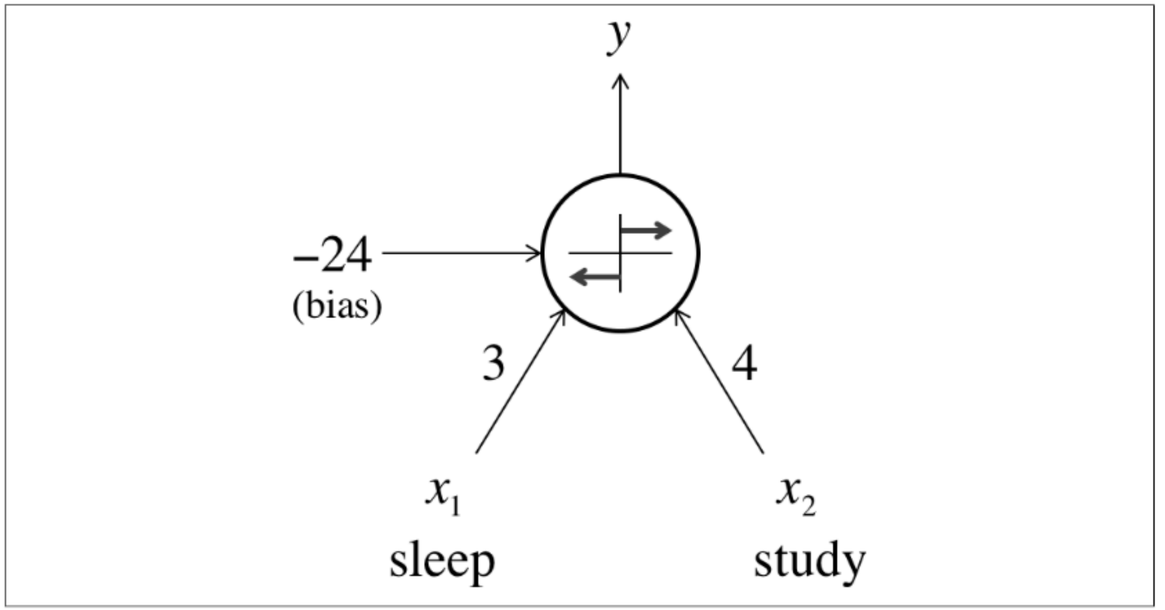


Figure 1-8. Expressing our exam performance perceptron as a neuron

Reference Book: ***Preview of O'Reilly's Fundamentals of Deep Learning***

Link : http://cdn2.hubspot.net/hubfs/443949/Fundamental_Deep_Learning_Preview_Zaloni.pdf

Linear Neurons and Their Limitations

Most neuron types are defined by the function f they apply to their logit z. Let's first consider layers of neurons that use a linear function in the form of $f(z) = az + b$. For example, a neuron that attempts to estimate a cost of a meal in a fast-food restaurant would use a linear neuron where $a = 1$ and $b = 0$. In other words, using $f(z) = z$ and weights equal to the price of each item, the linear neuron in Figure 1-10 would take in some ordered triple of servings of burgers, fries, and sodas and output the price of the combination.

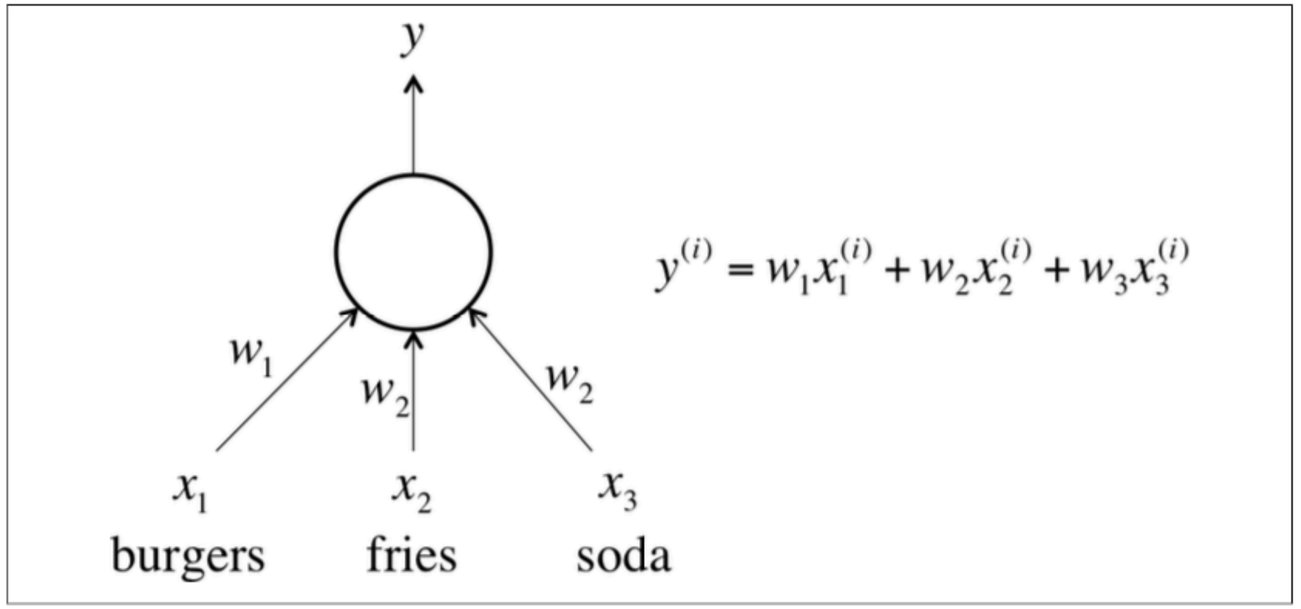


Figure 1-10. An example of a linear neuron

Sigmoid, Tanh, and ReLU Neurons

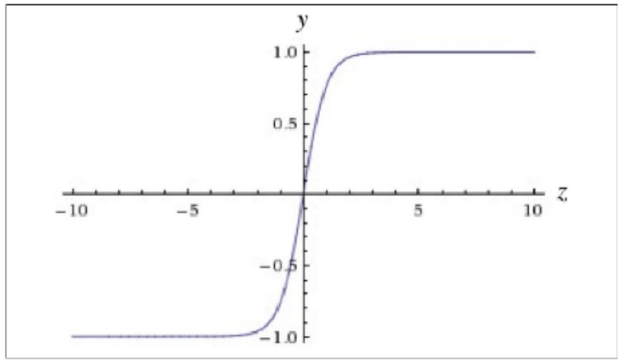


Figure 1-12. The output of a tanh neuron as z varies

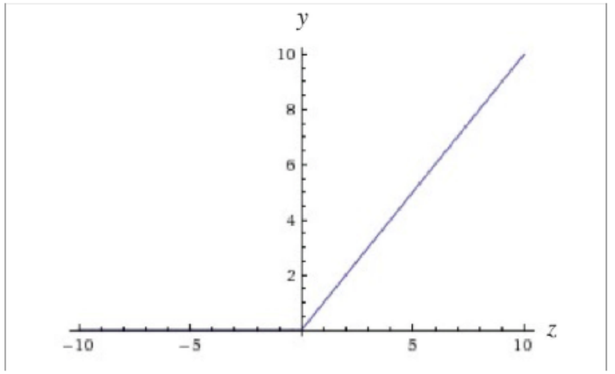


Figure 1-13. The output of a ReLU neuron as z varies

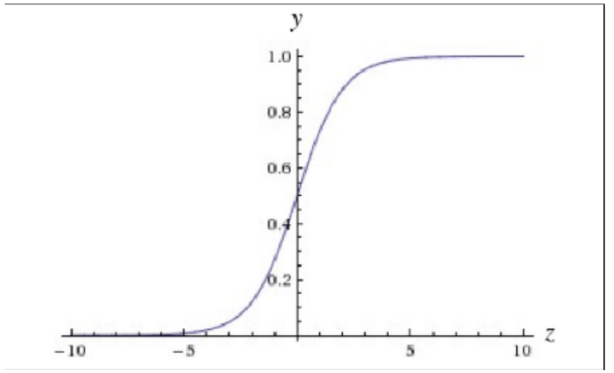


Figure 1-11. The output of a sigmoid neuron as z varies

Educational Video Series for Deep Learning Basics

Video 1
https://www.youtube.com/watch?v=BR9h47Jtqyw

Video 2
https://www.youtube.com/watch?v=ILsA4nyG7I0

Video 3
https://www.youtube.com/watch?v=oEGGr2K_v_4

Video 4
https://www.youtube.com/watch?v=Ukgii7Yd_cU

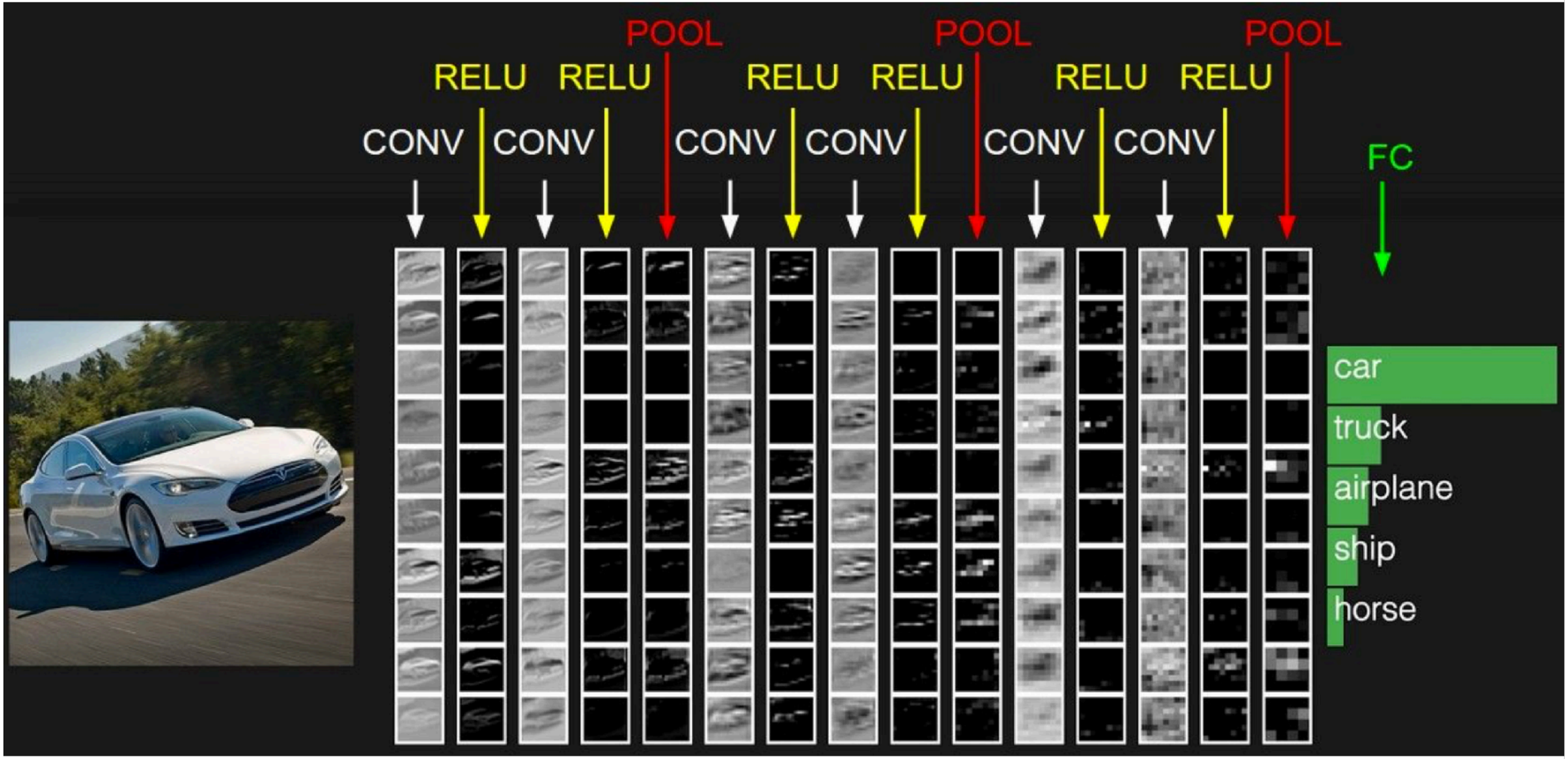
Video 5
https://www.youtube.com/watch?v=SQ67NBCLV98

Video 6
https://www.youtube.com/watch?v=_aCuOwF1ZjU

Video 7
https://youtu.be/wuo4JdG3SvU?list=PL9Hr9sNUjfsmEu1ZniY0XpHSzl5uihcXZ

Video 8
https://youtu.be/HMcx-zY8JSg?list=PL9Hr9sNUjfsmEu1ZniY0XpHSzl5uihcXZ

CONVOLUTIONAL NEURAL NETWORKS



e.g. 200K numbers → e.g. 10 numbers

CNN CASE STUDIES & RELATED PAPERS

CIFAR-10

who is the best in CIFAR-10 ?

CIFAR-10 49 results collected

Units: accuracy %

Classify 32x32 colour images.

Result	Method	Venue	Details
96.53%	Fractional Max-Pooling	arXiv 2015	Details
95.59%	Striving for Simplicity: The All Convolutional Net	ICLR 2015	Details
94.16%	All you need is a good init	ICLR 2016	Details
94%	Lessons learned from manually classifying CIFAR-10	unpublished 2011	Details
93.95%	Generalizing Pooling Functions in Convolutional Neural Networks: Mixed, Gated, and Tree	AISTATS 2016	Details
93.72%	Spatially-sparse convolutional neural networks	arXiv 2014	
93.63%	Scalable Bayesian Optimization Using Deep Neural Networks	ICML 2015	
93.57%	Deep Residual Learning for Image Recognition	arXiv 2015	Details
93.45%	Fast and Accurate Deep Network Learning by Exponential Linear Units	arXiv 2015	Details
93.34%	Universum Prescription: Regularization using	arXiv 2015	

Fractional Max-Pooling

Benjamin Graham
Dept of Statistics, University of Warwick, CV4 7AL, UK
b.graham@warwick.ac.uk

May 13, 2015

Abstract

Convolutional networks almost always incorporate some form of spatial pooling, and very often it is $\alpha \times \alpha$ max-pooling with $\alpha = 2$. Max-pooling act on the hidden layers of the network, reducing their size by an integer multiplicative factor α. The amazing by-product of discarding 75% of your data is that you build into the network a degree of invariance with respect to translations and elastic distortions. However, if you simply alternate convolutional layers with max-pooling layers, performance is limited due to the rapid reduction in spatial size, and the disjoint nature of the pooling regions. We have formulated a *fractional* version of max-pooling where α is allowed to take non-integer values. Our version of max-pooling is *stochastic* as there are lots of different ways of constructing suitable pooling regions. We find that our form of fractional max-pooling reduces overfitting on a variety of datasets: for instance, we improve on the state of the art for CIFAR-100 without even using dropout.

1 Convolutional neural networks

Convolutional networks are used to solve image recognition problems. They can be built by combining two types of layers:

- Layers of convolutional filters.
- Some form of spatial pooling, such as max-pooling.

Research focused on improving the convolutional layers has lead to a wealth of techniques such as dropout [10], DropConnect [12], deep networks with many small filters[2], large input layer filters for detecting texture [5], and deeply supervised networks [6].

By comparison, the humble pooling operation has been slightly neglected. For a long time 2×2 max-pooling (MP2 has been the default choice for building convolutional networks. There are many reasons for the popularity of MP2-pooling: it is fast, it quickly reduces the size of the hidden layers, and it encodes

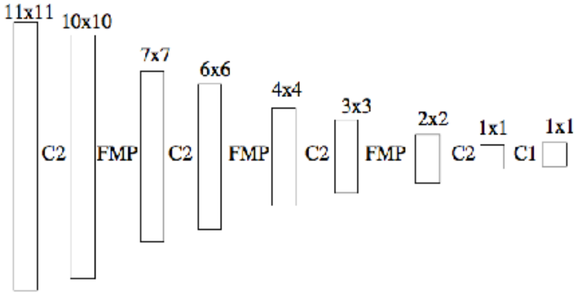


Figure 3: Layer sizes for a tiny FMP$\sqrt{2}$ network. The fractions $\frac{3}{2}$, $\frac{6}{4}$ and $\frac{10}{7}$ approximate $\sqrt{2}$.

Dataset and the number of repeat tests	pseudorandom disjoint	random disjoint	pseudorandom overlapping	random overlapping
MNIST, 1 test	0.54	0.57	**0.44**	0.50
MNIST, 12 tests	0.38	0.37	0.34	**0.32**
CIFAR-100, 1 test	31.67	32.06	**31.2**	31.45
CIFAR-100, 12 tests	28.48	27.89	28.16	**26.39**

Table 1: MNIST and CIFAR-100 % test errors.

Pooling method	None	Translations	Affine
6 layers of MP2	14.1	4.6	1.8
10 layers of FMP($\sqrt{2}$), 1 test	1.9	1.3	0.9
10 layers of FMP($\sqrt{2}$), 12 tests	**0.7**	**0.8**	**0.4**

Table 2: Assamese % test error with different type of data augmentation.

IMPLEMENTATION-1

- Implementing various filters to Lena.jpg

Salt & Pepper Problem

With Low Pass filter

With Gauss filter

Horizontal Sobel Filter

Diagonal Sobel Filter

Vertical Sobel Filter

Edge Filter

IMPLEMENTATION -2 MNIST

BASIC MNIST TUTORIAL

```python
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b

# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])

# The raw formulation of cross-entropy,
#
#   tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(tf.nn.softmax(y)),
#                                 reduction_indices=[1]))
#
# can be numerically unstable.
#
# So here we use tf.nn.softmax_cross_entropy_with_logits on the raw
# outputs of 'y', and then average across the batch.
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

sess = tf.InteractiveSession()
tf.global_variables_initializer().run()
# Train
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(500)
  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print(sess.run(accuracy, feed_dict={x: mnist.test.images,
                                    y_: mnist.test.labels}))

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str, default='/tmp/tensorflow/mnist/input_data',
                      help='Directory for storing input data')
```

rec | MNIST_advanced | linear_regression

```
/Users/Macbook/anaconda/python.app/Contents/MacOS/python /Users/Macbook/PycharmProjects/MNIST/rec.py
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
0.9209
```

ACCURACY 92,09 %

CONVOLUTIONAL MNIST TUTORIAL

```python
deepnn()
    x_image = tf.reshape(x, [-1, 28, 28, 1])

    # First convolutional layer - maps one grayscale image to 32 feature maps.
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])
    h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

    # Pooling layer - downsamples by 2X.
    h_pool1 = max_pool_2x2(h_conv1)

    # Second convolutional layer -- maps 32 feature maps to 64.
    W_conv2 = weight_variable([5, 5, 32, 64])
    b_conv2 = bias_variable([64])
    h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

    # Second pooling layer.
    h_pool2 = max_pool_2x2(h_conv2)

    # Fully connected layer 1 -- after 2 round of downsampling, our 28x28 image
    # is down to 7x7x64 feature maps -- maps this to 1024 features.
    W_fc1 = weight_variable([7 * 7 * 64, 1024])
    b_fc1 = bias_variable([1024])

    h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
    h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

    # Dropout - controls the complexity of the model, prevents co-adaptation of
    # features.
    keep_prob = tf.placeholder(tf.float32)
    h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

    # Map the 1024 features to 10 classes, one for each digit
    W_fc2 = weight_variable([1024, 10])
    b_fc2 = bias_variable([10])

    y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
    return y_conv, keep_prob
```

rec | MNIST_advanced | linear_regression

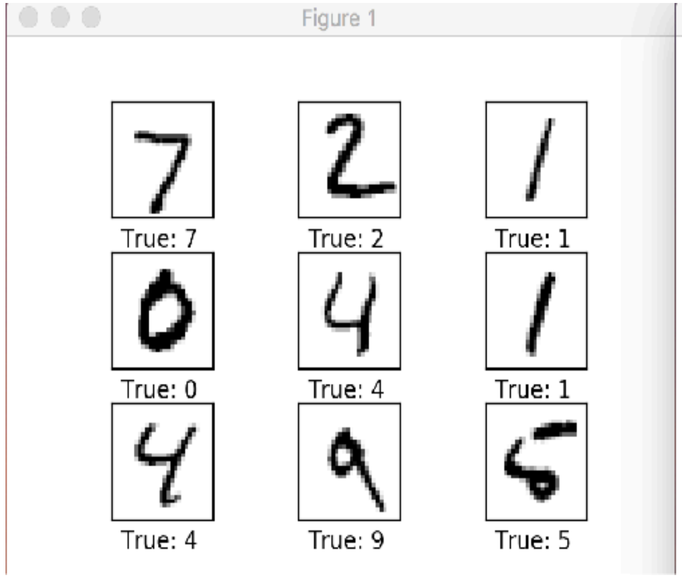
```
step 19995, training accuracy 1
step 19996, training accuracy 1
step 19997, training accuracy 1
step 19998, training accuracy 1
step 19999, training accuracy 1
test accuracy 0.9916

Process finished with exit code 0
```

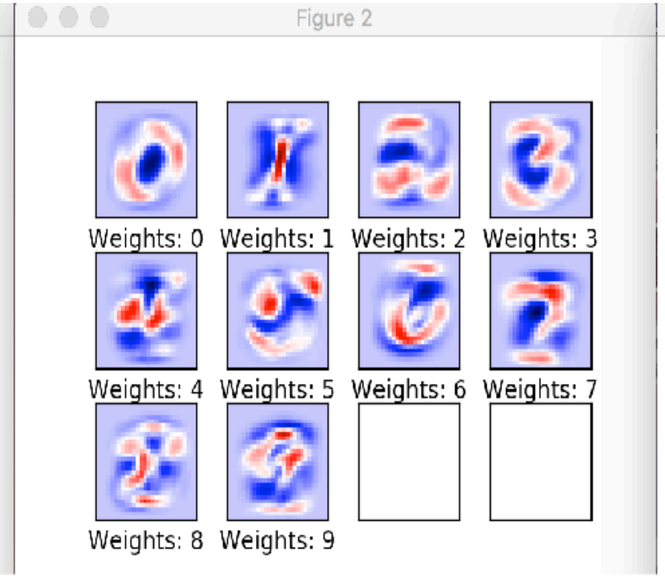
ACCURACY 99,06 %

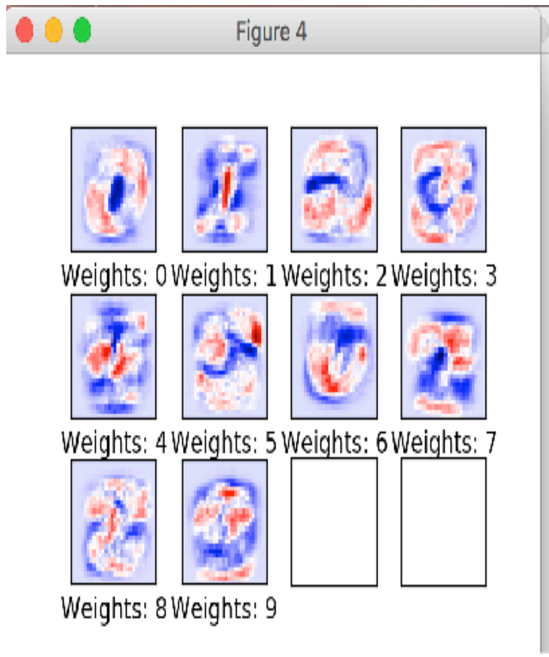
Figure 1-1. Image from MNIST handwritten digit dataset

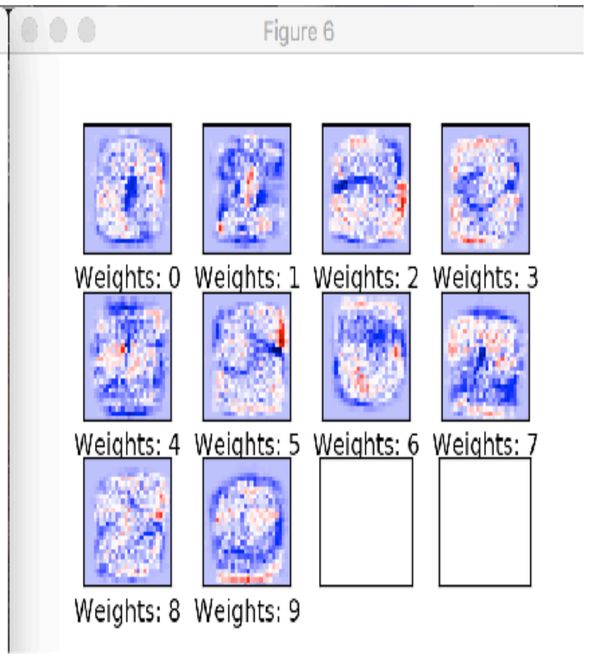
Weight Matrices of MNIST Processing

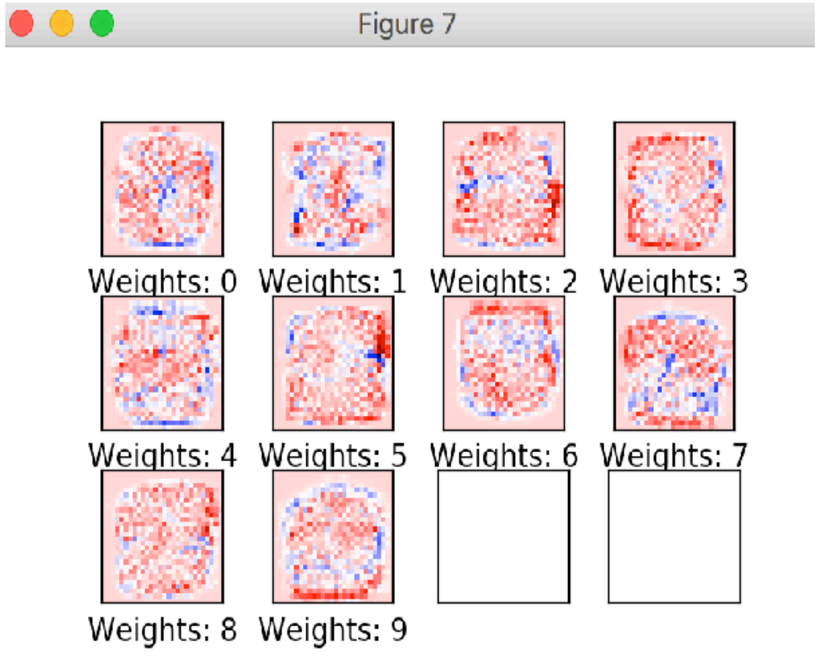
```
optimize(num_iterations=100)
plot_weights()
optimize(num_iterations=500)
plot_weights()
optimize(num_iterations=1000)
plot_weights()
optimize(num_iterations=5000)
plot_weights()
optimize(num_iterations=10000)
plot_weights()
optimize(num_iterations=50000)
plot_weights()
plt.show()
```



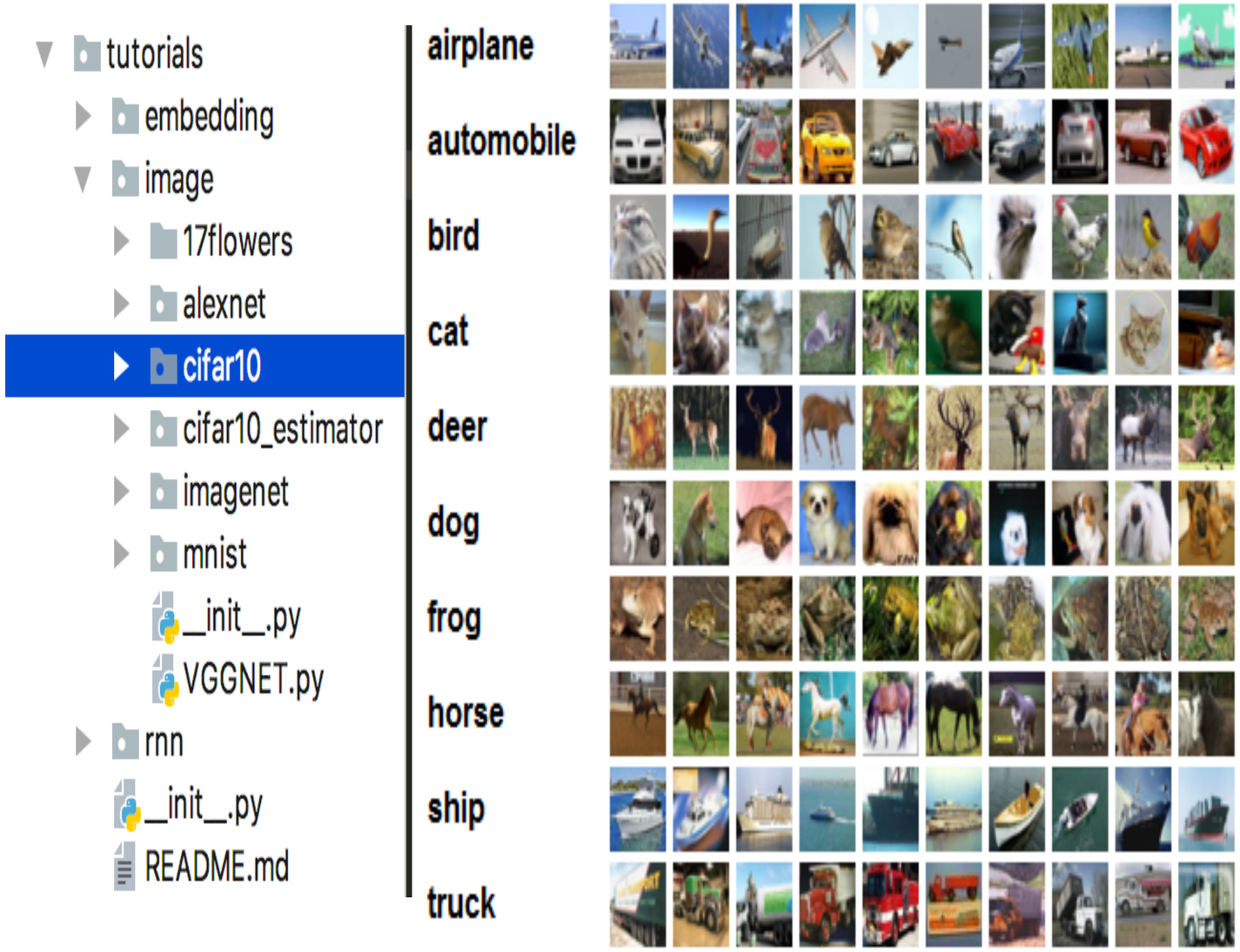








IMPLEMENTATION -3 IMAGE RECOGNITION TUTORIAL WITH CIFAR 10 AND IMAGENET DATASETS




```
  Args:
    image: Image file name.

  Returns:
    Nothing
  """
  if not tf.gfile.Exists(image):
    tf.logging.fatal('File does not exist %s', image)
  image_data = tf.gfile.FastGFile(image, 'rb').read()

  # Creates graph from saved GraphDef.
  create_graph()

  with tf.Session() as sess:
    # Some useful tensors:
    # 'softmax:0': A tensor containing the normalized prediction across
    #   1000 labels.
    # 'pool_3:0': A tensor containing the next-to-last layer containing 2048
    #   float description of the image.
    # 'DecodeJpeg/contents:0': A tensor containing a string providing JPEG
    #   encoding of the image.
    # Runs the softmax tensor by feeding the image_data as input to the graph.
    softmax_tensor = sess.graph.get_tensor_by_name('softmax:0')
    predictions = sess.run(softmax_tensor,
                           {'DecodeJpeg/contents:0': image_data})
    predictions = np.squeeze(predictions)

    # Creates node ID --> English string lookup.
    node_lookup = NodeLookup()

    top_k = predictions.argsort()[-FLAGS.num_top_predictions:][::-1]
    for node_id in top_k:
      human_string = node_lookup.id_to_string(node_id)
      score = predictions[node_id]
      print('%s (score = %.5f)' % (human_string, score))


def main(_):
  maybe_download_and_extract()
  image = (FLAGS.image_file if FLAGS.image_file else
           os.path.join(FLAGS.model_dir, 'cropped_panda.jpg'))
  run_inference_on_image(image)
```

rec | classify_image | linear_regression

```
giant panda, panda, panda bear, coon bear, Ailuropoda melanoleuca (score = 0.89107)
indri, indris, Indri indri, Indri brevicaudatus (score = 0.00779)
lesser panda, red panda, panda, bear cat, cat bear, Ailurus fulgens (score = 0.00296)
custard apple (score = 0.00147)
earthstar (score = 0.00117)
```

IMPLEMENTATION -4 CAT vs DOG CLASSIFICATION CHALLENGE

Input Image Shape = 64x64x3

All Images : **28088**

Train Images : **25279**

Test Images : **2809**

Cat = 14039 Dog = 14049

DATASET LINK

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

Model

```
# Input is a 32x32 image with 3 color channels (red, green and blue)
network = input_data(shape=[None, 64, 64, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug)

# 1: Convolution layer with 32 filters, each 3x3x3
conv_1 = conv_2d(network, 32, 3, activation='relu', name='conv_1')

# 2: Max pooling layer
network = max_pool_2d(conv_1, 2)

# 3: Convolution layer with 64 filters
conv_2 = conv_2d(network, 64, 3, activation='relu', name='conv_2')

# 4: Convolution layer with 64 filters
conv_3 = conv_2d(conv_2, 64, 3, activation='relu', name='conv_3')

# 5: Max pooling layer
network = max_pool_2d(conv_3, 2)

# 6: Fully-connected 512 node layer
network = fully_connected(network, 512, activation='relu')

# 7: Dropout layer to combat overfitting
network = dropout(network, 0.5)

# 8: Fully-connected layer with two outputs
network = fully_connected(network, 2, activation='softmax')

# Configure how the network will be trained
acc = Accuracy(name="Accuracy")
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.0005, metric=acc)

# Wrap the network in a model object
model = tflearn.DNN(network, checkpoint_path='model_cat_dog_7.tflearn', max_checkpoints=3,
                    tensorboard_verbose=3, tensorboard_dir='tmp/tflearn_logs/')

model.load('model_cat_dog_7.tflearn-10200')
###################################
# Train model for 100 epochs
###################################
model.fit(X, Y, validation_set=(X_test, Y_test), batch_size=250,
          n_epoch=100, run_id='model_cat_dog_7', show_metric=True)

model.save('model_cat_dog_7_final.tflearn')
```

Conv. Layer - 32 filter
Maxpooling Layer - 2x2
Conv. Layer - 64 filter
Conv. Layer - 64 filter
Maxpool - 2x2
FC Layer - 512 node
Dropout Layer-Rate = **0.5**
FC Layer - **2** node layer
Learning Rate = **0.0005**
Batch Size = **250**
Total Epoch = **250**

RESULTS

Test Score = **0.88**2164471766498 04

PREDICTED / ACTUAL	CAT	DOG	TOTAL
CAT	1192	**183**	1375
DOG	**148**	1286	1434

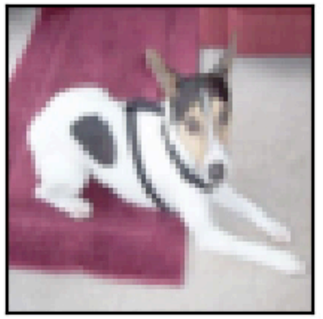
MISCLASSIFIED SAMPLES

0 - CAT

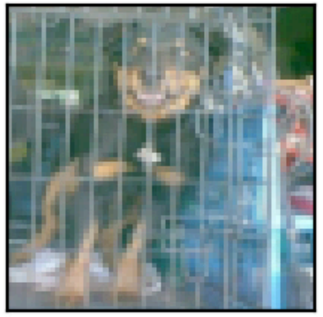
1 - DOG


True: 1, Pred: 0

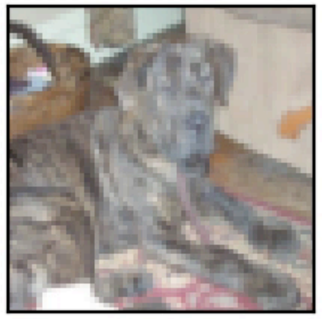

True: 0, Pred: 1

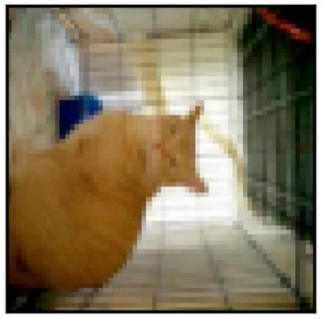

True: 1, Pred: 0


True: 1, Pred: 0


True: 1, Pred: 0


True: 0, Pred: 1


True: 1, Pred: 0


True: 0, Pred: 1

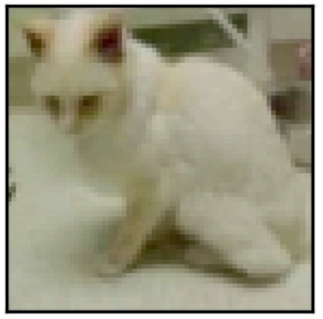

True: 0, Pred: 1

Image Retrieval

Image Representations

- Hand-Crafted Features (Before 2012)
  - SIFT
  - SIFT + BoW
  - VLAD
  - Fisher Vectors

Josef Sivic, Andrew Zisserman, et al. Video google: A text retrieval approach to object matching in videos. In ICCV, 2003.

Cortana Intelligence and Machine Learning Blog

Deep Learning Part 4: Content-Based Similar Image Retrieval using CNN

November 28, 2016 by Cortana Intelligence and ML Blog Team // 0 Comments

This post authored by Anusua Trivedi, Data Scientist, Microsoft.

Similar Image Retrieval

Once we have fine-tuned and retrained the pre-trained DCNN on the ACS image dataset, we follow the below steps:

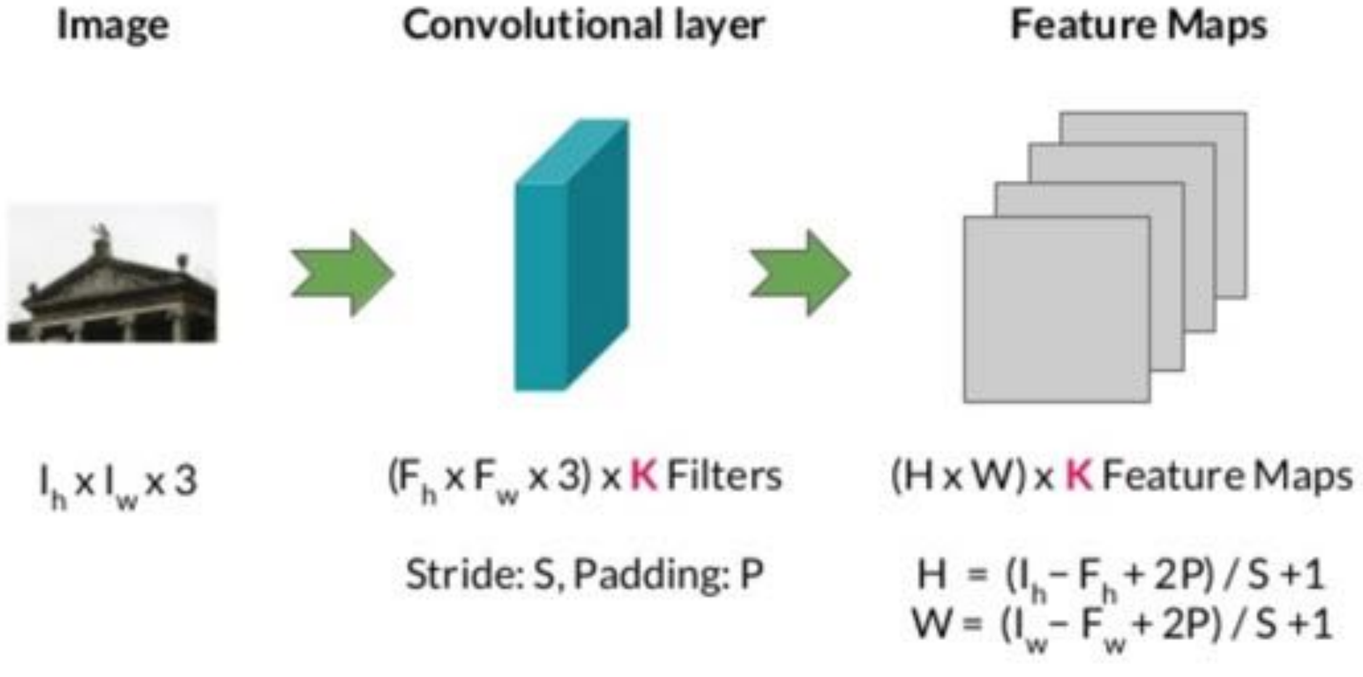
- Remove the last fully connected layer, which gives us the predicted label.
- Take the last pooled layer, which contains all the re-trained fashion image features.
- Calculate Euclidian distance among the extracted feature set to measure image similarity – less the distance, more similar the image.
- Run a traditional Nearest Neighbors Model on the extracted feature to calculate the Euclidian distance among the features [Figure 1].
- Create a scoring function to show the top three most similar images compared to the input/test image [Figure 2].

Here we have re-trained the pre-trained DCNN for fashion images. However, re-training of models is not always necessary, especially if there is a big overlap of pre-trained and test image sets. For example, you have an ImageNet pre-trained DCNN and you want to retrieve similar images of cats. The DCNN is already trained to identify cat images as part of the ImageNet dataset training. Hence no re-training is necessary in such cases. Just take an ImageNet pre-trained DCNN, get the image features from the last pool layer, and then calculate Euclidean distances on those pre-trained features.

Figure 3: Scoring function giving most similar image

Image Representations

From Feature Maps to Compact Representations



IMPLEMENTATION -5 IMAGE RETRIEVAL WITH OUR OWN DATASET FROM IMAGENET

CLASSES

PEOPLE

ANIMAL

FLOWER

ARTIFACT

SPORT

NATURE

FUNGUS

Total Class = ***7***

Train Images = ***1400***

(**200** images per class)

Test Images = ***137***

Input Image Shape

64x64x3

Dataset Link : http://image-net.org/explore

Model

```
################################
# Define network architecture
################################

# Input is a 64x64 image with 3 color channels (red, green and blue)
network = input_data(shape=[None, size_image, size_image, 3],
                     data_preprocessing=img_prep,
                     data_augmentation=img_aug)

# 1: Convolution layer with 64 filters, each 3x3x3
conv_1 = conv_2d(network, 64, 3, activation='relu', name='conv_1')

# 2: Max pooling layer
network = max_pool_2d(conv_1, 2)

# 3: Convolution layer with 128 filters
conv_2 = conv_2d(network, 128, 3, activation='relu', name='conv_2')

# 4: Convolution layer with 128 filters
conv_3 = conv_2d(conv_2, 128, 3, activation='relu', name='conv_3')

# 5: Max pooling layer
network = max_pool_2d(conv_3, 2)

# 6: Fully-connected 1024 node layer
network = fully_connected(network, 1024, activation='relu')

# 7: Dropout layer to combat overfitting
network = dropout(network, 0.75)

# 8: Fully-connected layer with 1400 outputs
network = fully_connected(network, class_size, activation='softmax')

# Configure how the network will be trained
acc = Accuracy(name="Accuracy")
network = regression(network, optimizer='adam',
                     loss='categorical_crossentropy',
                     learning_rate=0.0001, metric=acc)

# Wrap the network in a model object
model = tflearn.DNN(network, checkpoint_path='model_cbir_8.tflearn', max_checkpoints=3,
                    tensorboard_verbose=3, tensorboard_dir='/Users/Macbook/Desktop/imageretrievaldataset/tmp/tflearn_logsModel8/')
################################
# Train model for 200 epochs
################################
model.fit(X, Y, batch_size=32,
          n_epoch=200, run_id='model_cbir_8', show_metric=True)

model.save('model_cbir_8_final.tflearn')
```

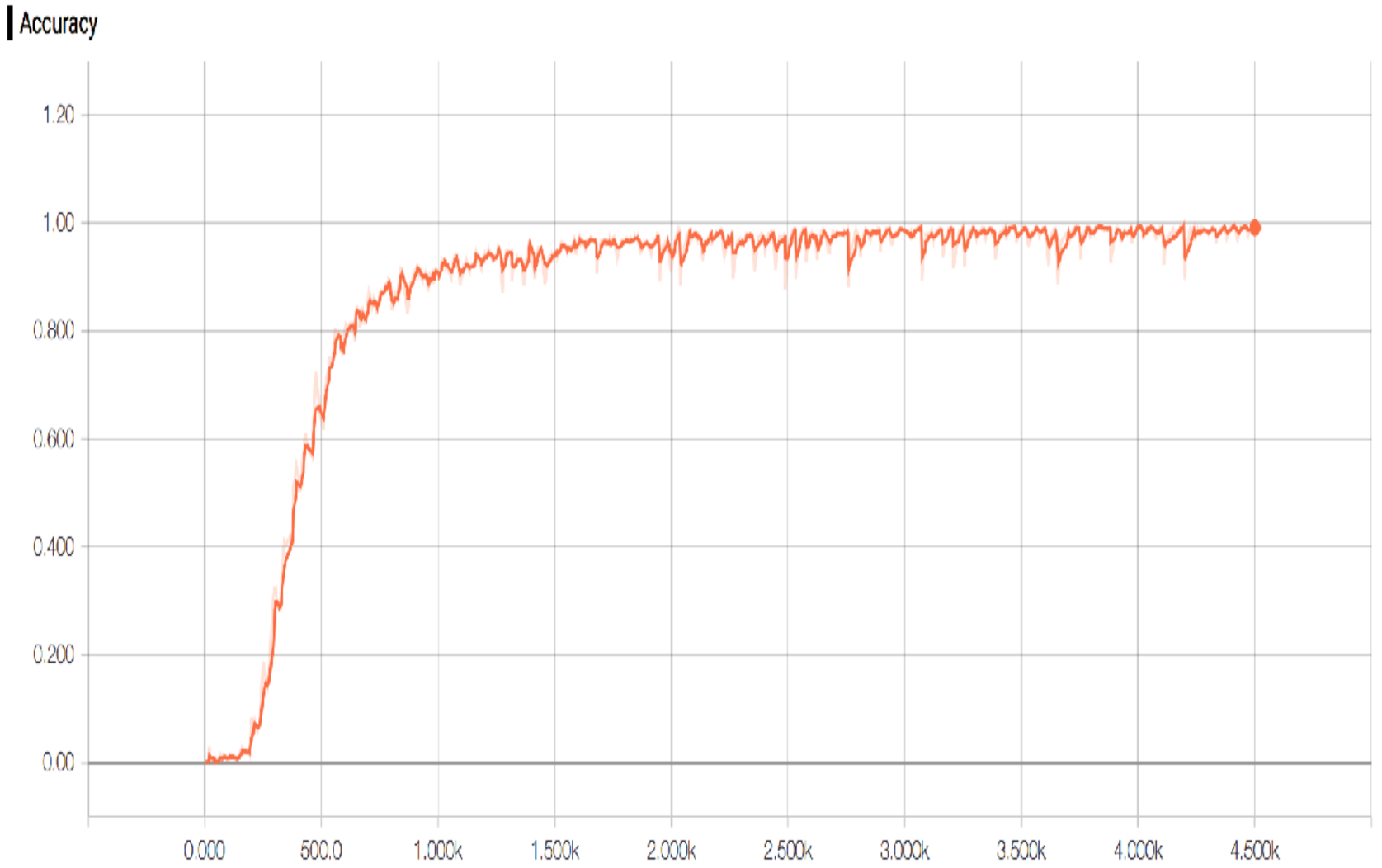
-Conv. Layer - 64 filter
-Maxpooling Layer - 2x2
-Conv. Layer - 128 filter
-Conv. Layer - 128 filter
-Maxpool - 2x2
-FC Layer - 1024 node
-Dropout Layer-Rate = **0.75**
-FC Layer - **1400** node layer

(assume every train image as class)

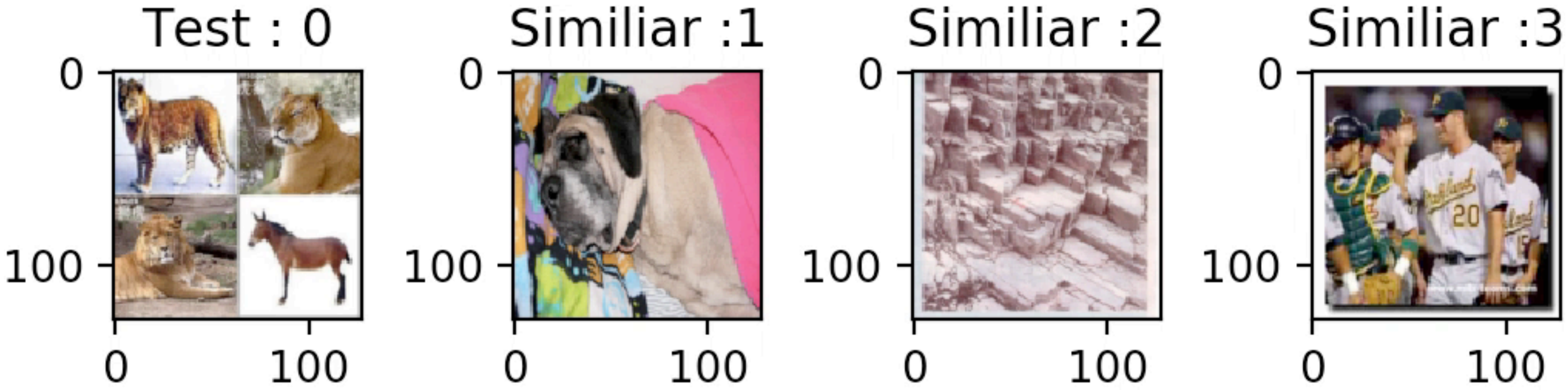
-Learning Rate = **0.0001**
-Batch Size = **32**
-Total Epoch = **200**

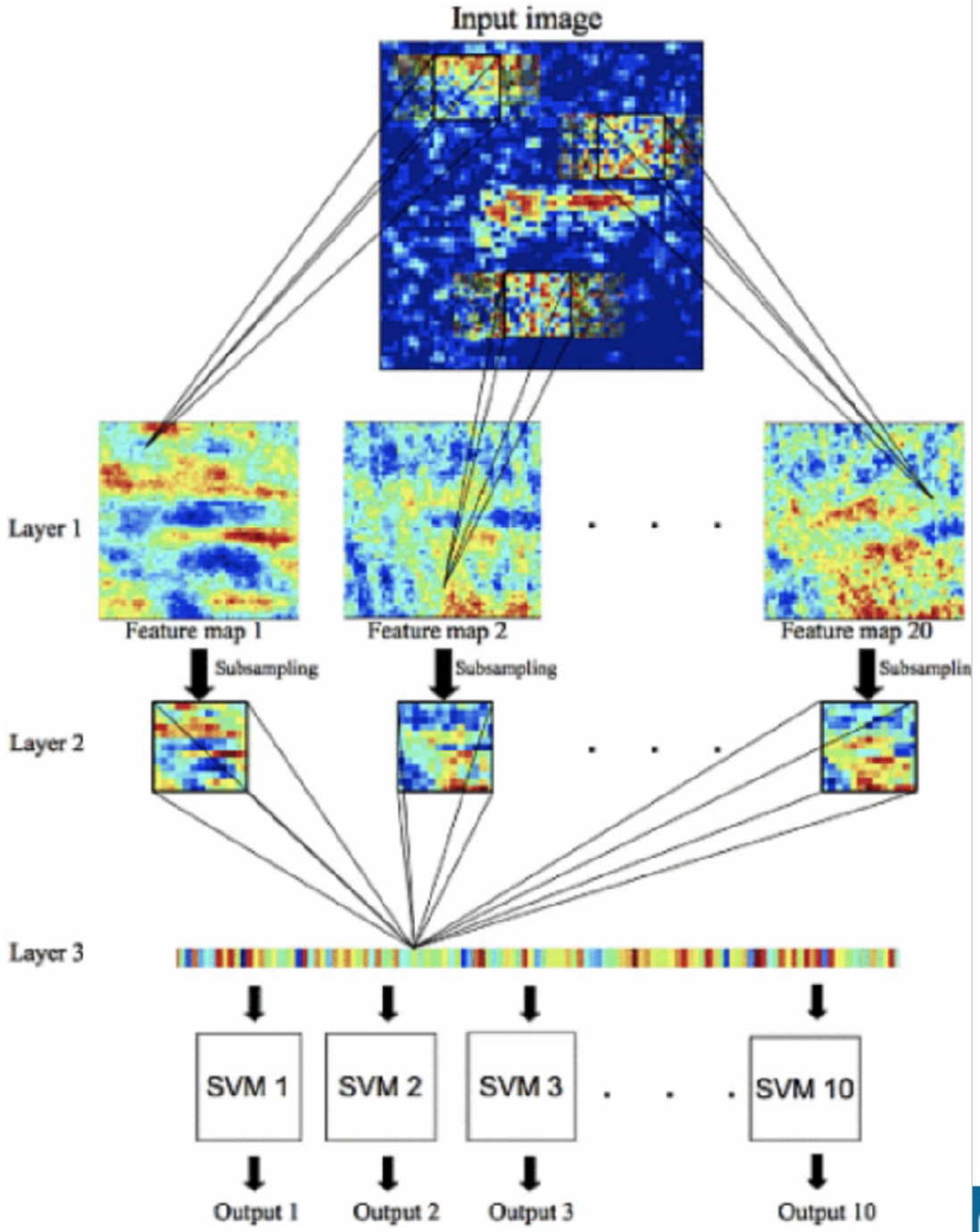
| Adam | epoch: 200 | loss: 0.02052 - Accuracy: 0.9991 -- iter: 0864/1400
Training Step: 8784 | total loss: 0.07611 | time: 77.598s
| Adam | epoch: 200 | loss: 0.07611 - Accuracy: 0.9960 -- iter: 0896/1400
Training Step: 8785 | total loss: 0.06918 | time: 80.426s
| Adam | epoch: 200 | loss: 0.06918 - Accuracy: 0.9964 -- iter: 0928/1400
Training Step: 8786 | total loss: 0.06234 | time: 83.239s
| Adam | epoch: 200 | loss: 0.06234 - Accuracy: 0.9968 -- iter: 0960/1400
Training Step: 8787 | total loss: 0.07161 | time: 85.925s
| Adam | epoch: 200 | loss: 0.07161 - Accuracy: 0.9940 -- iter: 0992/1400
Training Step: 8788 | total loss: 0.78645 | time: 88.732s
| Adam | epoch: 200 | loss: 0.78645 - Accuracy: 0.9602 -- iter: 1024/1400
Training Step: 8789 | total loss: 0.70829 | time: 91.566s
| Adam | epoch: 200 | loss: 0.70829 - Accuracy: 0.9642 -- iter: 1056/1400
Training Step: 8790 | total loss: 0.63773 | time: 94.391s
| Adam | epoch: 200 | loss: 0.63773 - Accuracy: 0.9678 -- iter: 1088/1400
Training Step: 8791 | total loss: 0.57468 | time: 97.174s
| Adam | epoch: 200 | loss: 0.57468 - Accuracy: 0.9710 -- iter: 1120/1400
Training Step: 8792 | total loss: 0.51731 | time: 100.022s
| Adam | epoch: 200 | loss: 0.51731 - Accuracy: 0.9739 -- iter: 1152/1400
Training Step: 8793 | total loss: 0.49251 | time: 102.815s

TensorBoard SCALARS IMAGES AUDIO GRAPHS DISTRIBUTIONS HISTOGRAMS EMBEDDINGS TEXT



RESULTS





Classifier	P_{CC}	P_{err}
CNN	90.48%	9.52%
CNN-SVM linear	89.14%	10.86%
CNN-SVM quadratic	97.75%	2.25%
CNN-SVM poly 3	98.16%	1.84%
CNN-SVM poly 4	98.22%	1.78%
CNN-SVM poly 5	93.97%	6.03%
CNN-SVM rbf 1	75.05%	24.95%
CNN-SVM rbf 2	98.56%	1.44%
CNN-SVM rbf 2.4	98.56%	1.44%
CNN-SVM rbf 3	98.00%	2.00%
CNN-SVM rbf 4	97.22%	2.78%
Gaussian Model [11]	97.18%	2.82%
Sparse Representation [12]	94.69%	5.31%
Nearest Neighbor [13]	86.78%	13.22%

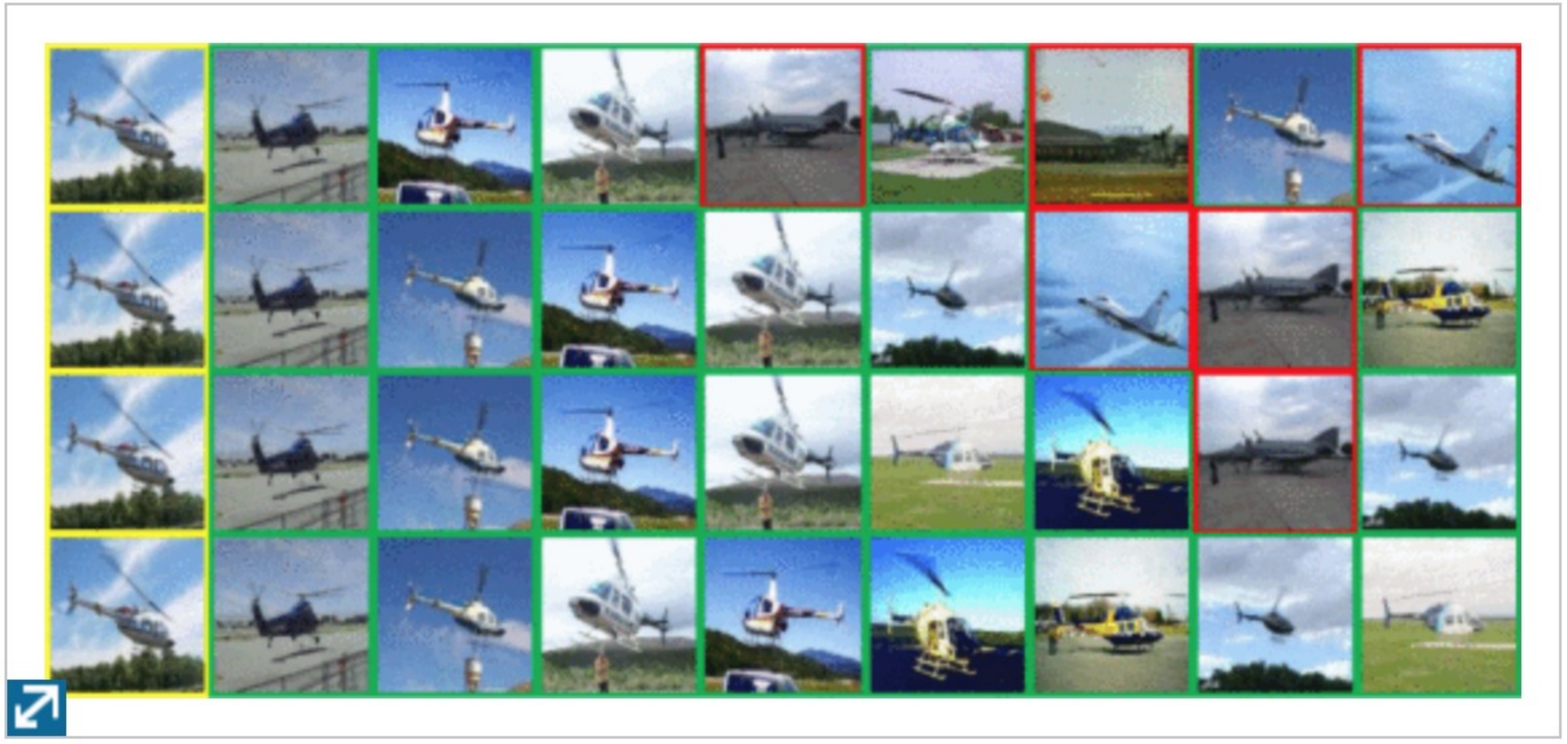
Figure 1.
The architecture of VGGnet in this paper.

Baseline

PCA128 ve SVM

PCA512 ve SVM

DF ve SVM



IMPLEMENTATION -6 IMAGE RETRIEVAL WITH OUR OWN MORE SPESIFIC DATASET FROM IMAGENET

CLASSES

LION

FISH

CELLPHONE

KEY

CAR

NATURE

RED FLOWER

Total Class = ***7***

Train Images = ***1400***

(**200** images per class)

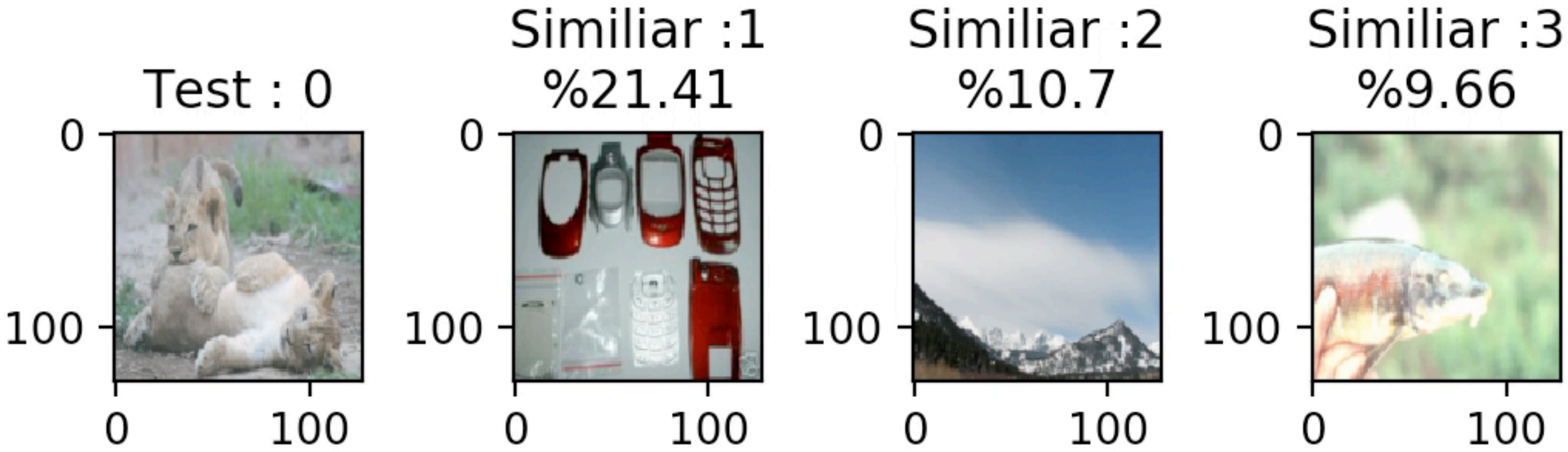
Test Images = ***140***

Input Image Shape

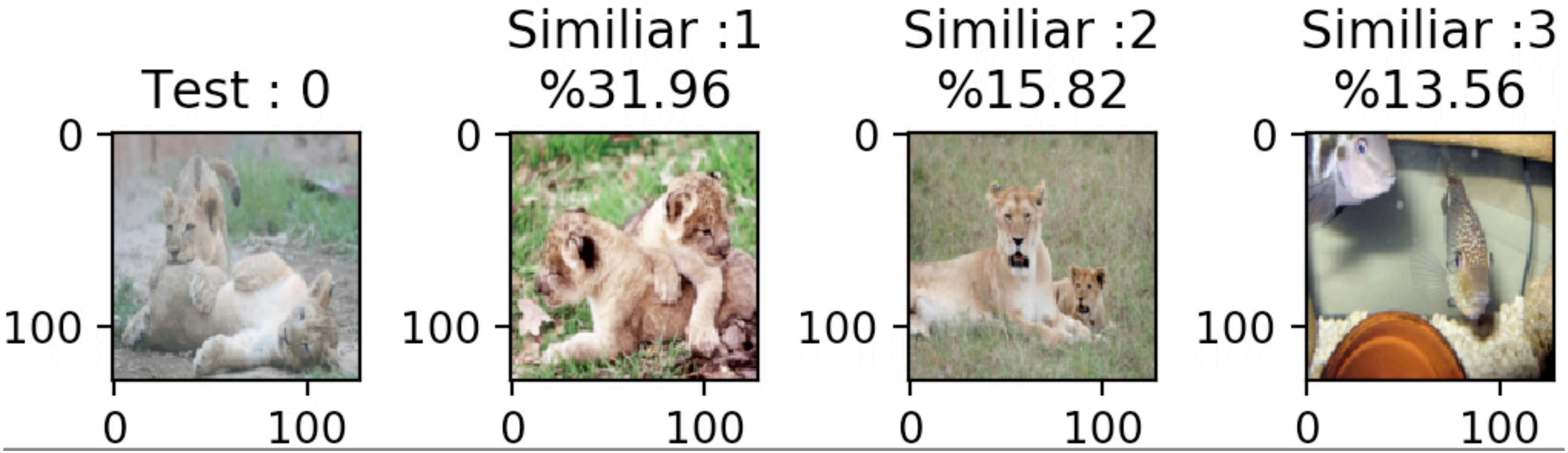
64x64x3

Dataset Link : *http://image-net.org/explore*

Different Dataset Results



50 epoch retrained model using old model weights



Data augmentation

An overfitting net can generally be made to perform better by using more training data. (And if your unregularized net does not overfit, you should probably make it larger.)

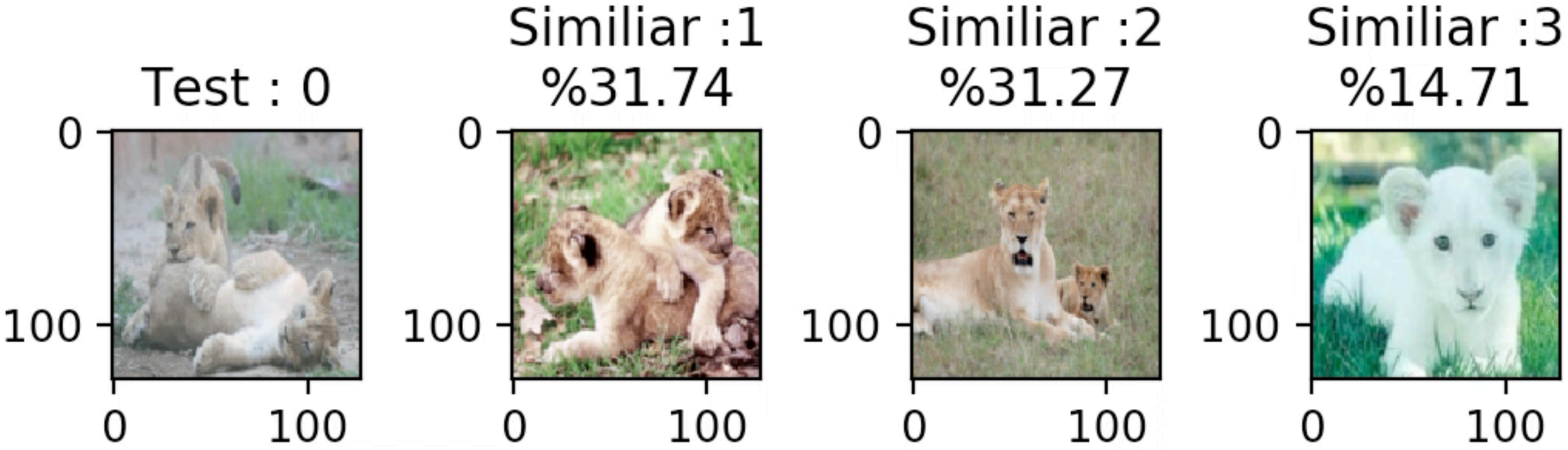
Data augmentation lets us artificially increase the number of training examples by applying transformations, adding noise etc. That's obviously more economic than having to go out and collect more examples by hand. Augmentation is a very useful tool to have in your deep learning toolbox.

We mentioned batch iterators already briefly. It is the batch iterator's job to take a matrix of samples, and split it up in batches, in our case of size 128. While it does the splitting, the batch iterator can also apply transformations to the data on the fly. So to produce those horizontal flips, we don't actually have to double the amount of training data in the input matrix. Rather, we will just perform the horizontal flips with 50% chance **while** we're iterating over the data. This is convenient, and for some problems it allows us to produce an infinite number of examples, without blowing up the memory usage. Also, transformations to the input images can be done while the GPU is busy processing a previous batch, so they come at virtually no cost.

```
# normalisation of images
img_prep = ImagePreprocessing()
img_prep.add_featurewise_zero_center()
img_prep.add_featurewise_stdnorm()

# Create extra synthetic training data by flipping & rotating images
img_aug = ImageAugmentation()
img_aug.add_random_flip_leftright()
img_aug.add_random_rotation(max_angle = 89.)
img_aug.add_random_blur(sigma_max=3.)
img_aug.add_random_flip_updown()
img_aug.add_random_90degrees_rotation(rotations = [0, 1, 2, 3])
```

Source : http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-keypoints-tutorial/#data-augmentation

Data Augmentation and Image Preprocessed Model



“Thank you for your attention”