KERAS

Gustavo Stein Mattos Araújo dos Santos
Humberto da Silva Neto

Cronograma

- Keras
  - O *framework*
  - Programando com Keras
  - Configurações e Compilação
  - Arquitetura e topologias
  - Vantagens e desvantagens
- Aplicação na literatura
- Outras aplicações

KERAS

O *framework*

Keras é uma biblioteca *open source* de rede neural escrita em Python.

- keras.io
- François Chollet
- Versão estável: 2.1.6 / 23 de Abril de 2018
- Linguagens: Python 2.7 e 3.6
- TensorFlow, CNTK ou Theano

O *framework*

- Princípios do Keras:
  - Ser amigável ao usuário
  - Modularidade
  - Extensibilidade
  - Totalmente declarado em Python

Programando com Keras

```
# Import libraries and packages:
import keras
from keras.models import Sequential
from keras.layers import Dense

# Initializing the ANN
classifier = Sequential()

# Adding the input layer and the first hidden layer
classifier.add(Dense(units = 6, activation = 'relu', kernel_initializer = 'uniform', input_dim = 11))

# Adding the second hidden layer
classifier.add(Dense(units = 6, activation = 'relu', kernel_initializer = 'uniform'))

# Adding the ouput layer
classifier.add(Dense(units = 1, activation = 'sigmoid', kernel_initializer = 'uniform'))

# Compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the Training set
classifier.fit(X_train, y_train, batch_size = 10, epochs = 100)

# Predicting the Test set results
y_pred = classifier.predict(X_test)
```

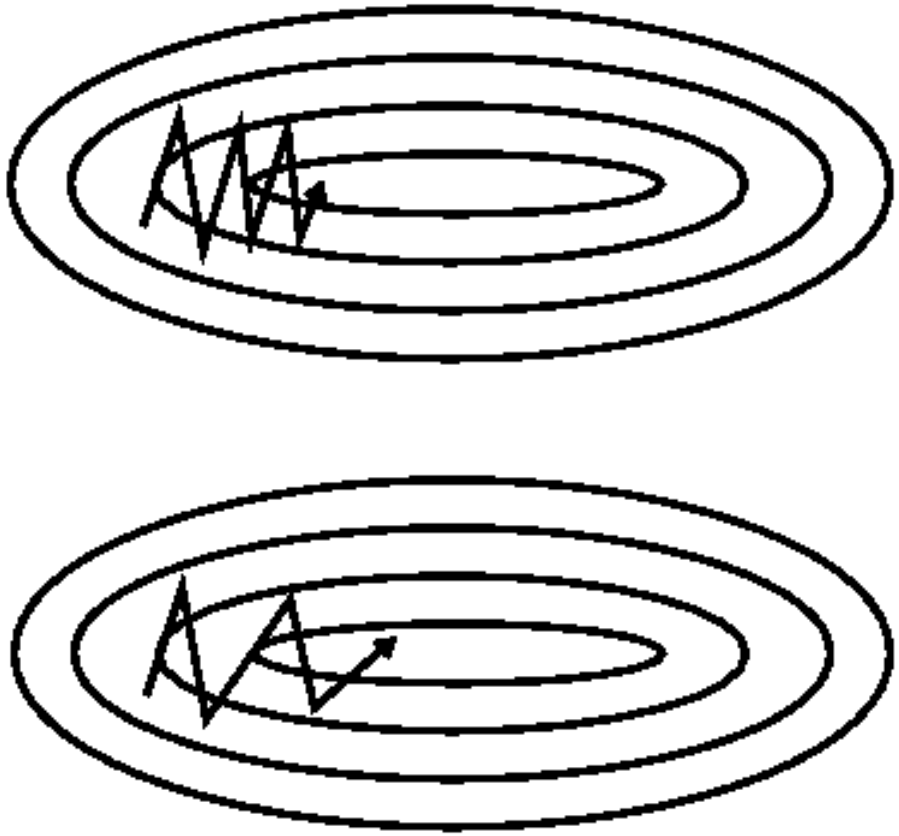
Configurações e Compilação

- A função de compilação recebe três argumentos:
  - Optimizer;
  - Loss;
  - Metrics (julgar a performance do modelo) ;

```
# For a multi-class classification problem
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# For a binary classification problem
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# For a mean squared error regression problem
model.compile(optimizer='rmsprop',
              loss='mse')

# For custom metrics
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

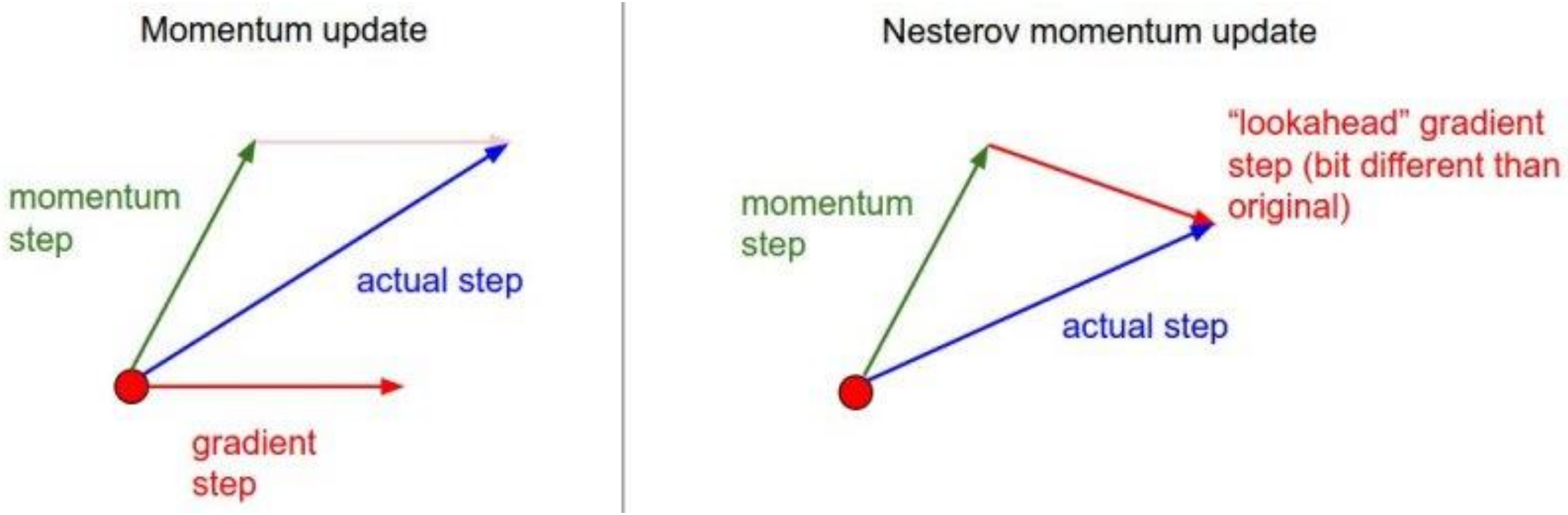
Otimizadores: SGD

- Stochastic Gradient Descent
  - Mini Batch
- Desafios:
  - Taxa de aprendizagem;
  - Adaptação da taxa de aprendizagem
  - Mínimos locais
- Momentum



Otimizadores: SGD

- Momentum: Nesterov Accelerated Gradient



Otimizadores: SGD

SGD [source]

```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

Stochastic gradient descent optimizer.

Includes support for momentum, learning rate decay, and Nesterov momentum.

Arguments

- **lr**: float >= 0. Learning rate.
- **momentum**: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- **decay**: float >= 0. Learning rate decay over each update.
- **nesterov**: boolean. Whether to apply Nesterov momentum.

Otimizadores: Adagrad

- Taxa de aprendizagem adaptativa
- Boa para conjunto de dados dispersos
- Desvantagem: Tende a diminuir a taxa de aprendizagem prematuramente (Soma do quadrado dos gradientes passados no denominador).

Otimizadores: Adagrad

Adagrad [source]

```
keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
```

Adagrad optimizer.

It is recommended to leave the parameters of this optimizer at their default values.

Arguments

- **lr**: float >= 0. Learning rate.
- **epsilon**: float >= 0. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

Otimizadores: Adadelta

- Pode ser interpretada como um extensão da Adagrad, onde busca-se minimizar a brusca diminuição da taxa de aprendizagem:
  - Ao invés de somar o quadrado dos gradientes passados, utiliza-se a média decaída exponencialmente dos gradientes, aumentando o período de aprendizagem do algoritmo
  - Não é necessário definir um coeficiente de aprendizagem inicial
  - RMSprop

Otimizadores: Adadelta

Adadelta [source]

```
keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
```

Adadelta optimizer.

It is recommended to leave the parameters of this optimizer at their default values.

Arguments

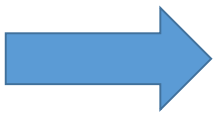
- **lr**: float >= 0. Learning rate. It is recommended to leave it at the default value.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

Otimizadores: Adam

- Adaptive Moment Estimation
- Similar a Adadelta e RMSprop, com adição de momentum (Heavy ball with friction)
- NAdam

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$



$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

Otimizadores: Adam

Adam [source]

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None
```

Adam optimizer.

Default parameters follow those provided in the original paper.

Arguments

- **lr**: float >= 0. Learning rate.
- **beta_1**: float, 0 < beta < 1. Generally close to 1.
- **beta_2**: float, 0 < beta < 1. Generally close to 1.
- **epsilon**: float >= 0. Fuzz factor. If `None`, defaults to `K.epsilon()`.
- **decay**: float >= 0. Learning rate decay over each update.

Loss

```
13  def mean_squared_error(y_true, y_pred):
14      return K.mean(K.square(y_pred - y_true), axis=-1)
15
16
17  def mean_absolute_error(y_true, y_pred):
18      return K.mean(K.abs(y_pred - y_true), axis=-1)
19
20
21  def mean_absolute_percentage_error(y_true, y_pred):
22      diff = K.abs((y_true - y_pred) / K.clip(K.abs(y_true),
23                                              K.epsilon(),
24                                              None))
25      return 100. * K.mean(diff, axis=-1)
26
27
28  def mean_squared_logarithmic_error(y_true, y_pred):
29      first_log = K.log(K.clip(y_pred, K.epsilon(), None) + 1.)
30      second_log = K.log(K.clip(y_true, K.epsilon(), None) + 1.)
31      return K.mean(K.square(first_log - second_log), axis=-1)
```

Arquitetura e topologias

SUPERVISED

FEED FORWARD

Feed Forward Network sometimes Referred to as MLP, is a fully connected dense model used as a simple classifier.

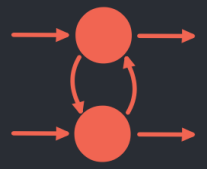
Convolutional Network assume that highly correlated features located close to each other in the input matrix and can be pooled and treated as one in the next layer.

Known for superior Image classification capabilities.

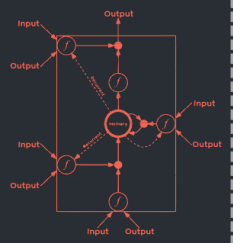
RECURRENT

Simple Recurrent Neural Network is a class of artificial neural network where connections between units form a directed cycle.

Hopfield Recurrent Neural Network It is a RNN in which all connections are symmetric. it requires stationary inputs.



Long Short Term Memory Network contains gates that determine if the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output



UNSUPERVISED

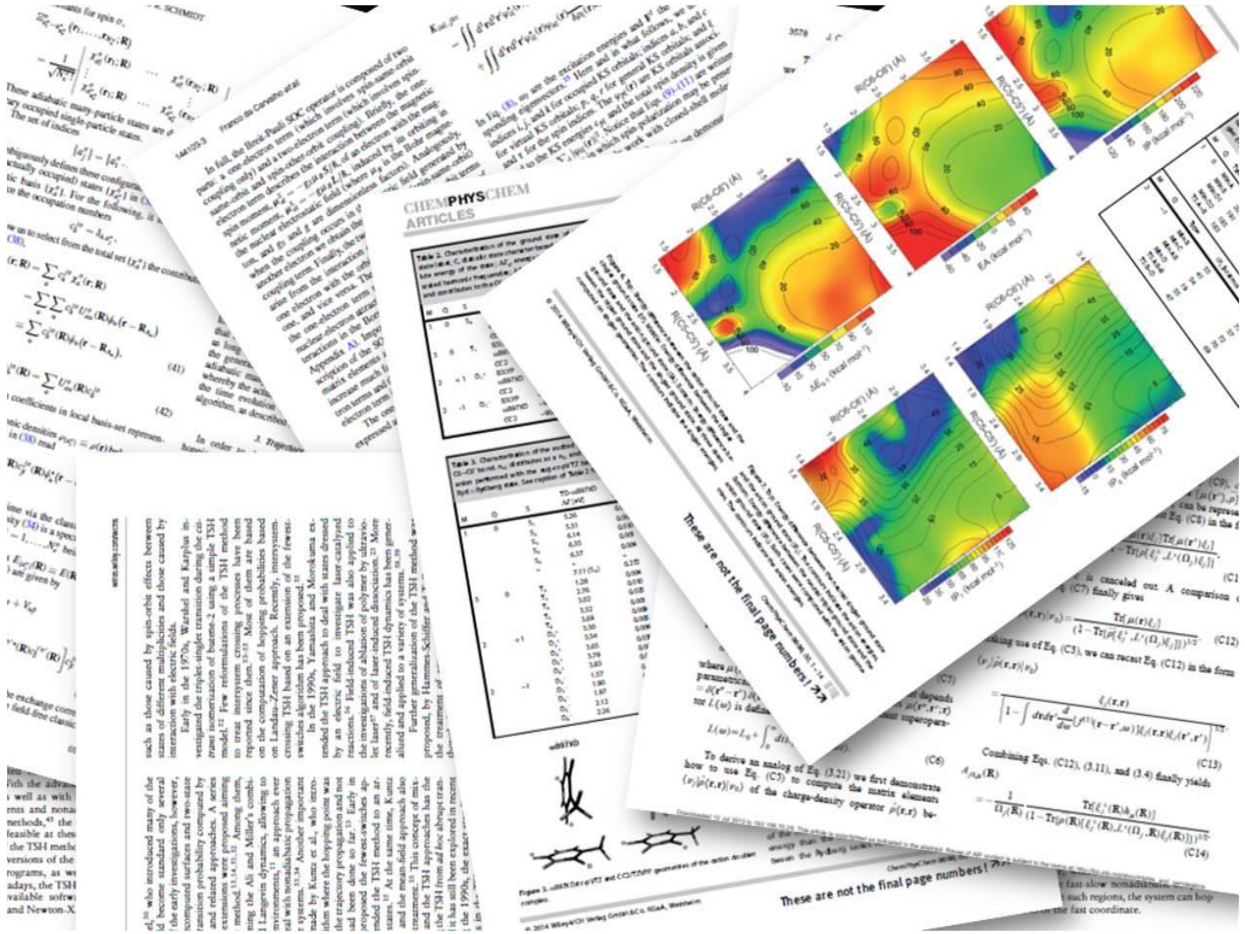
Auto Encoder aims to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction.

Restricted Boltzmann Machine can learn a probability distribution over its set of inputs..

Deep Belief Net is a composition of simple, unsupervised networks such as restricted Boltzmann machines ,where each sub-network's hidden layer serves as the visible layer for the next.

Vantagens e desvantagens

- Alto nível
- CPU e GPU
- Customização
- Não compatibilidade
- Retorna o último modelo



APLICAÇÃO NA LITERATURA

Using Convolutional Neural Networks for Determining Reticulocyte Percentage in Cats

- Determinar a proporção de reticulócitos agregados em um conjunto de dados de imagens de sangue de gatos.
- Shot MultiBox Detector (SSD)
  - SSD300 – Microsoft COCO dataset
  - 80 → 3 classes
  - Tuning:
    - 40 epochs
    - Adam optimizer
    - learning rate de 0.001
    - learning rate decay de 0.0005
- Resultados:
  - Câmera microscópica – 98,7%
  - Câmera do smartphone – 88,5%

- **aggregate reticulocytes**
- **punctate reticulocytes**
- **erythrocytes**

Using Convolutional Neural Networks for Determining Reticulocyte Percentage in Cats

Krunoslav Vinicki[1] Pierluigi Ferrari[2] Maja Belić[1] Romana Turk[1]

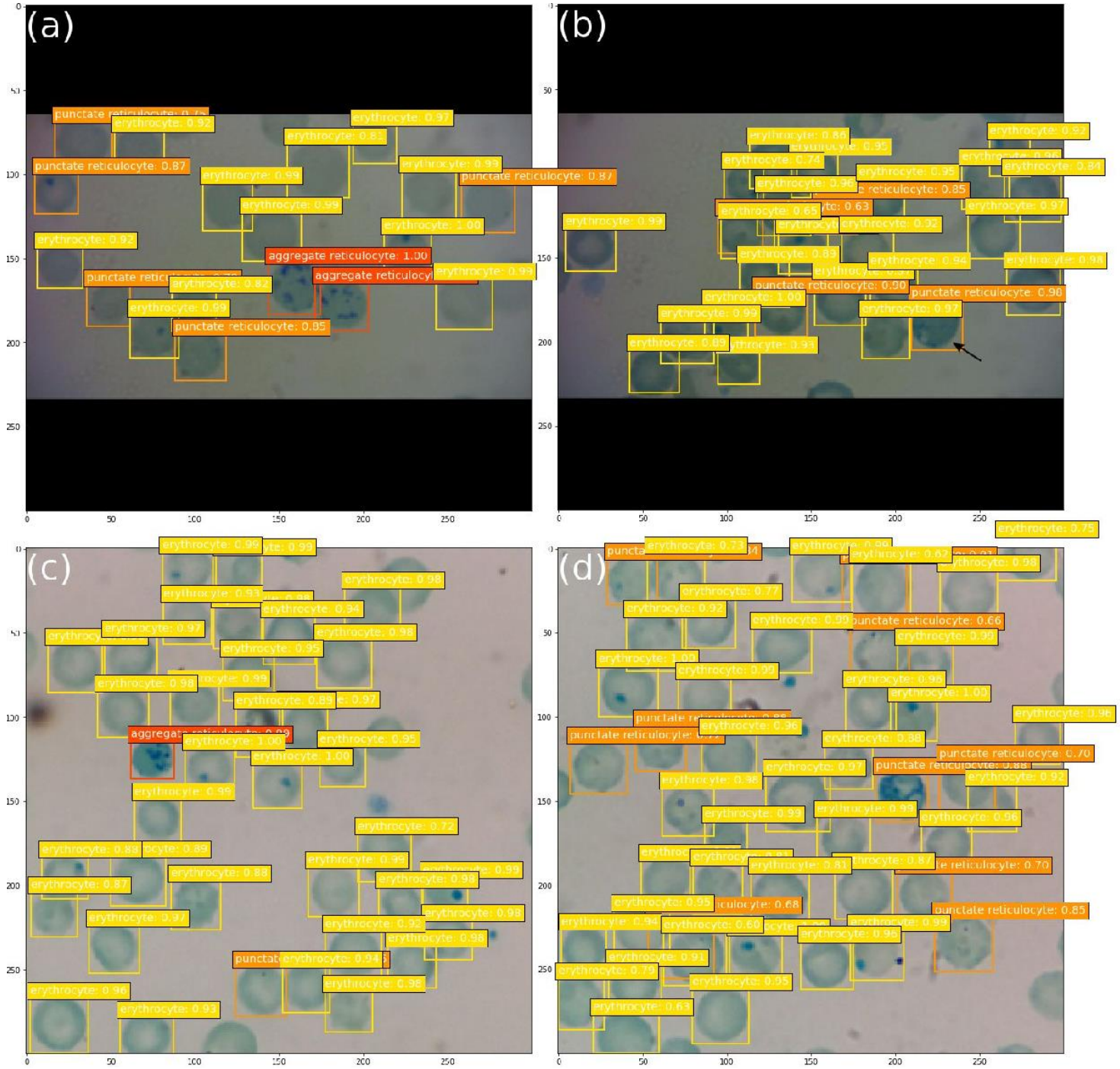
Abstract

Recent advances in artificial intelligence (AI), specifically in computer vision (CV) and deep learning (DL), have created opportunities for novel systems in many fields. In the last few years, deep learning applications have demonstrated impressive results not only in fields such as autonomous driving and robotics, but also in the field of medicine, where they have, in some cases, even exceeded human-level performance. However, despite the huge potential, adoption of deep learning-based methods is still slow in many areas, especially in veterinary medicine, where we haven't been able to find any research papers using modern convolutional neural networks (CNNs) in medical image processing. We believe that using deep learning-based medical imaging can enable more accurate, faster and less expensive diagnoses in veterinary medicine. In order to do so, however, these methods have to be accessible to everyone in this field, not just to computer scientists. To show the potential of this technology, we present results on a real-world task in veterinary medicine that is usually done manually: feline reticulocyte percentage. Using an open source Keras implementation of the Single-Shot MultiBox Detector (SSD) model architecture and training it on only 800 labeled images, we achieve an accuracy of 98.7% at predicting the correct number of aggregate reticulocytes in microscope images of cat blood smears. The main motivation behind this paper is to show not only that deep learning can approach or even exceed human-level performance on a task like this, but also that anyone in the field can implement it, even without a background in computer science.

1. Introduction

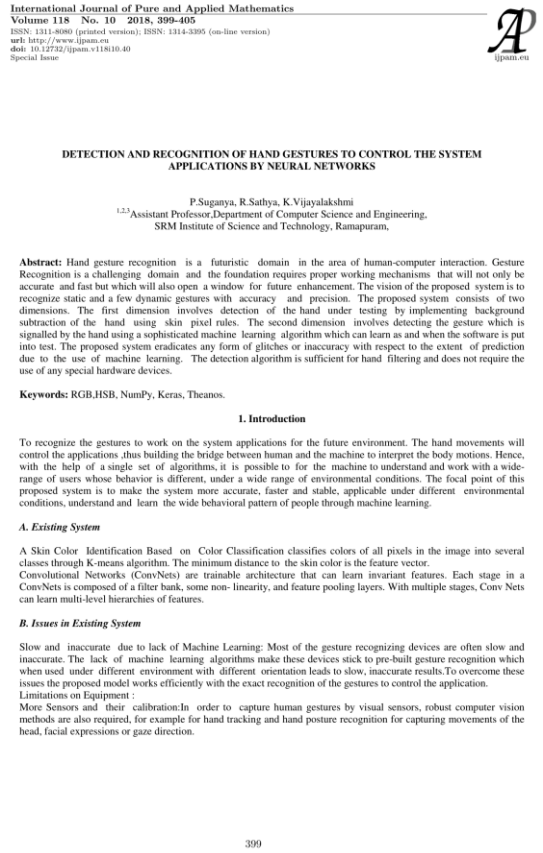
Convolutional Neural Networks (CNN) are a class of deep artificial neural networks that is loosely inspired by the structure and function of the mammalian visual cortex [1]. CNNs show similar levels of abstraction to our visual cortex: Earlier layers of the neural network detect simple features like edges, while later layers detect gradually more complex shapes and objects [2, 3]. These multiple levels of abstraction allow CNNs to learn to extract complex features directly from raw input images [3].

[1] University of Zagreb, Faculty of Veterinary Medicine
[2] Technical University of Berlin, Department of Mathematics
Correspondence to: Krunoslav Vinicki <kvinicki@gmail.com>, Pierluigi Ferrari <pierluigi.ferrari@gmx.com>.


(a)
(b)
(c)
(d)

Detection And Recognition Of Hand Gestures To Control The System Applications By Neural Networks

- Módulo de detecção:
  - Java
  - *Background subtraction* → RGB2HSB → Binarização
- Módulo de predição:
  - CNN: Keras com Theano no *backend*
  - Comparação com o dataset
- Aplicações:
  - Controlando o cursor de mouse
  - Para mudar de canal de televisão através de gestos
  - Interage com o ambiente de forma natural se o jogo suportar Realidade Aumentada / Virtual
  - Interpretar a linguagem de sinais.



International Journal of Pure and Applied Mathematics
Volume 118 No. 10 2018, 399-405
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
doi: 10.12732/ijpam.v118i10.40
Special Issue
ijpam.eu

DETECTION AND RECOGNITION OF HAND GESTURES TO CONTROL THE SYSTEM APPLICATIONS BY NEURAL NETWORKS

P.Suganya, R.Sathya, K.Vijayalakshmi
[1,2,3]Assistant Professor,Department of Computer Science and Engineering,
SRM Institute of Science and Technology, Ramapuram,

Abstract: Hand gesture recognition is a futuristic domain in the area of human-computer interaction. Gesture Recognition is a challenging domain and the foundation requires proper working mechanisms that will not only be accurate and fast but which will also open a window for future enhancement. The vision of the proposed system is to recognize static and a few dynamic gestures with accuracy and precision. The proposed system consists of two dimensions. The first dimension involves detection of the hand under testing by implementing background subtraction of the hand using skin pixel rules. The second dimension involves detecting the gesture which is signalled by the hand using a sophisticated machine learning algorithm which can learn as and when the software is put into test. The proposed system eradicates any form of glitches or inaccuracy with respect to the extent of prediction due to the use of machine learning. The detection algorithm is sufficient for hand filtering and does not require the use of any special hardware devices.

Keywords: RGB,HSB, NumPy, Keras, Theanos.

1. Introduction

To recognize the gestures to work on the system applications for the future environment. The hand movements will control the applications ,thus building the bridge between human and the machine to interpret the body motions. Hence, with the help of a single set of algorithms, it is possible to for the machine to understand and work with a wide-range of users whose behavior is different, under a wide range of environmental conditions. The focal point of this proposed system is to make the system more accurate, faster and stable, applicable under different environmental conditions, understand and learn the wide behavioral pattern of people through machine learning.

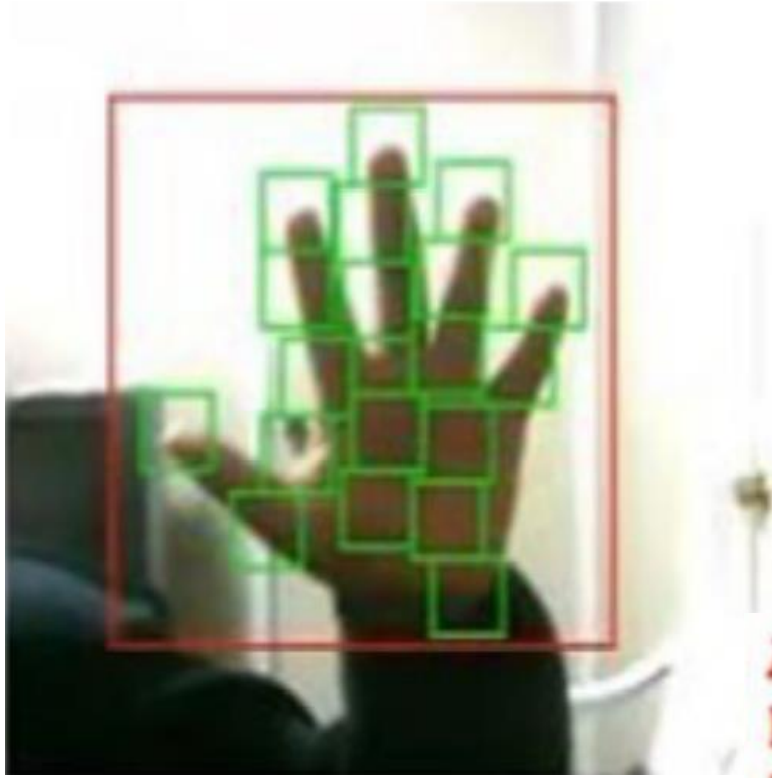
A. Existing System

A Skin Color Identification Based on Color Classification classifies colors of all pixels in the image into several classes through K-means algorithm. The minimum distance to the skin color is the feature vector.
Convolutional Networks (ConvNets) are trainable architecture that can learn invariant features. Each stage in a ConvNets is composed of a filter bank, some non- linearity, and feature pooling layers. With multiple stages, Conv Nets can learn multi-level hierarchies of features.

B. Issues in Existing System

Slow and inaccurate due to lack of Machine Learning: Most of the gesture recognizing devices are often slow and inaccurate. The lack of machine learning algorithms make these devices stick to pre-built gesture recognition which when used under different environment with different orientation leads to slow, inaccurate results.To overcome these issues the proposed model works efficiently with the exact recognition of the gestures to control the application.
Limitations on Equipment :
More Sensors and their calibration:In order to capture human gestures by visual sensors, robust computer vision methods are also required, for example for hand tracking and hand posture recognition for capturing movements of the head, facial expressions or gaze direction.

399



Ack: 4%
Fist: 0%
Hand: 92%
One: 0%
Straight: 5%
Palm: 3%
Thumbs: 4%
None: 6%
Swing: 5%

Leaf Recognition with Deep Learning and Keras using GPU computing

- Identificar a espécie de uma folha
- Metodologia:
  - Comparar redes pré-existentes
  - Datasets
- Conclusões:
  - Dados x Imagens
  - Tamanho das imagens

Kaggle

- 990 imagens
- 99 classes

MEW2012

- 9745 imagens
- 153 classes

Flavia

- 1907 imagens
- 32 classes

TFG IN COMPUTER ENGINEERING, ENGINEERING SCHOOL (ES), UAB UNIVERSITY (UAB)

gnition with Deep Learning and
as using GPU computing

Jesús López Barrientos

18/02/2018

bout using Deep Learning for leaf recognition using Keras and GPU compu-
s of "**Kaggle**" [1], a Machine Learning training webpage that using simple
lp people to learn how to use Deep Learning. Kaggle made a challenge
at was about Leaf Recognition. In that challenge more than 1500 users
ade teams to participate and we grab 17 codes of them to see how their
updated the codes, because they were written in 1.0 version of Keras, and
then we made a ranking to test their accuracy in leaf recognition using the
dataset provided by Kaggle. Also we downloaded two more datasets to make more tests with them.
On the other hand, we found two papers in "**the ImageClef Competition**" **[2]**, and we implemented
them from the beggining to see how simple is to transform a paper into code. ECOCUAN team,
from 2015, used a tunned AlexNet CNN and KDETUT, from 2017, used a ResNet50 modified CNN.
Another paper found on the net was one that use ResNet26 CNN, so we think that was a good idea
to make a ranking of the three of them to see which is better.

earning, Keras, CNN, Convolutional Neural Network, Tensorflow, Leaf
xNet, LeNet, Kaggle, ImageClef

That paper is divided in nine sections where we will ex-
plain state-of-the-art of Deep Learning through the time,
how all the codes are implemented, wich Convolutional
Neural Network is in the code and what is our method for
comparing the codes to make the ranking.

Also we will show you how we managed to make my
ranking of all the Kaggle codes using two datasets, Kaggle
dataset containing 990 binary leaf images of 99 classes of
leaves and Middle European Woods 2012 [3] dataset, con-
taining 9745 binary leaf images of 153 classes of leaves.

And our investigation of the three papers that we imple-
mented, using two datasets, Flavia [4] dataset containing
1907 fully color images of 32 classes of leaves.

puter machine the specie
the question of my work.
but, what is the thing that
e of that leaf? we can see
e so much similar leaf of
some species that are the same. So we can see the shape and
the color of the leaf, but there are also some other features
that can be learned to classify that leaf.

A Convolutional Neural Network makes that learning au-
tomatical making segmentation of the leaf image. The al-
extract the best features
he image and make a pre-
hm of automated learning
mber of classes that exists
lity to make people do a
es of all the leaf types. So
for itself.

2 A LITTLE CNN'S HISTORY

DEEP LEARNING was very hard to use before 90's because the requirements for implement a good neural network was not enough with the computers of that epoch. So in 1998, when the computers could do powerful calculations, Yann LeCunn et al. [5] create a neural network to make an OCR, a postal code digit recognition. They used de MNIST dataset to train their network and they archieved a 99,7% of accuracy in their predictions. Their Net was based on one Convolutional layer followed by one Max pool layer.

mpus.uab.cat

omputer Science Department)
arino@uab.cat

FEBRUARY 2018, Engineering School (UAB)

Kaggle DB Classification accuracy		
CNN	**Accuracy**	**Epochs**
a. mulgund(img+shapes)	0,99	139
a. lazarev (shapes)	0,99	299
yu-weichang(img+shapes)	0,97	81
anantzoid(img+shapes)	0,97	270
p. xue (Alexnet) (5 layers)	0,7	28
churandy (2 layers)	0,64	754
a. thakur (3 layers)	0,59	15
p. xue (Lenet) (2 layers)	0,51	35
v. lyunda (1 layer)	0,46	41
alutrin (4 layers)	0,43	17

Kaggle DB only features Classification accuracy		
CNN	**Accuracy**	**Epochs**
Picubeisnot30	1	861
Guravjoshi	0,99	991
Ibprofin	0,99	995
Konstantin Tumalevich	0,98	724
Churandy	0,97	949
Michael Semeniuk	0,97	996
Prateek	0,97	999

	precision	recall	f1-score	support
Acer Palmatum	1.00	0.80	0.89	5
Acer buergerianum Miq.	1.00	0.75	0.86	4
Aesculus chinensis	1.00	0.80	0.89	5
Berberis anhweiensis Ahrendt	0.75	0.60	0.67	5
Cedrus deodara (Roxb.) G. Don	1.00	1.00	1.00	7
Cercis chinensis	1.00	1.00	1.00	6
Chimonanthus praecox L.	1.00	0.50	0.67	4
Cinnamomum camphora (L.) J. Presl	1.00	0.83	0.91	6
Cinnamomum japonicum Sieb.	0.50	0.60	0.55	5
Citrus reticulata Blanco	1.00	1.00	1.00	5
Ginkgo biloba L.	1.00	1.00	1.00	5
Ilex macrocarpa Oliv.	0.71	1.00	0.83	5
Indigofera tinctoria L.	1.00	1.00	1.00	6
Kalopanax (Thunb. ex A.Murr.) Koidz.	1.00	1.00	1.00	5
Koelreuteria paniculata Laxm.	0.67	1.00	0.80	4
Lagerstroemia indica (L.) Pers.	1.00	1.00	1.00	5
Ligustrum lucidum Ait. f.	0.83	1.00	0.91	5
Liriodendron chinense (Hemsl.) Sarg.	1.00	0.75	0.86	4
Magnolia grandiflora L.	1.00	1.00	1.00	5
Mahonia bealei (Fortune) Carr.	1.00	1.00	1.00	5
Manglietia fordiana Oliv.	1.00	1.00	1.00	4
Nerium oleander L.	0.83	1.00	0.91	5
Osmanthus fragrans Lour.	1.00	1.00	1.00	4
Phoebe nanmu (Oliv.) Gamble	1.00	1.00	1.00	5
Phyllostachys edulis (Carr.)Houz.	0.80	0.80	0.80	5
Pittosporum tobira (Thunb.) Ait. f.	1.00	1.00	1.00	5
Podocarpus macrophyllus (Thunb.) Sweet	1.00	0.60	0.75	5
Populus ×canadensis Moench	0.83	1.00	0.91	5
Prunus persica (L.) Batsch	1.00	0.80	0.89	5
Prunus serrulata Lindl. var. lannesiana auct.	0.83	1.00	0.91	5
Tonna sinensis M. Roem.	0.71	1.00	0.83	5
Viburnum awabuki K.Koch	1.00	1.00	1.00	5
avg / total	0.92	0.91	0.90	159

DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers

- Gerar código-fonte de Deep Learning a partir de um documento de pesquisa.
- Dataset: 30K imagens
- Classificação:
  - Duas etapas
  - 4.096 características – VGG19
- Extração de conteúdo:
  - Detecção de fluxo
  - OCR (Optical Character Recognition)
- Gerando o código:
  - JSON

DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers

Akshay Sethi *
IIIT Delhi

Anush Sankaran, Naveen Panwar, Shreya Khare, Senthil Mani
IBM Research, India

Abstract

With an abundance of research papers in deep learning, reproducibility or adoption of the existing works becomes a challenge. This is due to the lack of open source implementations provided by the authors. Further, re-implementing research papers in a different library is a daunting task. To address these challenges, we propose a novel extensible approach, DLPaper2Code, to extract and understand deep learning design flow diagrams and tables available in a research paper and convert them to an abstract computational graph. The extracted computational graph is then converted into execution ready source code in both Keras and Caffe, in real-time. An *arXiv*-like website is created where the automatically generated designs is made publicly available for 5,000 research papers. The generated designs could be rated and edited using an intuitive drag-and-drop UI framework in a crowdsourced manner. To evaluate our approach, we create a simulated dataset with over 216,000 valid design visualizations using a manually defined grammar. Experiments on the simulated dataset show that the proposed framework provide more than 93% accuracy in flow diagram content extraction.

Introduction

The growth of deep learning (DL) in the field of artificial intelligence has been astounding in the last decade with about 35,800 research papers being published since 2010[1]. Keeping up with the growing literature has been a real struggle for researchers and practitioners. In one of the recent AI conferences, NIPS 2016, the maximum number of papers submitted (∼ 685/2500) were in the topic, "Deep Learning or Neural Networks". However, a majority of these research papers are not accompanied by their corresponding implementations. In NIPS 2016, **only** 101/567 (∼ 18%) papers made their source implementation available[2]. Implementing research papers takes at least a few days of effort for software engineers assuming that they have limited knowledge in DL (Sankaran et al. 2011).

Another major challenge is the availability of various libraries in multiple programming langauges to implement DL algorithms such as Tensorflow (Abadi et al. 2016), Theano (Bastien et al. 2012), Caffe (Jia et al. 2014), Torch (et al 2011), MXNet (Chen 2015), DLAJ (Gibson 2015), CNTK (Seide and Agarwal 2016) and wrappers such as Keras (Chollet and others 2015), Lasagne (Dieleman 2015), and PyTorch (Chintala 2016). The public implementations of the DL research papers are available in various libraries offering very little interoperability or communication among them. Consider a use-case for a researcher working in "image captioning", where three of the highly referred research papers for the problem of image captioning[3] are:

1. Show and Tell (Vinyals et al. 2015): Original implementation available in Theano; https://github.com/kelvinxu/arctic-captions
2. NeuralTalk2 (Karpathy and Fei-Fei 2015): Original implementation available in Torch; https://github.com/karpathy/neuraltalk2
3. LRCN (Donahue et al. 2015): Original implementation available in Caffe; http://jeffdonahue.com/lrcn/

As the implementations are available in different libraries, a researcher cannot directly combine the models. Also, for a practitioner having remaining of the code-base in Java (DLAJ) directly leveraging either of these public implementations would be daunting. Thus, we highlight two highly overlooked challenges in DL:

1. Lack of public implementation available for existing research works and thus, the time incurred in reproducing their results
2. Existing implementations are confined to a single (or few) libraries limiting portability into other popular libraries for DL implementation.

We observed that most of the research paper explains the DL model design either through a figure or a table. Thus,

*Akshay Sethi interened at IBM Research, India during this work.

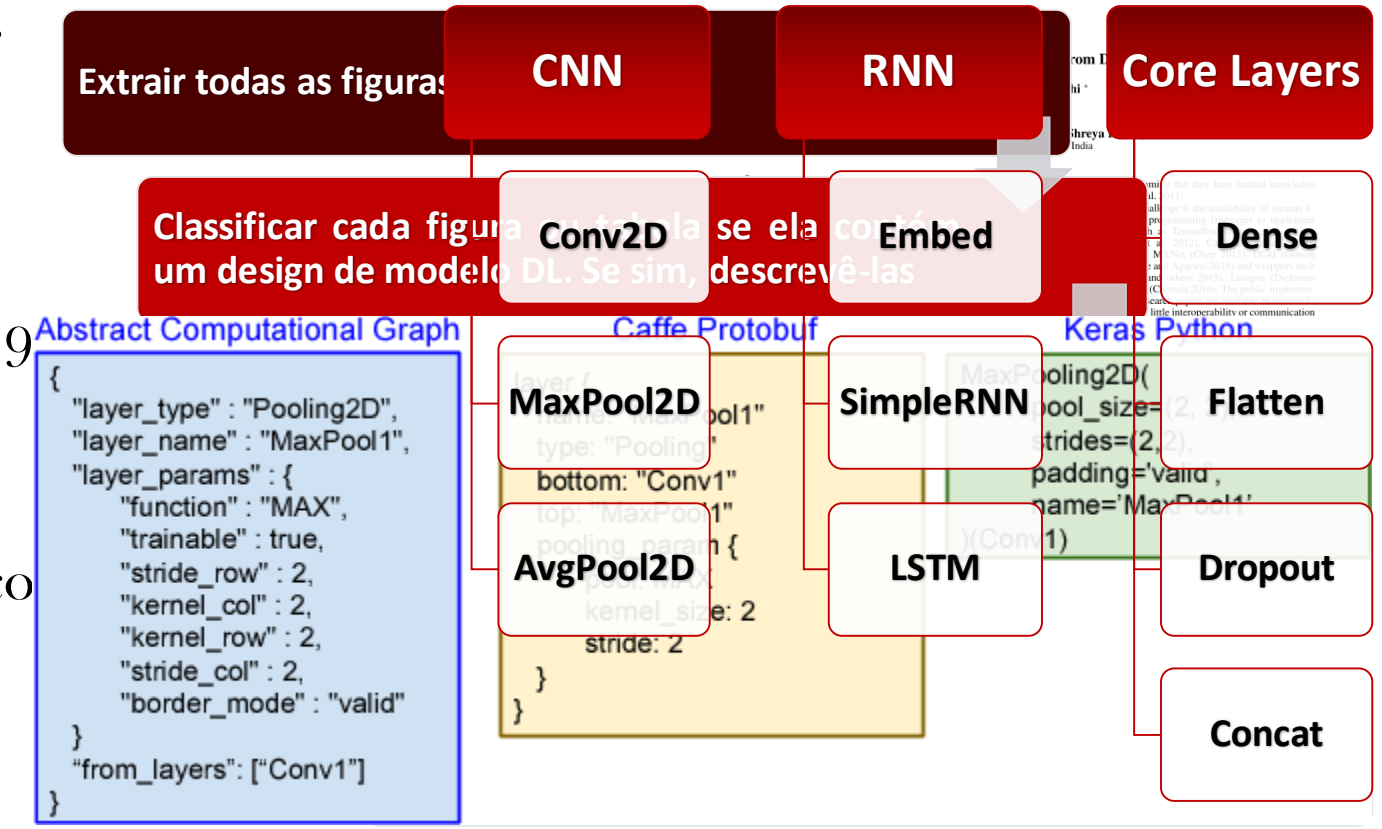
[1] https://scholar.google.co.in/scholar?as_sdt=1,5\&q=\%22deep+learning\%22\&hl=en\&as_ylo=2016\&as_vis=1

[2] https://www.kaggle.com/benhamner/nips-papers

[3] https://competitions.codalab.org/competitions/3221#results

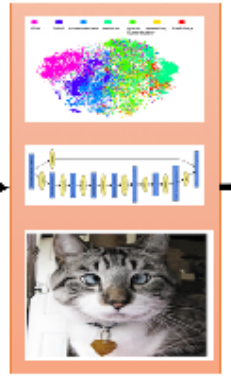
DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers

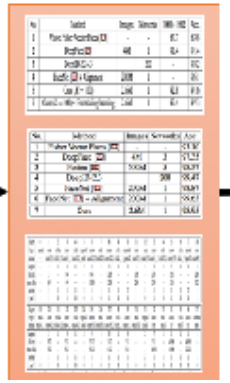
- Gerar código-fonte de Deep Learning a partir de um documento de pesquisa.
- Dataset: 30K imagens
- Classificação:
  - Duas etapas
  - 4.096 características – VGG19
- Extração de conteúdo:
  - Detecção de fluxo
  - OCR (Optical Character Reco
- Gerando o código:
  - JSON



1. Research Paper

2. Extract Figure & Tables





3. Type Classification

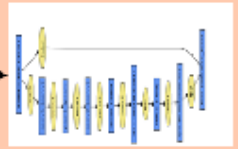
Contains a DL Model ?

Which Figure Type?

Contains a DL Model ?

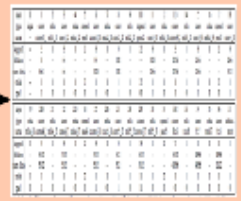
Which Table Type?

4. Extract Information



Extract Flow Information

Perform OCR



Extract Flow Information

LayerName Matching

5. Build Flow Graph

Abstract Computational Graph

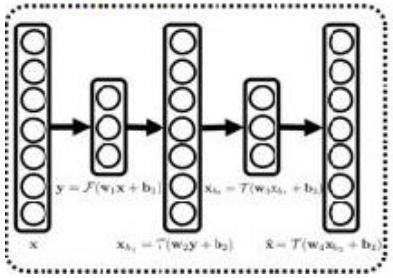
Abstract Computational Graph

6. Generate Code

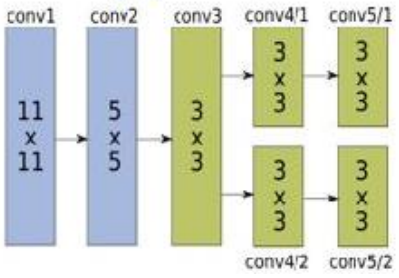
Keras Code

Caffe Code

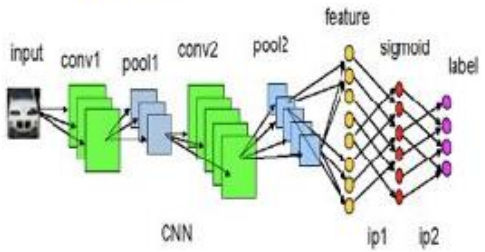
(a) Neurons Plot



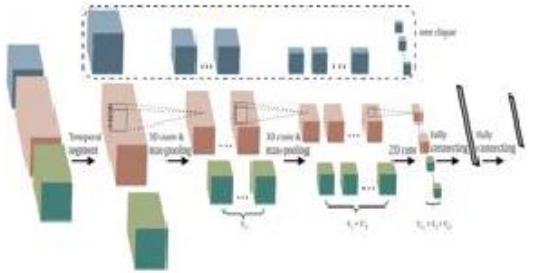
(b) 2D Box



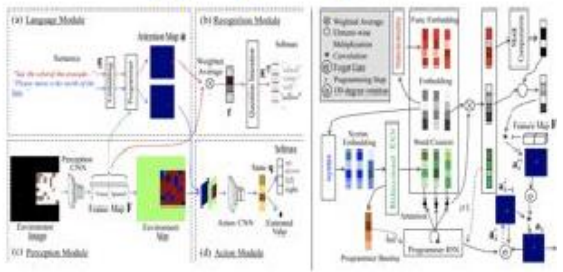
(c) Stacked2D Box



(d) 3D Box

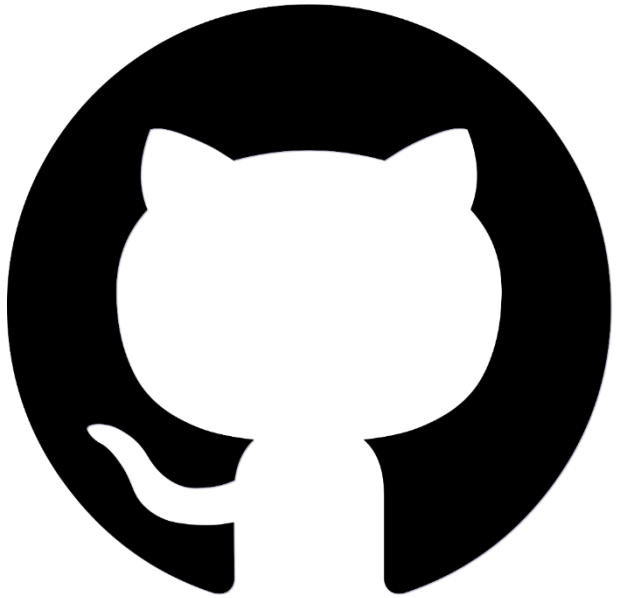


(e) Pipeline plot

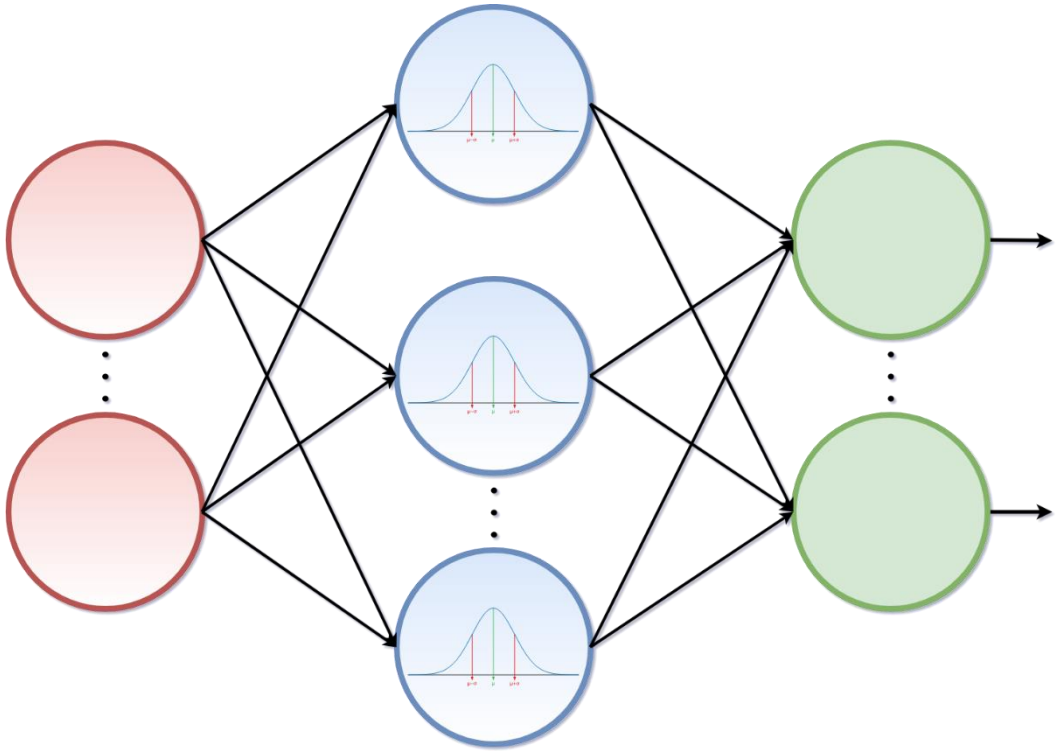


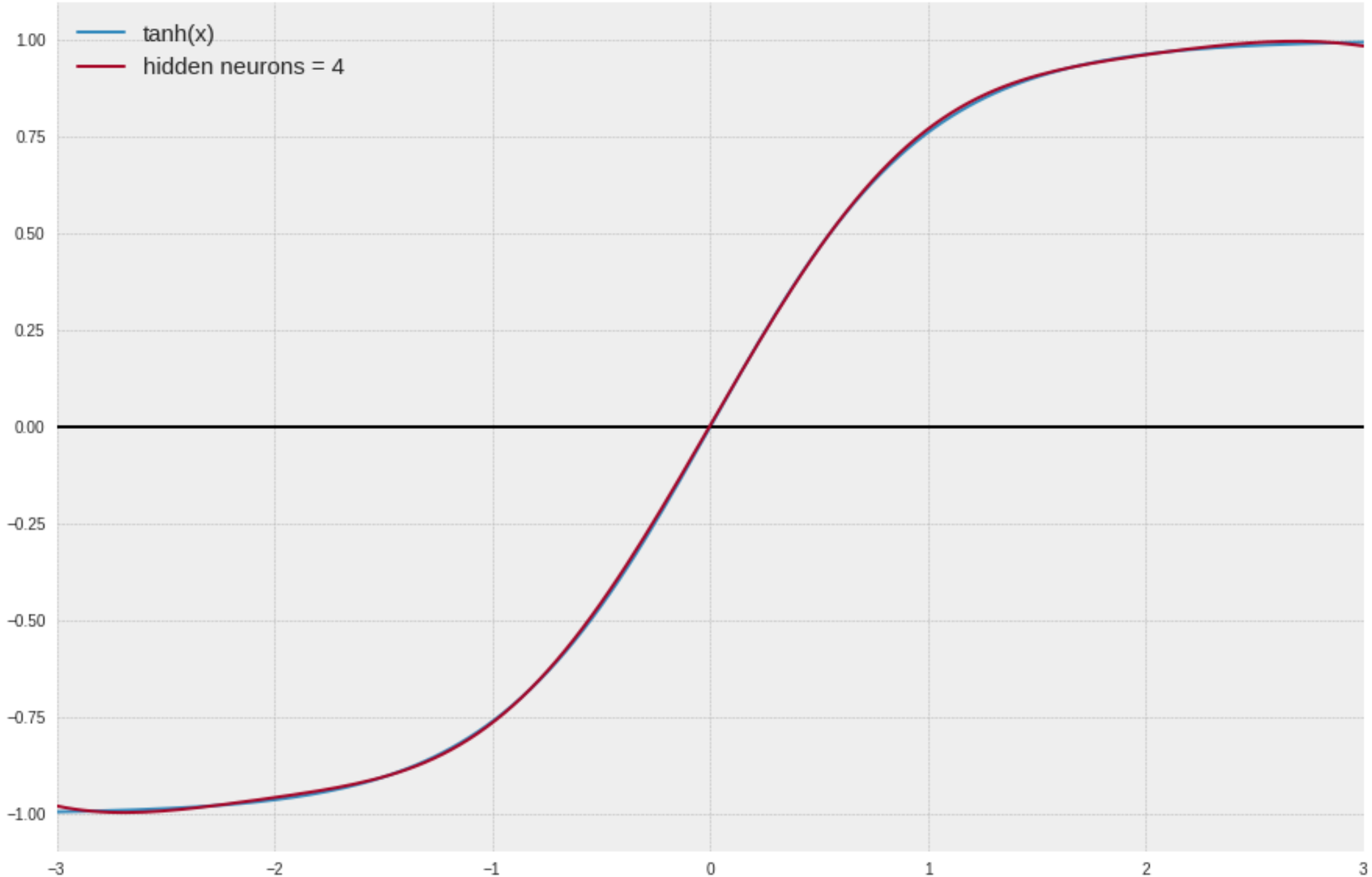
OUTRAS APLICAÇÕES

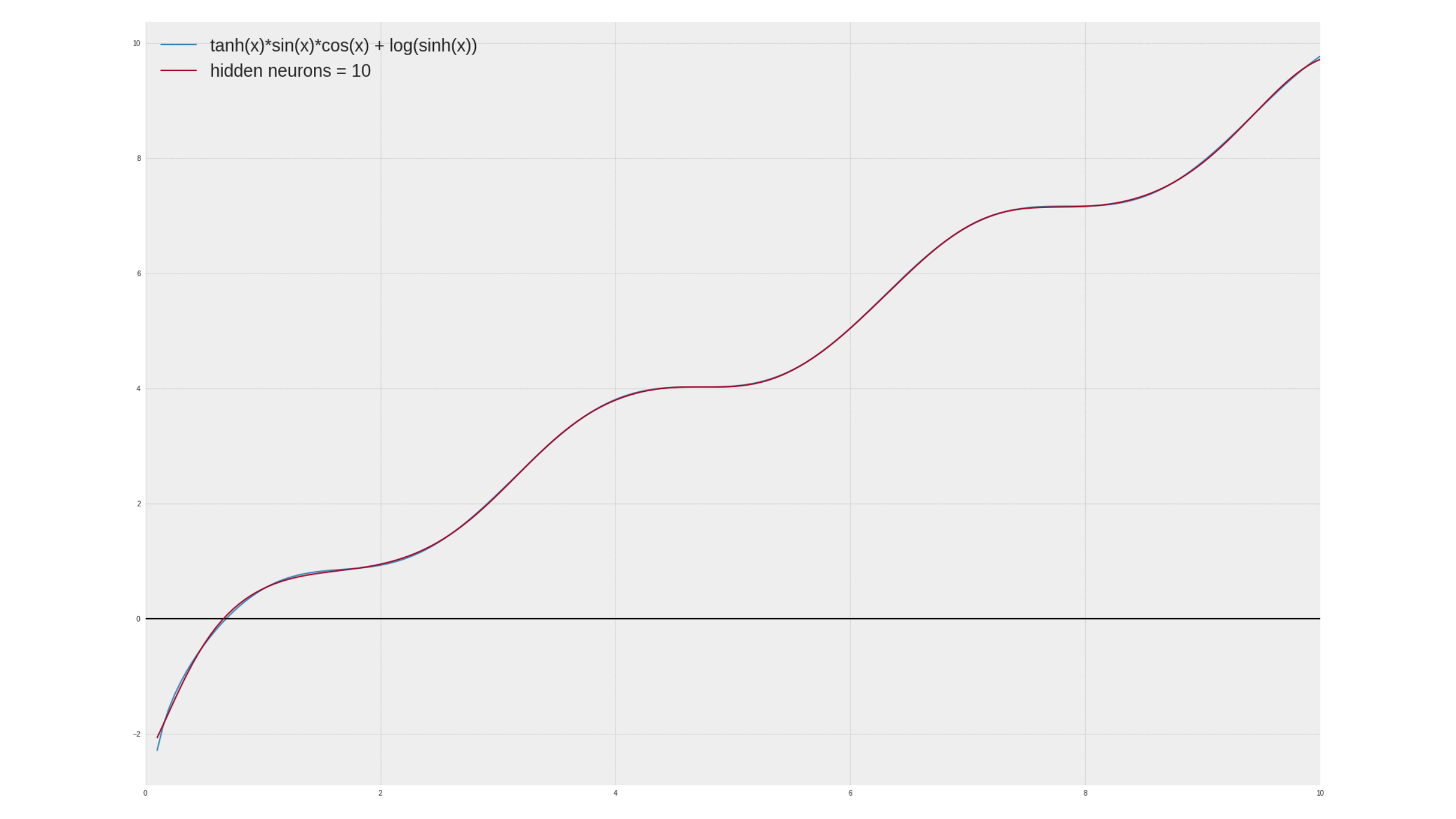
Redes de Função de Base Radial



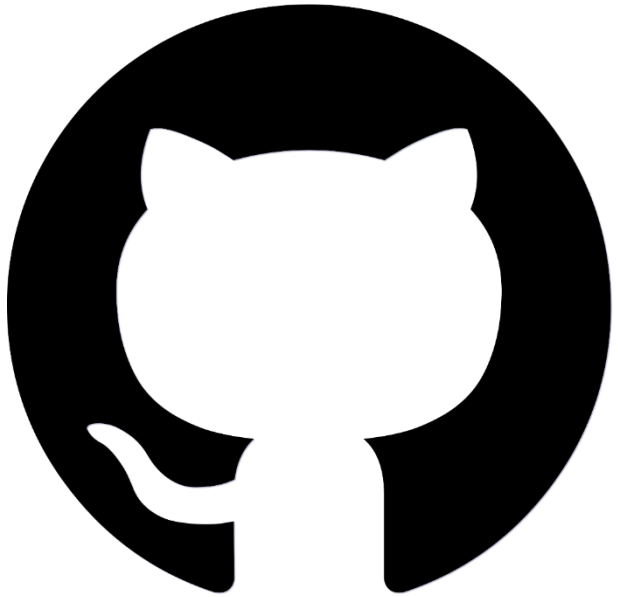
PetraVidnerova



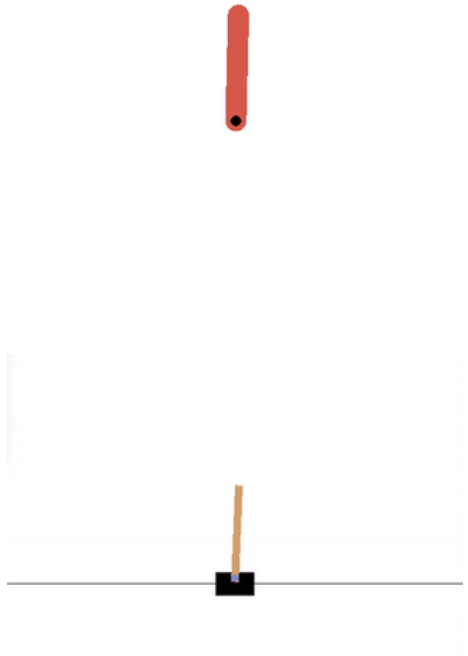

tanh(x)
hidden neurons = 4
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−3
−2
−1
0
1
2
3


tanh(x)*sin(x)*cos(x) + log(sinh(x))
hidden neurons = 10
10
8
6
4
2
0
−2
0
2
4
6
8
10

Deep Reinforcement Learning for Keras

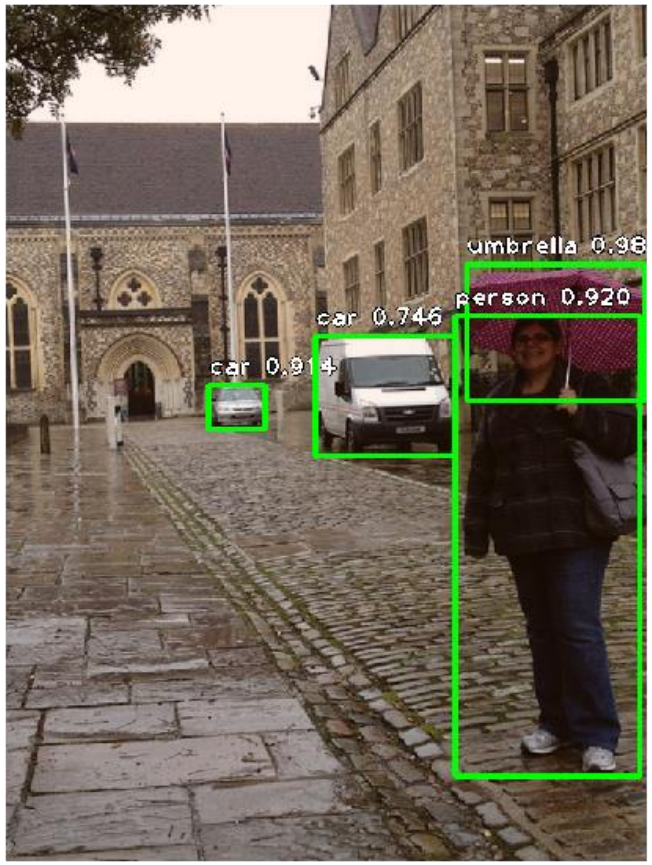


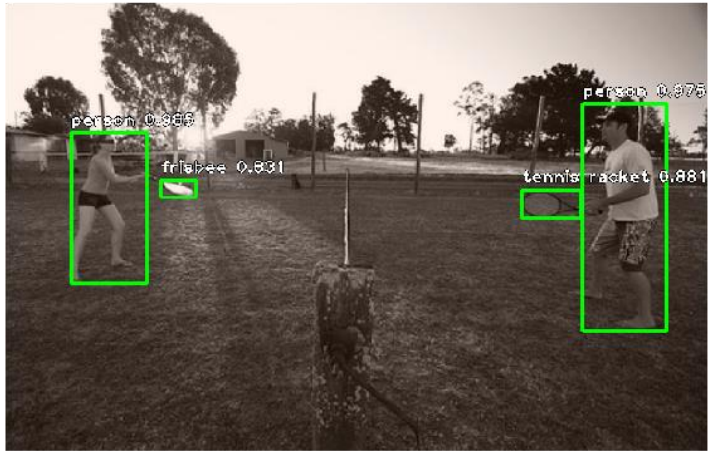
Keras-RL

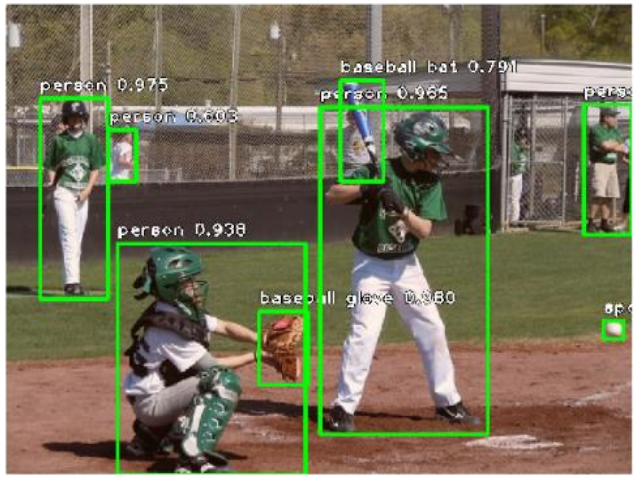


Keras RetinaNet

fizyr

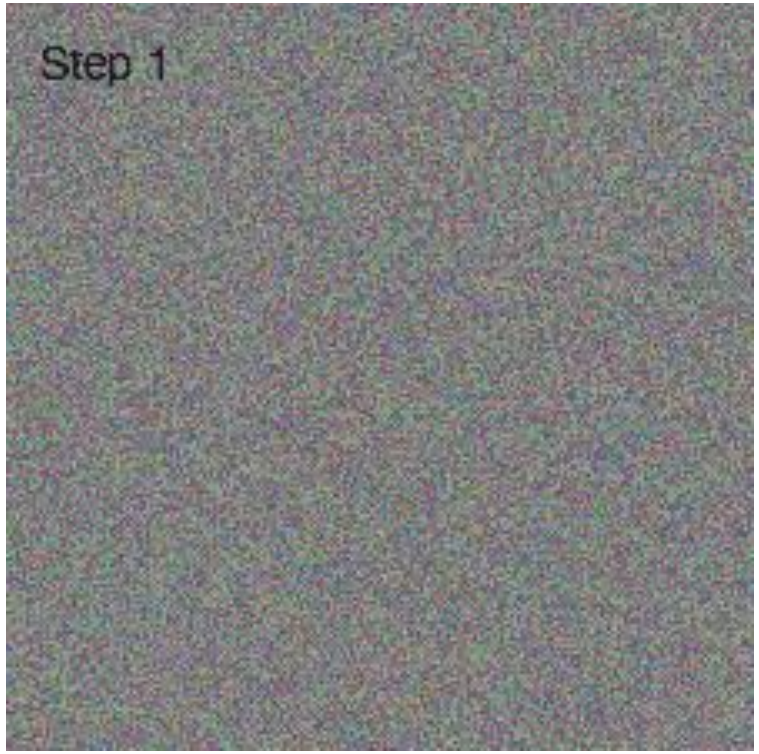






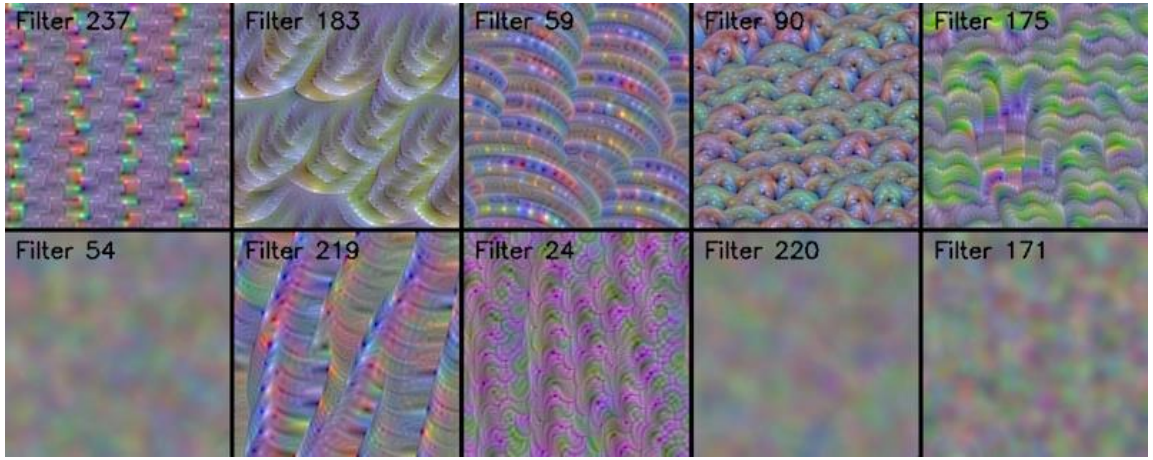
Keras Visualization Toolkit

raghakot

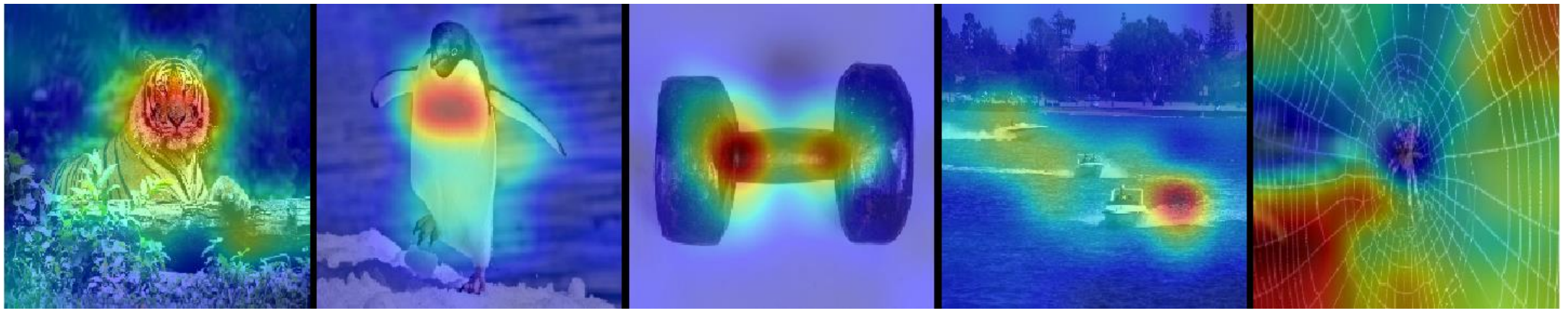


Keras Visualization Toolkit

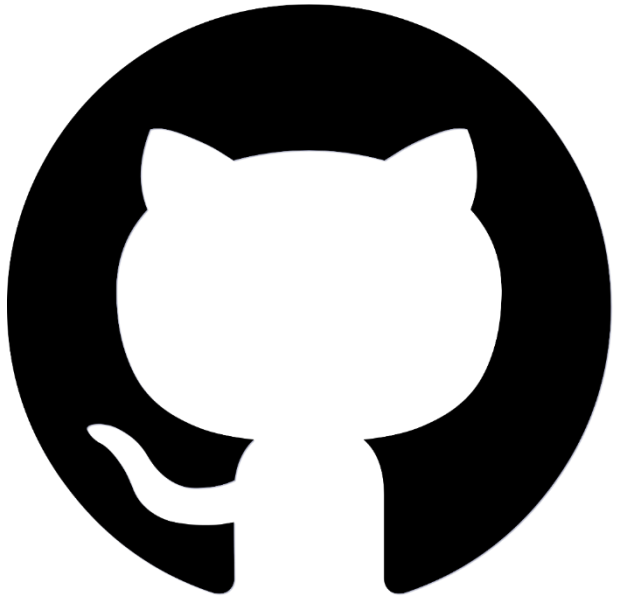
Conv filter visualization



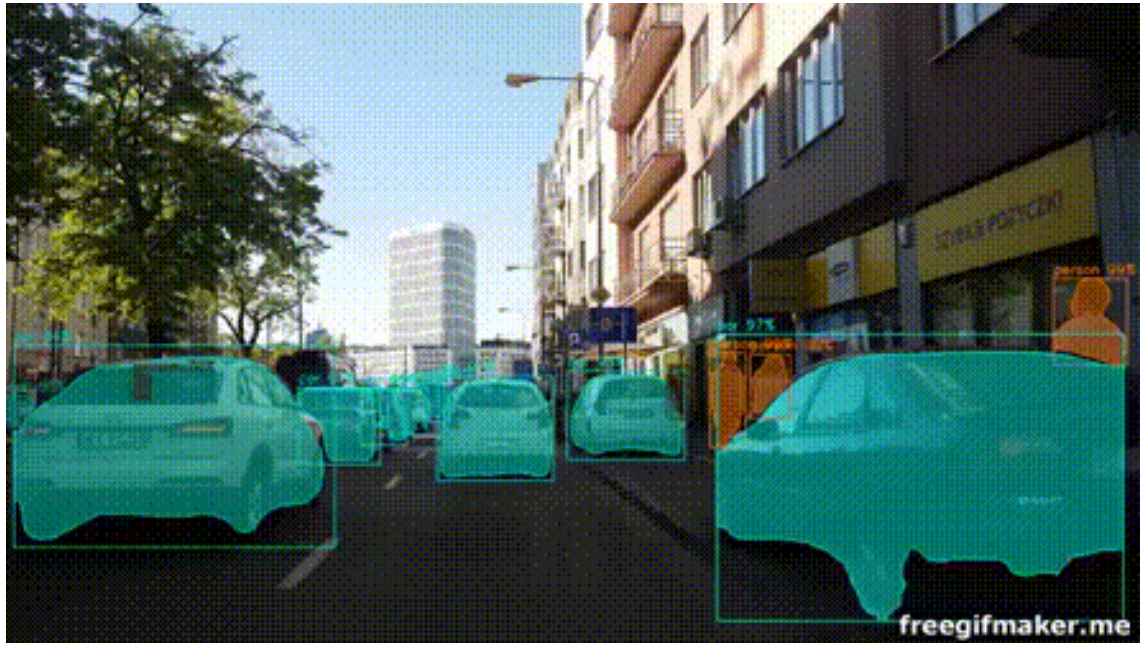
Attention Maps



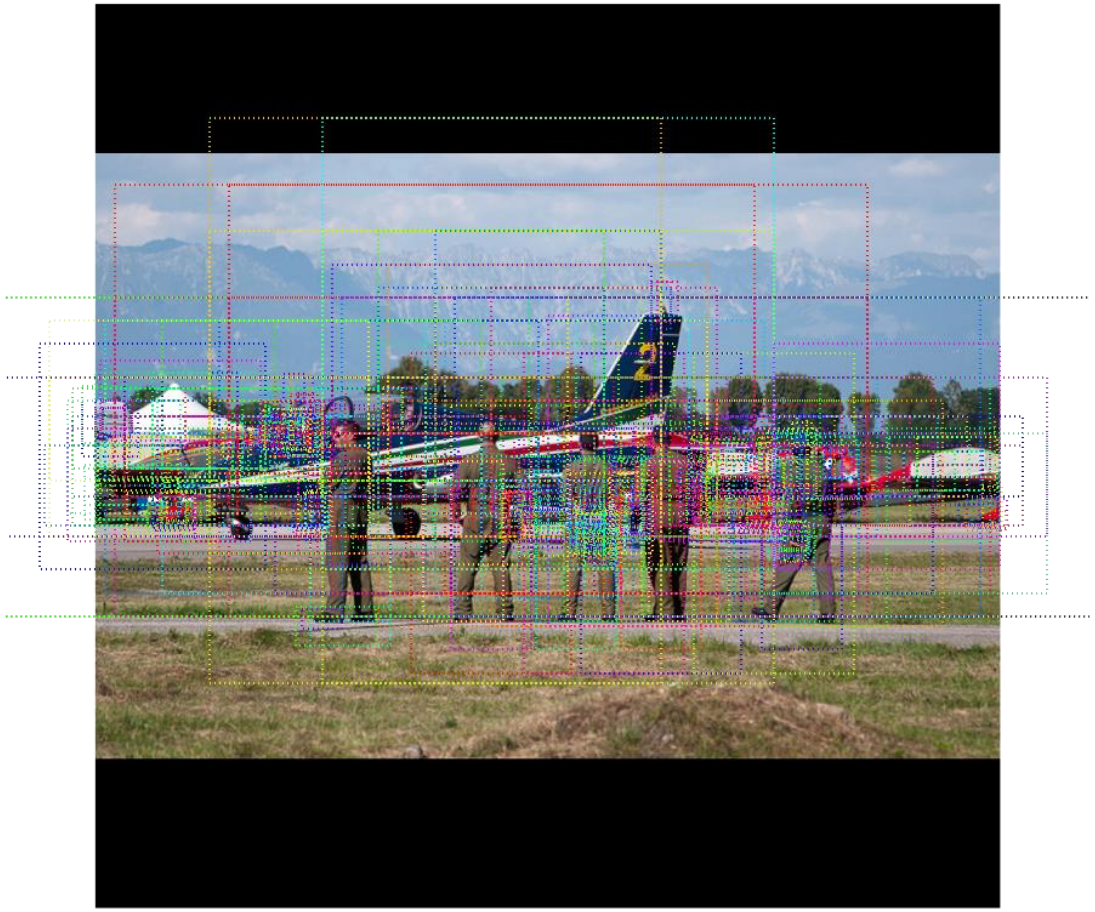
Mask R-CNN for Object Detection and Segmentation



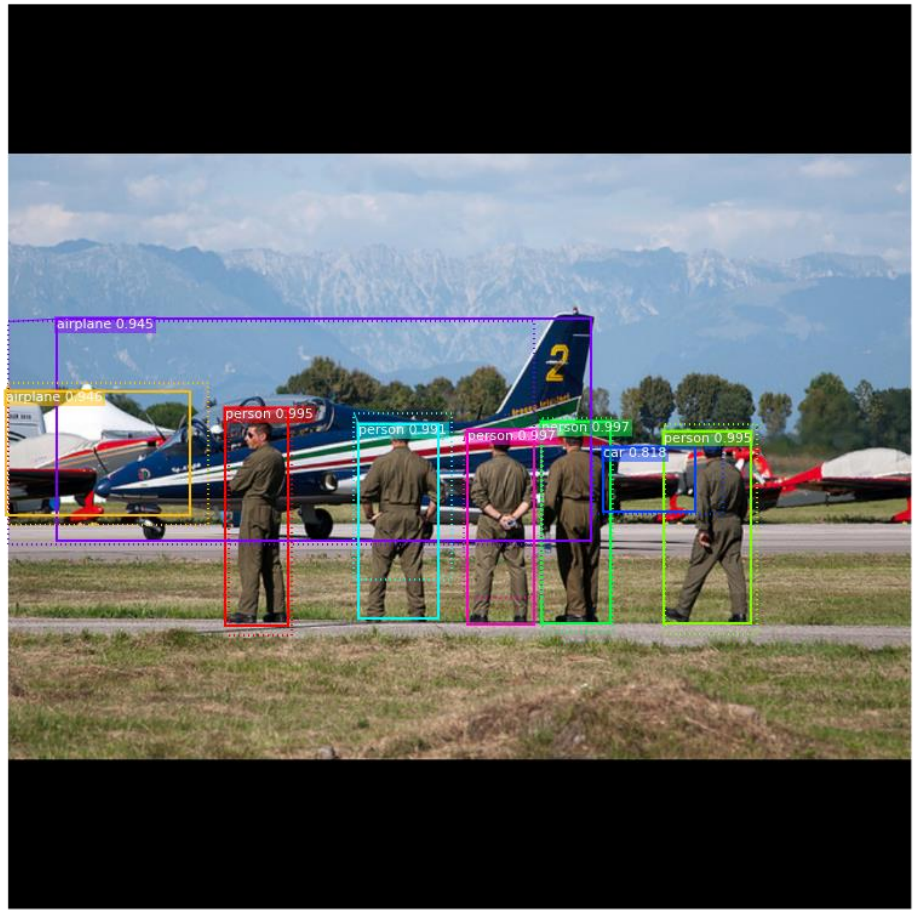
matterport



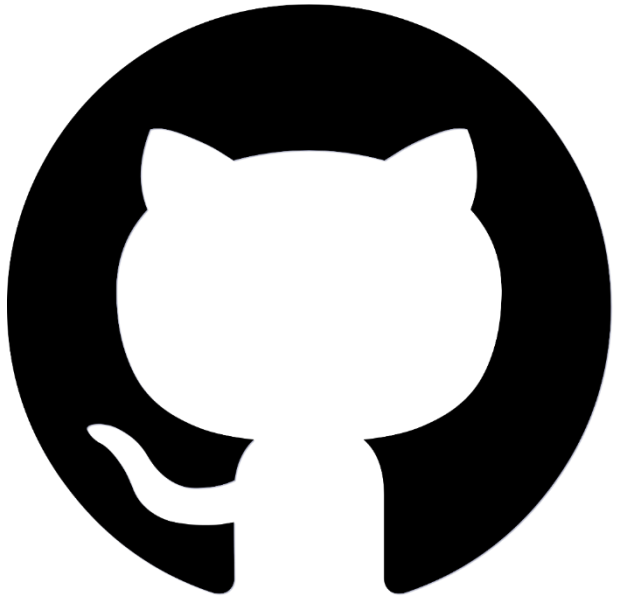
Mask R-CNN for Object Detection and Segmentation



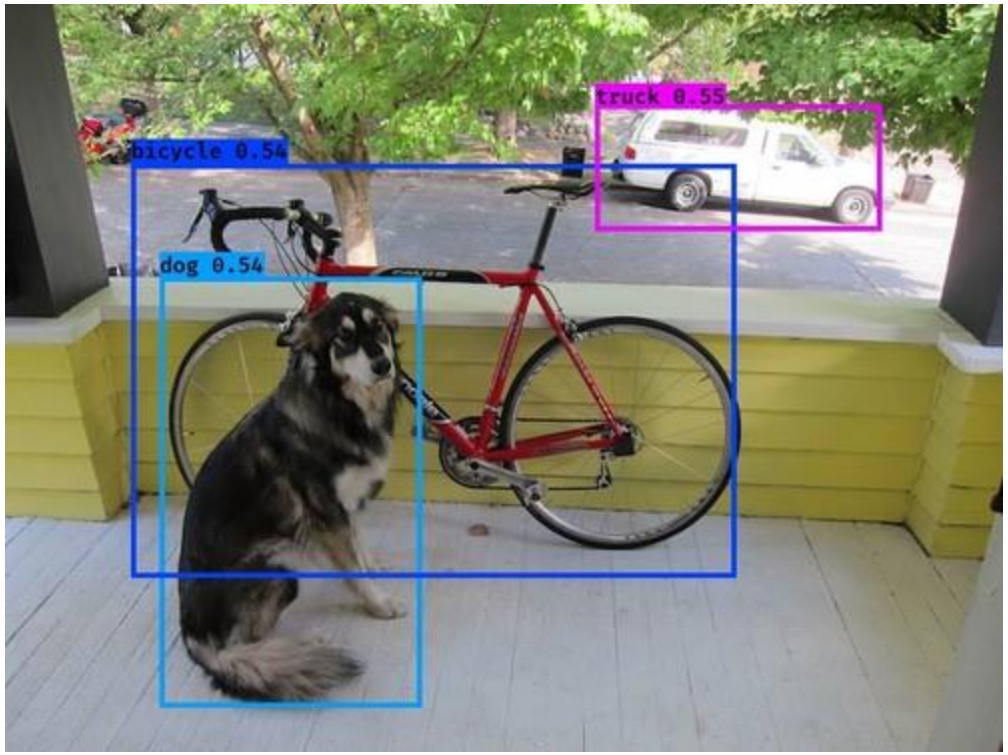
Detections after NMS



YAD2K: Yet Another Darknet 2 Keras



allanzelener



YOLO v2*

* Não escrito com em Python

Referências

- Barrientos, J. L., & Freire, F. V. (n.d.). *Leaf Recognition with Deep Learning and Keras using GPU computing*. Retrieved from https://ddd.uab.cat/record/188771
- Suganya, P., Sathya, R., & Vijayalakshmi, K. (2018). Detection and recognition of hand gestures to control the system applications by neural networks. *International Journal of Pure and Applied Mathematics*, *118*, 399–405.
- Sethi, A., Sankaran, A., Panwar, N., Khare, S., & Mani, S. (2017). DLPaper2Code: Auto-generation of Code from Deep Learning Research Papers. *CoRR*, *abs/1711.03543*. Retrieved from http://arxiv.org/abs/1711.03543
- Vinicki, K., Ferrari, P., Belic, M., & Turk, R. (2018). Using Convolutional Neural Networks for Determining Reticulocyte Percentage in Cats. *CoRR*, *abs/1803.04873*. Retrieved from http://arxiv.org/abs/1803.04873

Referências

- keras.io
- http://ruder.io/optimizing-gradient-descent/
- http://cs231n.github.io/neural-networks-3/
- https://github.com/PetraVidnerova/rbf_keras
- https://github.com/keras-rl/keras-rl
- https://github.com/fizyr/keras-retinanet
- https://github.com/raghakot/keras-vis
- https://github.com/matterport/Mask_RCNN
- https://github.com/allanzelener/YAD2K