WORCAP 2021 – XXI Workshop em Computação Aplicada do Instituto Nacional de Pesquisas Espaciais

INTRODUÇÃO À VISÃO COMPUTACIONAL

DIA 1

RAFAEL MARINHO DE ANDRADE

16 de setembro de 2021

WORCAP2021
Workshop em Computação Aplicada

COMPUTAÇÃO APLICADA

INPE
UNIDADE DE PESQUISA DO MCTI

AGENDA

DIA 1

1) O que é Visão Computacional;
2) Visão geral do processo de desenvolvimento;
3) Passo-a-passo;
    3.1) Obtenção de dados;
    3.2) Rotulagem dos dados;
    3.3) Ambiente de treinamento de uma rede convolucional;
    3.4) Processo de treinamento da rede convolucional;
    3.5) Ativação de uma rede.
4) Indo além.

MATERIAL DE APOIO

NÃO FIQUE DE MÃOS VAZIAS

Para os que se aventurarem a acompanhar com as mãos na massa:

TOY PROBLEM:
https://drive.google.com/drive/folders/1G3YWORqVE_ESF6mT_W4oWSPe1Kl9O3c0?usp=sharing

NOTEBOOK DO COLAB:
https://colab.research.google.com/drive/1WqAdqP6JczFDvg4IcDhjpmcE-2kQHOMe?usp=sharing

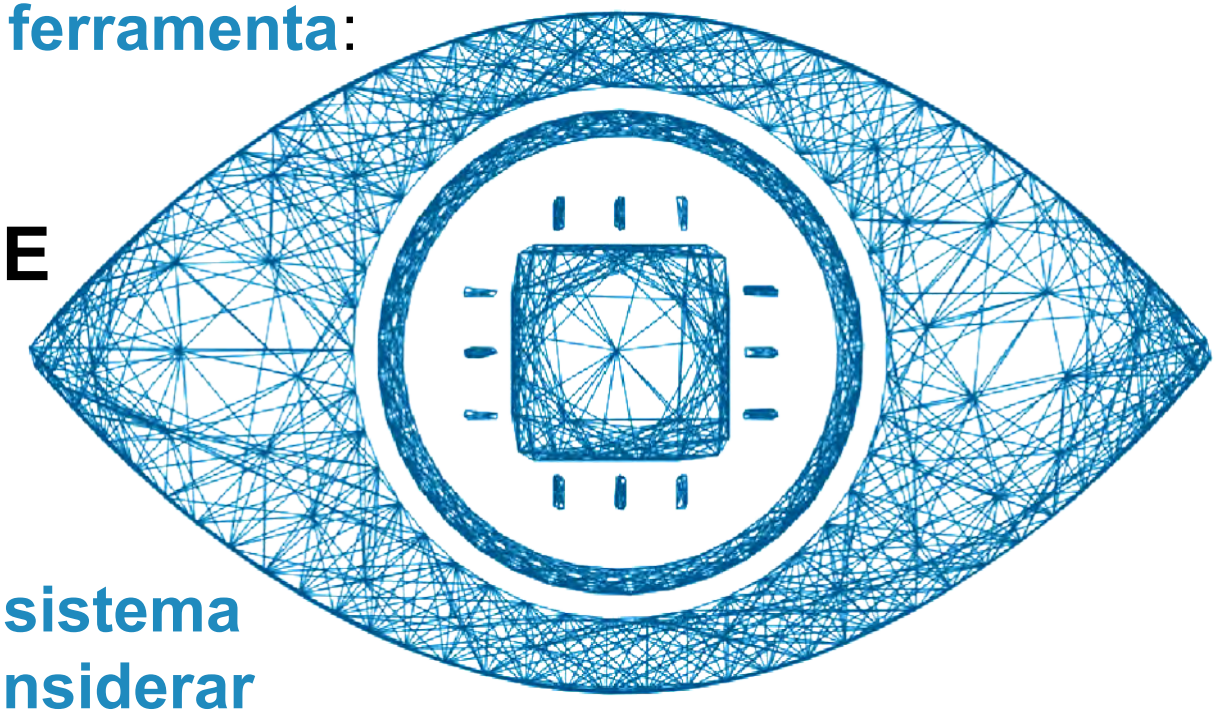
REPOSITÓRIO CONVTRAINING (GITHUB):
https://github.com/Rafael-Marinho/ConvTraining

Tudo isso, além desses slides, estão sendo postados no chat do **YouTube**.

O QUE É VISÃO COMPUTACIONAL?

O QUE É VISÃO COMPUTACIONAL

O conceito de visão se torna palpável ao considerarmos que **conseguimos atribuir a capacidade de visão à uma ferramenta**:



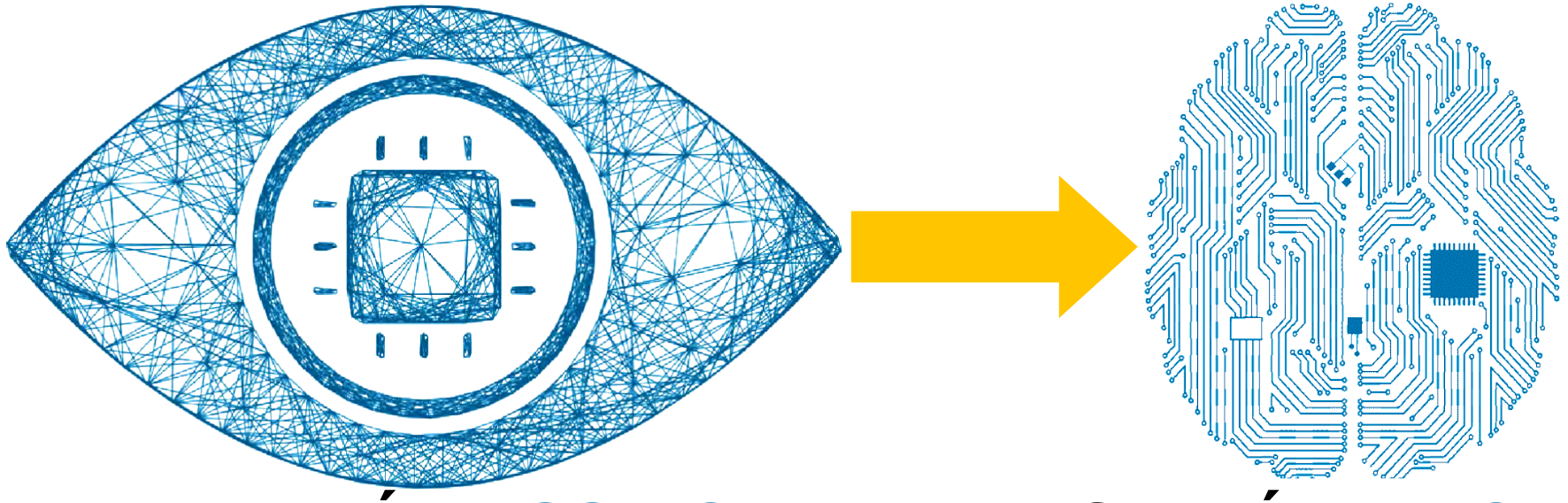
PODEMOS ENXERGAR PORQUE TEMOS UM PAR DE OLHOS.

Logo, **a presença de câmeras em um sistema computacional é o suficiente para considerar que tal máquina possui uma visão**?

Para entender melhor, precisamos saber a diferença entre **ver** e **enxergar**.

O QUE É VISÃO COMPUTACIONAL

A diferença está na “**profundidade**”, na **complexidade** da visão:



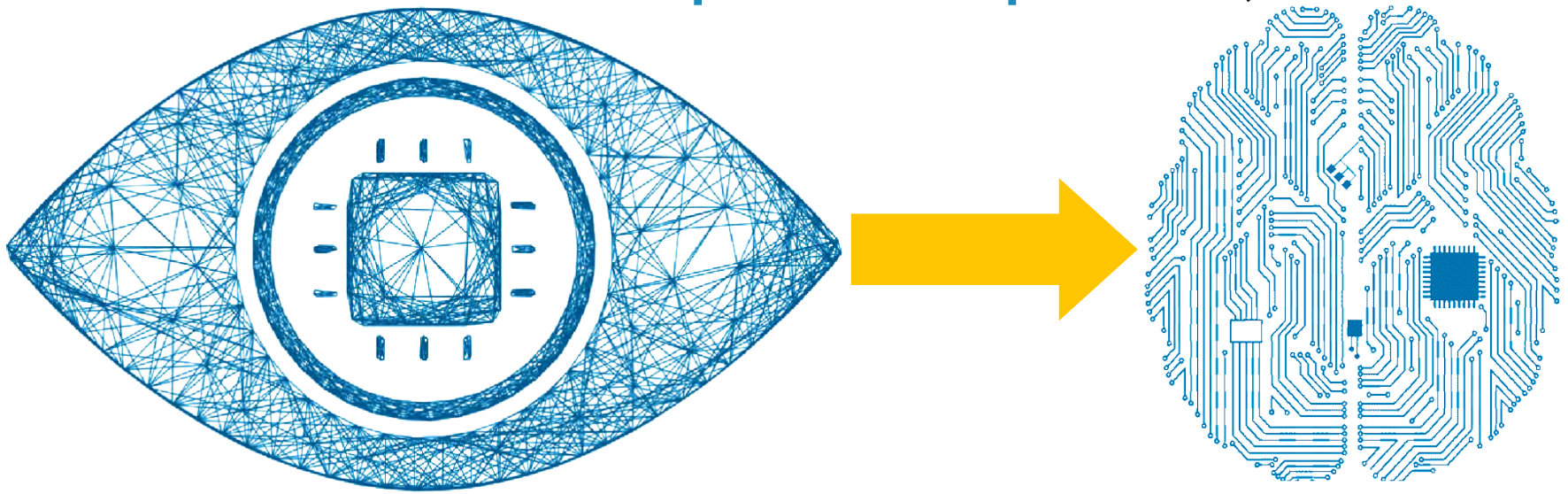
VER É PASSIVO E ENXERGAR É ATIVO.

O ser humano é capaz de ver porque tem um par de olhos
mas **é capaz de enxergar porque tem um cérebro**.

O QUE É VISÃO COMPUTACIONAL

Quando uma câmera é equipada em um computador, o computador **é capaz de extrair imagens** do mundo ao seu redor.

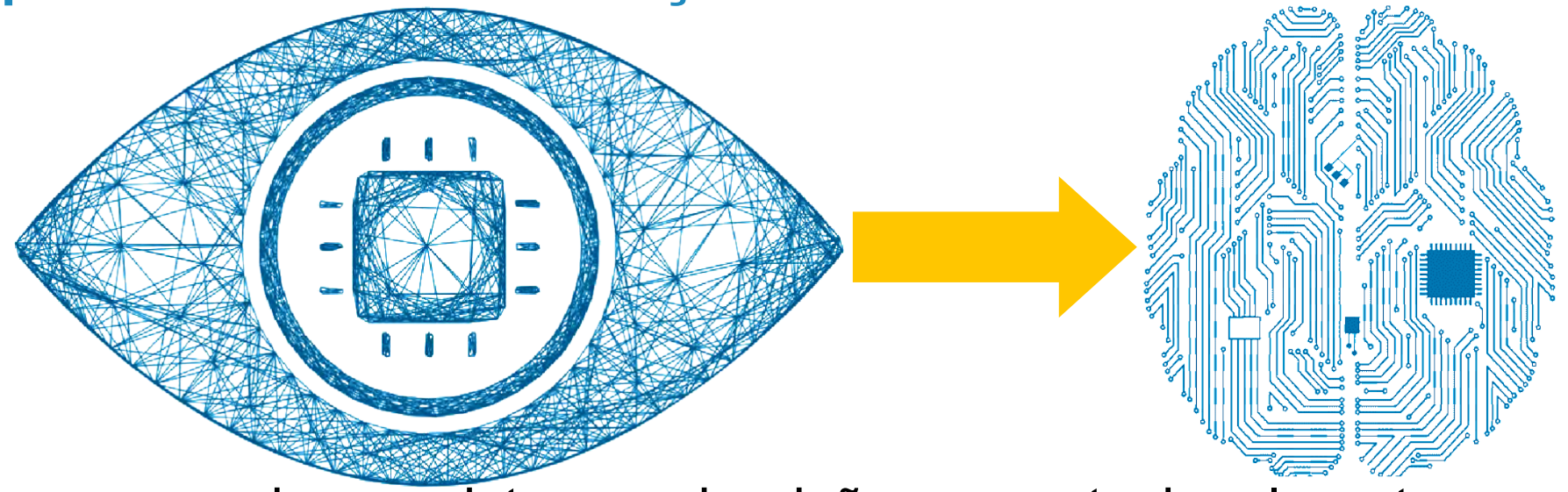
Isso **não o torna capaz de interpretá-las**, no entanto.



Então o computador, através de uma câmera, vê. Para enxergar, no entanto, **o computador precisa ser capaz de processar e interpretar essas imagens**.

O QUE É VISÃO COMPUTACIONAL

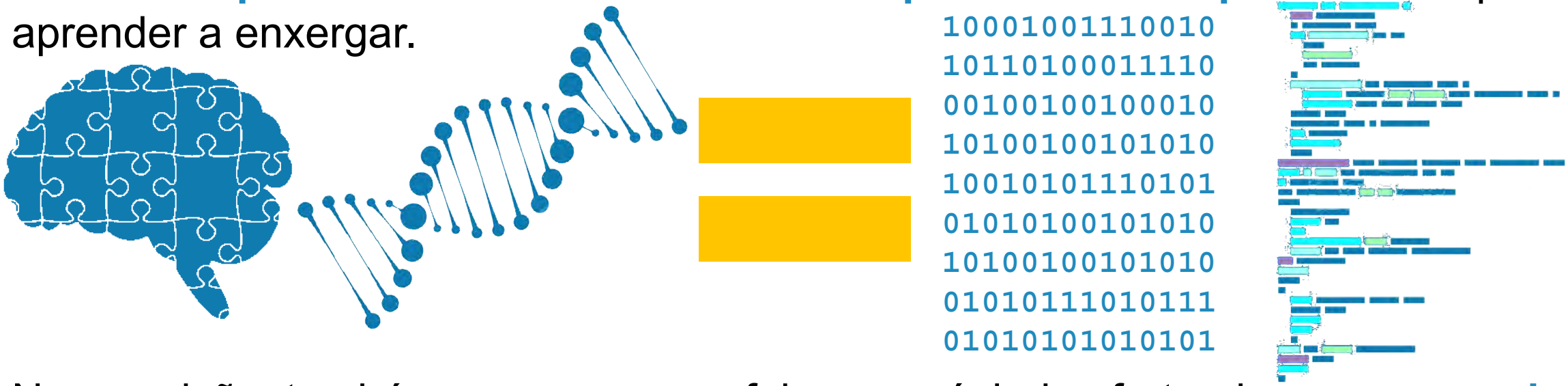
Somos dotados de ferramentas que nos permitem enxergar, portanto: **sensores** como os olhos para **adquirir informações visuais** e um **cérebro** para **processar essas informações**.



Do mesmo modo, os sistemas de visão computacional contam com sensores como **câmeras** para adquirir informações visuais e recursos computacionais para processá-las (como **hardware e software**).

O QUE É VISÃO COMPUTACIONAL

E assim como **aprendemos a enxergar**, usando informações e conhecimentos adquiridos em vida e herdados em nosso código genético, os **sistemas de visão computacional necessitam de um processo de aprendizado** para aprender a enxergar.

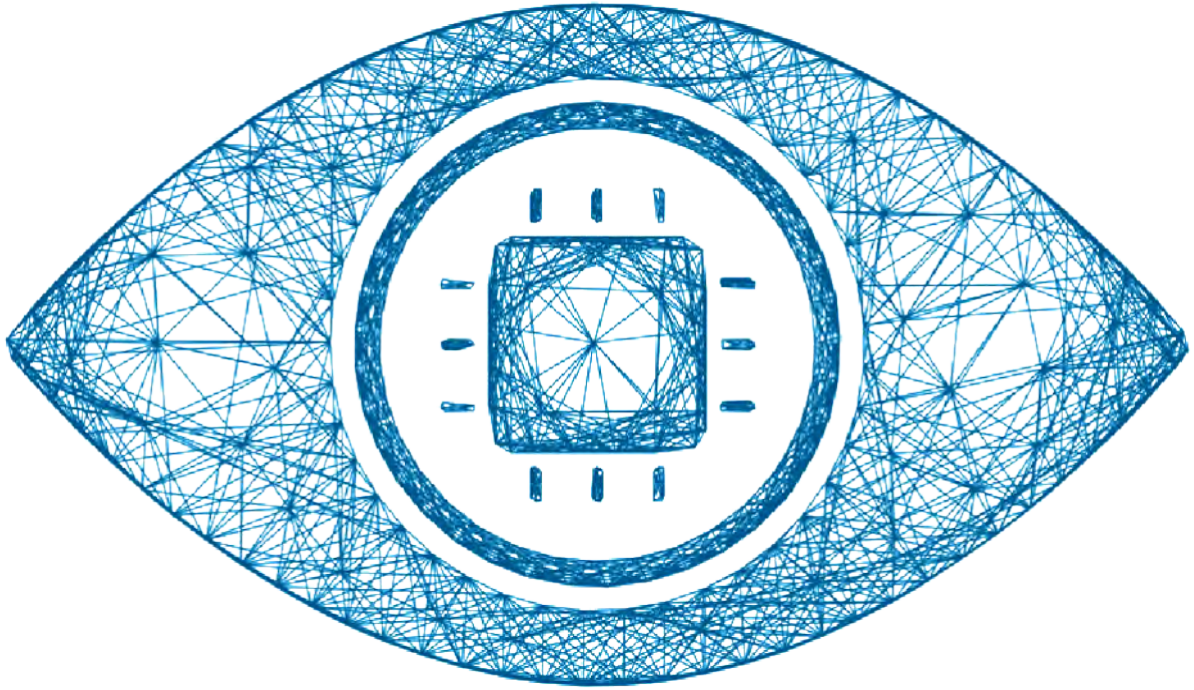


Nossa visão também nem sempre foi como é hoje, fruto do **processo de evolução** do ser humano. Assim também são as técnicas de **desenvolvimento das ferramentas e técnicas computacionais ligadas à visão**.

O QUE É VISÃO COMPUTACIONAL

GROSSEIRAMENTE FALANDO, PORTANTO:

A VISÃO COMPUTACIONAL BUSCA **REPLICAR A CAPACIDADE HUMANA DE ENXERGAR**, FAZENDO USO DE RECURSOS COMPUTACIONAIS PARA ISSO.



O QUE É VISÃO COMPUTACIONAL

AS CAPACIDADES DA VISÃO

O ser humano tem capacidades de percepção muito complexas e igualmente natas, como **detectar**, **discriminar**, **segmentar**, **analisar**, **assimilar**, **contextualizar**, **relacionar** e **associar**.

CLASSIFICAÇÃO

DETECÇÃO + CLASSIFICAÇÃO

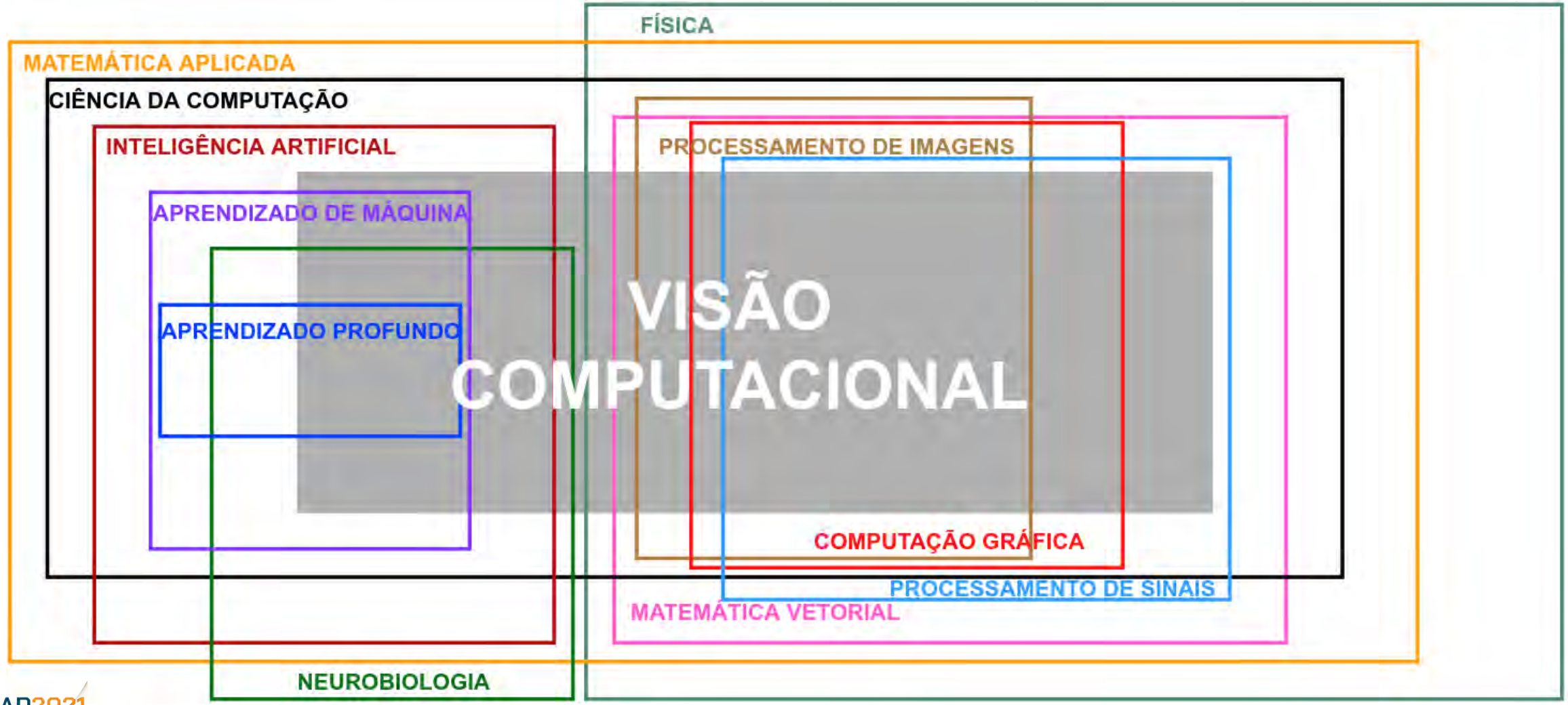
SEGMENTAÇÃO SEMÂNTICA

SEGMENTAÇÃO INSTANCIAL

Essas habilidades são consideradas os **principais objetivos da visão computacional**.

O QUE É VISÃO COMPUTACIONAL

ONDE SITUA-SE A VISÃO COMPUTACIONAL

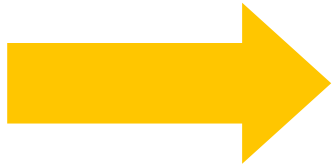


O QUE É POSSÍVEL FAZER COM VISÃO COMPUTACIONAL

O QUE É POSSÍVEL

Tal como com a ciência e tecnologia em geral, as possibilidades da visão computacional ficam limitadas pela imaginação, conhecimento técnico e recursos disponíveis. Os últimos 50 anos de evolução da Visão Computacional nos permitem hoje **experimentar suas aplicações para praticamente qualquer problema concebível**, seja a partir de réplicas da visão humana ou ainda além dela.

VEJAMOS A SEGUIR ALGUNS EXEMPLOS TANGÍVEIS



https://www.pyimagesearch.com/blog/

O QUE É POSSÍVEL

EXEMPLOS

CORREÇÕES E APRIMORAMENTO DE IMAGENS

Colorização

Eliminação de rasuras

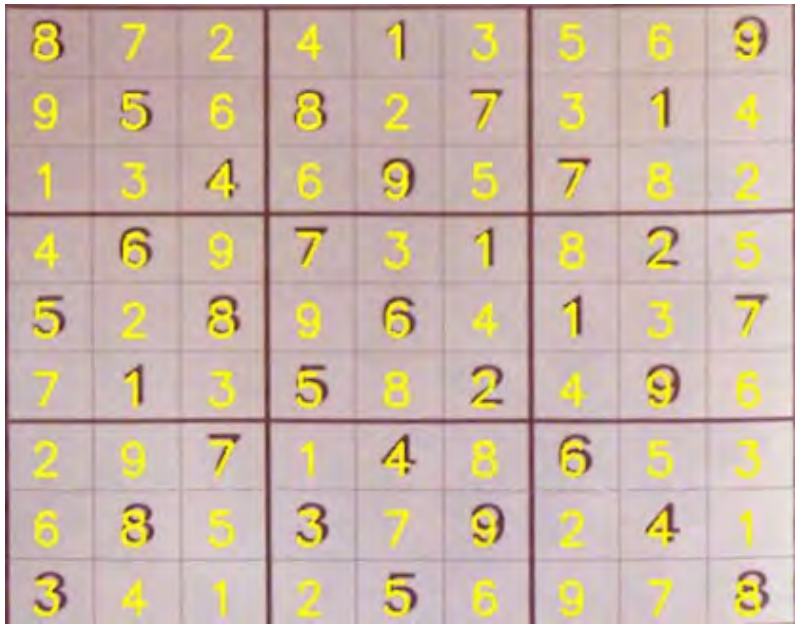
Superresolução

https://www.pyimagesearch.com/2020/11/09/opencv-super-resolution-with-deep-learning/
https://www.pyimagesearch.com/2020/05/18/image-inpainting-with-opencv-and-python/
https://www.pyimagesearch.com/2019/02/25/black-and-white-image-colorization-with-opencv-and-deep-learning/

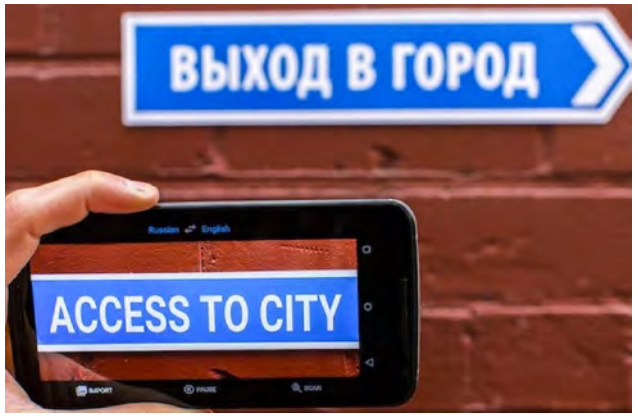
O QUE É POSSÍVEL

EXEMPLOS

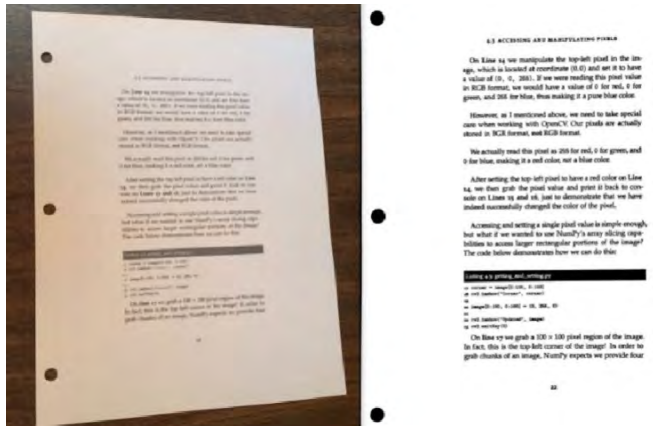
AGILIZAR COISAS DO COTIDIANO



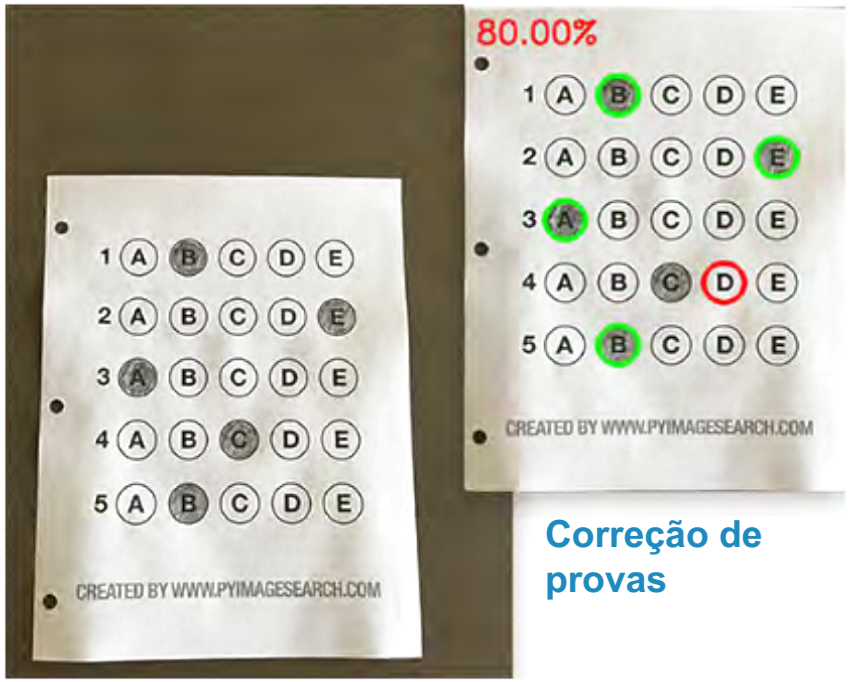
Resolução de jogos



Tradução simultânea de termos



Digitalização de documentos



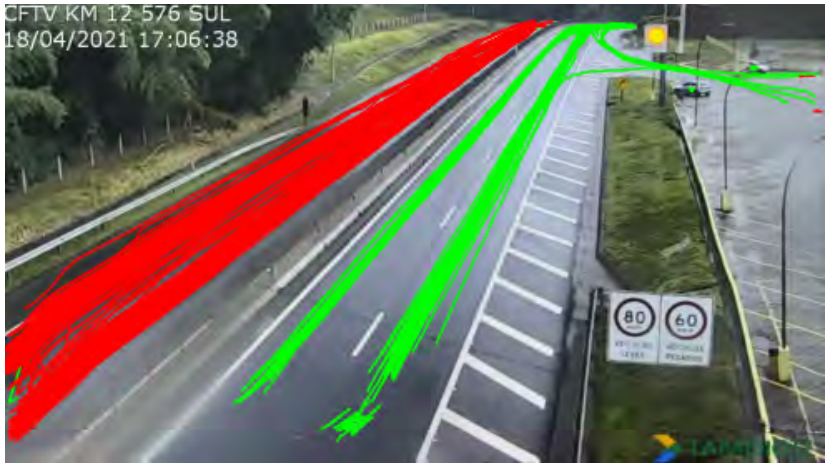
Correção de provas

https://www.pyimagesearch.com/2020/08/10/opencv-sudoku-solver-and-ocr/
https://www.pyimagesearch.com/2016/10/03/bubble-sheet-multiple-choice-scanner-and-test-grader-using-omr-python-and-opencv/
https://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/

O QUE É POSSÍVEL

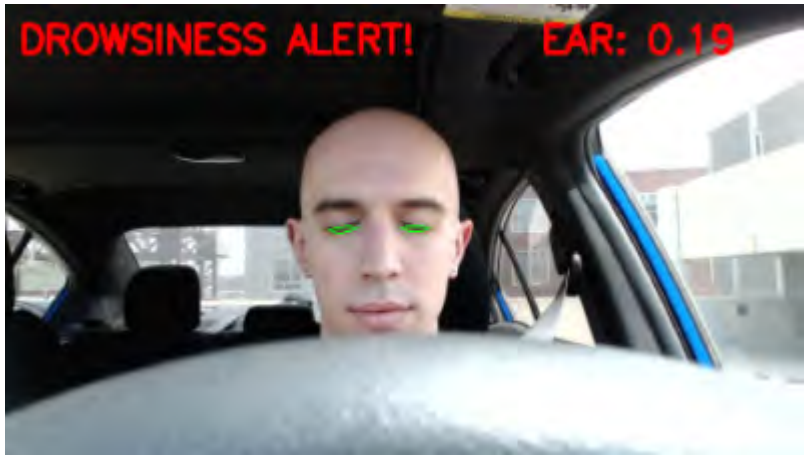
EXEMPLOS

Detecção de pontos de referência



Detecção de rotas e rastreio

Estimação de velocidade



Detecção de sonolência

CONTEXTO DE VEÍCULOS RODOVIÁRIOS

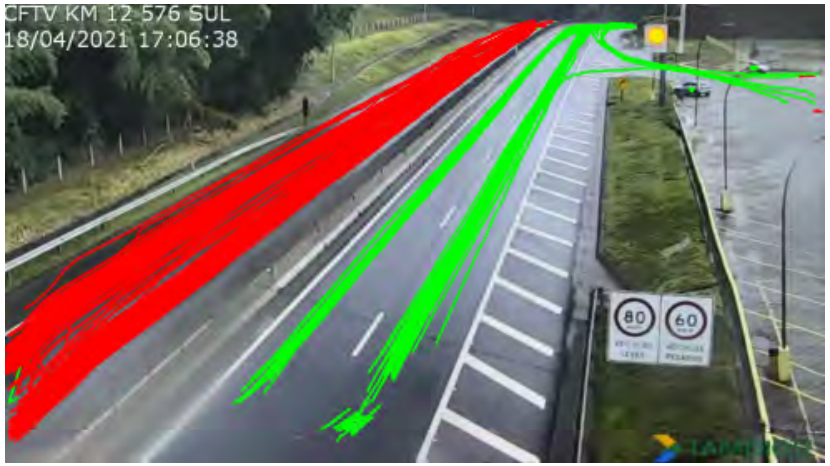
Identificação de placas de trânsito

https://www.pyimagesearch.com/2019/12/02/opencv-vehicle-detection-tracking-and-speed-estimation/
https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/
https://www.pyimagesearch.com/2017/05/08/[illegible]

O QUE É POSSÍVEL

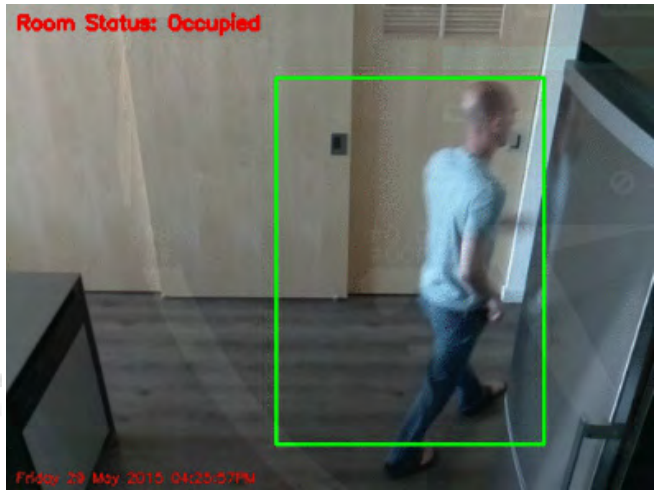
EXEMPLOS

SEGURANÇA E VIGILÂNCIA



Rastreio de veículos por placa

Autenticação de identidade



Detecção de atividade

https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/
https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/
https://www.pyimagesearch.com/2019/11/25/human-activity-recognition-with-opencv-and-deep-learning/

O QUE É POSSÍVEL

EXEMPLOS

SEGURANÇA E VIGILÂNCIA

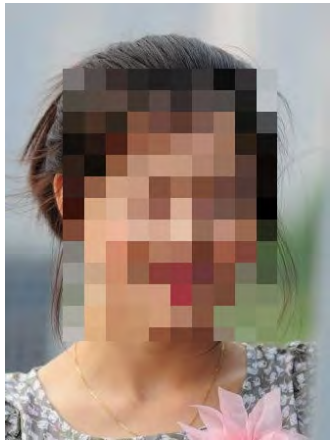
https://www.pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/
https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/
https://www.pyimagesearch.com/2019/11/25/[illegible]

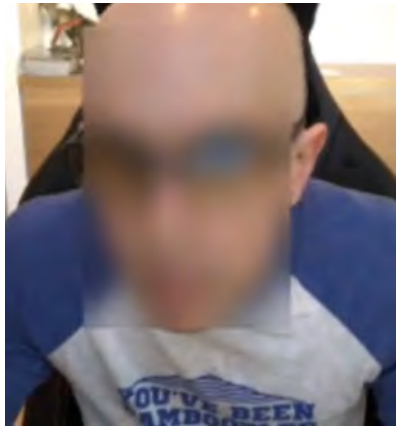
O QUE É POSSÍVEL

EXEMPLOS

INTEGRIDADE

Detecção de menores de idade em condição de combate militar.



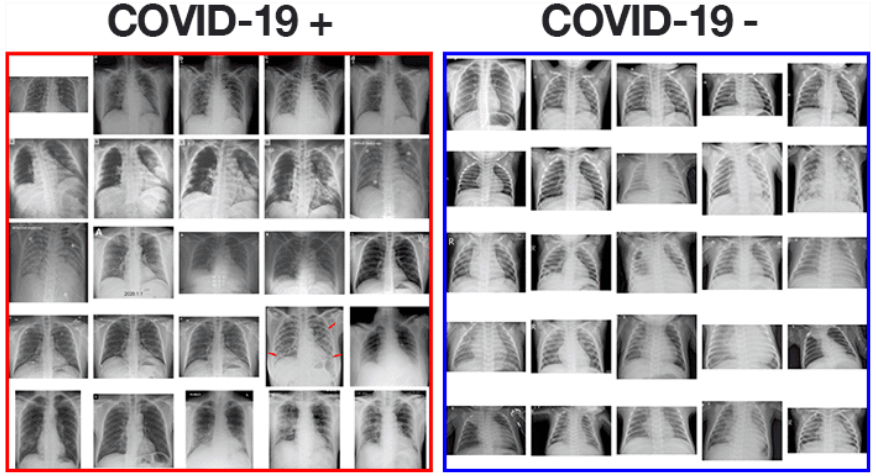


Censura e anonimização

https://www.pyimagesearch.com/2020/05/11/an-ethical-application-of-computer-vision-and-deep-learning-identifying-child-soldiers-through-automatic-age-and-military-fatigue-detection/
https://www.pyimagesearch.com/2020/04/06/blur-and-anonymize-faces-with-opencv-and-python/

O QUE É POSSÍVEL

EXEMPLOS



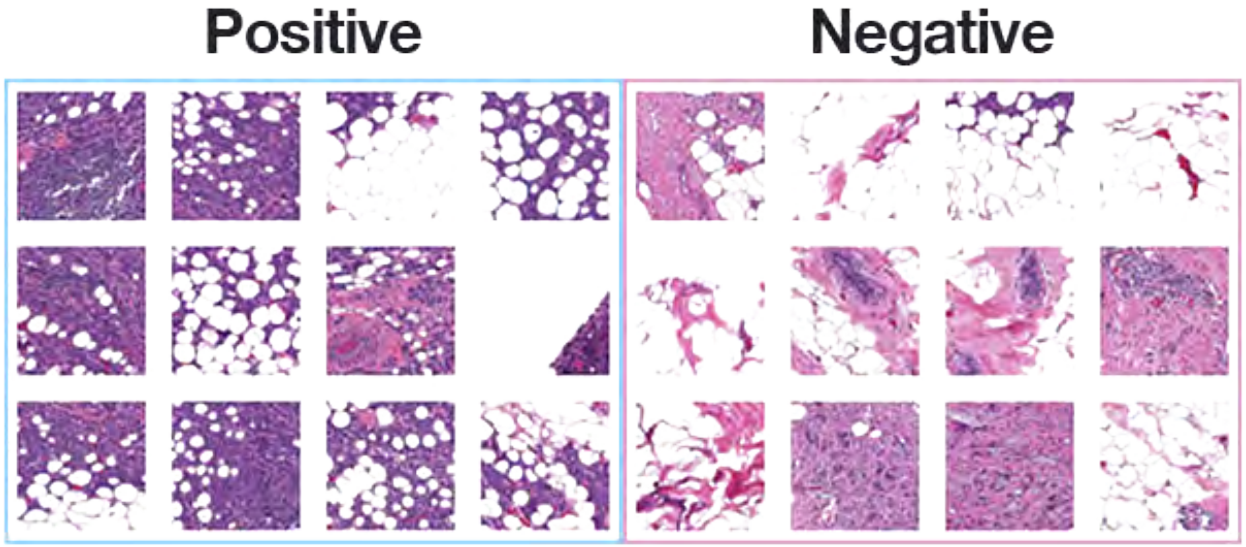
PANDEMIA COVID-19

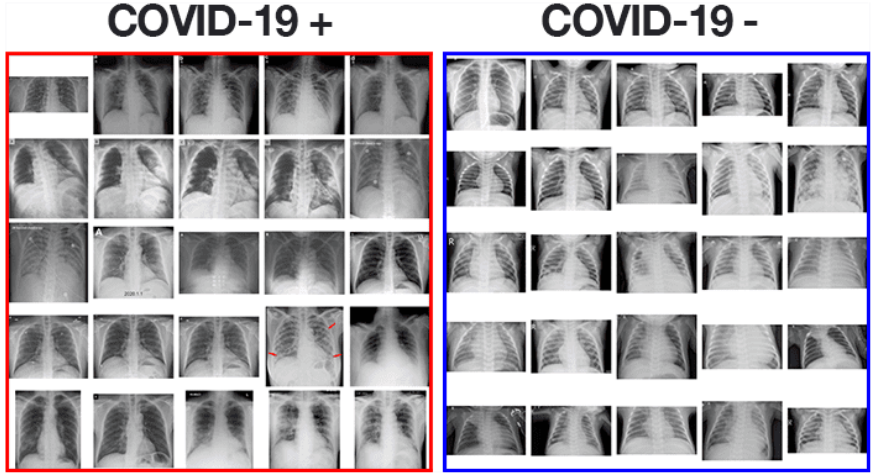
https://www.pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/
https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/
https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/
https://www.pyimagesearch.com/2019/02/18/breast-cancer-classification-with-keras-and-deep-learning/
https://www.pyimagesearch.com/2018/12/03/deep-learning-and-medical-image-analysis-with-keras/

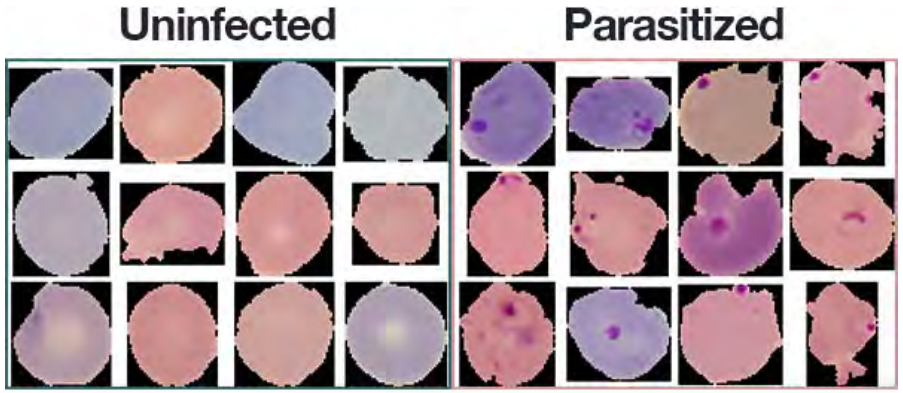
O QUE É POSSÍVEL

EXEMPLOS

Os princípios para detecção de COVID-19 podem ser aplicados inclusive para **detecção de outras doenças**, como **câncer de mama** e **malária**.





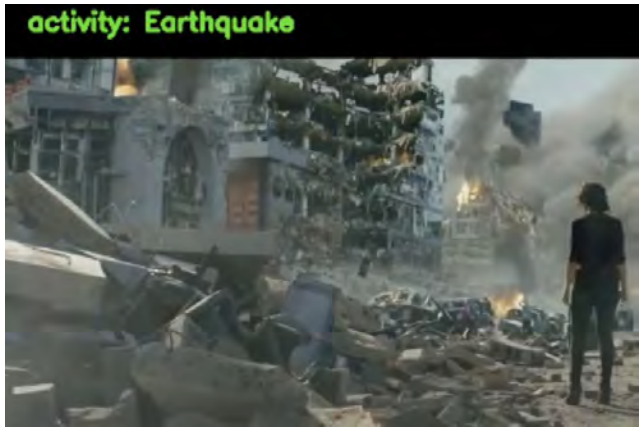


https://www.pyimagesearch.com/2020/06/01/opencv-social-distancing-detector/
https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/
https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/
https://www.pyimagesearch.com/2019/02/18/breast-cancer-classification-with-keras-and-deep-learning/
https://www.pyimagesearch.com/2018/12/03/deep-learning-and-medical-image-analysis-with-keras/

O QUE É POSSÍVEL

EXEMPLOS

DETECÇÃO DE DESASTRES

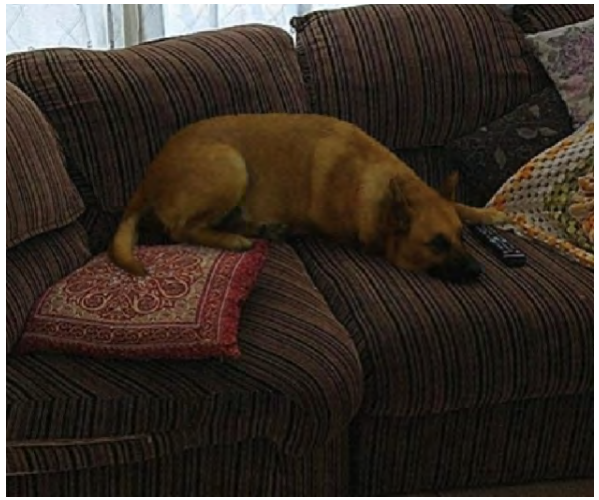


https://www.pyimagesearch.com/2019/11/18/fire-and-smoke-detection-with-keras-and-deep-learning/
https://www.pyimagesearch.com/2019/11/11/detecting-natural-disasters-with-keras-and-deep-learning/

O QUE É POSSÍVEL

EXEMPLOS

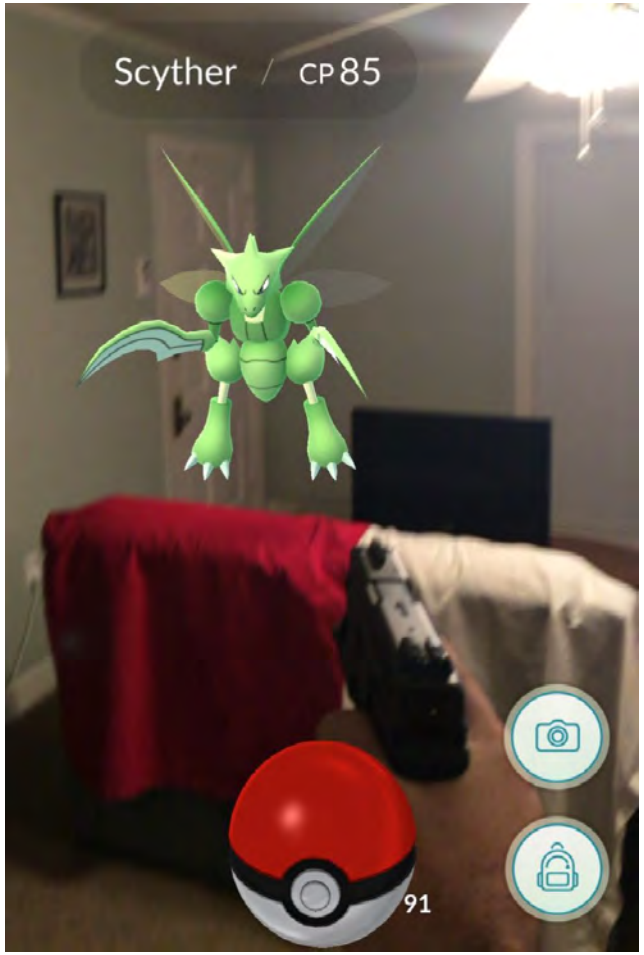
REALIDADE AUMENTADA

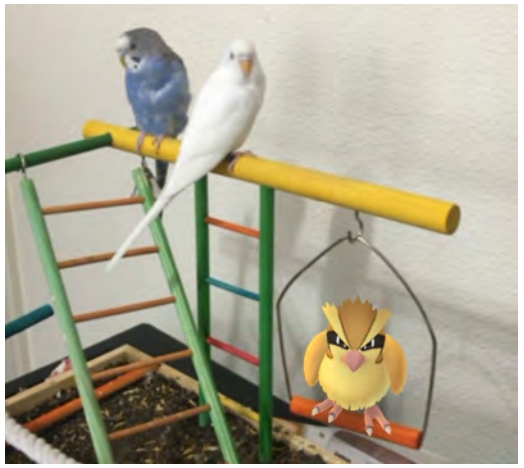


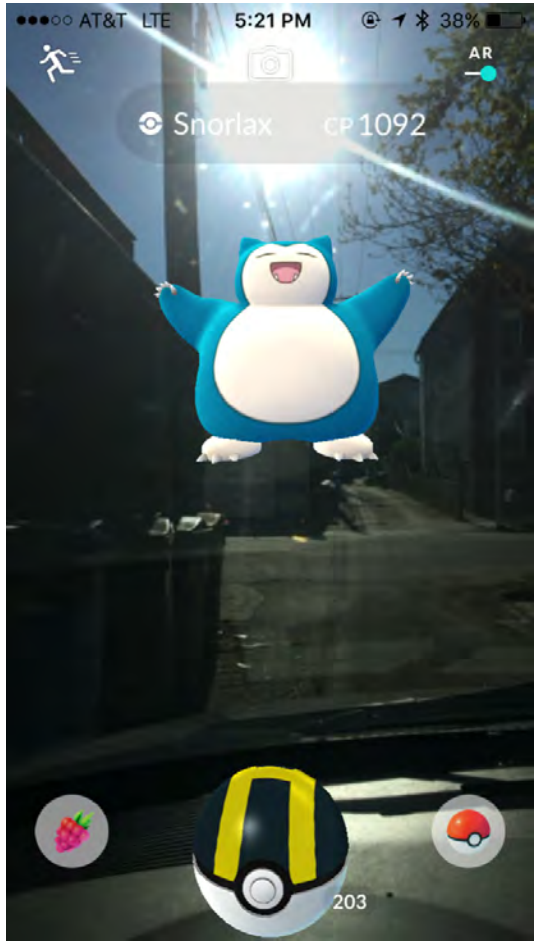
https://www.pyimagesearch.com/2021/01/11/opencv-video-augmented-reality/

O QUE É POSSÍVEL

EXEMPLOS







https://www.pyimagesearch.com/2021/01/11/opencv-video-augmented-reality/

O QUE É POSSÍVEL

EXEMPLOS

Pokédex

INVENTE.
ALOPRE.

Catalogação de deck de Yu-Gi-Oh (com duelos)

https://www.pyimagesearch.com/2014/05/19/building-pokedex-python-comparing-shape-descriptors-opencv/
https://www.pyimagesearch.com/2021/03/03/an-interview-with-anthony-lowhur-recognizing-10000-yugioh-cards-with-computer-vision-and-deep-learning/

O QUE É POSSÍVEL

EXEMPLOS

HISTÓRIAS DA ÉPOCA DA FACULDADE...

INVENTE.
ALOPRE.

MORAL DA HISTÓRIA:
CONTROLE DE ESTOQUE

O QUE É POSSÍVEL

EXEMPLOS

O QUE MAIS PODERIA SER FEITO?

SEJAM CRIATIVOS

COMENTEM

SONHEM

VISÃO GERAL DO PROCESSO

VISÃO GERAL DO PROCESSO

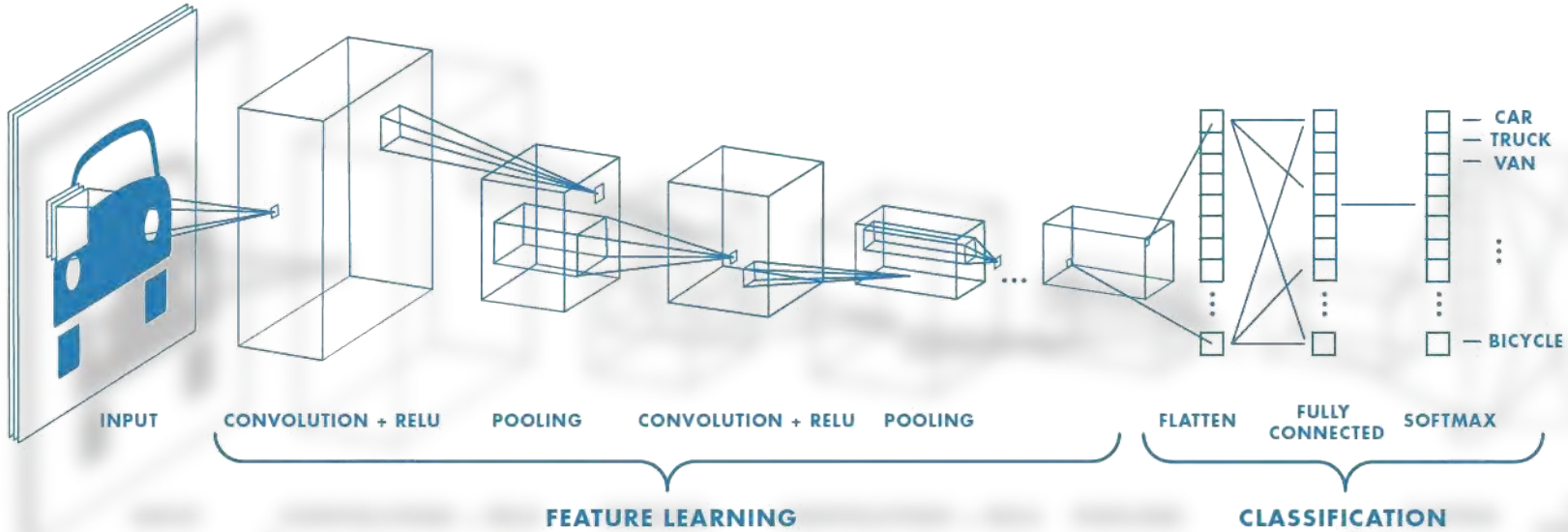
PASSO-A-PASSO

VOCÊ VAI PRECISAR:

EM LINHAS GERAIS, O PROCESSO É O MESMO DE QUALQUER OUTRA REDE NEURAL. É O PADRÃO DO APRENDIZADO DE MÁQUINA.

1) obter os dados (imagens);
2) preparar os dados para serem “compreendidos” durante o treinamento da rede convolucional, o que envolve:
    2.1) a rotulagem dos objetos presentes em imagens para exemplificação das classes; e então
    2.2) segmentação dos dados em subconjuntos de treinamento, validação e, possivelmente, teste;
3) modelar a rede convolucional;
4) treinar a rede convolucional do passo 3) com os dados do passo 2.2);
5) aplicar a rede convolucional treinada em alguma aplicação (ativação); e
6) desenvolver a aplicação para resolver o problema proposto.

TECNOLOGIAS EMPREGADAS



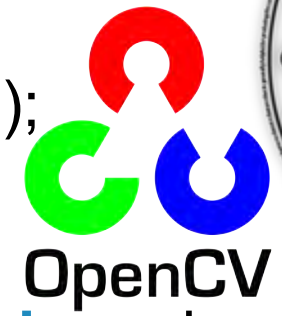
TECNOLOGIAS EMPREGRADAS

PASSO A PASSO

As tecnologias empregadas são:

- Rotulagem (labeling): **LabelImg**;
- Framework de rede neural: **Darknet**;
- Rede convolucional de detecção: **YOLOv4**;
- Framework de execução: **OpenCV** ($\geq$ 4.4.0);
- Linguagem de programação: **Python3**, **C++**.



Além dessas tecnologias, o ambiente **Google Colab** pode ser fundamental para o treinamento da rede convolucional em tempo viável.

TREINANDO SUA PRIMEIRA CNN

TREINANDO SUA PRIMEIRA CNN

ABORDAGEM DE PROBLEMAS

O primeiro passo para o desenvolvimento é ter um problema para ser resolvido com visão computacional: estamos falando de problemas onde, de modo geral, **você precisa detectar algo específico** e poderia delegar tal tarefa à uma máquina.

Se você tem um problema a ser abordado com visão computacional, **você pode estar prestes a dar o primeiro passo**.

Mas se você não tem um problema e quer acompanhar com as mãos na massa, fique à vontade para explorar o **toy problem** proposto.

TREINANDO SUA PRIMEIRA CNN

ABORDAGEM DE PROBLEMAS

Contextualização do toy problem:

Durante a pandemia de COVID-19, muitas cidades decidiram adotar medidas de restrições para minimizar as interações humanas presenciais. Uma delas foi o lockdown, que deixou as ruas vazias.
As ruas vazias se mostraram, então, um ambiente onde animais selvagens (muitos deles perigosos) imperassem.

O toy problem consiste, então, em **detectar animais selvagens em ambientes urbanos**.

TREINANDO SUA PRIMEIRA CNN

ABORDAGEM DE PROBLEMAS

DADOS

DADOS

A rede neural convolucional que será treinada será por fim ativada em imagens, permitindo a detecção de **objetos de interesse**.

ASSIM COMO NÓS, A REDE NÃO PODE DETECTAR E TÃO POUCO CLASSIFICAR ALGO QUE NÃO CONHECE.

Para isso, claro, ela precisa antes aprender **o que são esses objetos.**

Antes de tudo, então, o primeiro ponto a ser confrontado é:

O QUE A APLICAÇÃO A SER DESENVOLVIDA DEVERÁ DETECTAR?

DADOS

Tendo em mente o que deverá ser detectado, então, **o primeiro passo é reunir as imagens.**

Em linhas gerais, o conteúdo para a resolução do problema tende a ser fácil de ser concebido. É importante, ainda assim, levar em consideração alguns aspectos ao longo da busca pelos dados.

1) REPRESENTABILIDADE: as imagens do treinamento devem ser o mais representativas possível às imagens consumidas durante a ativação da rede;

- **1.1) AMPLITUDE:** quanto mais imagens, melhor;
- **1.2) QUALIDADE:** qualidade é mais importante que quantidade;
- **1.3) EQUILÍBRIO:** classes incomuns são detectadas com mais dificuldade.

DADOS

2) EXATIDÃO: detalhes irrelevantes nas imagens consumidas (conteúdo dentro das bounding boxes) durante o treinamento podem deteriorar a qualidade final da rede;

3) RIGOR: o operador responsável tanto pela busca das imagens quanto pela rotulagem é sujeito à falhas, então ele deve ser cauteloso e filtrar tais falhas para que não acometam o conteúdo consumido durante o treinamento.

- **3.1) DISTRIBUIÇÃO:** distribuir tais tarefas podem reduzir o tempo despendido para suas realizações, mas é importante que todos os envolvidos sejam rigorosos e alinhados em seus métodos (vide QUALIDADE e EXATIDÃO).

DADOS

PORTANTO, PERGUNTE-SE CONSTANTEMENTE:

1º: Posso obter as imagens pela Internet (em um dataset) ou imagens capturadas *ad hoc* são indispensáveis? e

2º: Uma vez tendo as imagens, elas são

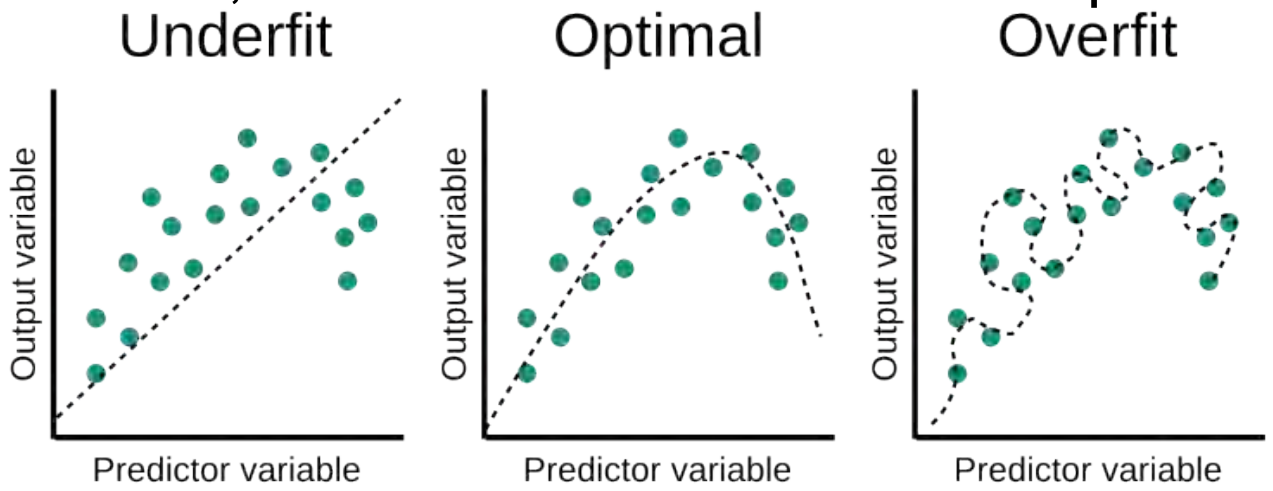
- em **volume suficiente** para permitir a rede aprender a identificar as características que melhor definem os objetos?
- **diversas o suficiente** para a rede vir a ser generalista (ao menos para as aplicações onde será ativada)? e
- com **qualidade o suficiente** para não deteriorar a rede neural durante o treinamento?

DADOS

ADEQUAÇÃO

A importância de seguir tais aspectos é, principalmente, evitar problemas de underfitting na rede convolucional.

UNDERFITTING: ocorre quando os dados não são representativos o suficiente, fazendo com que a rede neural não aprenda as características dos objetos de interesse de fato, e assim tornando a rede incapaz de detectá-los.

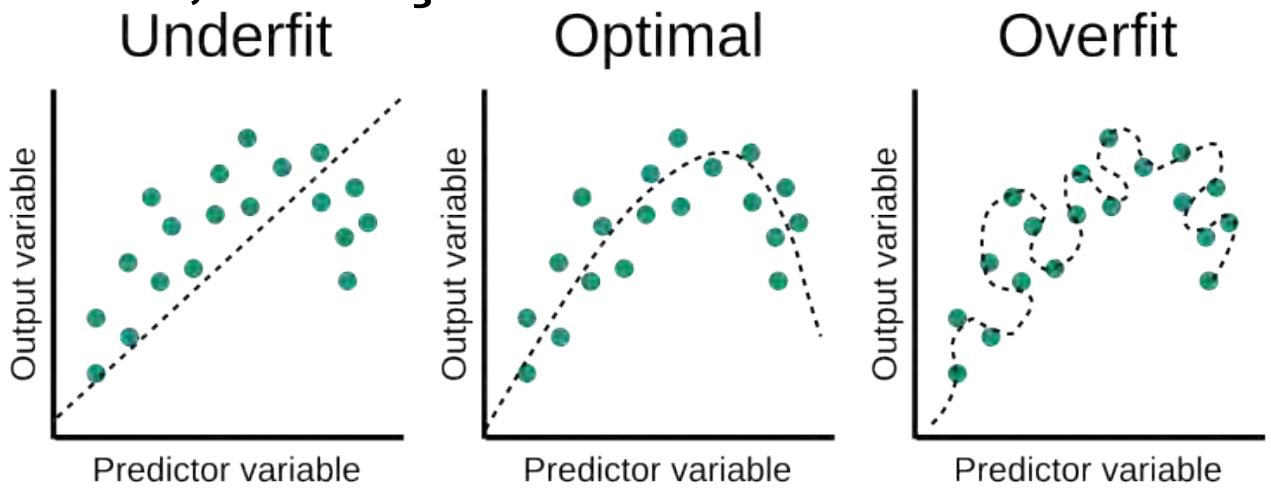


DADOS

ADEQUAÇÃO

OVERFITTING: ocorre quando os dados de treinamento são representativos demais aos dados de validação apenas, tornando-a não generalista e atuando mal nos cenários de ativação apesar de bons indicadores no treinamento.

O overfitting pode ser evitado com boas estratégias de segmentação dos dados entre treinamento, validação e teste.

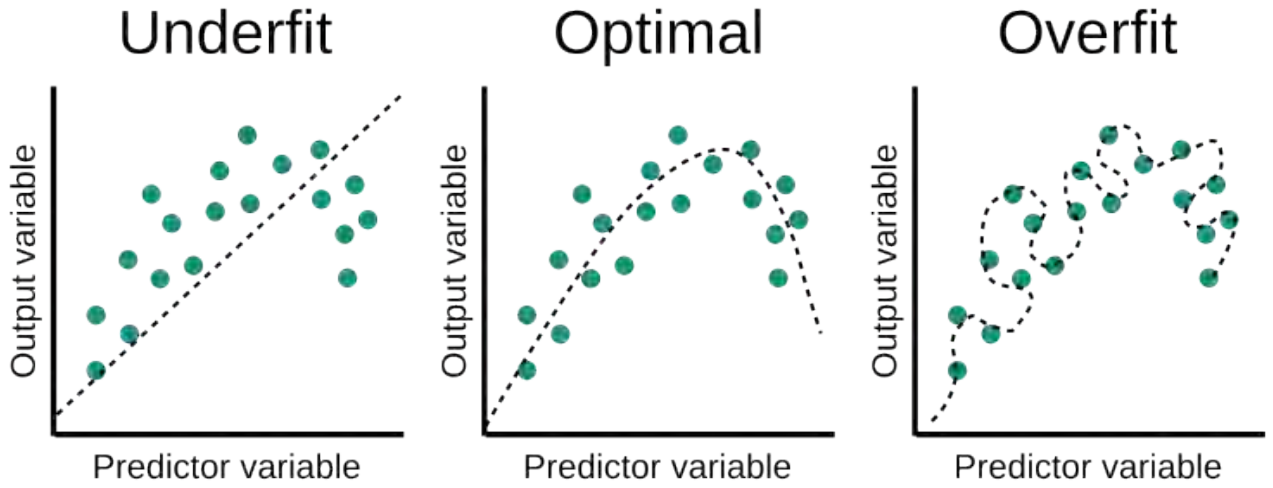


DADOS

ADEQUAÇÃO

EM RESUMO:

O treinamento é como estudar para uma prova. O underfitting ocorre quando a rede vai mal na prova porque não aprendeu, enquanto o overfitting ocorre quando a rede vai bem na prova porque decorou as respostas, sem aprender o conteúdo de fato.



DADOS

TENDO FEITO ISSO:

Você deve ter **uma pasta cheia de imagens**, provavelmente em formato JPG ou PNG.

Com isso, temos encerrada a **Fase 1 - Obtenção dos dados**, tal como disponibilizado no **toy problem**.

ROTULAGEM DOS DADOS

ROTULAGEM DOS DADOS

O QUE É

O processo de rotulagem consiste na literal **classificação dos dados de treinamento e validação**, que servem de exemplos para a rede durante o treinamento: na metáfora da prova, os rótulos são os professores.

Há diversas formas de rotular as imagens para o processo de treinamento, e o mais preciso é a **marcação**, onde os dados referente aos objetos de interesse dentro das imagens são definidos. É possível fazer isso:

1) manualmente -- **não recomendado mas completamente praticável**;
2) de forma automatizada (*data gathering*) -- **mais rápido**; e
3) com auxílio de uma ferramenta -- **permite maior flexibilidade, precisão e, portanto, qualidade**.

ROTULAGEM DOS DADOS

POR ONDE COMEÇAR

Existem algumas ferramentas disponíveis para a marcação, sendo algumas de código aberto e outras proprietárias. Ocorre também a prática de desenvolvimento de ferramentas *ad hoc* para marcação.

Dentre as ferramentas disponíveis, uma que se destaca é o **LabelImg** que pode ser baixada e utilizada gratuitamente.

Vamos abordar o LabelImg.

ROTULAGEM DOS DADOS

LABELIMG

Framework **open-source** escrito em **Python** com recursos Qt que recebe como entrada imagens (ou diretórios contendo imagens) e arquivos contendo as classes atribuídas (opcional), e tem como saída arquivos em formato TXT para cada imagem contendo as coordenadas de cada objeto marcado e a classe a eles atribuídos.

Seu uso é **muito simples e** intuitivo, recomendado para o treinamento de redes convolucionais especializadas usando conjunto de dados particulares.

Disponível para Linux, Windows e Mac OS.

ROTULAGEM DOS DADOS

LABELIMG

No Windows e Mac OS é necessário baixar o repositório.
No Linux, é possível também instalá-lo pelo PIP.

LabelImg

https://github.com/tzutalin/labelImg

LINUX

```
pip3 install labelImg
```

OU (necessário baixar o repositório)

```
sudo apt-get install pyqt5-dev-tools
sudo pip3 install -r requirements/requirements-linux-python3.txt
make qt5py3
```

MAC OS

```
brew install qt
brew install libxml2
```

OU

```
pip3 install pyqt5 lxml
make qt5py3
```

WINDOWS

```
pyrcc4 -o libs/resources.py resources.qrc
For pyqt5, pyrcc5 -o libs/resources.py resources.qrc
```

ROTULAGEM DOS DADOS

LABELIMG

SE INSTALADO PELO REPOSITÓRIO

(dentro da pasta do LabelImg):

```
python labelImg.py [IMAGE_PATH] [CLASS_FILE]
```

SE INSTALADO PELO PIP

```
labelImg [IMAGE_PATH] [CLASS_FILE]
```

FLAGS:

`[IMAGE_PATH]` → Diretório onde ficam localizadas as imagens a serem marcadas.
`[CLASS_FILE]` → Arquivo que contém as classes.

Na ausência dessas flags, o [IMAGE_PATH] é acessado pelo usuário na interface gráfica e o [CLASS_FILE] é gerado a partir de um template já formado. **Recomenda-se usar essas flags, para evitar conflitos com um arquivo de classes inadequado.**

Nota: se o computador possui ambos o Python2 e Python3 instalado, o comando deve chamar `python3`.

ROTULAGEM DOS DADOS

LABELIMG

Antes de abrir o LabelImg:

- Crie um arquivo chamado **classes.txt**;
- insira os nomes das classes a serem identificadas, **uma classe por linha**; e
- salve o arquivo.

classes.txt

```
1  alligator
2  bear
3  goat
4  lion
5  monkey
6  puma
7  warthog
8  wolf
9
```

Uma vez tendo o arquivo de classes, abra o LabelImg com as flags 1) para o caminho das imagens e 2) o classes.txt, exatamente nessa ordem:

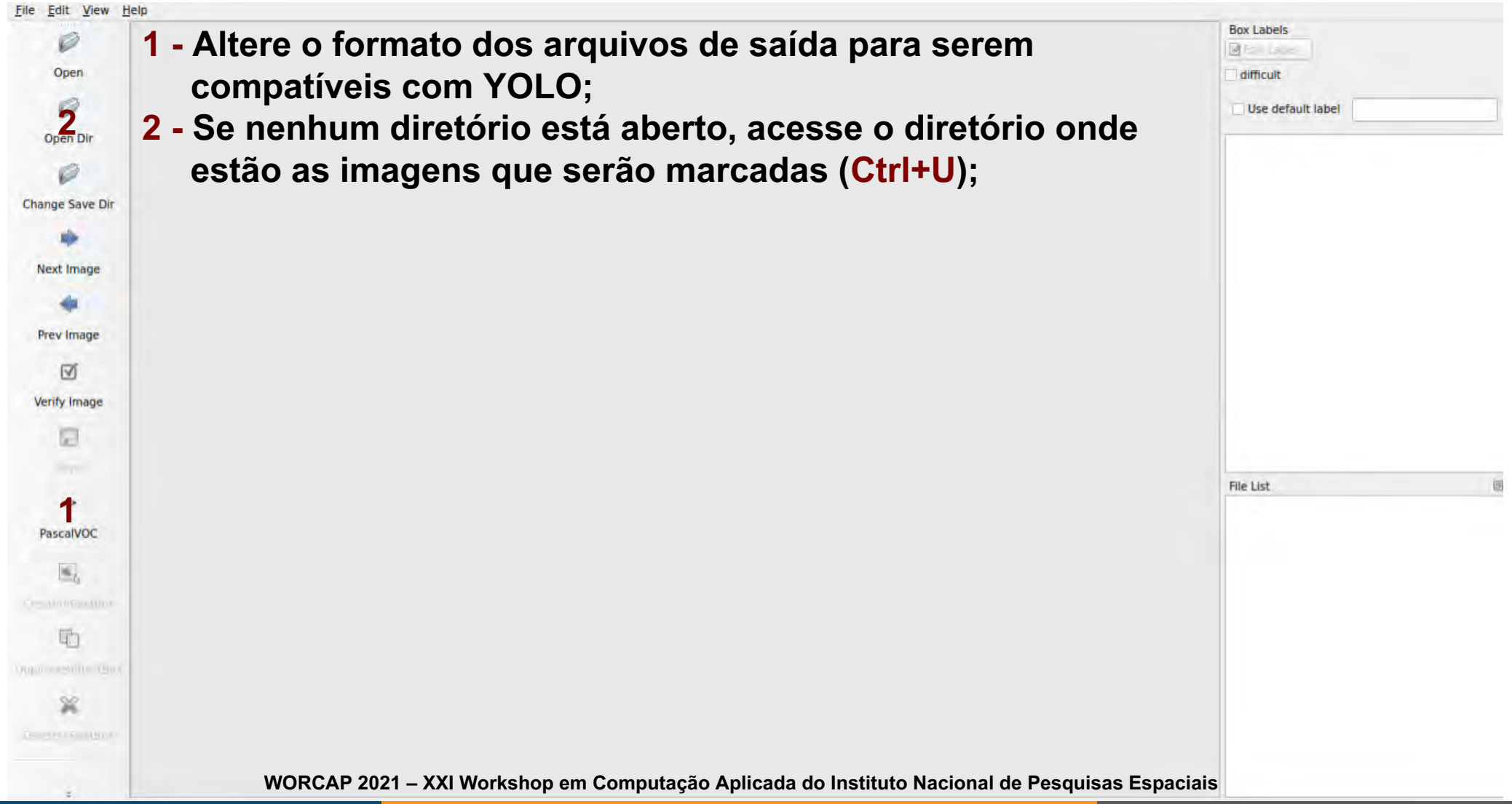
```
labelImg [IMAGE_PATH] classes.txt
```

ROTULAGEM DOS DADOS

LABELIMG

ROTULAGEM DOS DADOS

LABELIMG



1 - Altere o formato dos arquivos de saída para serem compatíveis com YOLO;

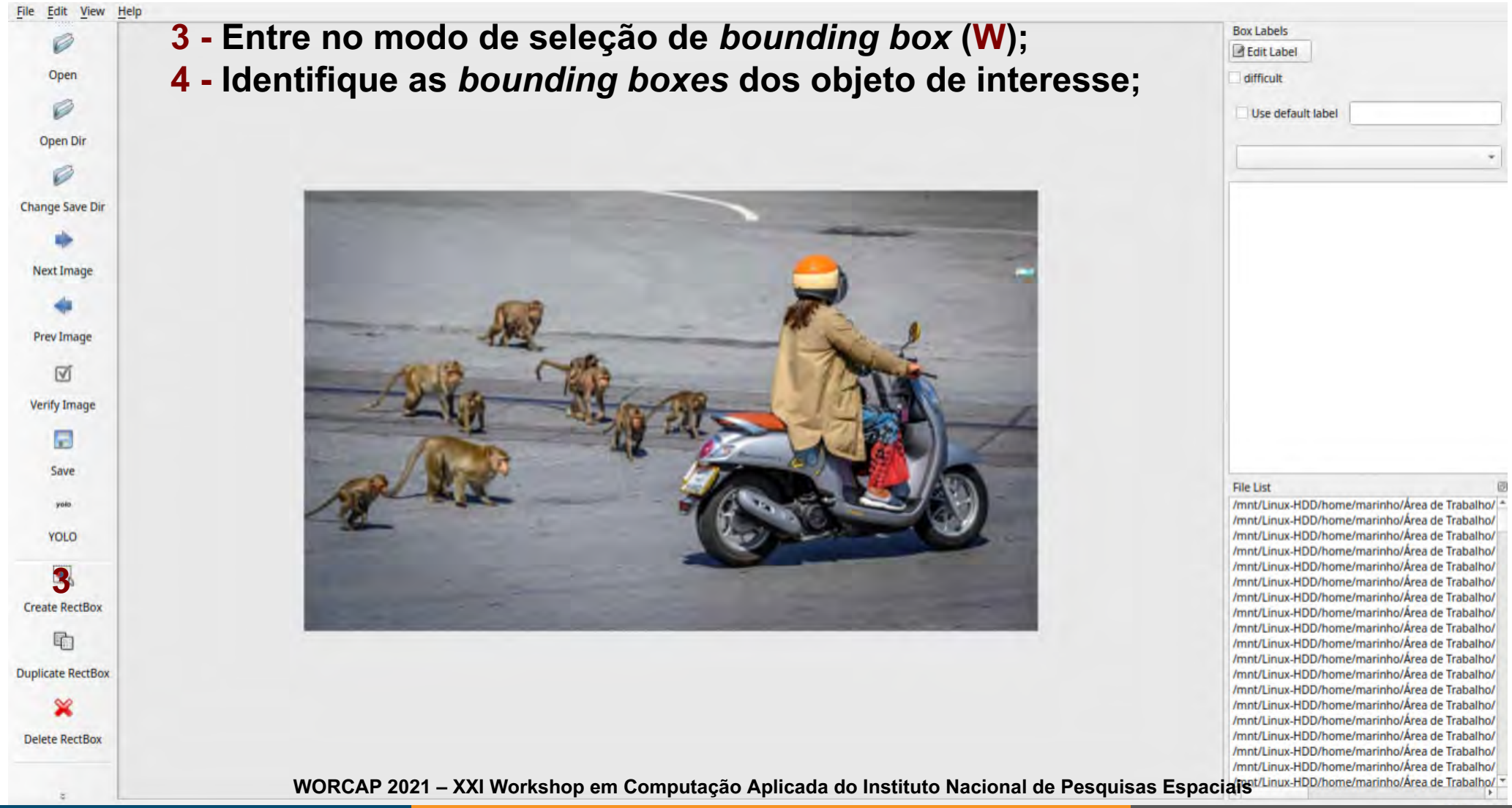
2 - Se nenhum diretório está aberto, acesse o diretório onde estão as imagens que serão marcadas (Ctrl+U);

ROTULAGEM DOS DADOS

LABELIMG

3 - Entre no modo de seleção de *bounding box* (W);

4 - Identifique as *bounding boxes* dos objeto de interesse;



ROTULAGEM DOS DADOS

LABELIMG

5 - Rotule o objeto marcado;

ROTULAGEM DOS DADOS

LABELIMG

6 - Repita os passos 3, 4 e 5 até que todos os objetos de interesse da imagem estejam marcados;

ROTULAGEM DOS DADOS

LABELIMG

7 - Salve as marcações realizadas (Ctrl+S);

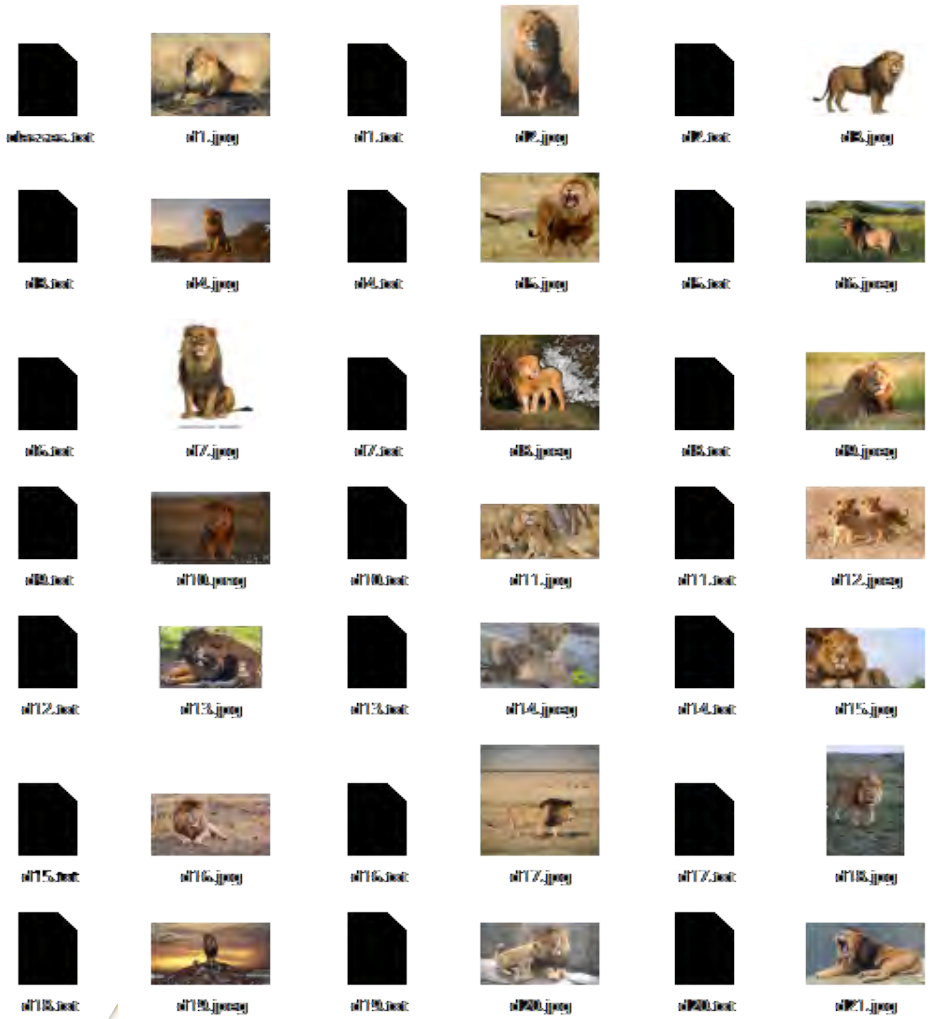
8 - Prossiga para a próxima imagem;

ROTULAGEM DOS DADOS

LABELIMG

ROTULAGEM DOS DADOS

COMO FUNCIONA



classes.txt

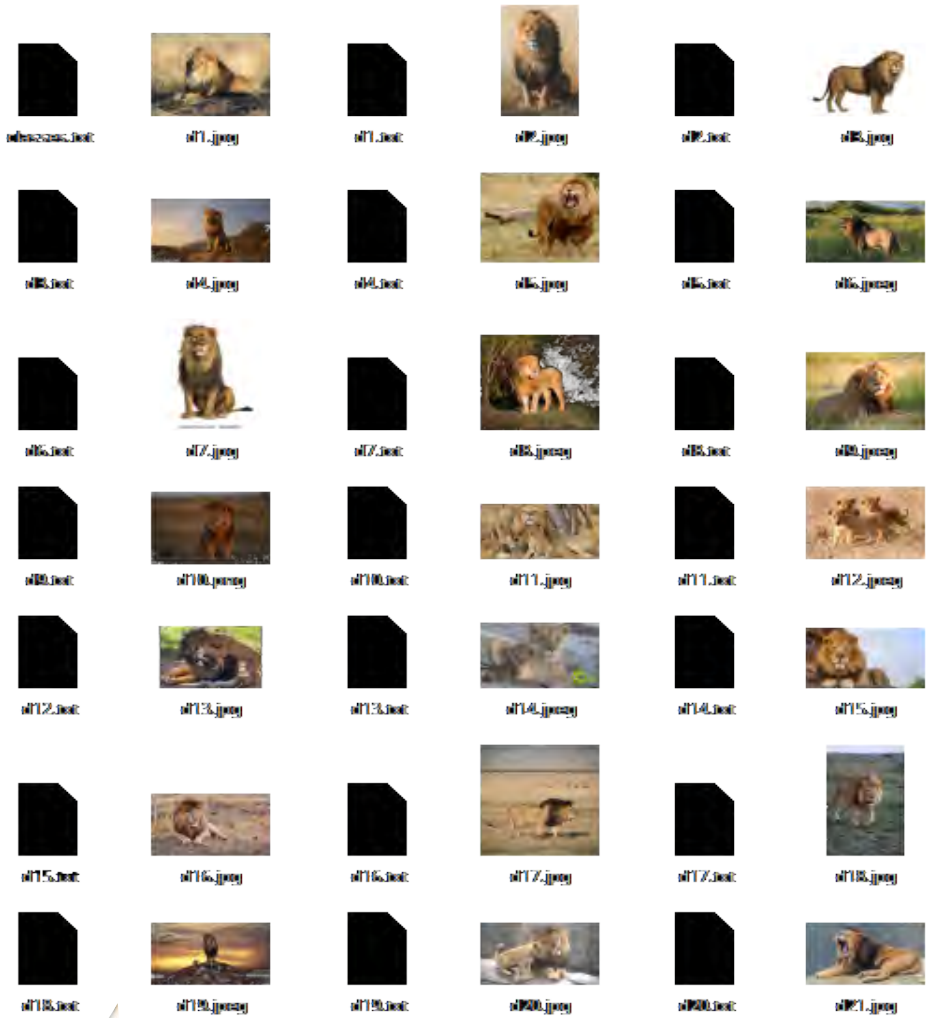
1	alligator
2	bear
3	goat
4	lion
5	monkey
6	puma
7	warthog
8	wolf
9	

100.txt

1	5	0.260671	0.707143	0.085366	0.225714
2	0	0.443979	0.459286	0.040396	0.104286
3	2	0.854421	0.612857	0.128049	0.202857
4	3	0.573933	0.289286	0.030488	0.052857
5	2	0.586128	0.162143	0.035061	0.070000
6	0	0.617759	0.167143	0.023628	0.037143
7	6	0.621570	0.124286	0.008384	0.028571
8	0	0.583079	0.089286	0.016768	0.027143
9	2	0.559070	0.063571	0.026677	0.075714
10	4	0.579268	0.057143	0.018293	0.045714
11	3	0.557165	0.017143	0.012195	0.020000
12	0	0.549543	0.005000	0.009146	0.007143
13	2	0.420351	0.234286	0.038872	0.085714
14	0	0.443216	0.130714	0.017530	0.032857
15	3	0.475991	0.082857	0.016006	0.037143
16	2	0.457317	0.044286	0.016768	0.031429
17	0	0.456555	0.027857	0.013720	0.015714
18	0	0.476753	0.035714	0.011433	0.022857
19	0	0.443979	0.016429	0.012957	0.018571
20	5	0.459985	0.012857	0.012957	0.022857

ROTULAGEM DOS DAODS

COMO FUNCIONA



Após as marcações, o diretório das imagens contém:

1) as imagens marcadas, que são arquivos de entrada;

2) um arquivo TXT homônimo para cada imagem, resultante do processo de marcação, sendo, portanto, arquivos de saída; e

3) o classes.txt, que contém os rótulos definidos preferencialmente antes do processo de marcação, e que pode ser modificado durante a marcação.

ROTULAGEM DOS DADOS

COMO FUNCIONA

Cada um dos arquivos TXT (não classes.txt) tem a seguinte organização:

- Cada linha é referente à uma instância (objeto marcado) da imagem;
- a primeira coluna é composta por ordenais referentes às classes no arquivo classes.txt; e
- as duas colunas seguintes são as coordenadas do ponto mínimo das bounding boxes que delimitam as instâncias.
- as duas últimas colunas são as dimensões das bounding boxes que delimitam as instâncias.

classes.txt

```
1  alligator
2  bear
3  goat
4  lion
5  monkey
6  puma
7  warthog
8  wolf
9
```

100.txt

```
 1  5 0.260671 0.707143 0.085366 0.225714
 2  0 0.443979 0.459286 0.040396 0.104286
 3  2 0.854421 0.612857 0.128049 0.202857
 4  3 0.573933 0.289286 0.030488 0.052857
 5  2 0.586128 0.162143 0.035061 0.070000
 6  0 0.617759 0.167143 0.023628 0.037143
 7  6 0.621570 0.124286 0.008384 0.028571
 8  0 0.583079 0.089286 0.016768 0.027143
 9  2 0.559070 0.063571 0.026677 0.075714
10  4 0.579268 0.057143 0.018293 0.045714
11  3 0.557165 0.017143 0.012195 0.020000
12  0 0.549543 0.005000 0.009146 0.007143
13  2 0.420351 0.234286 0.038872 0.085714
14  0 0.443216 0.130714 0.017530 0.032857
15  3 0.475991 0.082857 0.016006 0.037143
16  2 0.457317 0.044286 0.016768 0.031429
17  0 0.456555 0.027857 0.013720 0.015714
18  0 0.476753 0.035714 0.011433 0.022857
19  0 0.443979 0.016429 0.012957 0.018571
20  5 0.459985 0.012857 0.012957 0.022857
```

(As coordenadas e dimensões são relativas, em escala 0 ⊢ 1)

ROTULAGEM DOS DADOS

COMO FUNCIONA

Cada um dos arquivos TXT (não classes.txt) tem a seguinte organização:

- Cada linha é referente à uma instância (objeto marcado) da imagem;
- a primeira coluna é composta por ordenais referentes às classes no arquivo classes.txt; e
- as duas colunas seguintes são as coordenadas do ponto mínimo das bounding boxes que delimitam as instâncias.
- as duas últimas colunas são as dimensões das bounding boxes que delimitam as instâncias.

classes.txt

```
1  alligator 0
2  bear      1
3  goat      2
4  lion      3
5  monkey    4
6  puma      5
7  warthog   6
8  wolf      7
9
```

(As coordenadas e dimensões são relativas, em escala 0 ⊢ 1)

100.txt

```
1   5 0.260671 0.707143 0.085366 0.225714
2   0 0.443979 0.459286 0.040396 0.104286
3   2 0.854421 0.612857 0.128049 0.202857
4   3 0.573933 0.289286 0.030488 0.052857
5   2 0.586128 0.162143 0.035061 0.070000
6   0 0.617759 0.167143 0.023628 0.037143
7   6 0.621570 0.124286 0.008384 0.028571
8   0 0.583079 0.089286 0.016768 0.027143
9   2 0.559070 0.063571 0.026677 0.075714
10  4 0.579268 0.057143 0.018293 0.045714
11  3 0.557165 0.017143 0.012195 0.020000
12  0 0.549543 0.005000 0.009146 0.007143
13  2 0.420351 0.234286 0.038872 0.085714
14  0 0.443216 0.130714 0.017530 0.032857
15  3 0.475991 0.082857 0.016006 0.037143
16  2 0.457317 0.044286 0.016768 0.031429
17  0 0.456555 0.027857 0.013720 0.015714
18  0 0.476753 0.035714 0.011433 0.022857
19  0 0.443979 0.016429 0.012957 0.018571
20  5 0.459985 0.012857 0.012957 0.022857
```

Classe

ROTULAGEM DOS DADOS

COMO FUNCIONA

Cada um dos arquivos TXT (não classes.txt) tem a seguinte organização:

- Cada linha é referente à uma instância (objeto marcado) da imagem;
- a primeira coluna é composta por ordenais referentes às classes no arquivo classes.txt; e
- as duas colunas seguintes são as coordenadas do ponto mínimo das bounding boxes que delimitam as instâncias.
- as duas últimas colunas são as dimensões das bounding boxes que delimitam as instâncias.

classes.txt

```
1  alligator 0
2  bear      1
3  goat      2
4  lion      3
5  monkey    4
6  puma      5
7  warthog   6
8  wolf      7
9
```

(As coordenadas e dimensões são relativas, em escala 0 ⊢ 1)

100.txt

```
1   5 0.260671 0.707143 0.085366 0.225714
2   0 0.443979 0.459286 0.040396 0.104286
3   2 0.854421 0.612857 0.128049 0.202857
4   3 0.573933 0.289286 0.030488 0.052857
5   2 0.586128 0.162143 0.035061 0.070000
6   0 0.617759 0.167143 0.023628 0.037143
7   6 0.621570 0.124286 0.008384 0.028571
8   0 0.583079 0.089286 0.016768 0.027143
9   2 0.559070 0.063571 0.026677 0.075714
10  4 0.579268 0.057143 0.018293 0.045714
11  3 0.557165 0.017143 0.012195 0.020000
12  0 0.549543 0.005000 0.009146 0.007143
13  2 0.420351 0.234286 0.038872 0.085714
14  0 0.443216 0.130714 0.017530 0.032857
15  3 0.475991 0.082857 0.016006 0.037143
16  2 0.457317 0.044286 0.016768 0.031429
17  0 0.456555 0.027857 0.013720 0.015714
18  0 0.476753 0.035714 0.011433 0.022857
19  0 0.443979 0.016429 0.012957 0.018571
20  5 0.459985 0.012857 0.012957 0.022857
```

Classe Coordenadas

ROTULAGEM DOS DADOS

COMO FUNCIONA

Cada um dos arquivos TXT (não classes.txt) tem a seguinte organização:

- Cada linha é referente à uma instância (objeto marcado) da imagem;
- a primeira coluna é composta por ordenais referentes às classes no arquivo classes.txt; e
- as duas colunas seguintes são as coordenadas do ponto mínimo das bounding boxes que delimitam as instâncias.
- as duas últimas colunas são as dimensões das bounding boxes que delimitam as instâncias.

classes.txt

```
1  alligator 0
2  bear      1
3  goat      2
4  lion      3
5  monkey    4
6  puma      5
7  warthog   6
8  wolf      7
9
```

(As coordenadas e dimensões são relativas, em escala 0 ⊢ 1)

100.txt

```
1   5 0.260671 0.707143 0.085366 0.225714
2   0 0.443979 0.459286 0.040396 0.104286
3   2 0.854421 0.612857 0.128049 0.202857
4   3 0.573933 0.289286 0.030488 0.052857
5   2 0.586128 0.162143 0.035061 0.070000
6   0 0.617759 0.167143 0.023628 0.037143
7   6 0.621570 0.124286 0.008384 0.028571
8   0 0.583079 0.089286 0.016768 0.027143
9   2 0.559070 0.063571 0.026677 0.075714
10  4 0.579268 0.057143 0.018293 0.045714
11  3 0.557165 0.017143 0.012195 0.020000
12  0 0.549543 0.005000 0.009146 0.007143
13  2 0.420351 0.234286 0.038872 0.085714
14  0 0.443216 0.130714 0.017530 0.032857
15  3 0.475991 0.082857 0.016006 0.037143
16  2 0.457317 0.044286 0.016768 0.031429
17  0 0.456555 0.027857 0.013720 0.015714
18  0 0.476753 0.035714 0.011433 0.022857
19  0 0.443979 0.016429 0.012957 0.018571
20  5 0.459985 0.012857 0.012957 0.022857
```

Classe Coordenadas Dimensões

ROTULAGEM DOS DADOS

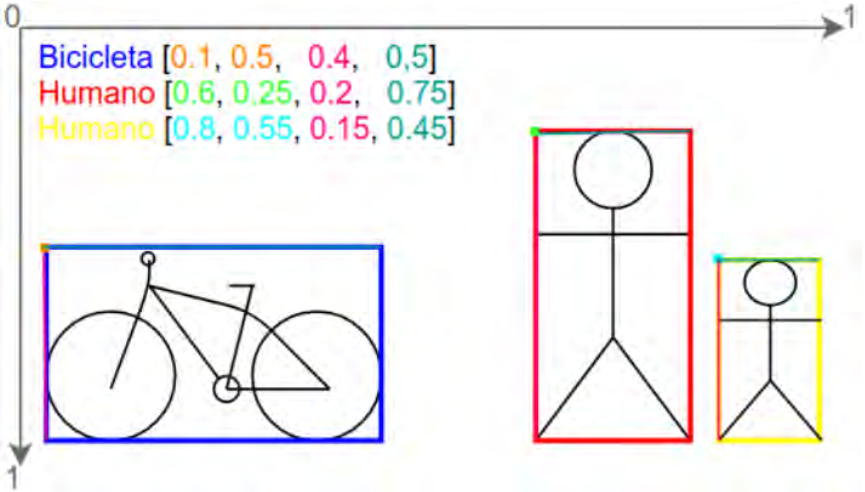
COMO FUNCIONA

Cada um dos arquivos TXT (não classes.txt) tem a seguinte organização:

- Cada linha é referente à uma instância (objeto marcado) da imagem;
- a primeira coluna é composta por ordenais referentes às classes no arquivo classes.txt; e
- as duas colunas seguintes são as coordenadas do ponto mínimo das bounding boxes que delimitam as instâncias.
- as duas últimas colunas são as dimensões das bounding boxes que delimitam as instâncias.

classes.txt

```
1  alligator 0
2  bear      1
3  goat      2
4  lion      3
5  monkey    4
6  puma      5
7  warthog   6
8  wolf      7
9
```




```
10  4 0.579268 0.057143 0.018293 0.045714
11  3 0.557165 0.017143 0.012195 0.020000
12  0 0.549543 0.005000 0.009146 0.007143
13  2 0.420351 0.234286 0.038872 0.085714
14  0 0.443216 0.130714 0.017530 0.032857
15  3 0.475991 0.082857 0.016006 0.037143
16  2 0.457317 0.044286 0.016768 0.031429
17  0 0.456555 0.027857 0.013720 0.015714
18  0 0.476753 0.035714 0.011433 0.022857
19  0 0.443979 0.016429 0.012957 0.018571
20  5 0.459985 0.012857 0.012957 0.022857
```

(As coordenadas e dimensões são relativas, em escala 0 ⊢ 1)

Classe Coordenadas Dimensões

ROTULAGEM DOS DADOS

TOY PROBLEM

TENDO FEITO ISSO:

Você deve ter **uma pasta cheia de imagens**, provavelmente em formato JPG ou PNG, e **um arquivo com instâncias de objetos para cada uma** dessas imagens, além do **arquivo de rótulos** de fato (classes.txt).
Com isso, temos encerrada a **Fase 2 - Rotulagem dos dados**, tal como disponibilizado no **toy problem**.

AMBIENTE DE TREINAMENTO

AMBIENTE DE TREINAMENTO

CONTEXTUALIZAÇÃO

Com todas as imagens para o treinamento da rede convolucional reunidas e devidamente marcadas e rotuladas, **é necessário fazer a Darknet entender o que deve fazer.**

Para que o treinamento possa ser realizado, os **subconjuntos de treinamento** e **validação** devem ser segmentados, a **arquitetura da rede** YOLO deve ser formada e os **pesos iniciais** devem ser obtidos.

Trata-se do processo de **formação do ambiente de treinamento**.

AMBIENTE DE TREINAMENTO

ORGANIZAÇÃO

models

PESOS DA REDE CONVOLUCIONAL

data

train

SUBCONJUNTO DE TREINAMENTO

validation

SUBCONJUNTO DE VALIDAÇÃO

DIRETIVAS DO TREINAMENTO

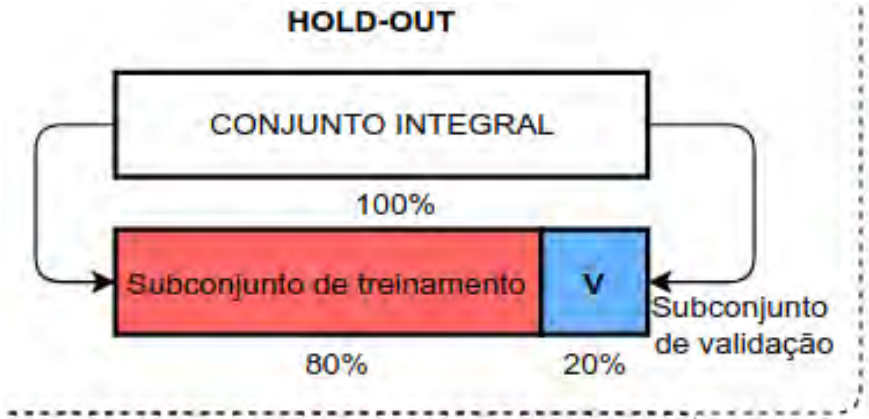
ARQUITETURA DA REDE E PARAFERNÁLIA

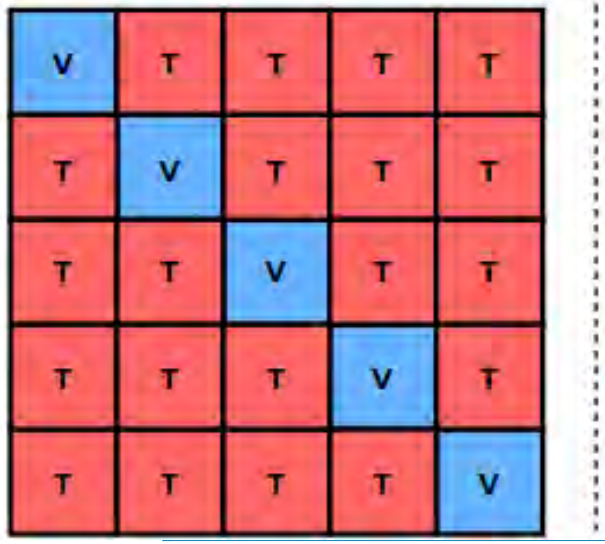
AMBIENTE DE TREINAMENTO

SEGMENTAÇÃO DO CONJUNTO DE TREINAMENTO

AMBIENTE DE TREINAMENTO

SEGMENTAÇÃO DO CONJUNTO DE TREINAMENTO





AMBIENTE DE TREINAMENTO

SEGMENTAÇÃO DO CONJUNTO DE TREINAMENTO

models

data

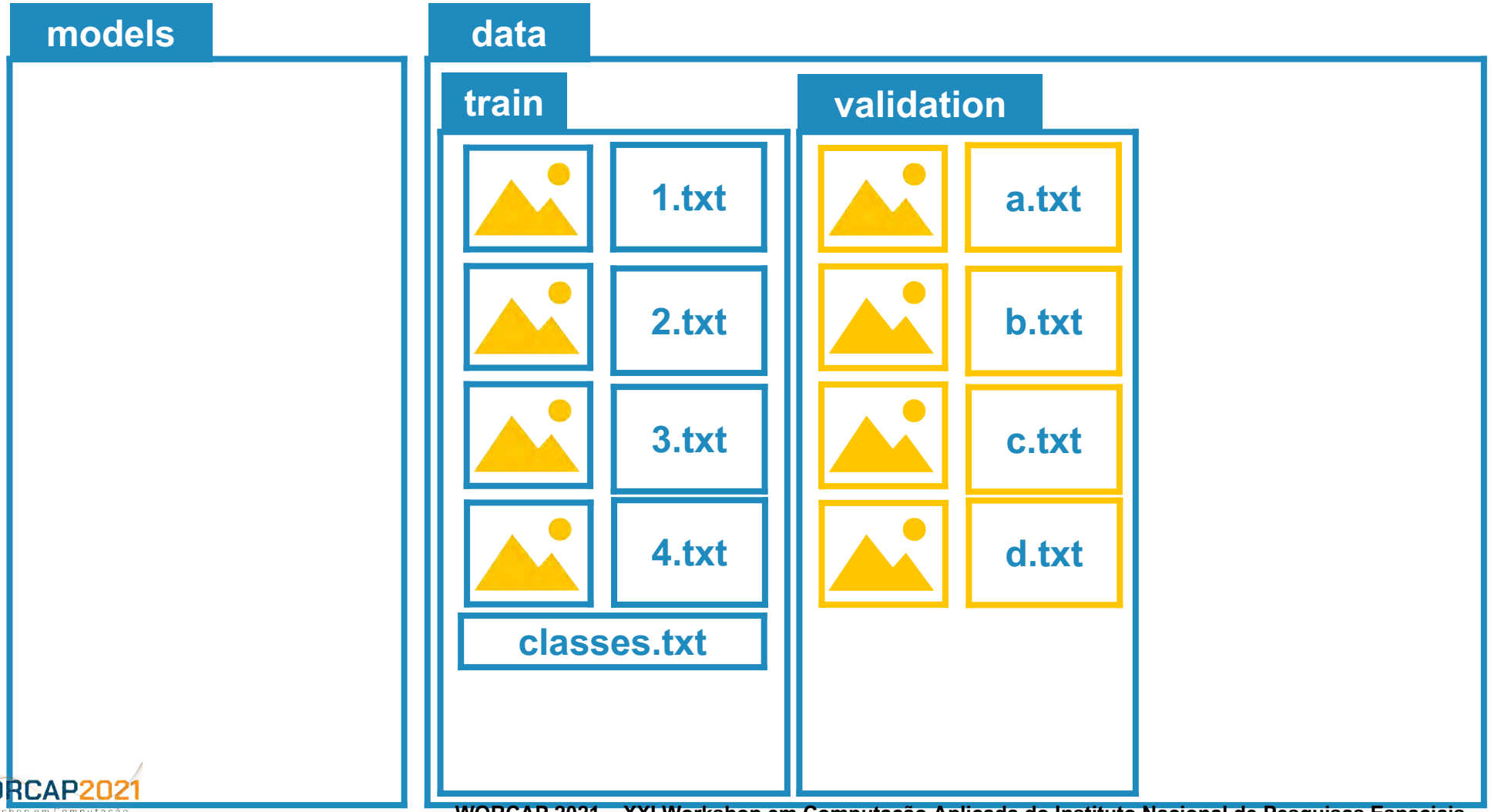
train

Na pasta **train** fica o **subconjunto de treino do processo de treinamento**.

Este diretório deve ser alimentado com boa parte das **imagens** trabalhadas durante o processo de rotulagem, com seus respectivos **arquivos TXT** contendo os **metadados das marcações**, além do **classes.txt** contendo os **rótulos**.

AMBIENTE DE TREINAMENTO

SEGMENTAÇÃO DO CONJUNTO DE TREINAMENTO

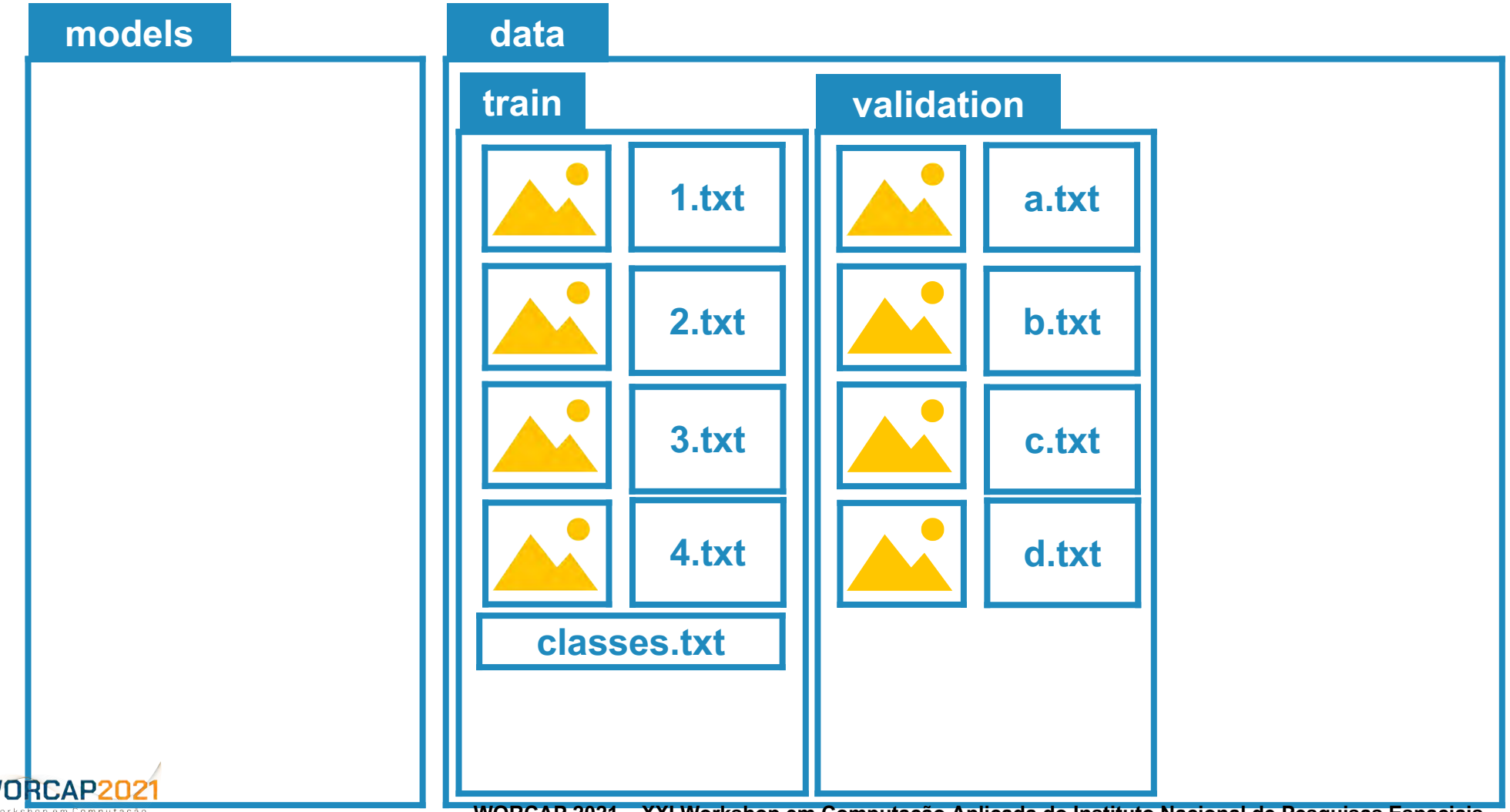


Em **validation** deve ficar o **subconjunto de validação do processo de treinamento**.

Este subconjunto deve ser análogo ao presente em **train**, contendo as **imagens** e seus respectivos **arquivos TXT contendo os metadados das instâncias dos objetos marcados**.

AMBIENTE DE TREINAMENTO

PARAMETRIZAÇÃO E DIRETIVAS

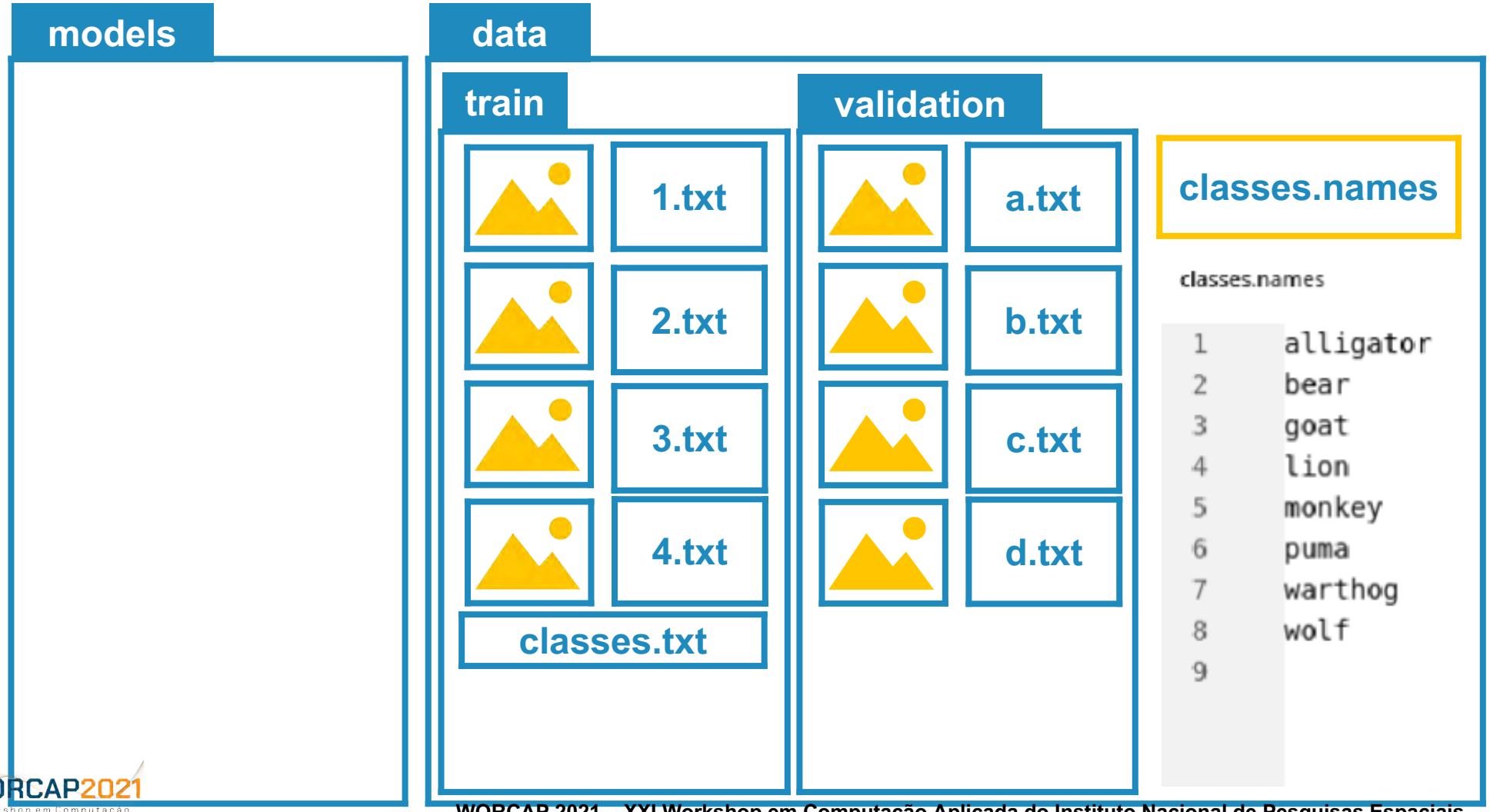


generate_stuff.py

Os demais arquivos são gerados pelo **generate_stuff.py**, script Python que reduz o restante do processo a uma fração de segundo.

AMBIENTE DE TREINAMENTO

PARAMETRIZAÇÃO E DIRETIVAS



classes.names

classes.names

```
1  alligator
2  bear
3  goat
4  lion
5  monkey
6  puma
7  warthog
8  wolf
9
```

generate_stuff.py

Um desses arquivos gerados é o **classes.names**, que nada mais é do que uma **cópia do classes.txt**, porém com qualquer “ ” (espaço) substituído por “_” (underline).

AMBIENTE DE TREINAMENTO

PARAMETRIZAÇÃO E DIRETIVAS

models

train.txt

```
1   data/train/a23.jpg
2   data/train/e6.jpg
3   data/train/b15.jpeg
4   data/train/b19.jpeg
5   data/train/c11.jpg
6   data/train/a6.jpg
7   data/train/e20.jpg
8   data/train/f3.jpg
9   data/train/e18.jpg
10  data/train/c3.jpeg
11  data/train/e15.jpeg
12  data/train/h5.jpg
13  data/train/f7.jpg
14  data/train/b16.jpeg
15  data/train/h2.jpeg
16  data/train/e11.jpg
17  data/train/f5.jpg
18  data/train/g5.jpg
19  data/train/e14.jpg
20  data/train/e17.jpg
```

data

train

classes.txt

validation

classes.names

train.txt

generate_stuff.py

É gerado ainda o **train.txt**, que é uma lista com os **caminhos** (relativos à raíz do ambiente) para cada imagem do **subconjunto de treinamento.**

As imagens são organizadas de forma **embaralhada**.

AMBIENTE DE TREINAMENTO

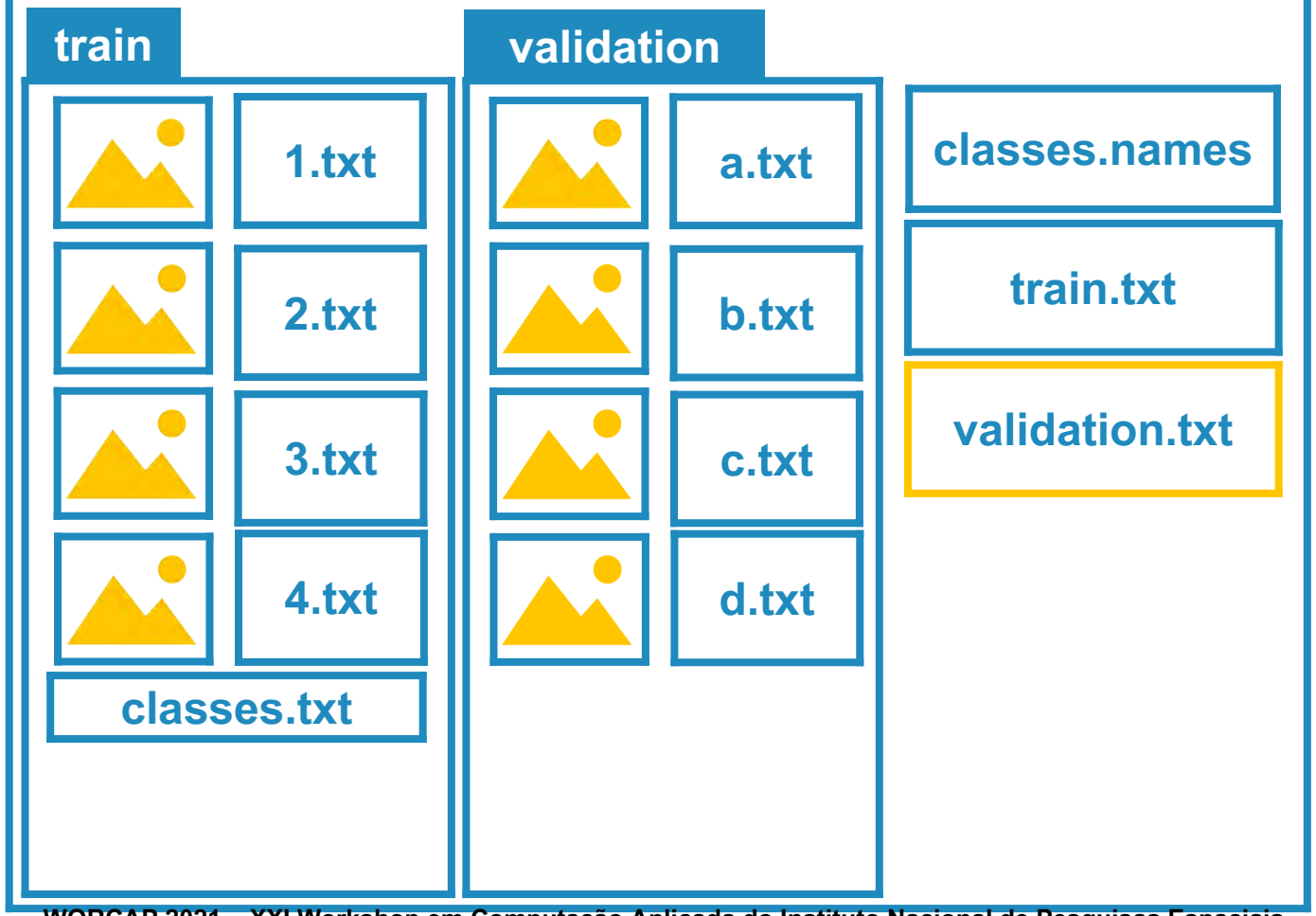
PARAMETRIZAÇÃO E DIRETIVAS

models

validation.txt

```
1   data/validation/g30.jpeg
2   data/validation/f30.jpeg
3   data/validation/b25.jpg
4   data/validation/h30.jpg
5   data/validation/b26.jpeg
6   data/validation/b33.jpg
7   data/validation/c30.jpg
8   data/validation/b34.jpg
9   data/validation/g26.jpg
10  data/validation/g25.jpeg
11  data/validation/d28.jpg
12  data/validation/e29.jpeg
13  data/validation/c27.jpg
14  data/validation/d27.jpeg
15  data/validation/h27.jpeg
16  data/validation/h26.jpg
17  data/validation/e27.jpeg
18  data/validation/a25.jpg
19  data/validation/d30.jpg
20  data/validation/d29.jpeg
```

data



Do mesmo modo, é gerado o **validation.txt**, que é uma **lista com os caminhos** (também relativos à raíz do ambiente) para cada imagem presente no **subconjunto de validação**, tudo embaralhado.

AMBIENTE DE TREINAMENTO

PARAMETRIZAÇÃO E DIRETIVAS

models

directives.data

```
1  classes = 8
2  train = data/train.txt
3  valid = data/validation.txt
4  names = data/classes.names
5  backup = models
```

data

train


1.txt


2.txt


3.txt


4.txt

classes.txt

validation


a.txt


b.txt


c.txt


d.txt

classes.names

train.txt

validation.txt

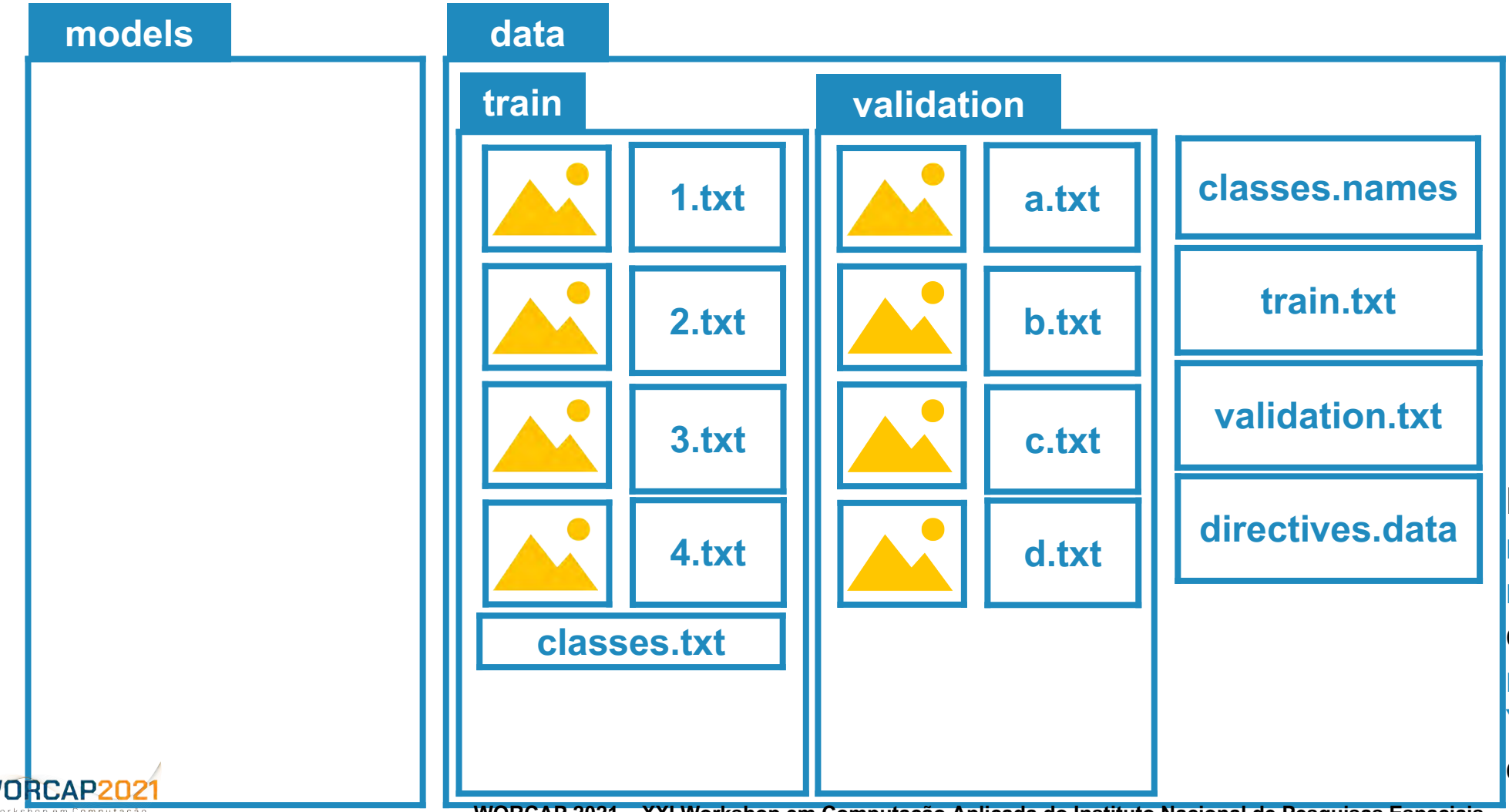
directives.data

generate_stuff.py

Gera o **directives.data**, que contém os **parâmetros e diretivas para o treinamento**:
1) o número de classes, 2) o caminho para as listas de imagens de treinamento e de validação, 3) o arquivo dos rótulos das classes, e 4) o diretório dos pesos da rede convolucional.

AMBIENTE DE TREINAMENTO

ARQUITETURA DA REDE YOLOv4



E, por fim, gera o **model_full.cfg** e **model_tiny.cfg**, que contêm a **arquitetura da rede convolucional YOLO** nos formatos **Full** e **Tiny**, respectivamente.

AMBIENTE DE TREINAMENTO

ARQUITETURA DA REDE YOLOv4

Uma arquitetura de rede convolucional profunda para visão computacional deve seguir **uma lógica complexa e pouco amigável para novatos**.

A definição dos hiperparâmetros segue as **instruções de Alexey Bochkovskiy** (desenvolvedor por trás da YOLOv4), as quais consideram:

- **classes** = *C*;
- **batch** = 64;
- **subdivisions** = 16;
- **width** = 416;
- **height** = 416;
- ***max_batches*** = (*C* * 2000), 6000 se *C* ≤ 3;
- **steps** = (*max_batches* * 0.8), (*max_batches* * 0.9); e
- **filters** = ((*C* + 5) * 3),

onde *C* é o **número de classes**.

generate_stuff.py

model_full.cfg

model_tiny.cfg

Nota: esses híperparâmetros seguem a mesma lógica tanto na rede YOLOv4 Full quanto YOLOv4 Tiny.

AMBIENTE DE TREINAMENTO

PESOS DA REDE YOLOv4

models

- model_full.weights
- model_tiny.weights

data

train


1.txt


2.txt


3.txt


4.txt

classes.txt

validation


a.txt


b.txt


c.txt


d.txt

- classes.names
- train.txt
- validation.txt
- directives.data

- generate_stuff.py
- model_full.cfg
- model_tiny.cfg

Por último, é necessário haver o arquivo de **pesos iniciais** para a rede convolucional, de onde é realizada uma **transferência de aprendizagem**.

AMBIENTE DE TREINAMENTO

TOY PROBLEM

TENDO FEITO ISSO:

Você deve ter uma pasta contendo a **rede convolucional YOLOv4** em ambos seus formatos (Full e Tiny), sendo a arquitetura, pesos e rótulos das classes discrimináveis, além do **conjunto de treinamento** dividido para validação e treino, além das **diretivas** para tal.
Com isso, temos encerrada a **Fase 3 - Ambiente de treinamento**, tal como disponibilizado no **toy problem**.

TREINAMENTO

TREINAMENTO DA REDE

Com o ambiente para o treinamento devidamente organizado e configurado, a Darknet pode trabalhar em cima dele para treinar a rede.

O treinamento de uma rede convolucional é uma tarefa de **alta complexidade computacional**, envolvendo **muitos tensores** e preferencialmente um **alto volume de dados**, então é necessária uma máquina capaz de lidar com tudo isso.

Fica implícito também que **é um processo demorado** diante do atual estado tecnológico.

TREINAMENTO DA REDE

Mesmo em uma máquina com os melhores recursos de hardware, portanto, o treinamento **pode demandar horas ou mesmo dias**.

De modo geral, é **essencial** uma máquina equipada com uma **GPGPU**.

Muitas pessoas não têm à disposição uma máquina com uma GPGPU dedicada, mas isso pode ser contornado com **IaaS**.
A IaaS mais proeminente para a tarefa é o **Google Colab**.

TREINAMENTO DA REDE

GOOGLE COLABORATORY

Formalmente batizado de Colaboratory, o Google Colab é um **ambiente de desenvolvimento colaborativo** (web) desenvolvido pela Google Research.

Apresenta um ambiente com **interface Jupyter** que **serve tanto como IDE**, ao utilizar os recursos da máquina onde está sendo acessada, **como também IaaS** ao dispor de recursos de firmware e hardware hospedados no mainframe compartilhado da Google.

TREINAMENTO DA REDE

GOOGLE COLABORATORY

VANTAGENS

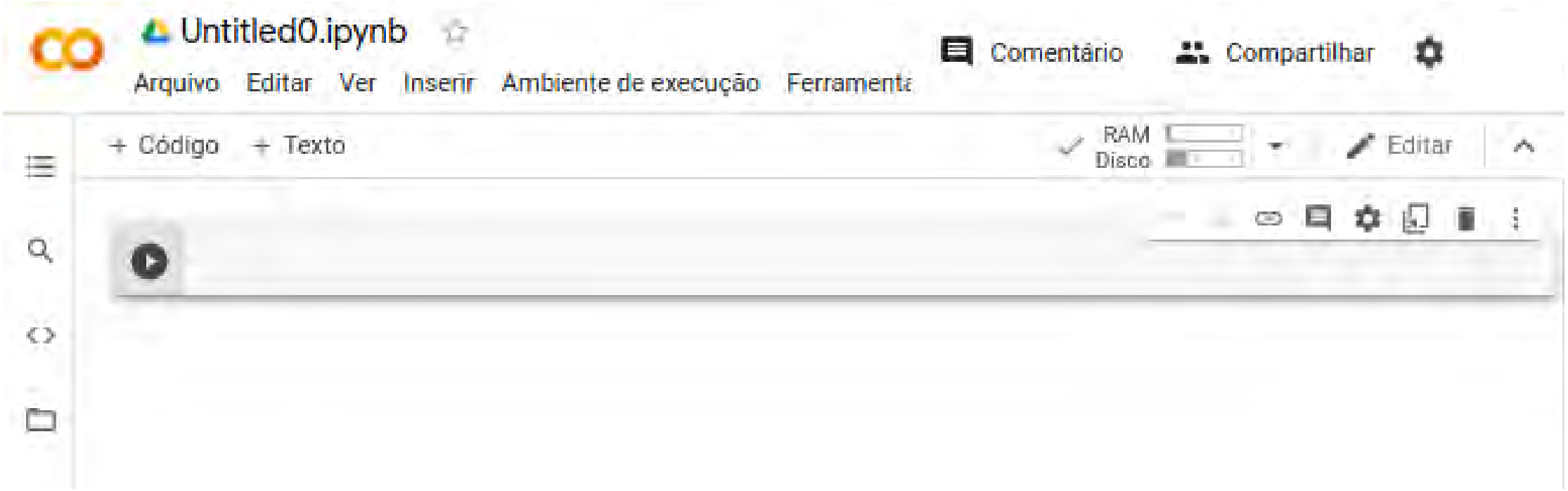
- Acessibilidade em múltiplos pontos de acesso, mesmo de forma simultanea;
- dispõe de recursos de hardware e firmware avançados, como a GPGPU NVIDIA K80 (inclusive como TPU) e NVIDIA CUDA;
- gratuíto.

DESVANTAGENS

- Os recursos de GPGPU não são ilimitados, disponíveis por até 12 horas por dia;
- é possível assinar maior prioridade no uso dos recursos (inclusive as GPGPUs mais rápidas), mas eles ainda não são garantidos como ininterrúptos;
- dispõe de RAM e alocação de memória limitados.

TREINAMENTO DA REDE

GOOGLE COLABORATORY



TREINAMENTO DA REDE

GOOGLE COLABORATORY

Há disponível um **template** para o treinamento no Google Colab; você pode acessá-lo e **salvar uma cópia no seu Google Drive**.

TREINAMENTO DA REDE

QUAL FORMATO YOLO ESCOLHER

FULL

- Arquitetura de 137 camadas, sendo
- 3 camadas “YOLO”,
- com volume de 12 KB;
- Pesos com 244,3 MB.

TINY

- Arquitetura de 29 camadas, sendo
- 2 camadas “YOLO”,
- com volume de 2,9 KB;
- Pesos com 22,5 MB.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE

models

- model_full.weights
- model_tiny.weights

data

train


1.txt


2.txt


3.txt


4.txt

classes.txt

validation


a.txt


b.txt


c.txt


d.txt

- classes.names
- train.txt
- validation.txt
- directives.data

- generate_stuff.py
- model_full.cfg
- model_tiny.cfg

Acesse o **Google Drive** e crie uma pasta na raíz com o nome de **WorCAP2021**.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE

models

- model_full.weights
- model_tiny.weights

- model_last.weights

data

train

- 1.txt

- 2.txt

- 3.txt

- 4.txt
- classes.txt

validation

- a.txt

- b.txt

- c.txt

- d.txt

- classes.names
- train.txt
- validation.txt
- directives.data
- generate_stuff.py
- model_full.cfg
- model_tiny.cfg

Escolha se você vai treinar uma rede YOLOv4 **Full** ou **Tiny**, e então renomeie o arquivo de pesos escolhido para **model_last.weights**.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE

models

model_last.weights

data

train


1.txt


2.txt


3.txt


4.txt

classes.txt

validation


a.txt


b.txt


c.txt


d.txt

classes.names

train.txt

validation.txt

directives.data

generate_stuff.py

model_full.cfg

model_tiny.cfg

model.cfg

Do mesmo modo, escolha e renomeie o arquivo de arquitetura escolhido para model.cfg.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE

models

- model_last.weights

data

train


- 1.txt


- 2.txt


- 3.txt


- 4.txt
- classes.txt

validation


- a.txt


- b.txt


- c.txt


- d.txt

- classes.names
- train.txt
- validation.txt
- directives.data

- generate_stuff.py
- model.cfg

Para o treinamento, os arquivos **generate_stuff.py** e **classes.txt** não são necessários, então eles podem ser ignorados.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE

models

- model_last.weights

data

train

- 1.txt
- 2.txt
- 3.txt
- 4.txt

validation

- a.txt
- b.txt
- c.txt
- d.txt

- classes.names
- train.txt
- validation.txt
- directives.data
- model.cfg

Faça o upload do ambiente para dentro da recém-criada pasta **WorCAP2021**.

TREINAMENTO DA REDE

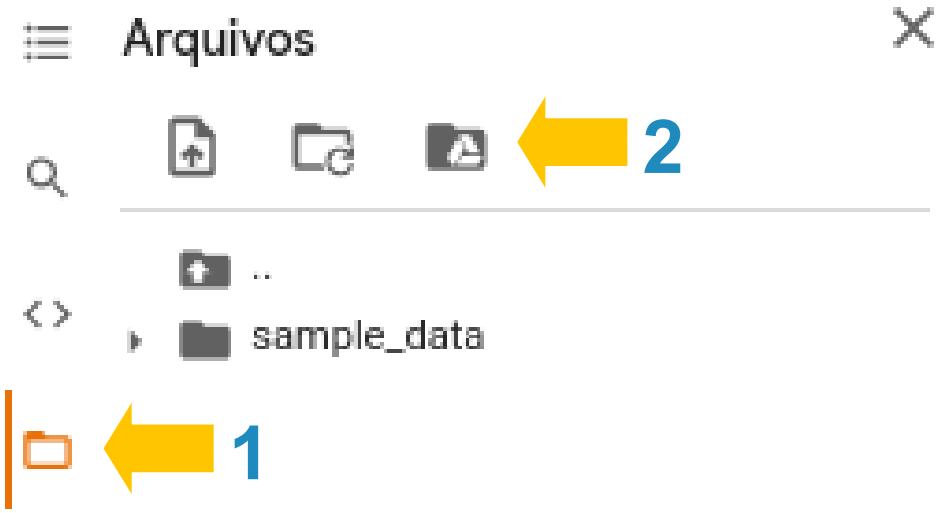
GOOGLE COLABORATORY

Com o ambiente hospedado no **Google Drive**, é necessário acessá-lo pelo Google Colab.

Tal função já é embutida no sistema do Colab.

TREINAMENTO DA REDE

AMBIENTE DE TREINAMENTO NO DRIVE



1- Acesse a aba de **Arquivos**;

2- Monte o Google **Drive**;
2.1- Talvez o Colab insira um bloco para solicitar permissão de acesso ao Drive. Se assim for, **simplesmente siga as instruções**.

3- Aguarde até que esteja disponível a pasta com nome “drive” no ambiente.

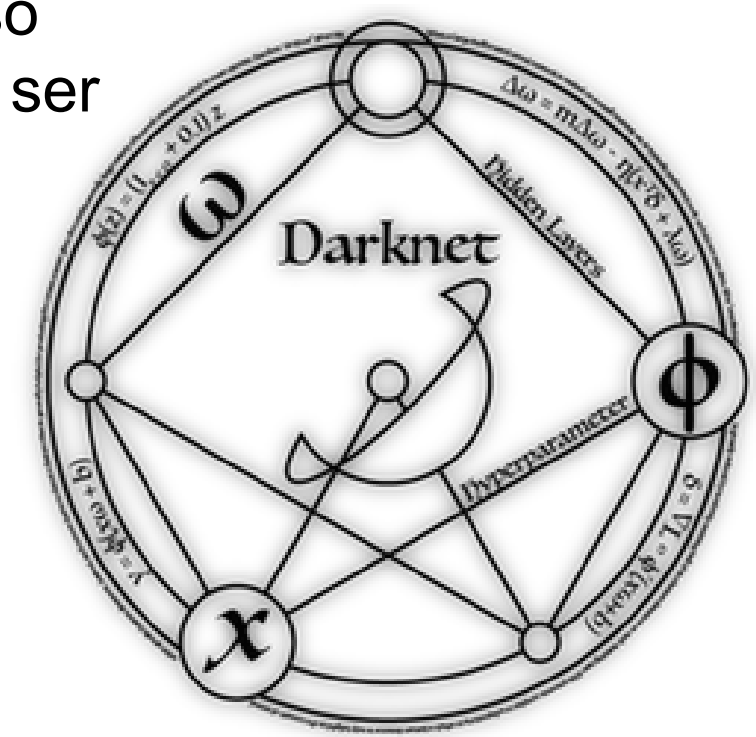
Nota: a conexão com o Drive pode ser defeita após desconexões.

TREINAMENTO DA REDE

A DARKNET

O ambiente do Colab agora conta então com a **rede convolucional YOLO** desejada (**pesos** e **arquitetura**), os **subconjuntos de treinamento e validação** e toda a parafernália envolvida no processo (**parâmetros** e **diretivas**), e tudo foi organizado para ser trabalhado pela Darknet.

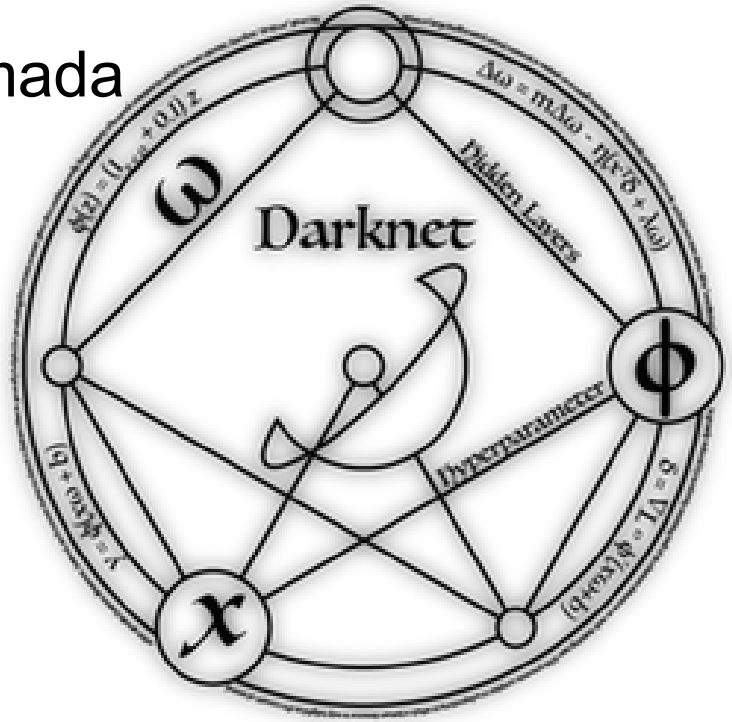
ESTÁ FALTANDO, É CLARO, A DARKNET.



TREINAMENTO DA REDE

A DARKNET

A Darknet é um **framework de redes neurais open source** escrito em C, otimizado para o uso com recursos CUDA, abrigando um ecossistema de recursos que compreendem, dentre elas, a **YOLO**.
Foi criada e abandonada por Joseph Redmon, e retomada por Alexey Bochkovskiy e demais personalidades da comunidade.



É considerada **de fácil uso e muito robusta**.

Apesar de parecer complicado, o processo de instalação da Darknet no Colab com **recursos CUDA** é **tão fácil quanto num computador Linux local**.

TREINAMENTO DA REDE

A DARKNET

Baixe o repositório da Darknet:

```
!git clone https://github.com/AlexeyAB/darknet
```

Configure o MAKEFILE para o uso do CUDA:

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
```

Instala a Darknet:

```
!make
```

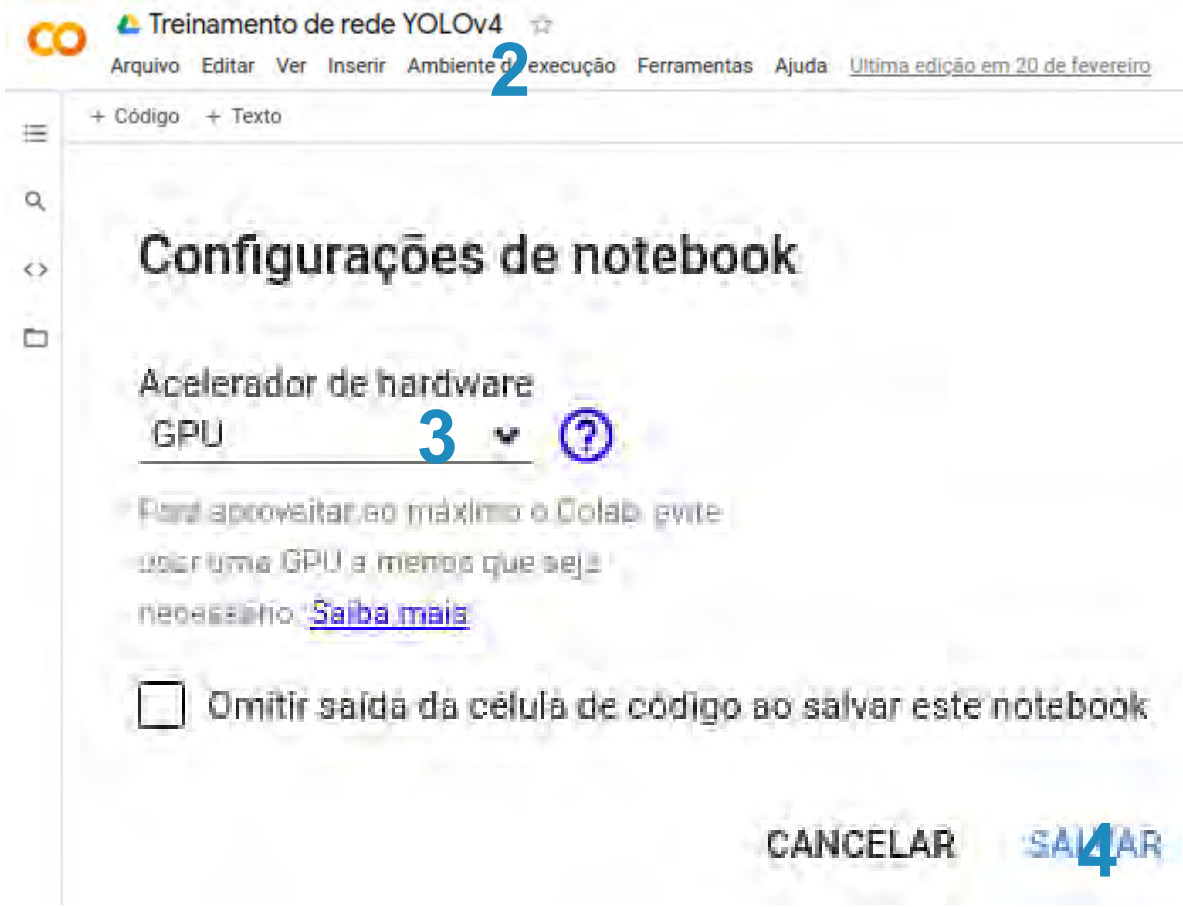
Verifica o CUDA:

```
!/usr/local/cuda/bin/nvcc --version
```

Nota: para executar em uma máquina Linux local, basta retirar os ”!” e “%”.

TREINAMENTO DA REDE

GOOGLE COLABORATORY



1- Conecte-se ao ambiente de desenvolvimento em nuvem;

2- Clique em "**Ambiente de execução**" e acesse a opção "**Alterar tipo de ambiente de execução**";

3- Defina a opção "**GPU**";

4- Salve a configuração.

TREINAMENTO DA REDE

GOOGLE COLABORATORY

1

```
# Baixa o repositório da Darknet:
!git clone https://github.com/AlexeyAB/darknet

# Configura o MAKEFILE para o uso do CUDA (DEPENDE DE GPGPU):
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile

# Verifica o CUDA:
!/usr/local/cuda/bin/nvcc --version

# Instala a Darknet:
!make
```

1- Clique no ► para executar o bloco;

2- Aguarde até que a instalação esteja pronta (o processo deve demorar cerca de **um minuto**).

TREINAMENTO DA REDE

GOOGLE COLABORATORY

Agora que a Darknet está devidamente configurada e instalada com os recursos CUDA, **basta inicializar o treinamento**.

Para isso:

1- Acesse o diretório do ambiente de treinamento no Drive pelo seguinte comando:

```
%cd "/content/drive/My Drive/WorCAP2021"
```

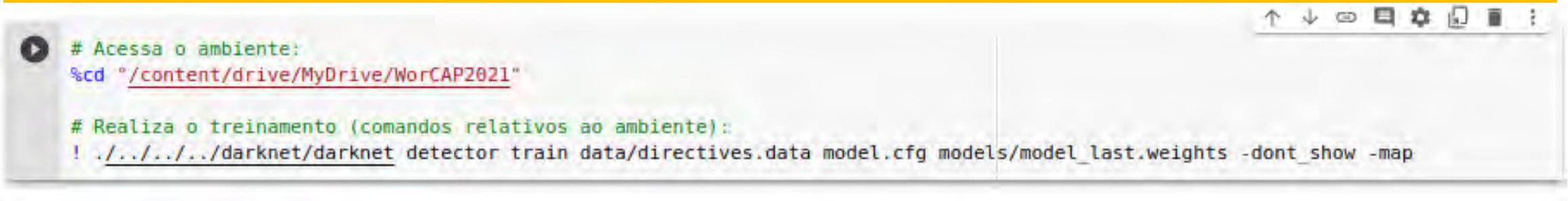
2- Execute a Darknet em modo de treino, apontando para o arquivo de diretivas, arquitetura da rede e pesos iniciais:

```
! ./../../../darknet/darknet detector train data/directives.data model.cfg models/model_last.weights -dont_show -map
```

Dica: além do dump, a opção -map gera um gráfico com o andamento do treinamento, que é armazenado na raíz do ambiente de treinamento (drive/WorCAP2021/).

TREINAMENTO DA REDE

GOOGLE COLABORATORY



```
# Acessa o ambiente:
%cd "/content/drive/MyDrive/WorCAP2021"

# Realiza o treinamento (comandos relativos ao ambiente):
! ./../../../darknet/darknet detector train data/directives.data model.cfg models/model_last.weights -dont_show -map
```

DICA: Acesse o terminal do Chrome (Ctrl+Shift+I) e execute esse código para evitar ser derrubado por inatividade:

```
function ClickConnect(){
console.log("Working");
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
  .click()
}
setInterval(ClickConnect,60000)
```


```
function ClickConnect(){
console.log("Working");
document
  .querySelector('#top-toolbar > colab-connect-button')
  .shadowRoot.querySelector('#connect')
  .click()
}
setInterval(ClickConnect,60000)
```

TREINAMENTO DA REDE

ACOMPANHANDO O TREINAMENTO

AO LONGO DO TREINAMENTO, PODEMOS ANALISAR INFORMAÇÕES SOBRE SEU PROGRESSO.

As duas métricas que devemos prestar atenção é o **accuracy mAP** e **loss**.

As flags aplicadas no comando da Darknet nos disponibiliza duas formas de analisar o progresso do treinamento:

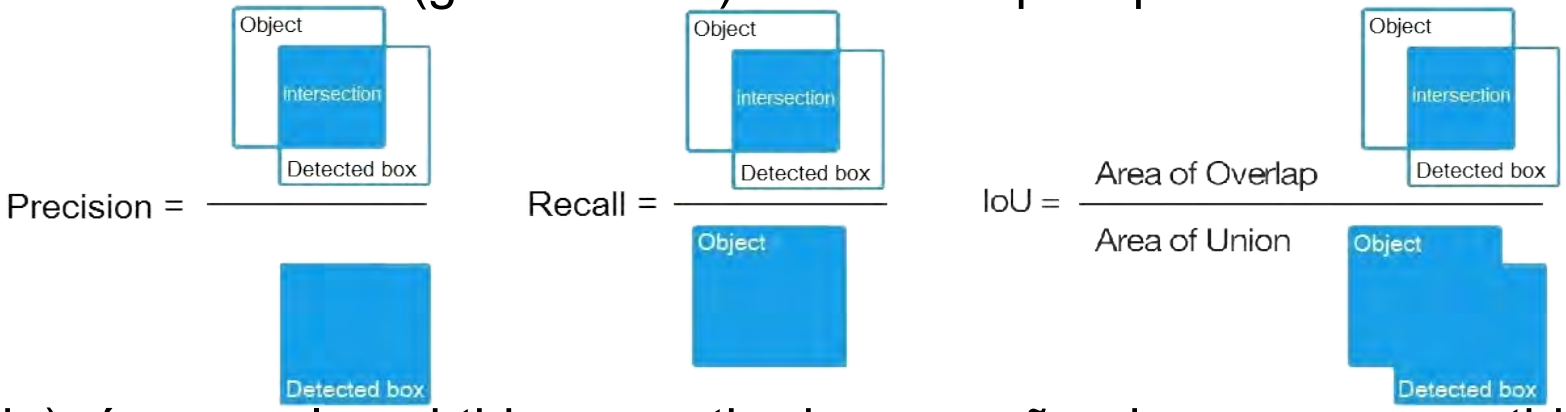
1) o **dump** (saída imediata ao bloco do Colab), que emite **informações precisas de cada época do treinamento**, e
2) o **gráfico**, que apresenta a **plotagem dos dados de accuracy mAP e loss** ostensivos no dump.

O gráfico é gerado dentro do **diretório WorCAP2021**, no seu Drive.

TREINAMENTO DA REDE

ACOMPANHANDO O TREINAMENTO

- O **accuracy mAP** (precisão média geral) trata-se do **desempenho nominal atingido pela rede** durante o processo de validação, sendo um valor baseado na comparação entre o que e onde detecta com os dados rotulados (ground truth). O ideal é que apresente os valores mais elevados possível.



- O **loss** (perda) é um valor obtido a partir da correção dos erros cometidos ao longo do treinamento, que **tende a cair conforme a rede aprende** (sobrando assim cada vez menos coisas para aprender, teoricamente) e está, portanto, intimamente ligado à taxa de aprendizado da rede. Logo, **o loss diz respeito à capacidade da rede se aprimorar**.

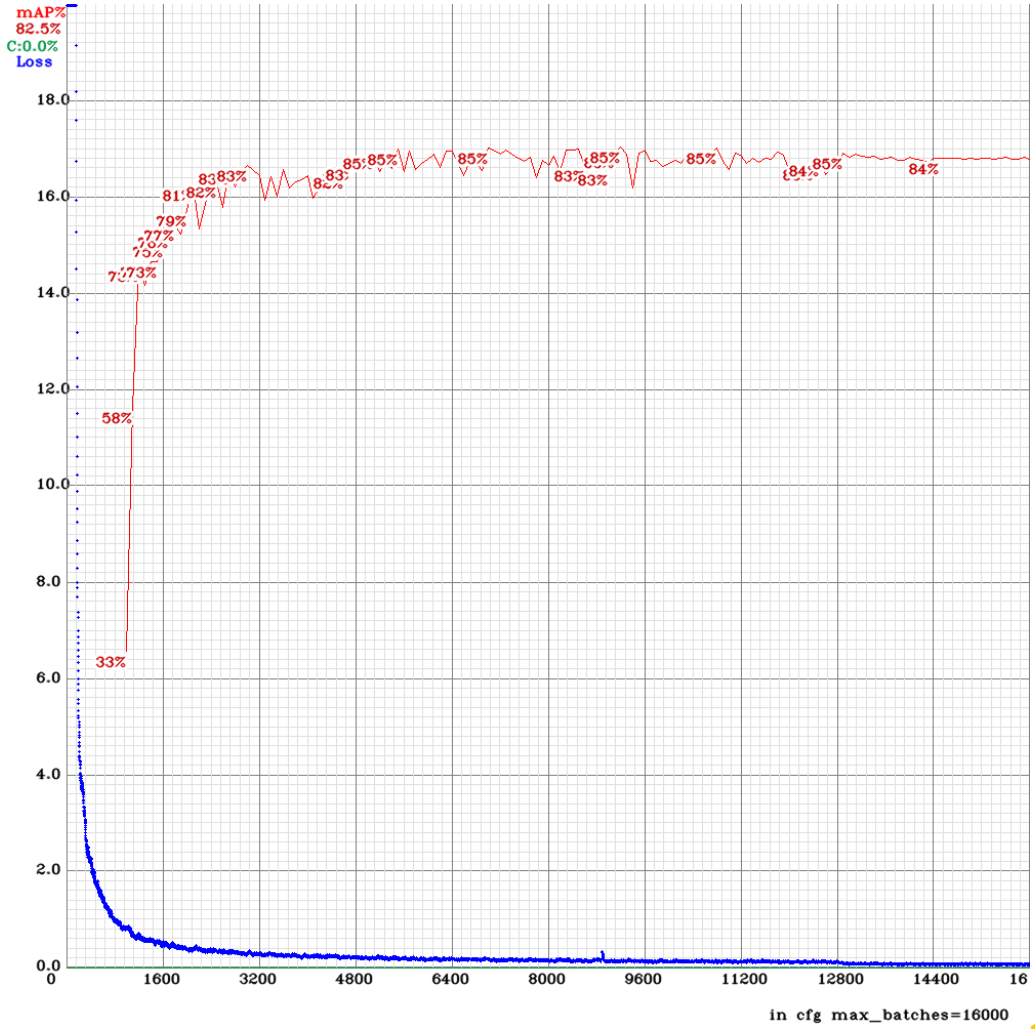
Diversos tipos de loss podem ser analisados como métrica de análise, e a Darknet confia no loss de classe e interseção sobre união -- isto é, o que e onde detecta, respectivamente.

TREINAMENTO DA REDE

ACOMPANHANDO O TREINAMENTO

O mAP e o loss têm, portanto, uma relação inversamente proporcional, onde **o mAP sobe apenas enquanto o loss estiver caindo** e, uma vez que o loss se estabiliza, não sobra perspectiva de crescimento para o mAP.

O **ângulo da evolução do loss indica a taxa de aprendizado e, mais do que isso, a necessidade de mais dados**: ângulos mais agudos indicam uma taxa de aprendizado muito elevada ou dados insuficientes.
Ângulos mais obtusos indicam que a rede pode aproveitar melhor longos períodos de treinamento e atingir resultados melhores, e são, portanto, mais desejados -- a menos que você precise de resultados mais imediatos, mesmo que mais modestos.



TREINAMENTO DA REDE

ACOMPANHANDO O TREINAMENTO

PRESTE ATENÇÃO NO DUMP DO TREINAMENTO:

mAP
Loss
Época

YOLOv4 Full

Tensor Cores are disabled until the first 3000 interations are reached.
Last accuracy mAP@0.5 = 62.51 %, best = 68.58 %
2566: 0.702676, 0.970859 avg loss, 0.001000 rate, 7.205960 seconds, 292224 images, 24.082905 hours left
Loaded: 2.448205 seconds - performance bottleneck on CPU or Disk HDD/SSD
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.842614), count: 9, **class_loss** = 0.023178, **iou_loss** = 2.568325, **total_loss** = 5.378532
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.842614), count: 9, **class_loss** = 0.000246, **iou_loss** = 3.683245, **total_loss** = 7.367212
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.907881), count: 1, **class_loss** = 0.000018, **iou_loss** = 0.027446, **total_loss** = 0.268646
total_bbox = 3985338, rewritten_bbox = 4.501877 %

YOLOv4 Tiny

Tensor Cores are used.
Last accuracy mAP@0.5 = 48.32 %, best = 51.21 %
14566: 0.702676, 0.970859 avg loss, 0.001000 rate, 7.205960 seconds, 292224 images, 1.285632 hours left
Loaded: 2.448205 seconds - performance bottleneck on CPU or Disk HDD/SSD
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 150 Avg (IOU: 0.842614), count: 9, **class_loss** = 0.000326, **iou_loss** = 5.770300, **total_loss** = 5.770626
v3 (iou loss, Normalizer: (iou: 0.07, obj: 1.00, cls: 1.00) Region 161 Avg (IOU: 0.907881), count: 1, **class_loss** = 0.000008, **iou_loss** = 0.097644, **total_loss** = 0.097652
total_bbox = 3985338, rewritten_bbox = 4.501877 %

TREINAMENTO DA REDE

ACOMPANHANDO O TREINAMENTO

PRESTE ATENÇÃO NO DUMP DO TREINAMENTO:

A cada 100 épocas, o treinamento passa por uma fase de **análise de desempenho** nos batches e insere os dados no gráfico. Nesta fase também é impresso dump um relato do atual estado do treinamento para cada classe, com valores de **precisão média** (ap) e matriz de confusão (**verdadeiros positivo e falsos-positivo**), que denunciam classes desequilibradas no conjunto de treinamento e validação. Também há demais métricas tradicionais de avaliação, como **precisão**, **recorrência** e **F1**.

```
class_id = 0, name = alligator, ap = 43.34%   (TP = 21, FP = 4)
class_id = 1, name = bear, ap = 99.78%        (TP = 46, FP = 2)
class_id = 2, name = goat, ap = 92.33%        (TP = 75, FP = 5)
class_id = 3, name = lion, ap = 94.43%        (TP = 24, FP = 10)
class_id = 4, name = monkey, ap = 91.89%      (TP = 196, FP = 11)
class_id = 5, name = puma, ap = 91.61%        (TP = 31, FP = 8)
class_id = 6, name = warthog, ap = 81.53%     (TP = 25, FP = 9)
class_id = 7, name = wolf, ap = 76.31%        (TP = 16, FP = 6)
```

```
for conf_thresh = 0.25, precision = 0.89, recall = 0.83, F1-score = 0.86
for conf_thresh = 0.25, TP = 434, FP = 55, FN = 86, average IoU = 78.58 %
IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
mean average precision (mAP@0.50) = 0.839023, or 83.90 %
Total Detection Time: 14 Seconds
```

$$Acurácia = \frac{Verdadeiros\ Positivos\ (TP) + Verdadeiros\ Negativos\ (VN)}{Total}$$

$$Precisão = \frac{Verdadeiros\ Positivos\ (TP)}{Verdadeiros\ Positivos\ (TP) + FalsosPositivos\ (FP)}$$

$$Recall = \frac{Verdadeiros\ Positivos\ (TP)}{Verdadeiros\ Positivos\ (TP) + Falsos\ Negativos\ (FN)}$$

$$F1 = \frac{2 * precisão * recall}{precisão + recall}$$

TREINAMENTO DA REDE

PÓS-TREINO

O treinamento irá perdurar até que 1) a Darknet atinja os valores considerados ótimos, 2) você o interrompa ou 3) o Colab derrube seu ambiente -- lembre-se: os recursos são disponibilizados por até 12 horas diárias, mas **seu ambiente pode ser arbitrariamente derrubado**.

De todo modo, ao longo do treinamento, **são gerados arquivos contendo pesos**, armazenados por padrão na pasta **models** no ambiente. São utilizadas as seguintes terminologias:

- **final.weights:** pesos tidos como definitivos pela Darknet após o treinamento estar encerrado pela mesma;
- **best.weights:** pesos com melhores valores de mAP entre todo o processo de treinamento até então;
- **last.weights:** pesos gerados como checkpoint a cada 100 quantidade de épocas de treinamento, idealmente formado para ser usado como peso inicial para retomar o treinamento após uma eventual interrupção; e
- **X.weights:** onde X é um número referente à época de treinamento quando foi gerado, sendo mais um tipo de arquivo de checkpoint.

Dito isso, **basta acessar a pasta models no Drive e baixar o arquivo de pesos desejado** e usá-lo para ativar a rede em sua aplicação.

TREINAMENTO DA REDE

TOY PROBLEM

TENDO FEITO ISSO:

Você deve ter uma pasta contendo a **rede convolucional YOLOv4 no formato desejado** (Full ou Tiny) junto com os **dados e diretivas para o treinamento**, tudo pronto para que a rede possa ser treinada pela Darknet.

Com isso, temos encerrada a **Fase 4 - Treinamento da rede convolucional**, tal como disponibilizado no **toy problem**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

CONTEXTUALIZAÇÃO

Um bom tempo foi dedicado ao desenvolvimento da rede, desde reunir imagens até o treinamento. **Finalmente, chegou a hora de ativar sua rede convolucional.**

O conceito de "ativação" da rede consiste em colocá-la para funcionar em uma aplicação. Existe uma infinidade de formas de se ativar a rede convolucional, e a grande maioria delas convergem à **programação com frameworks de visão computacional**.

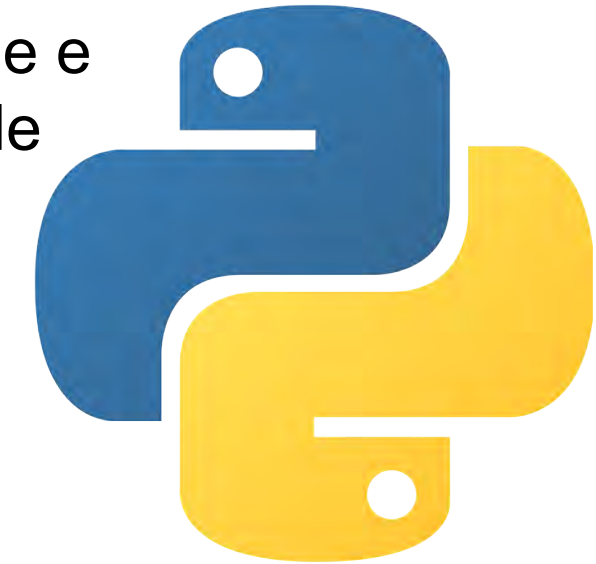
As linguagens de programação mais utilizadas para aplicações de visão computacional são **Python** e **C**, enquanto o framework de visão computacional mais utilizado é o **OpenCV**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

PYTHON

Aqui trataremos de uma aplicação simples desenvolvida em **Python**.

O Python é uma linguagem de programação **open source interpretada**, **multiparadigma**, de **alto nível** e de **propósito geral**, altamente robusta, ágil, portável e de **rápido aprendizado**, sendo também muito recomendada como primeira linguagem de programação devido sua simplicidade e **grande aceitação** em uma gama muito ampla e profunda de comunidades, inclusive a **científica**.



A versão de Python a ser utilizada será **Python3**.

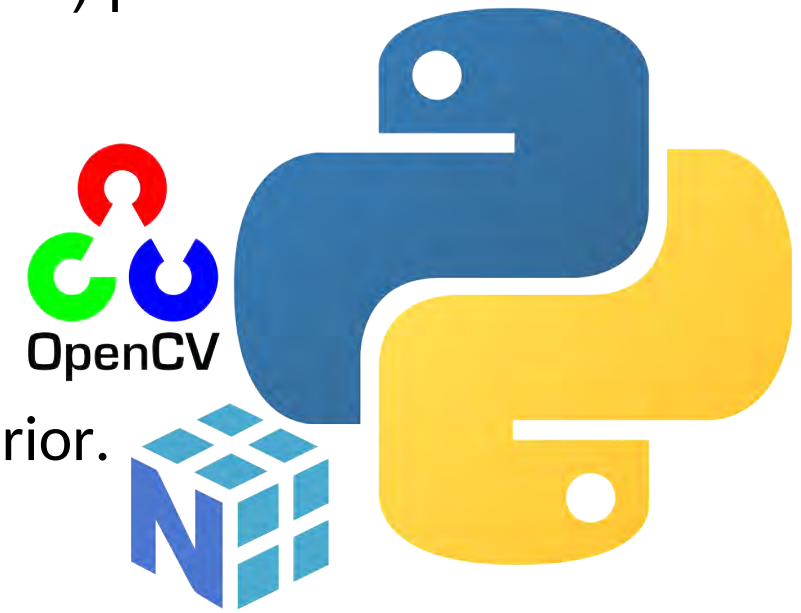
ATIVAÇÃO DAS REDES CONVOLUCIONAIS

PYTHON

E não apenas Python puro será utilizado; serão utilizadas também as bibliotecas **NumPy** e **OpenCV**.

Uma vez tendo o Python instalado, ambas as bibliotecas podem ser instaladas pelo Python Index Package (mais conhecido como PIP) pelo comando:

```
pip install opencv-python numpy
```



Nota: a versão do OpenCV instalada deve ser 4.4 ou superior.

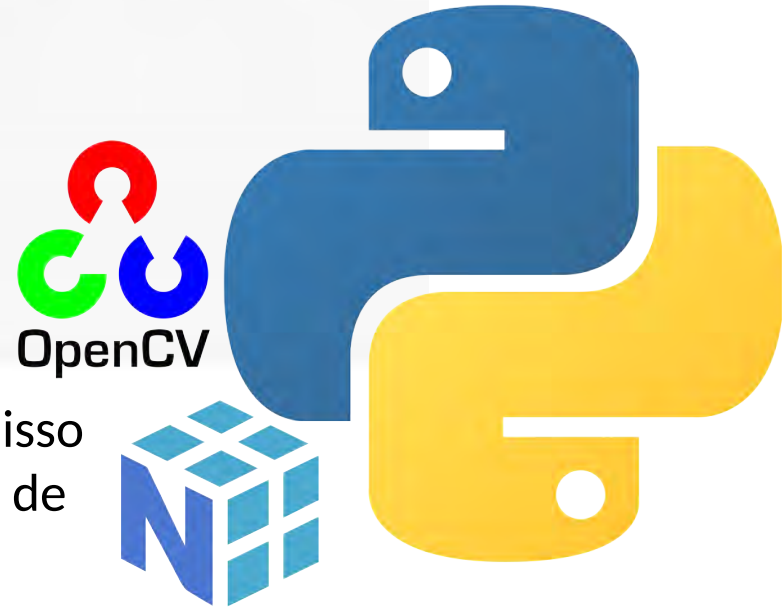
ATIVAÇÃO DAS REDES CONVOLUCIONAIS

PYTHON

ATIVAÇÃO DA REDE

```
[ ] # Instala uma versão do OpenCV compatível com a Darknet:
    !pip install opencv-python==4.5.3.56
```

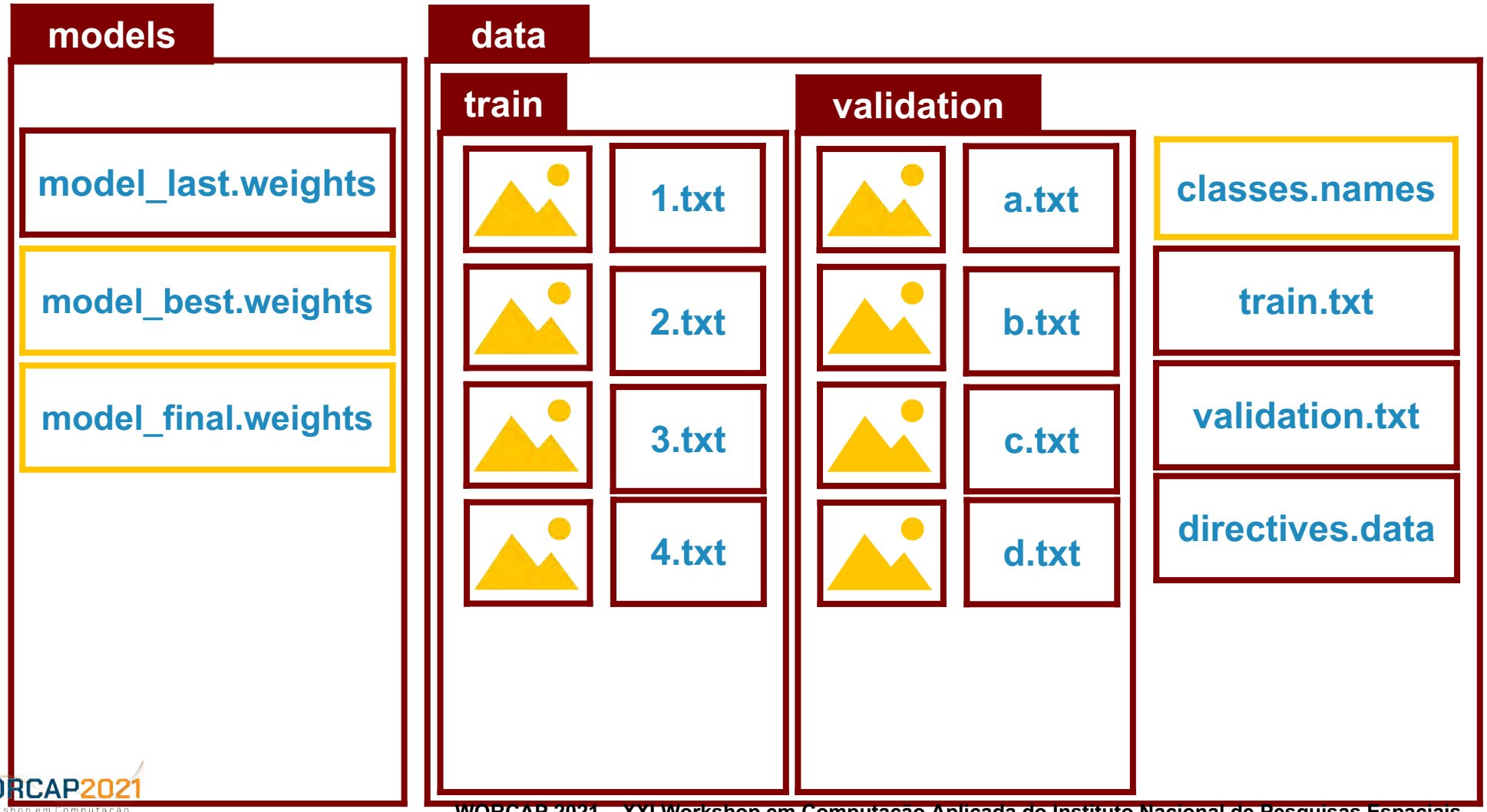
```
[ ] # Importa as bibliotecas utilizadas para a ativação:
    from os import listdir
    from cv2 import imread, imwrite, VideoWriter, VideoWriter_fourcc, VideoCapture, CAP_PROP_FPS, dnn, FONT_HERSHEY_PLAIN, rectangle, putText
    from google.colab.patches import cv2_imshow as imshow
    from numpy import argmax


    # Define o caminho para o ambiente de ativação no Drive:
    path = "/content/drive/MyDrive/WorCAP2021/activation/"

    # Carrega a rede convolucional (arquitetura e pesos):
    net = dnn.readNet(path + "networks/model.cfg", path + "networks/model.weights")

    # Carrega as classes discrimináveis:
    classes = []
    with open(path + "networks/model.names", 'r') as f:
        classes = f.read().splitlines()
```



É razoável executar a ativação da rede em sua máquina pessoal, mas isso **agora também é possível no Colab**. Imediatamente abaixo do código de treinamento, pode ser executada a ativação da rede.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

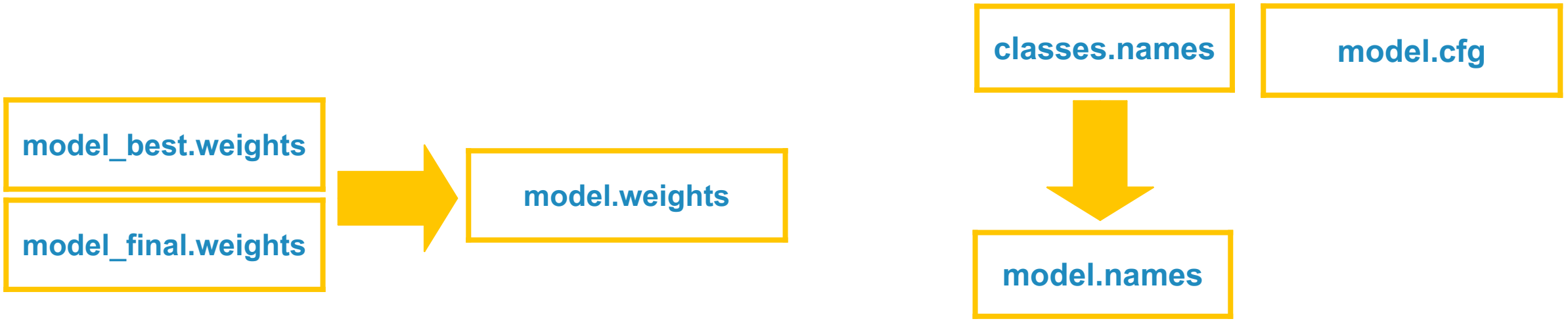
AMBIENTE DE ATIVAÇÃO



Do ambiente de treinamento, você precisará apenas da rede convolucional de fato: rótulos das classes (**classes.names**), arquitetura (**model.cfg**) e pesos resultantes (**model_final.weights** ou **model_best.weights**).

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AMBIENTE DE ATIVAÇÃO

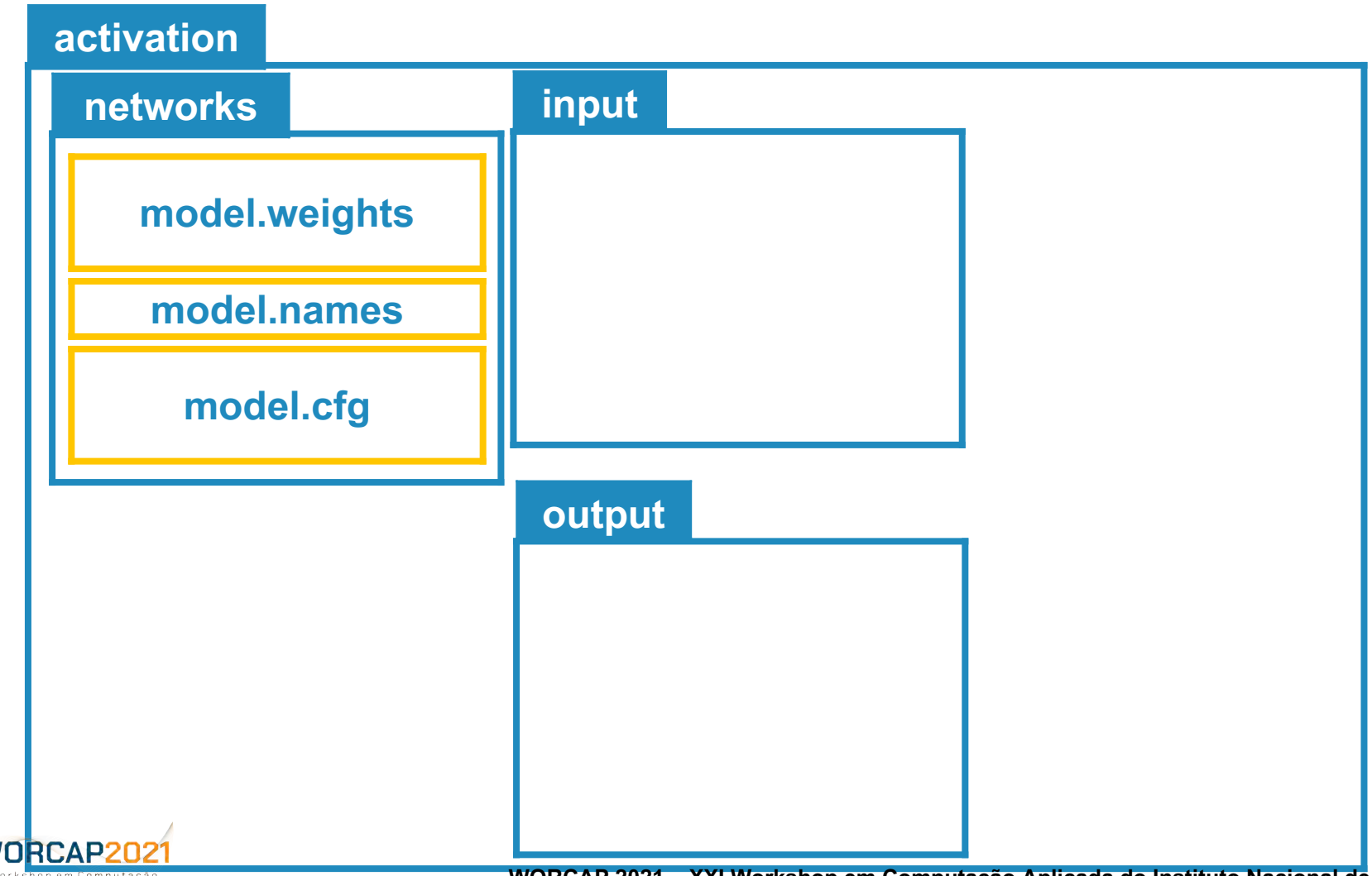


O arquivo de pesos escolhido deve ser renomeado para **model.weights** e o arquivo de rótulos deve ser renomeado para **model.names**.

Então, o **ambiente de ativação deve ser gerado no Drive**: dentro da pasta **WorCAP2021**, crie uma pasta chamada **activation**.
Dentro de **activation**, deve haver também pastas nomeadas **networks**, **input** e **output**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

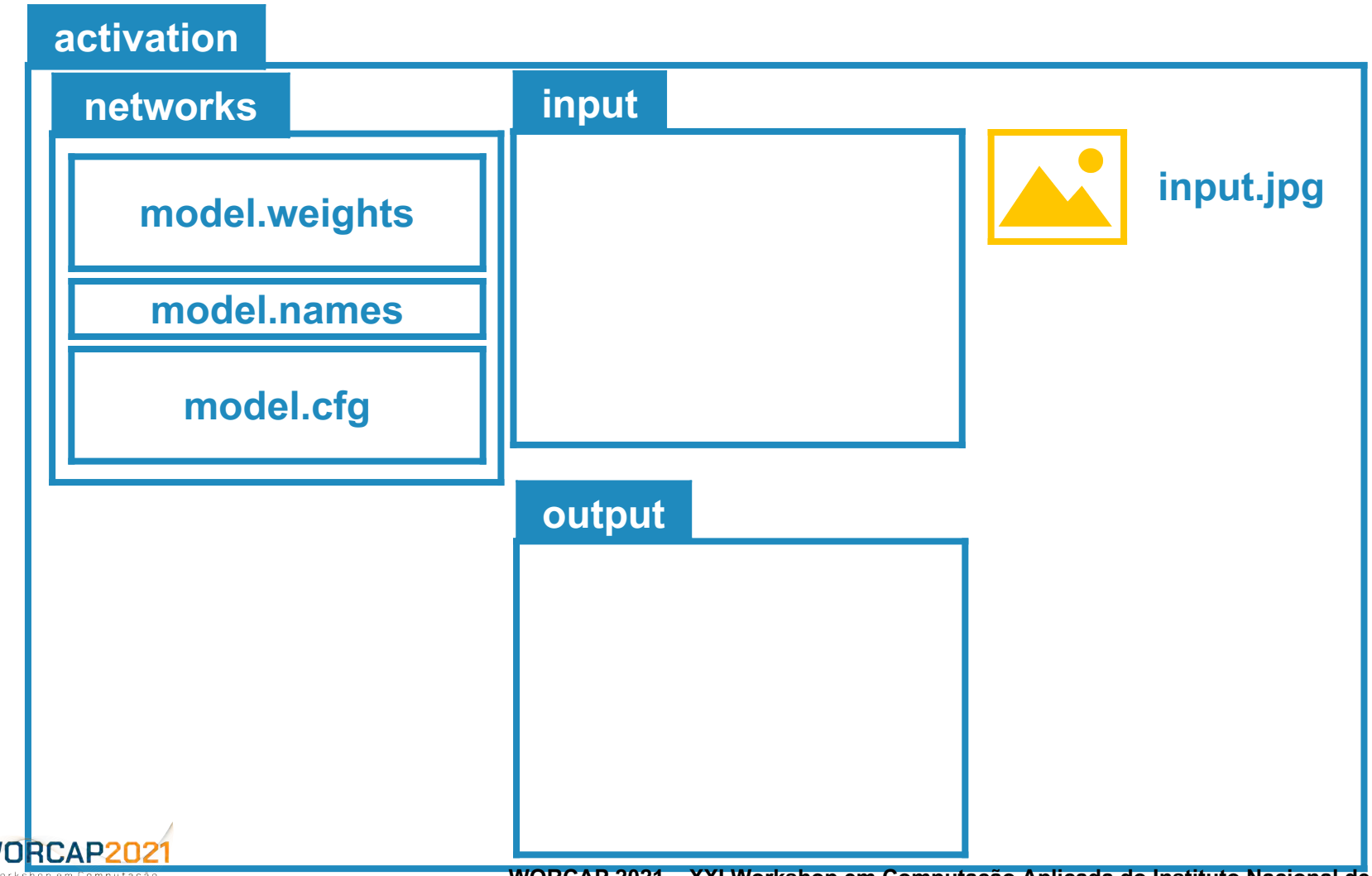
AMBIENTE DE ATIVAÇÃO



- A rede convolucional deve ser inserida na pasta **networks**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AMBIENTE DE ATIVAÇÃO

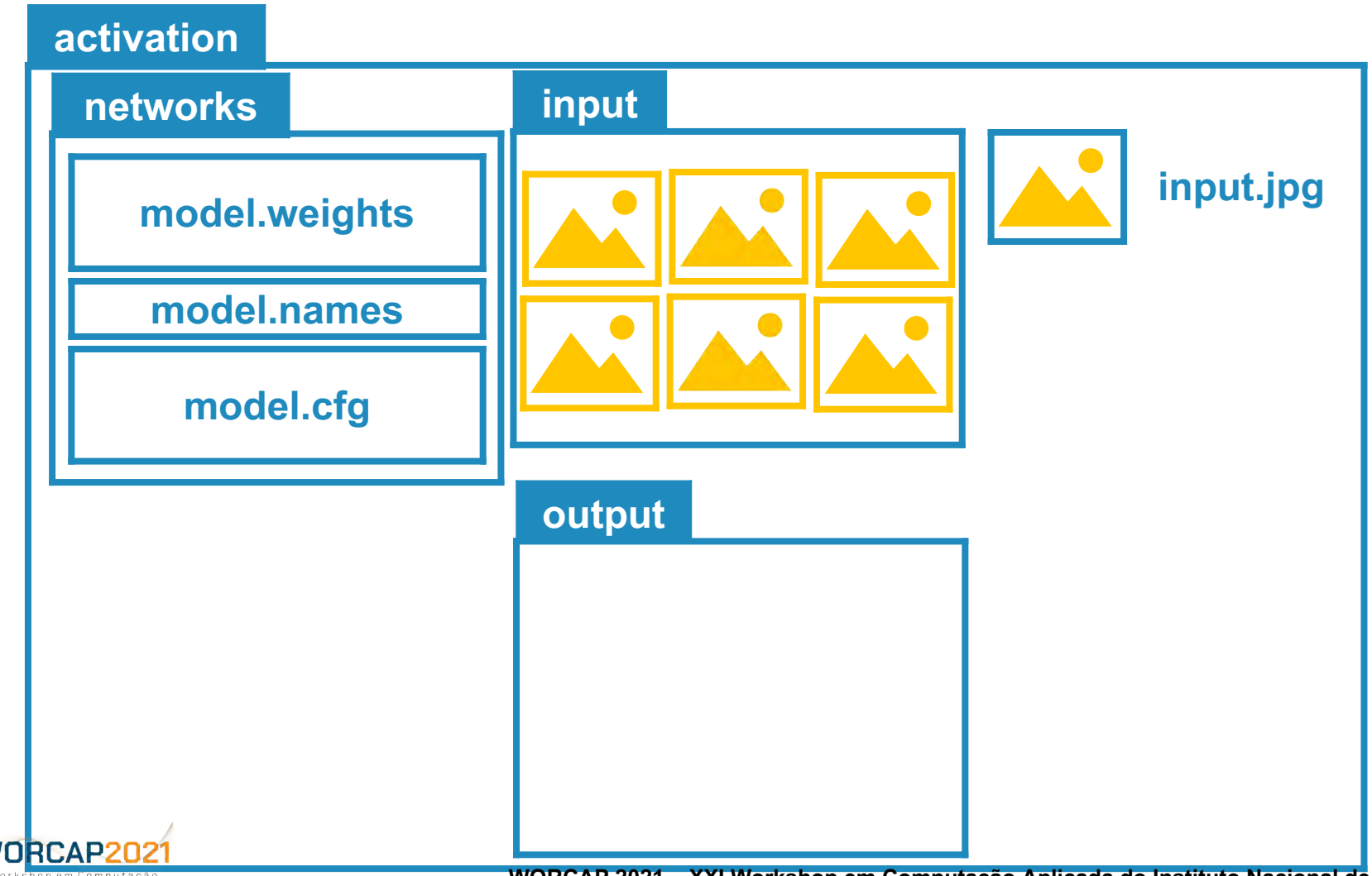


- A rede convolucional deve ser inserida na pasta **networks**.

- Se a rede será ativada em **uma única imagem**, essa imagem deve estar na **raíz do activation** e deve ser nomeada como **input.jpg**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AMBIENTE DE ATIVAÇÃO



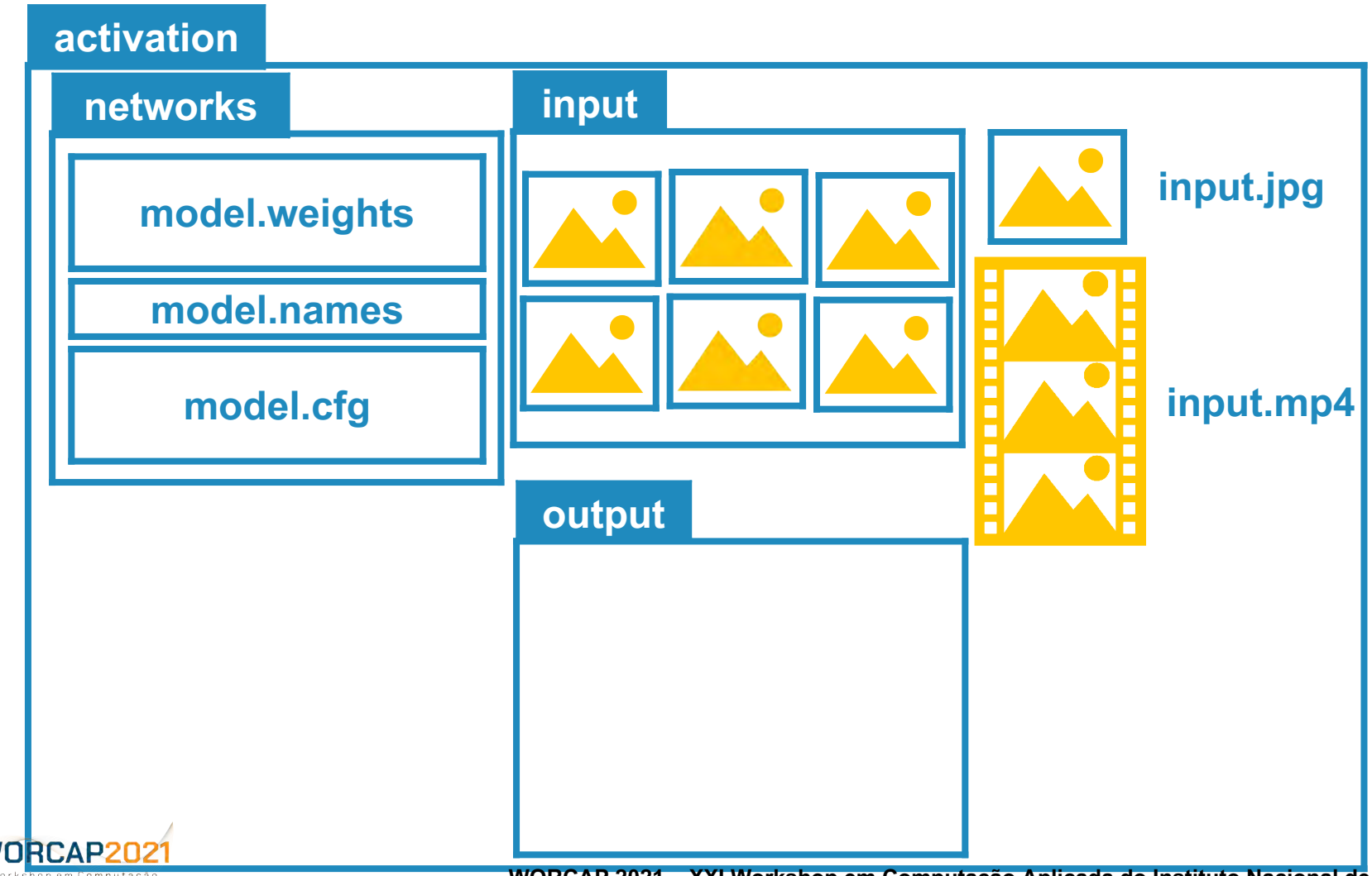
- A rede convolucional deve ser inserida na pasta **networks**.

- Se a rede será ativada em **uma única imagem**, essa imagem deve estar na **raíz do activation** e deve ser nomeada como **input.jpg**.

- Se a rede será ativada em **uma série de imagens**, elas devem ser todas inseridas na pasta **input**, sem nenhum nome espeífico.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AMBIENTE DE ATIVAÇÃO



- A rede convolucional deve ser inserida na pasta **networks**.

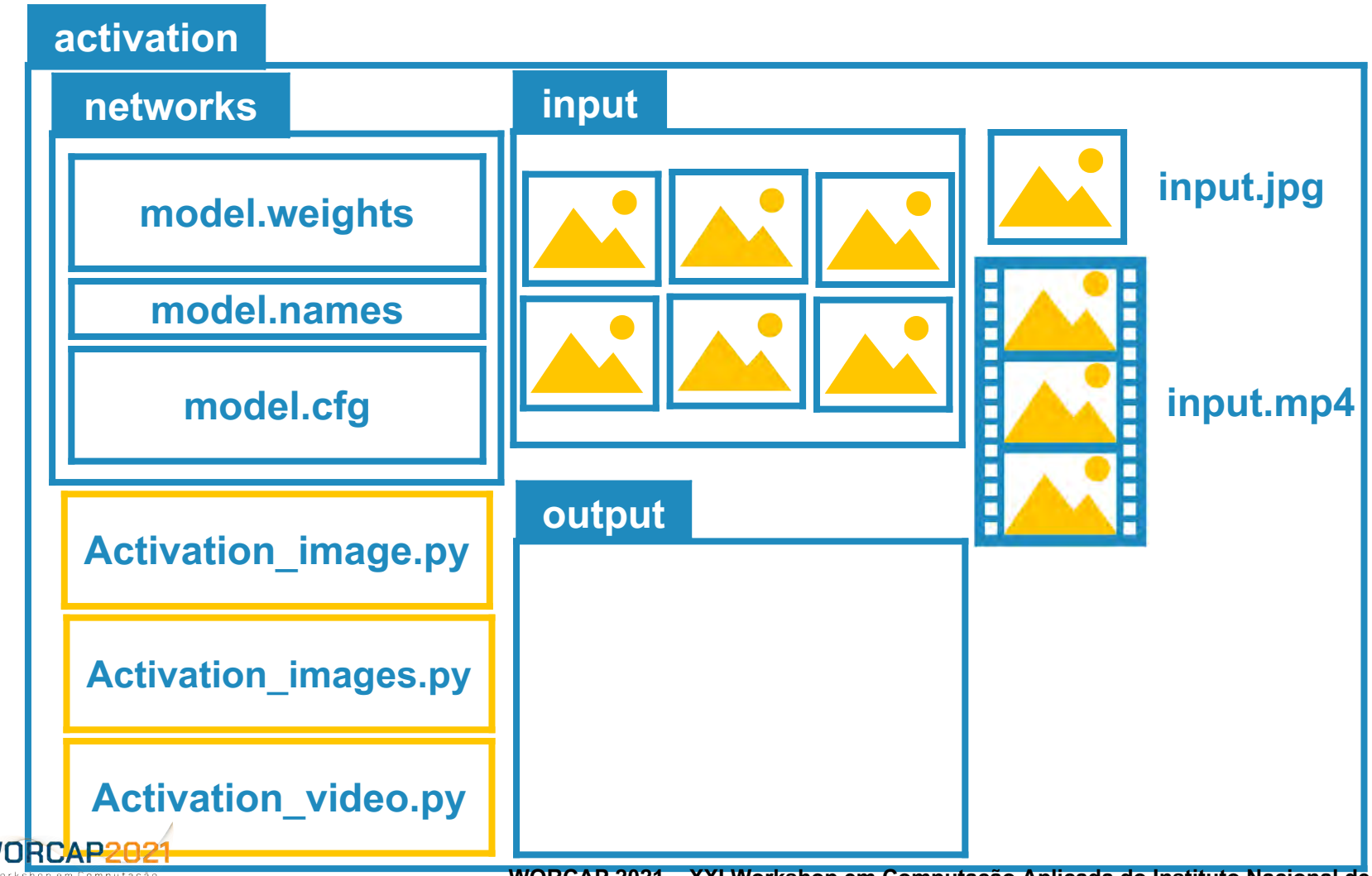
- Se a rede será ativada em **uma única imagem**, essa imagem deve estar na **raíz do activation** e deve ser nomeada como **input.jpg**.

- Se a rede será ativada em **uma série de imagens**, elas devem ser todas inseridas na pasta **input**, sem nenhum nome espeífico.

- Se a rede será ativada em **um vídeo**, esse vídeo deve estar na **raíz do activation** e deve ser nomeada como **input.mp4**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AMBIENTE DE ATIVAÇÃO



Se você quiser executar a ativação em sua máquina, você pode baixar os três códigos Python no repositório **ConvTraining** no GitHub:

- **Activation_image.py** realiza a ativação em uma única imagem.
- **Activation_images.py** realiza a ativação nas imagens em input; e
- **Activation_video.py** realiza a ativação em vídeo.

https://github.com/Rafael-Marinho/ConvTraining

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

OPÇÕES DE ATIVAÇÃO

Finalmente, uma vez tendo o ambiente de ativação preparado e organizado, **é hora de ativar a sua rede convolucional**.
Para isso, **você pode executar em sua máquina pessoal** o algoritmo de ativação com o **Python** como, por exemplo, em:

```
python3 Activation_image.py
python3 Activation_images.py
python3 Activation_video.py
```

Uma imagem
Uma pasta inteira de imagens
Um vídeo

A ativação da rede convolucional, nos computadores atuais, deve ser de **pelo menos uma imagem por segundo**. Isso **varia**, principalmente, com as **dimesões das imagens** e o **hardware** onde é executada a ativação e como o **software** (sistema operacional) aloca seus recursos para a tarefa.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

OPÇÕES DE ATIVAÇÃO

OU você pode executar os blocos para isso no Google Colab.

Diferente do treinamento, **a ativação não exige o uso da GPU** (ainda que seja recomendado), não correndo assim o risco de ser derrubado do servidor.

```
[ ] # Instala uma versão do OpenCV compatível com a Darknet:
    !pip install opencv-python==4.5.3.56
```

```
[ ] # Importa as bibliotecas utilizadas para a ativação:
    from os import listdir
    from cv2 import imread, imwrite, VideoWriter, VideoWriter_fourcc, VideoCapture, CAP_PROP_FPS, dnn, FONT_HERSHEY_PLAIN, rectangle, putText
    from google.colab.patches import cv2_imshow as imshow
    from numpy import argmax


    # Define o caminho para o ambiente de ativação no Drive:
    path = "/content/drive/MyDrive/WorCAP2021/activation/"

    # Carrega a rede convolucional (arquitetura e pesos):
    net = dnn.readNet(path + "networks/model.cfg", path + "networks/model.weights")

    # Carrega as classes discrimináveis:
    classes = []
    with open(path + "networks/model.names", 'r') as f:
        classes = f.read().splitlines()
```

O primeiro passo para usar qualquer um dos algoritmos é **instalar uma versão do OpenCV compatível com a Darknet** e **carregar a rede convolucional YOLO** presente no Drive.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

O algoritmo segue a seguinte rotina:

1) Carrega a rede neural convolucional, as classes e a imagem onde a rede é ativada;
2) detecta as instâncias de objetos e identifica sua classe (e também a taxa de confiança de cada identificação);
3) renderiza as bounding boxes, rótulos de classe e suas respectivas taxas de confiança de cada instância capturada no quadro;
4) dispõe o quadro, já com os objetos da tarefa 3 inclusas; e
5) adiciona o quadro em um arquivo de saída.

Essa é a rotina para uma única imagem.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

O algoritmo segue a seguinte rotina:

1) Carrega a rede neural convolucional, as
ativada;
2) detecta as instâncias de objetos e identif
de confiança de cada identificação);
3) renderiza as bounding boxes, rótulos de
de confiança de cada instância capturada n
4) dispõe o quadro, já com os objetos da ta
5) adiciona o quadro em um arquivo de saí

Essa é a rotina para um

ATIVAÇÃO EM UMA ÚNICA IMAGEM:

```
# Carrega a imagem:
img = imread(path + "input.jpg")

# Define variáveis de execução, ponteiros e afins:
height, width, _ = img.shape

net.setInput(dnn.blobFromImage(img, (1 / 255), (415, 416), (0, 0, 0), swapRB=True, crop=False))
boxes = []
confidences = []
class_ids = []

# Detecção e armazenamento das instâncias de objetos:
for output in net.forward(net.getUnconnectedOutLayersNames()):
    for detection in output:
        scores = detection[5:]
        class_id = argmax(scores)
        confidence = scores[class_id]
        if (confidence > 0.5):
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)
            x = int(center_x - (w / 2))
            y = int(center_y - (h / 2))
            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

# Renderização no quadro dos delimitadores, classe e índice de confiança de cada instância detectada:
indexes = dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
if (len(indexes) > 0):
    for i in indexes.flatten():
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = str(round(confidences[i], 2))
        rectangle(img, (x, y), ((x + w), (y + h)), (0, 0, 200), 2)
        rectangle(img, (x, y - 20), (x + ((len(label) + len(str(confidence))) * 10), y), (0, 0, 200), -1)
        putText(img, (label + ' ' + confidence), (x, (y - 5)), FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)

# Disposição e armazenamento em disco (no Drive) do quadro e conteúdo nele renderizado:
imwrite(path + "output.png", img)
imshow(img)
```

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

Se, ao invés de uma única imagem, for em um conjunto de imagens:

1) Carrega a rede neural convolucional, as classes e a imagem onde a rede é ativada;
2) lê o diretório input, criando uma lista de imagens onde deve ser ativada;
LOOP
3) detecta as instâncias de objetos e identifica sua classe (e também a taxa de confiança de cada identificação);
4) renderiza as bounding boxes, rótulos de classe e suas respectivas taxas de confiança de cada instância capturada no quadro; e
5) adiciona o quadro em um arquivo de saída no diretório output.

Cada iteração no loop é uma imagem do diretório input.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

Se, ao invés de uma única imagem, for em

1) Carrega a rede neural convolucional, as cl
ativada;
2) lê o diretório input, criando uma lista de ima
LOOP
3) detecta as instâncias de objetos e identi
de confiança de cada identificação);
4) renderiza as bounding boxes, rótulos de
de confiança de cada instância capturad
5) adiciona o quadro em um arquivo de saí

Cada iteração no loop é uma imag

ATIVAÇÃO EM DIRETÓRIO CONTENDO VÁRIAS IMAGENS:

```
# Carrega a lista de imagens no diretório de ativação:
image_files = []
for filename in listdir(path + "input"):
    image_files.append(filename)

# Executa a ativação em cada imagem dentro do diretório de ativação:
for image in image_files:
    # Carrega a imagem:
    img = imread(path + "input/" + image)

    # Define variáveis de execução, ponteiros e afins:
    height, width, _ = img.shape

    net.setInput(dnn.blobFromImage(img, (1 / 255), (415, 416), (0, 0, 0), swapRB=True, crop=False))
    boxes = []
    confidences = []
    class_ids = []

    # Detecção e armazenamento das instâncias de objetos:
    for output in net.forward(net.getUnconnectedOutLayersNames()):
        for detection in output:
            scores = detection[5:]
            class_id = argmax(scores)
            confidence = scores[class_id]
            if (confidence > 0.5):
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - (w / 2))
                y = int(center_y - (h / 2))
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # Renderização no quadro dos delimitadores, classe e índice de confiança de cada instância detectada:
    indexes = dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    if (len(indexes) > 0):
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = str(round(confidences[i], 2))
            rectangle(img, (x, y), ((x + w), (y + h)), (0, 0, 200), 2)
            rectangle(img, (x, y - 20), (x + ((len(label) + len(str(confidence))) * 10), y), (0, 0, 200), -1)
            putText(img, (label + ' ' + confidence), (x, (y - 5)), FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)

    # Armazenamento da imagem resultante em disco (no Drive):
    imwrite((path + "output/" + image), img)
```

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

Se, ao invés de uma única imagem, for um vídeo:

1) Carrega a rede neural convolucional, as classes e o vídeo onde a rede é ativada, e inicia o ponteiro de escrita do vídeo de saída;

LOOP

- **2)** detecta as instâncias de objetos e identifica sua classe (e também a taxa de confiança de cada identificação);
- **3)** renderiza as bounding boxes, rótulos de classe e suas respectivas taxas de confiança de cada instância capturada no quadro;
- **4)** dispõe o quadro, já com os objetos da tarefa 3 inclusas;
- **5)** adiciona o quadro em um arquivo de saída; e

6) libera os ponteiros de leitura e escrita.

Cada iteração no loop é um quadro do vídeo.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

O ALGORÍTMO

Se, ao invés de uma única imagem, for um víde

1) Carrega a rede neural convolucional, as class
ativada, e inicia o ponteiro de escrita do vídeo de s

LOOP

2) detecta as instâncias de objetos e identifica
de confiança de cada identificação);

3) renderiza as bounding boxes, rótulos de cla
de confiança de cada instância capturada nc

4) dispõe o quadro, já com os objetos da tarefa

5) adiciona o quadro em um arquivo de saída;

6) libera os ponteiros de leitura e escrita.

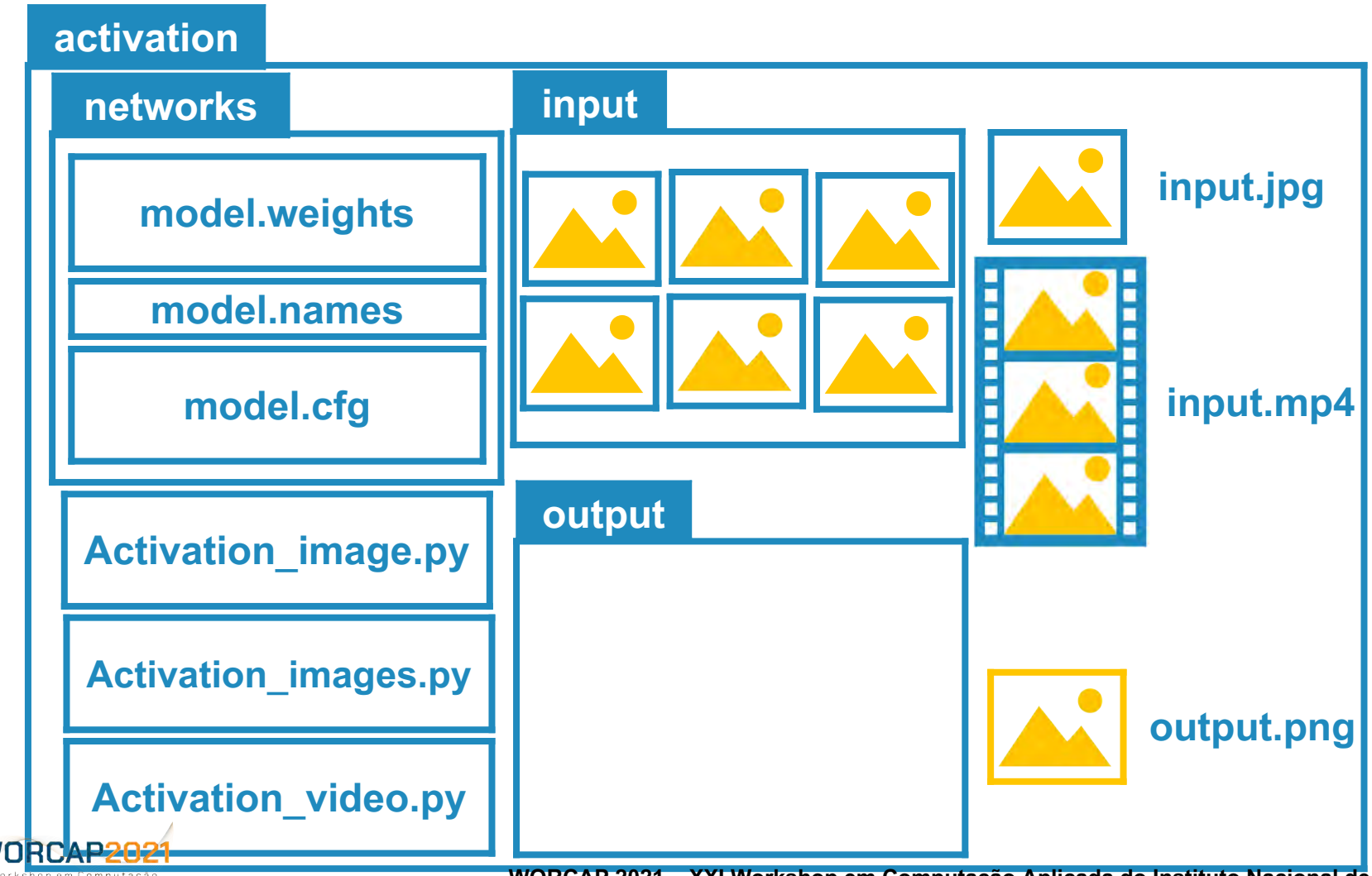
Cada iteração no loop é um qua

ATIVAÇÃO EM FLUXO DE VÍDEO:

```
# Define ponteiros de leitura e escrita do vídeo de entrada e saída, respectivamente:
cap = VideoCapture(path + "input.mp4")
out = VideoWriter(path + 'output.mp4', VideoWriter_fourcc(*"mp4v"), 20, (int(cap.get(3)), int(cap.get(4))))

# Loop responsável pelo fluxo de vídeo:
while (1 < 2):

    # Define variáveis de execução, ponteiros e afins:
    _, img = cap.read()

    try:
        height, width, _ = img.shape
    except(AttributeError):
        break
    net.setInput(dnn.blobFromImage(img, (1 / 255), (415, 416), (0, 0, 0), swapRB=True, crop=False))
    boxes = []
    confidences = []
    class_ids = []

    # Detecção e armazenamento das instâncias de objetos:
    for output in net.forward(net.getUnconnectedOutLayersNames()):
        for detection in output:
            scores = detection[5:]
            class_id = argmax(scores)
            confidence = scores[class_id]
            if (confidence > 0.5):
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - (w / 2))
                y = int(center_y - (h / 2))
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    # Renderização no quadro dos delimitadores, classe e índice de confiança de cada instância detectada:
    indexes = dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    if (len(indexes) > 0):
        for i in indexes.flatten():
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            confidence = str(round(confidences[i], 2))
            rectangle(img, (x, y), ((x + w), (y + h)), (0, 0, 200), 2)
            rectangle(img, (x, y - 20), (x + ((len(label) + len(str(confidence))) * 10), y), (0, 0, 200), -1)
            putText(img, (label + ' ' + confidence), (x, (y - 5)), FONT_HERSHEY_PLAIN, 1, (255, 255, 255), 2)

    # Disposição e armazenamento do quadro no fluxo de vídeo em disco (no Drive) e conteúdo nele renderizado:
    imshow(img)
    out.write(img)

# Encerramento dos ponteiros de leitura e escrita:
cap.release()
out.release()
```

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

APÓS A ATIVAÇÃO

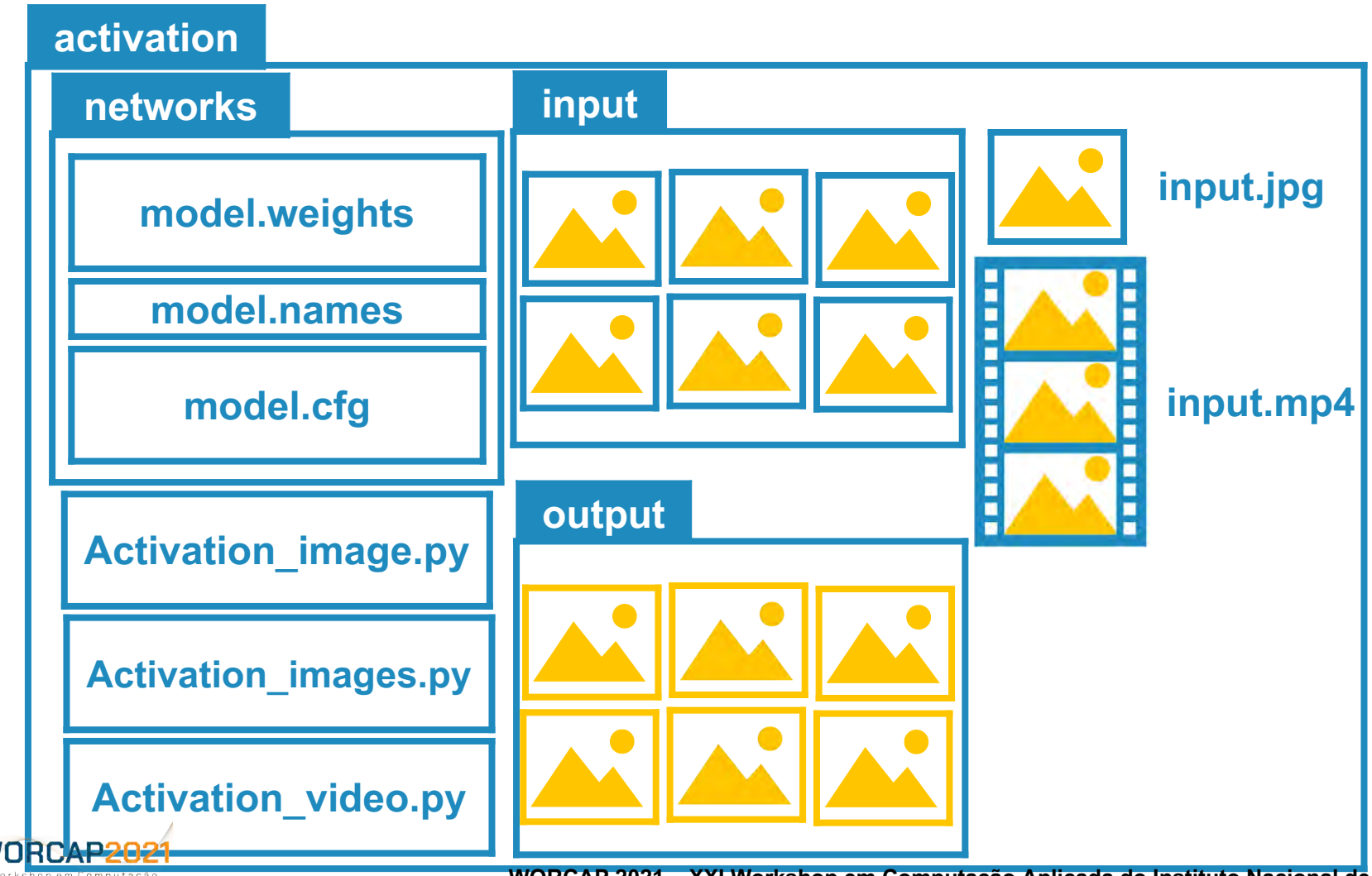


Cada algoritmo gera seus próprios **arquivos de saída**:

- Se a rede será ativada em **uma única imagem**, o arquivo de saída será uma imagem chamada **output.png**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

APÓS A ATIVAÇÃO



Cada algoritmo gera seus próprios **arquivos de saída**:

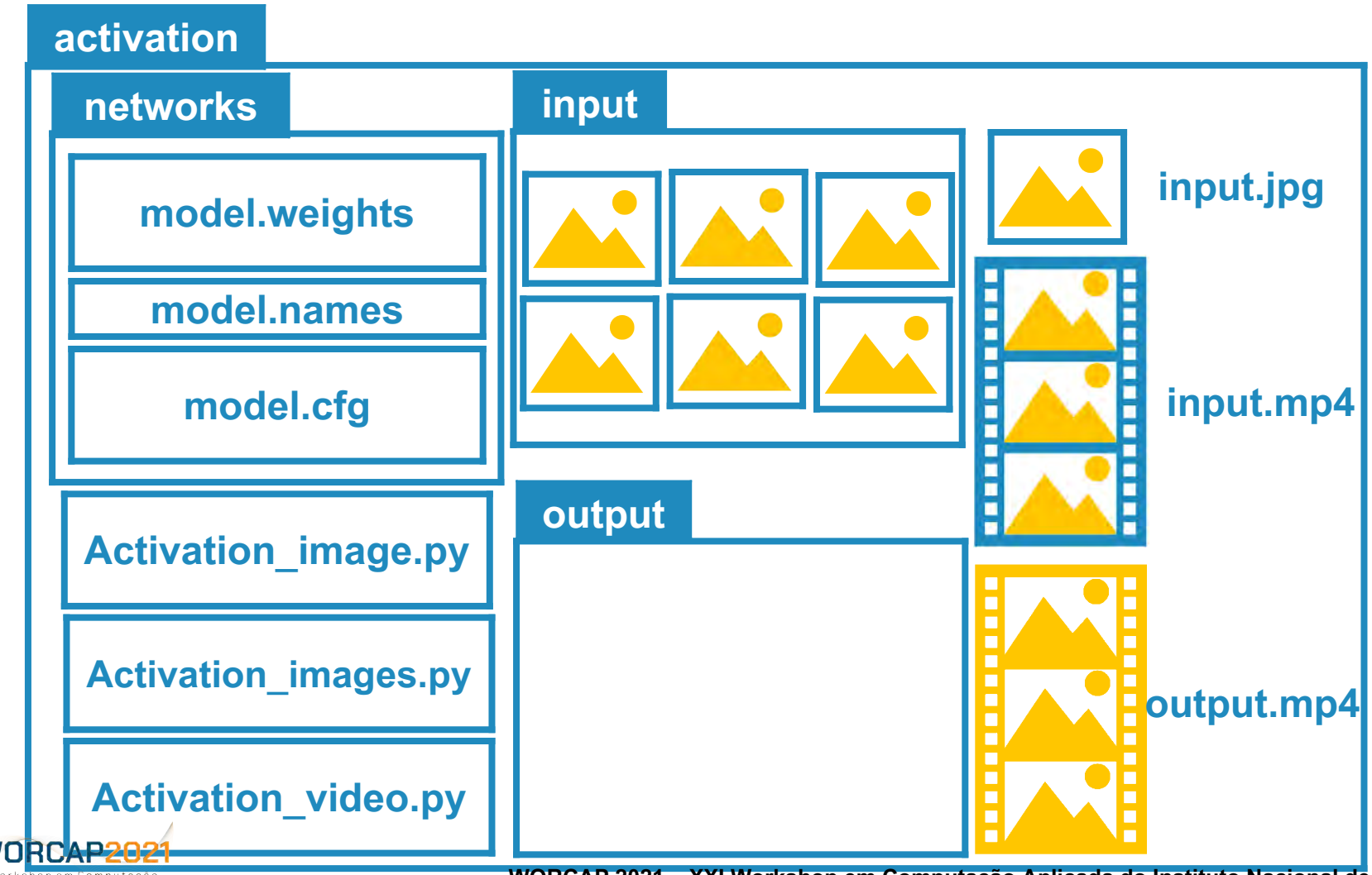
- Se a rede ser ativada em **uma única imagem**, o arquivo de saída será uma imagem chamada **output.png**.

- Se a rede ser ativada nas **imagens do diretório input**, o diretório **output** será alimentado com **uma imagem para cada imagem presente em input**.

https://github.com/Rafael-Marinho/Conv-Training

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

APÓS A ATIVAÇÃO



Cada algoritmo gera seus próprios **arquivos de saída**:

- Se a rede ser ativada em **uma única imagem**, o arquivo de saída será uma imagem chamada **output.png**.

- Se a rede ser ativada nas **imagens do diretório input**, o diretório **output** será alimentado com **uma imagem para cada imagem presente em input**.

- Se a rede ser ativada em **um vídeo**, o arquivo de saída será um vídeo chamada **output.mp4**.

https://github.com/Rafael-Marinho/Conv-Training

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

TOY PROBLEM

TENDO FEITO ISSO:

Você deve ter uma pasta contendo a **rede convolucional YOLOv4 já treinada** e **pronta para ser ativada em arquivos de imagem e vídeo**, e tão logo os arquivos de saída das ativações de fato.

Com isso, temos encerrada a **Fase 5 - Ativação da rede**, tal como disponibilizado no **toy problem**.

ATIVAÇÃO DAS REDES CONVOLUCIONAIS

AVALIANDO OS RESULTADOS

Tendo as imagens e vídeos resultantes da ativação, é pertinente avaliar seus resultados.

É importante considerar que os vídeos gerados **têm uma frequência de 20 quadros por segundo**. A maioria das capturas de vídeo são em 30 ou 60 quadros por segundo, e a taxa de 20 FPS foi arbitrariamente escolhida para **facilitar as análises** da ativação, não dizendo respeito ao desempenho da rede de forma alguma; **esse híperparâmetro pode ser modificado livremente**.

A avaliação da classificação considera a seguinte métrica:

Classe discriminada:	Incorreta	Incorreta	Correta	Correta
Taxa de confiança:	100%	50%	50%	100%

A avaliação da detecção é mais natural: se detecta algum objeto onde ele deveria estar e essa detecção é estável entre quadros, então a detecção é satisfatória.

INDO ALÉM - QUAIS OS PRÓXIMOS PASSOS

PRÓXIMOS PASSOS

PLUS ULTRA

EIS O QUE VOS ESPERA AMANHÃ:

- **TEORIA:** o que são redes neurais e redes convolucionais, como funcionam e por que funcionam.

- **CÓDIGO:** análise dos códigos de ativação e entendimento do algoritmo.

- **INTRODUÇÃO AO OPENCV:** computação gráfica e dados vetoriais na visão computacional.

O QUE VOCÊ PODE FAZER ATÉ AMANHÃ:

Até a tarde de amanhã é possível reunir e rotular dados, organizar o ambiente e **treinar sua primeira rede convolucional** para aplicações de visão computacional.

Se você não quiser se aprofundar tanto (ou mesmo não tem com o que se aprofundar), **sinta-se livre para brincar com o toy problem**.

VÁ ALÉM

PERGUNTAS?

WORCAP2021
Workshop em Computação Aplicada

COMPUTAÇÃO APLICADA

INPE
UNIDADE DE PESQUISA DO MCTI

MUITO OBRIGADO

AMANHÃ TEM MAIS

Contato: rafael.andrade23@fatec.sp.gov.br
LAC - sala 14.

WORCAP 2021 – XXI Workshop em Computação Aplicada do Instituto Nacional de Pesquisas Espaciais

INTRODUÇÃO À VISÃO COMPUTACIONAL

DIA 2

RAFAEL MARINHO DE ANDRADE

17 de setembro de 2021

WORCAP2021
Workshop em Computação Aplicada

COMPUTAÇÃO APLICADA

INPE
UNIDADE DE PESQUISA DO MCTI

AGENDA

DIA 2

1) Teoria, história e esclarecimentos;
   1.1) Redes neurais;
   1.2) Redes convolucionais;
2) O código para ativação da YOLO;
3) Introdução ao OpenCV;
4) Discussão.

MATERIAL DE APOIO

NÃO FIQUE DE MÃOS VAZIAS

Para os que se aventurarem a acompanhar com as mãos na massa:

SLIDES DA AULA PASSADA:
https://drive.google.com/file/d/1p0y6G0TdDesooiFmvPdhrYxUOJQtNNXr/view?usp=sharing

TOY PROBLEM:
https://drive.google.com/drive/folders/1G3YWORqVE_ESF6mT_W4oWSPe1Kl9O3c0?usp=sharing

NOTEBOOK NO COLAB:
https://colab.research.google.com/drive/1WqAdqP6JczFDvg4IcDhjpmcE-2kQHOMe?usp=sharing

REPOSITÓRIO CONVTRAINING (GITHUB):
https://github.com/Rafael-Marinho/ConvTraining

TEORIA, HISTÓRIA E ESCLARECIMENTOS

(NÃO NECESSARIAMENTE NESSA ORDEM)

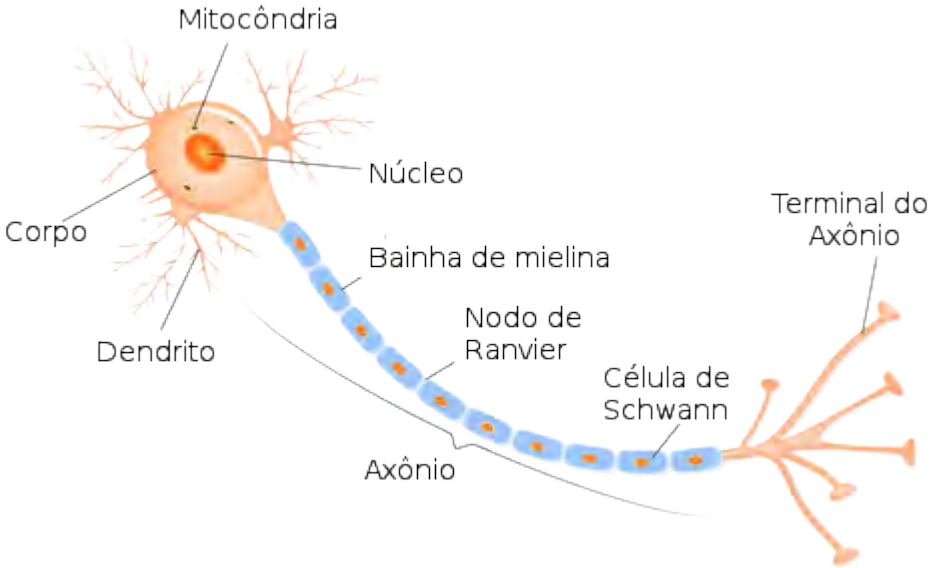
A COMEÇAR PELAS

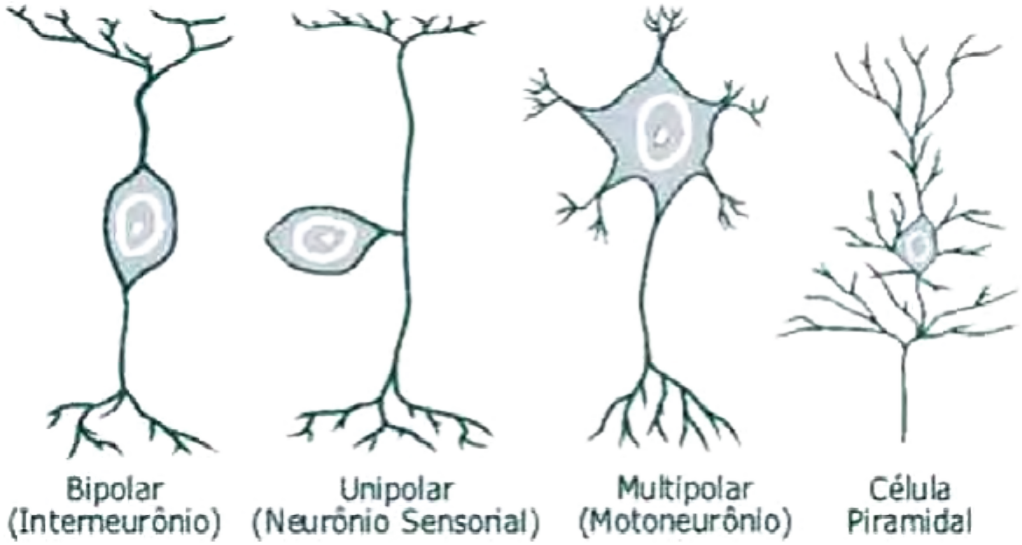
REDES NEURAIS

REDES NEURAIS

A NEUROCOMPUTAÇÃO

As redes neurais compreendem uma escola da inteligência artificial chamada **conexionismo** e abrange uma área conhecida como **neurocomputação**. É inspirado nos sistemas neurais, compreendendo uma área híbrida entre a computação (primordialmente matemática) e neurociência.



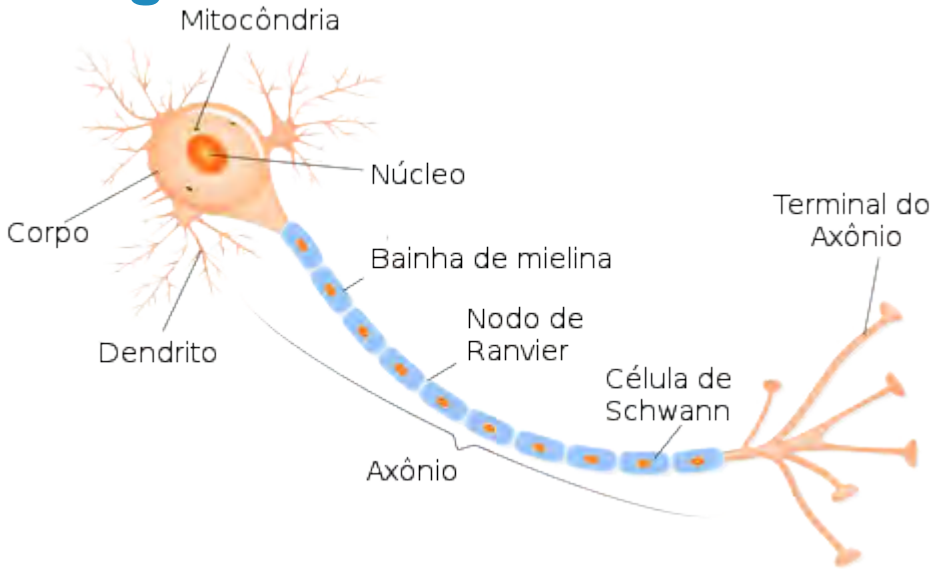


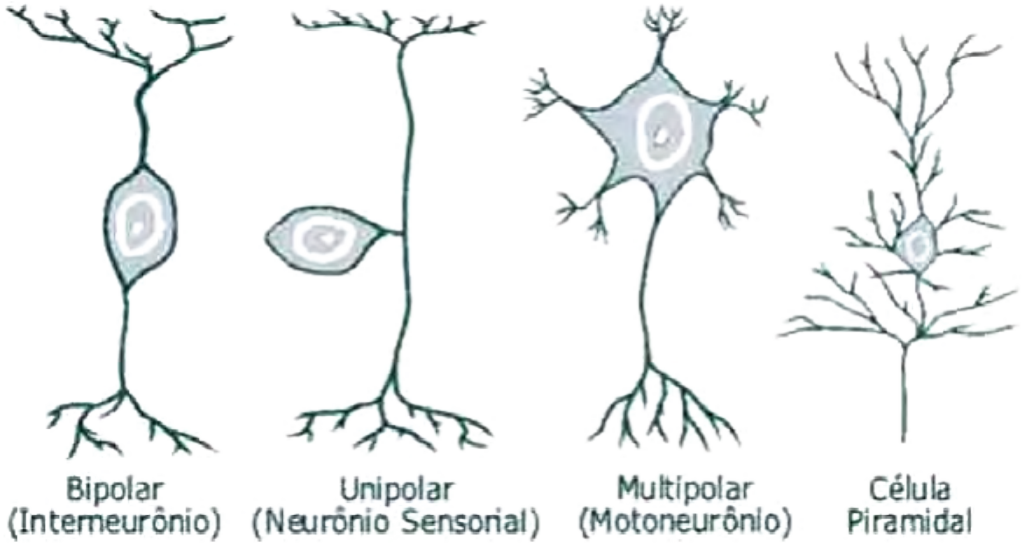
REDES NEURAIS

A IDÉIA

Baseia-se na ideia de **reproduzir matematicamente um cérebro** a partir de sua estrutura neural, assim como todo o resto do sistema nervoso que nos permite perceber o mundo que nos circunda.

Ainda que a história da neurociência remonte tempos imemoriais, ela começou a tomar formato no século XIX. **A neurocomputação só viria a surgir no século seguinte**.





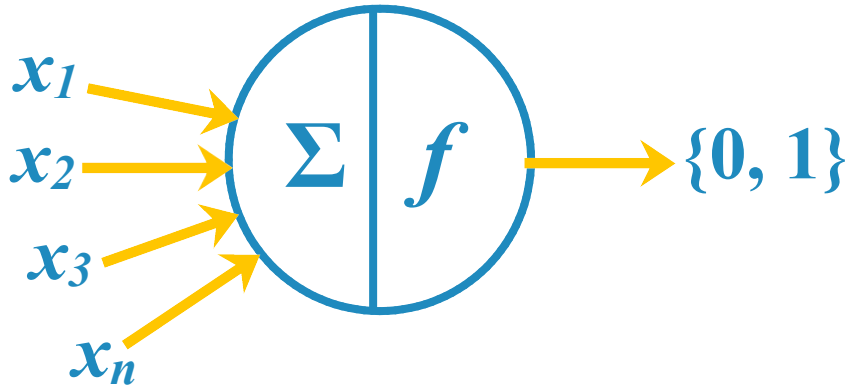
REDES NEURAIS

O NEURÔNIO DE MCCULLOCH E PITTS

As redes neurais se tornaram possíveis quando, em **1943**, **Warren McCulloch** e **Walter Pitts** definiram matematicamente um neurônio:

$$\sum_{i=1}^{n} x_i$$

Essa modelagem pode ser também representada pelo seguinte diagrama:

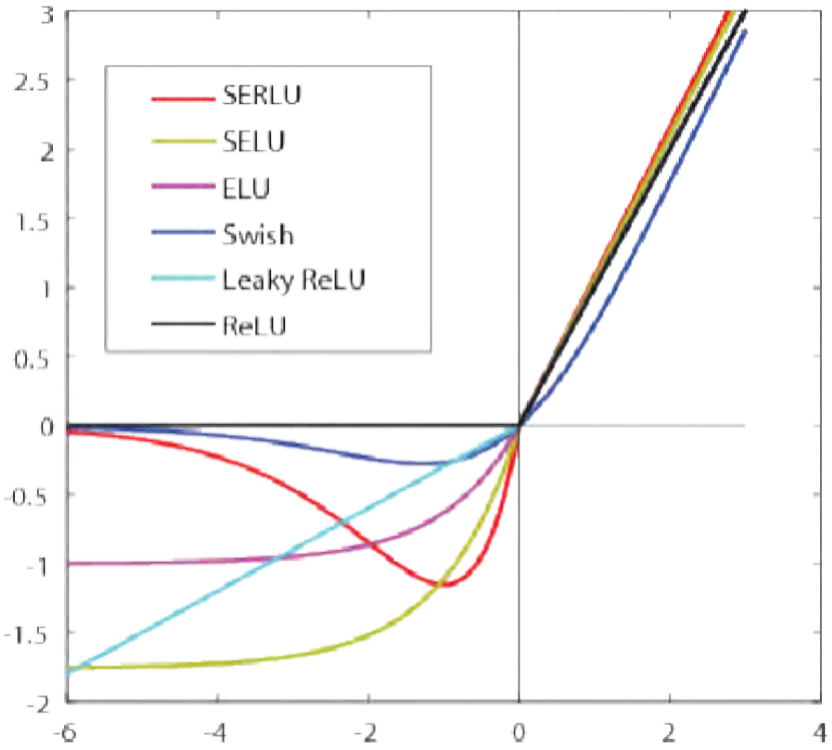


onde x é um estímulo inserido no neurônio e f é a função de ativação aplicada para formar a resposta neural.

REDES NEURAIS

FUNÇÃO DE ATIVAÇÃO

A **função de ativação** do neurônio pode ser literalmente **qualquer função matematicamente concebível**. Com isso, a função mais adequada varia de acordo com diversos fatores, para **convergir ao propósito da rede**, tanto para do ponto de vista micro (neurônio) quanto para o macro (rede inteira).



Um exemplo simples e muitas vezes prático de função de ativação é a **função degrau**, que retorna um valor binário de acordo com a relação da soma dos estímulos e um limiar: **se houver estímulos suficientes, o neurônio é ativado**.

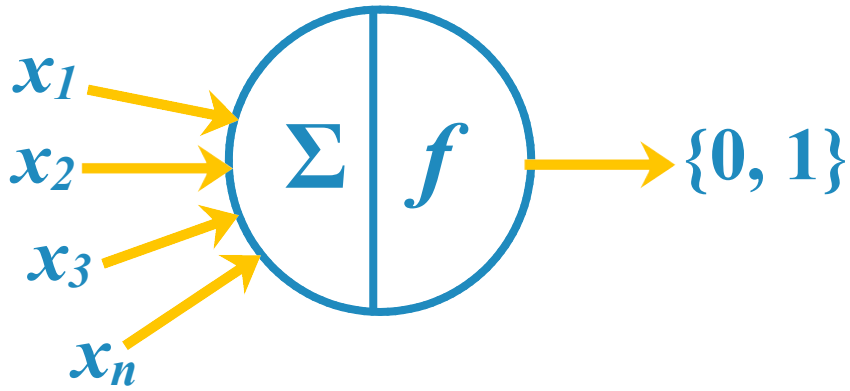
REDES NEURAIS

PRINCÍPIOS DE FUNCIONAMENTO

Ainda que tenham sido desenvolvidas metodologias diversas de redes neurais, **os princípios do funcionamento de qualquer rede neural são os mesmos princípios do neurônio de McCulloch e Pitts**:

haverão sempre

- entradas;
- conexões;
- funções de ativação; e
- saída.

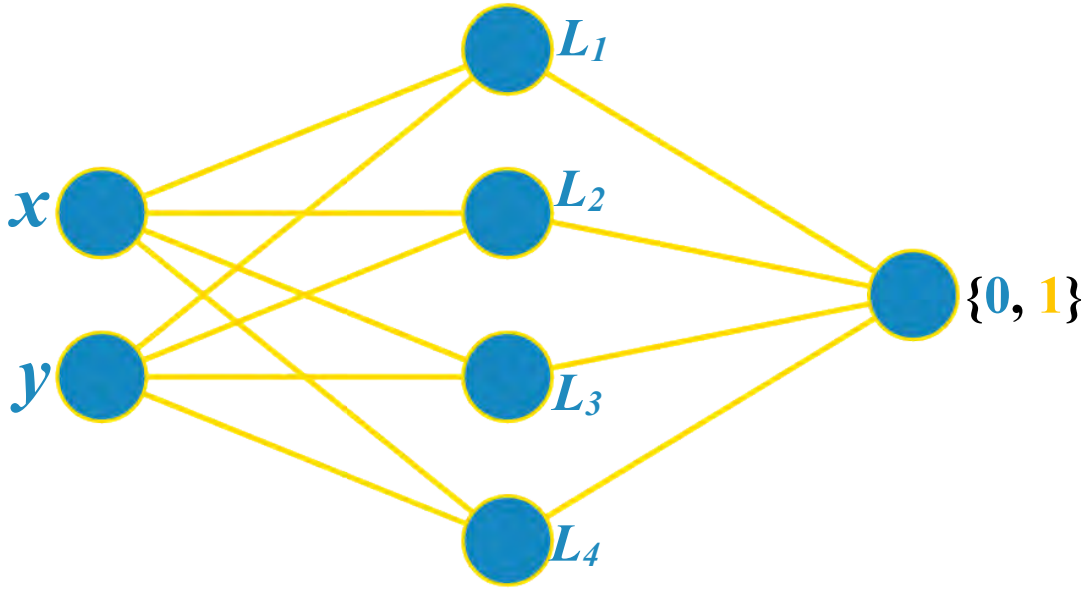


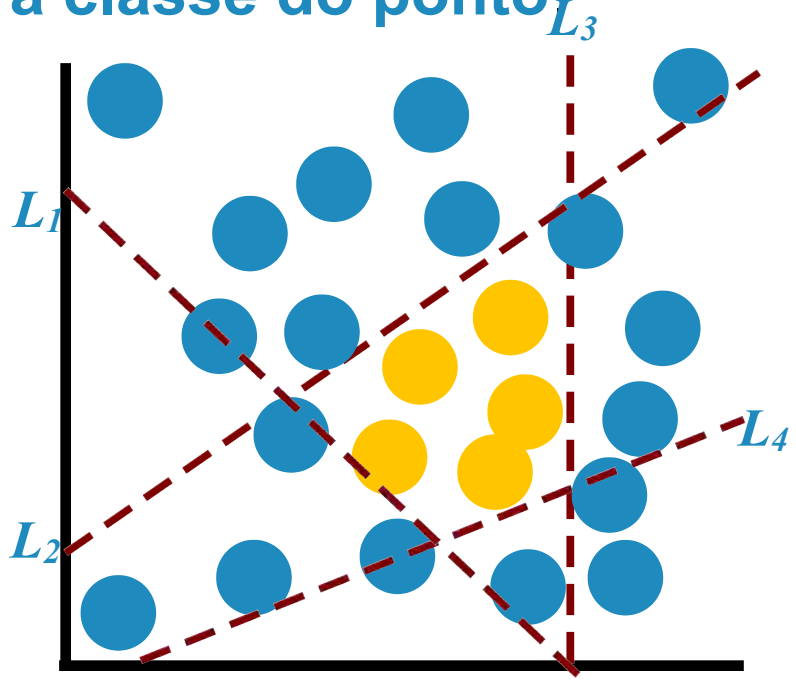
As entradas são os **atributos** do problema; as conexões são as **operações** quais esses atributos são submetidos; as funções de ativação são exatamente quais os **tipos de operação** que as entradas são submetidas; e a saída é o **resultado** que deve ser fornecido (predição).

REDES NEURAIS

PRINCÍPIOS DE FUNCIONAMENTO

No problema abaixo, por exemplo, **são inseridos como entrada as coordenadas** de cada ponto do conjunto; **as conexões levam essas coordenadas a neurônios que identificam sua relação com classificadores lineares**; e a **saída retorna um valor que identifique a classe do ponto**



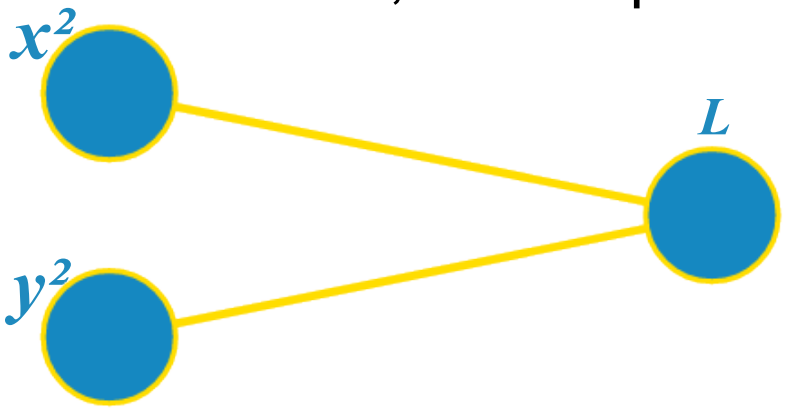


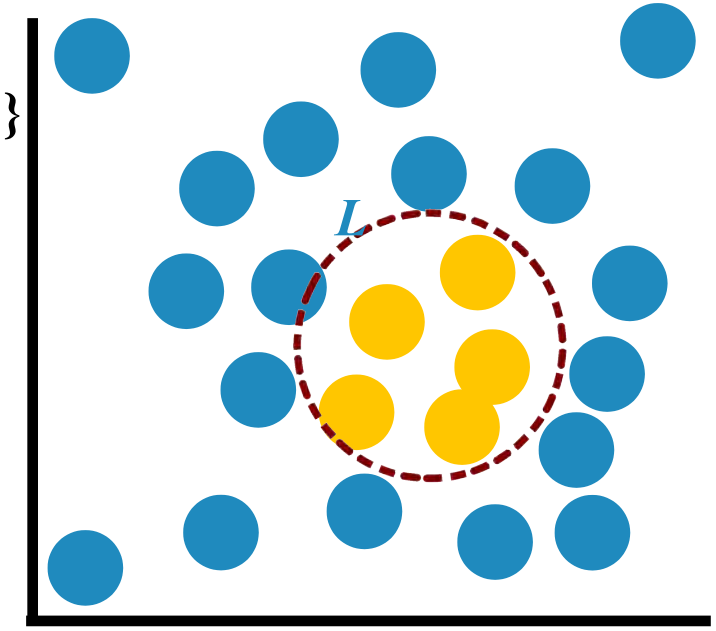
REDES NEURAIS

PRINCÍPIOS DE FUNCIONAMENTO

A modelagem da rede deve levar em consideração as propriedades do problema: **uma mesma solução pode ser atingida com abordagens variadas**, que variam, por suas vezes, em simplicidade e robustez.

A abordagem ideal é a mais simples a prover uma solução minimamente robusta.





Naturalmente, **as soluções devem apresentar um grau de adequação equilibrado**, sendo não muito justo (overfitting) e nem muito genérico (underfitting).

REDES NEURAIS

Mas até aí, o neurônio artificial de McCulloch e Pitts **era apenas um modelo de como um neurônio e, por extrapolação, uma rede neural funciona**. Demoraria mais uma década e meia até a coisa ganhar tração. O que possibilitou isso, no entanto, é fundamentado ainda de 1949 quando **Donald Hebb evidenciou que os estímulos em conexões neurais são fortificados quando exercitados**.

Em resumo, a descoberta de Hebb pôs luz ao fato de que **as conexões neurais são ponderáveis** e, portando, treinadas a partir da experiência. Cativado pela descoberta de Hebb, **Frank Rosenblatt** deu origem ao **perceptron** em **1957**:

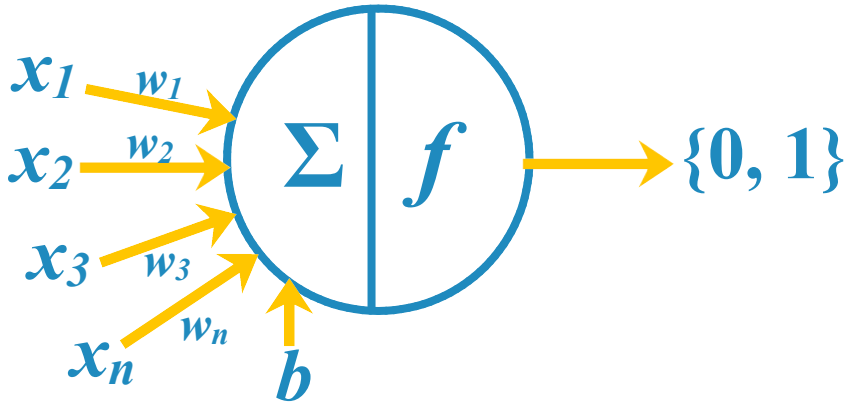
$$\sum_{i=1}^{n}(x_i w_i + b)$$

REDES NEURAIS

O PERCEPTRON DE ROSENBLATT

O **perceptron de Rosenblatt** é praticamente uma evolução do neurônio de McCulloch e Pitts, onde foram adicionados pesos para cada conexão neural (w) e um valor de viés (b).

$$\sum_{i=1}^{n}(x_i w_i + b)$$



Logo, **cada conexão neural era ponderada** (como sugerido por Hebb) e o neurônio **tem embutido um valor basal**, de forma que uma resposta neural seria possível mesmo sem nenhum estímulo inserido no neurônio.

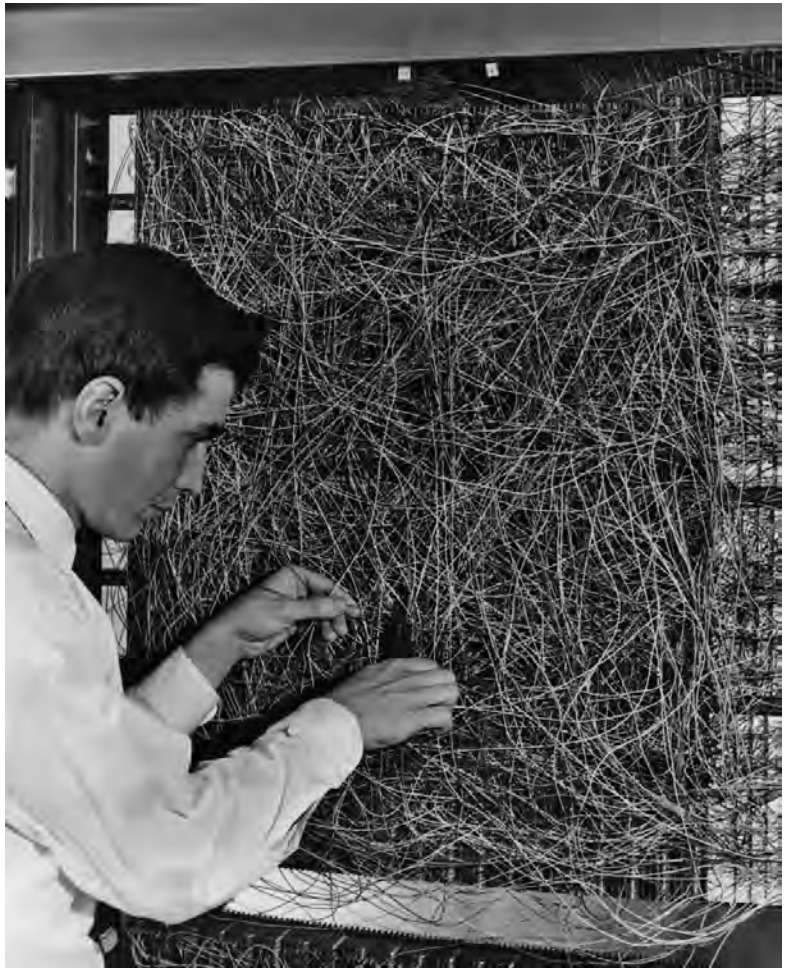
REDES NEURAIS

O PERCEPTRON DE ROSENBLATT

Em outras palavras, **o perceptron podia ser treinado**. Essa idéia logo desencadeou toda uma cultura de pesquisa e desenvolvimento em torno da neurocomputação.

Ainda em 1957, Frank Rosenblatt **implementou um cérebro eletrônico em hardware** onde os pesos iniciais eram definidos por potenciômetros, enquanto os ajuste desses pesos eram realizados por motores elétricos.

(Isso vai fazer sentido mais tarde)

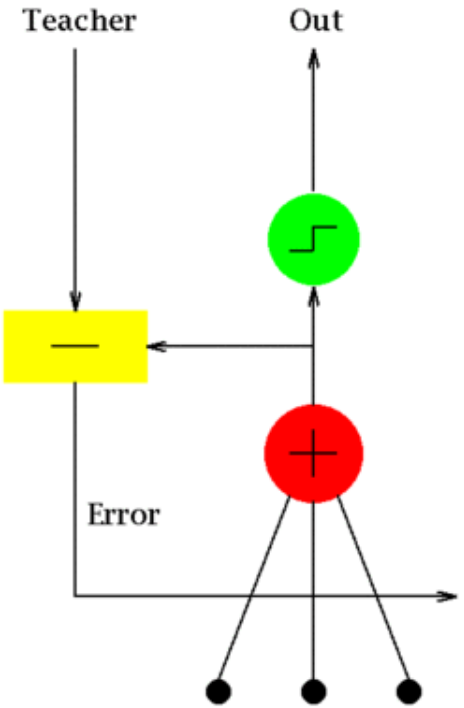


Frank Rosenblatt e o Mark-1 Perceptron, 1957

REDES NEURAIS

APRENDIZADO SUPERVISIONADO

Ainda em 1959, **Bernard Widrow e Marcian Hoff desenvolveram os modelos ADALINE e MADALINE**, objetivamente para serem aplicados em problemas práticos e reais. Respectivamente, tais termos são siglas para ADAptive LINear Element, e Multilayer ADALINE.
Esses modelos são a consolidação da técnica de **aprendizado supervisionado que é praticada até hoje**.



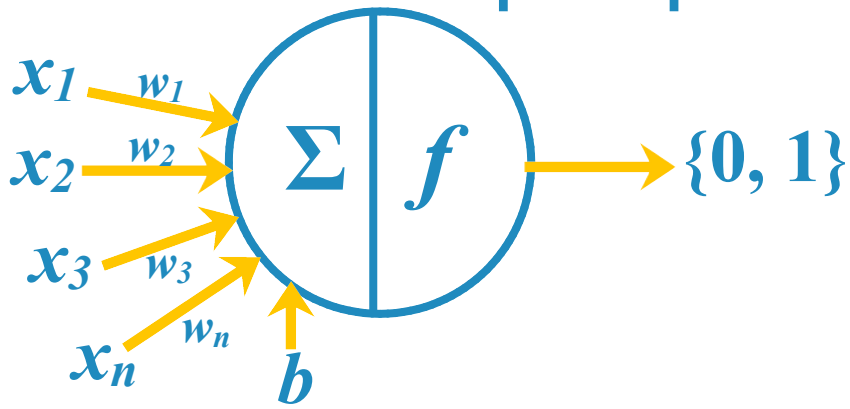
Na década de 1960, após anos de experimentação e desenvolvimento dos fundamentos, as redes neurais **passaram a ser experimentados em problemas do mundo real**.

PRINCÍPIOS DO TREINAMENTO

APRENDIZADO SUPERVISIONADO

Assim como o princípio do funcionamento não mudou desde o neurônio de McCulloch e Pitts, **o princípio de treinamento de todas as redes neurais são**, em síntese, **o mesmo princípio do treinamento do perceptron de Rosenblatt**.

O treinamento, por sua vez, **nada mais é do que o processo de ajuste dos pesos**.

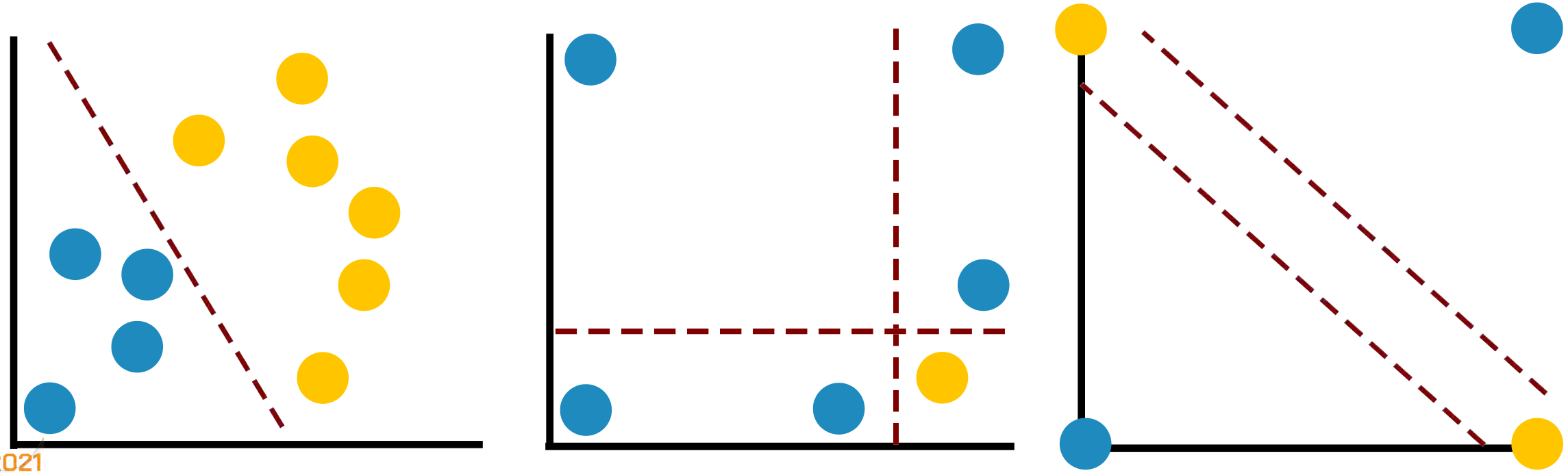


A definição da **arquitetura da rede**, as **funções de ativação**, **valores de viés** e quais atributos são utilizados como **entrada**, por via de regra, **não são abordados durante o treinamento da rede**, sendo estaticamente definidos antes do treinamento. Existem também, no entanto, abordagens sofisticadas para de remodelar a rede durante o treinamento.

REDES NEURAIS

LIMITAÇÕES

Apesar da crescente cultura em torno do perceptron de Rosenblatt, ganhou fama em 1969 a **restrição matemática dos perceptrons à problemas linearmente separáveis**. Ou seja, um perceptron não é capaz de resolver problemas de disjunção exclusiva, considerado matematicamente simples.



REDES NEURAIS

INVERNO DAS REDES NEURAIS

Dessa limitação simples, veio a crença de que, por mera extrapolação, **implementações mais complexas de perceptrons, mesmo em rede, não contornariam essa limitação**. Apesar de presunçoso, **essa crença ganhou popularidade** e deu a entender que a neurocomputação não seria aplicável em cenários práticos, **tornando o perceptron menos desejável**.

Assim, **o paradigma conexionista foi condenado à estagnação durante uma década inteira**.

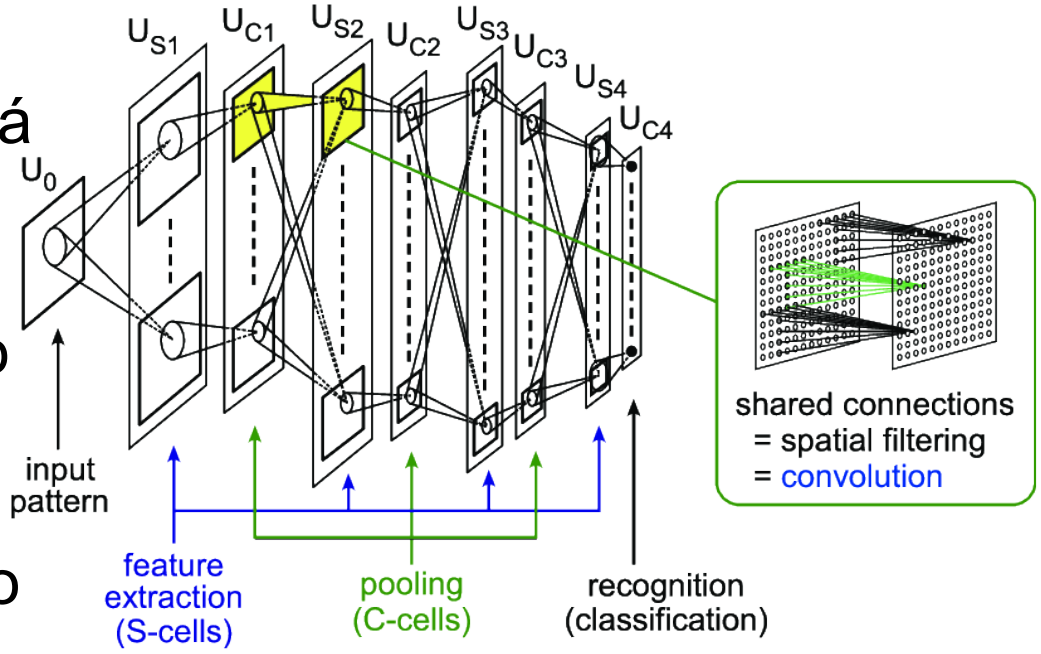
Esse período (década de 1970) ficou conhecido como
Inverno das Redes Neurais.

REDES NEURAIS

O NOVO COMEÇO

A década de 1980 iniciou-se com um novo começo para as redes neurais quando **Kunihiko Fukushima deu origem ao neocognitron**, uma rede neural multicamadas.

O neocognitron foi **desenvolvido para a detecção de caracteres manuscritos**, que já era visto desde então como uma aplicação bem prática para as redes neurais, rendendo mais fôlego e reformando a esperança para o paradigma conexionista.



Mais do que apenas isso, o neocognitron foi o **precursor das redes convolucionais**.

ENQUANTO ISSO...

(POR TRÁS DA FACHADA DESTA APARENTEMENTE INOCENTE LIVRARIA)

SURGIAM OS ALICERCES DAS

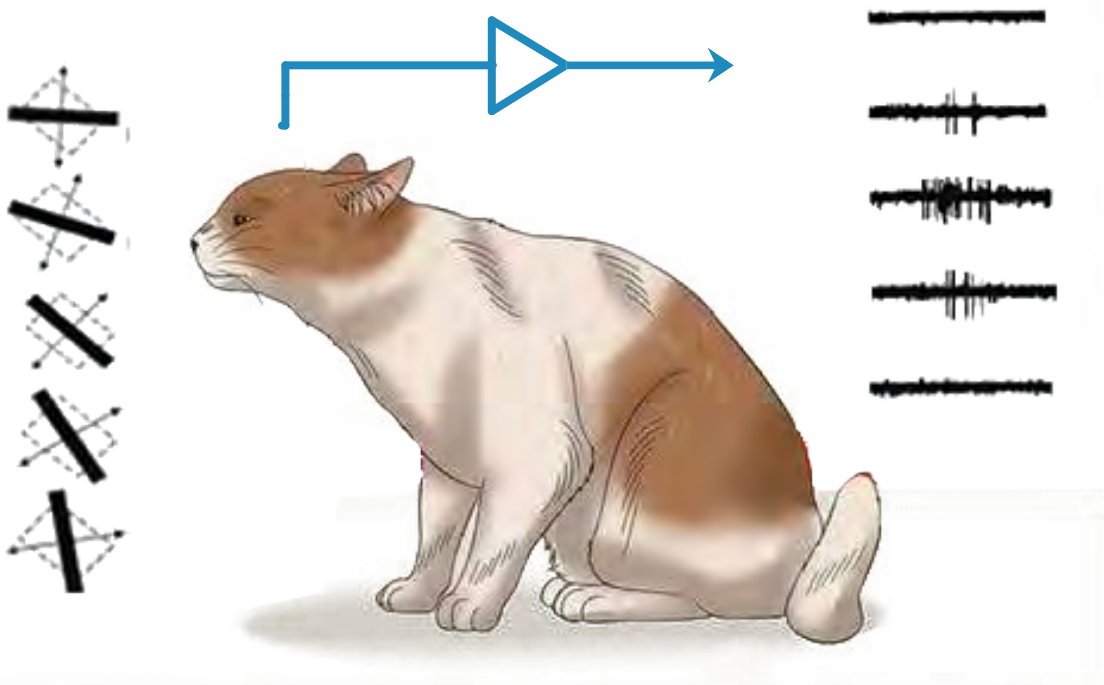
REDES CONVOLUCIONAIS

REDES CONVOLUCIONAIS

O EXPERIMENTO DE HUBEL E WIESEL

Durante a década de 1950, os experimentos de **David Hubel** e **Torstein Wiesel** demarcaram os alicerces da visão computacional de fato, em especial um com um gato em **1959**. Nesse experimento, foram ligados eletrôdos ao córtex visual de **um gato que foi submetido a uma apresentação de slides** com padrões visuais diversos.

A resposta neural do gato foi enviada para um osciloscópio. Finalmente, foi denotado que **a resposta neural segue padrões de acordo com o padrão visual observado**: alguns neurônios são especialmente estimulados por determinadas feições em específico.



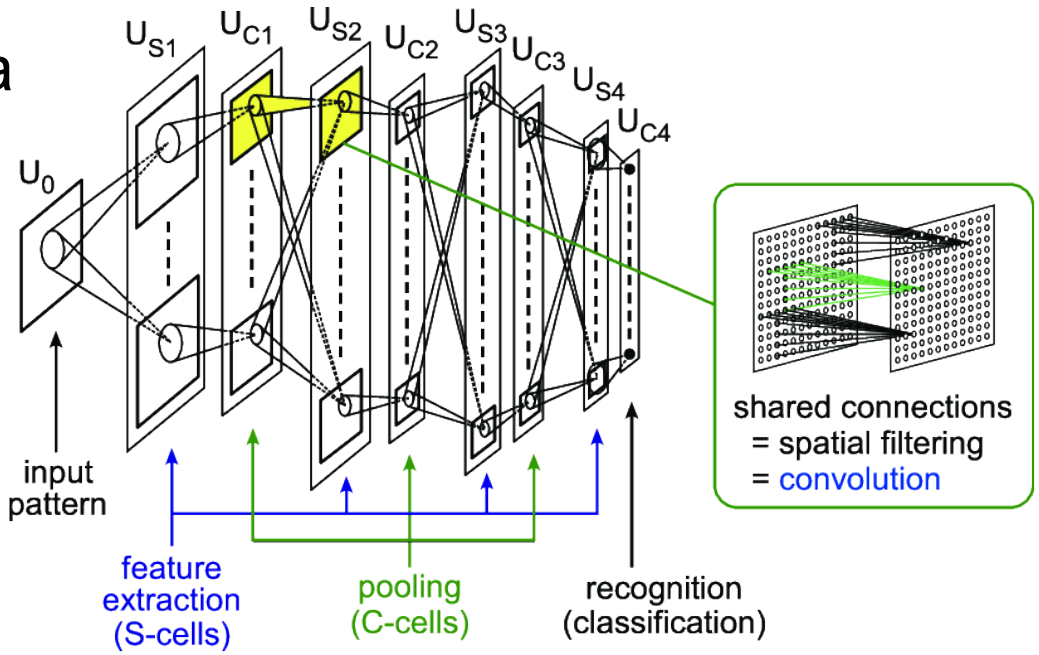
REDES CONVOLUCIONAIS

O NEOCOGNITRON DE FUKUSHIMA

Foi inspirado por Hubel e Wiesel que **Kunihiko Fukushima deu origem ao neocognitron**, no que diz respeito à resposta neural para detectar padrões visuais.

Parte do vislumbre da solução de Fukushima veio a ele como uma **varredura**, que surgiu naturalmente ao considerar que **os padrões interessantes tendem a estar em um local específico da imagem**.

A **convolução**, por sua vez, cumpre o papel da varredura e foi uma solução **para que as feições mais marcantes fossem filtradas**.



REDES CONVOLUCIONAIS

NA ÉPOCA DE ROSENBLATT

Parte da inspiração também remonta ao **Mark-1 Perceptron**: Frank Rosenblatt e sua equipe a implementaram com uma câmera **matriz 20x20 de fotossensores** de sulfeto de cádmio (CdS).

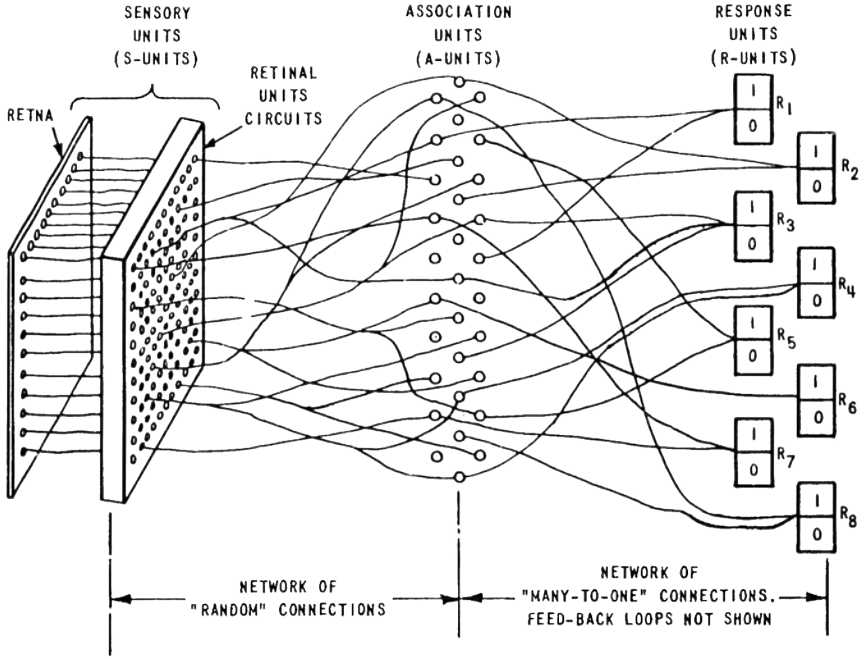
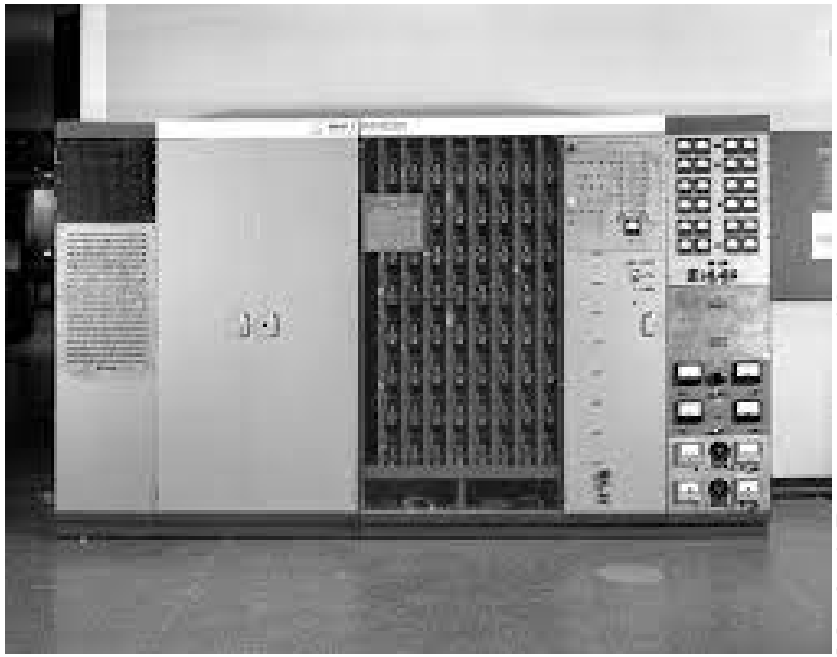


Figure I ORGANIZATION OF THE MARK I PERCEPTRON



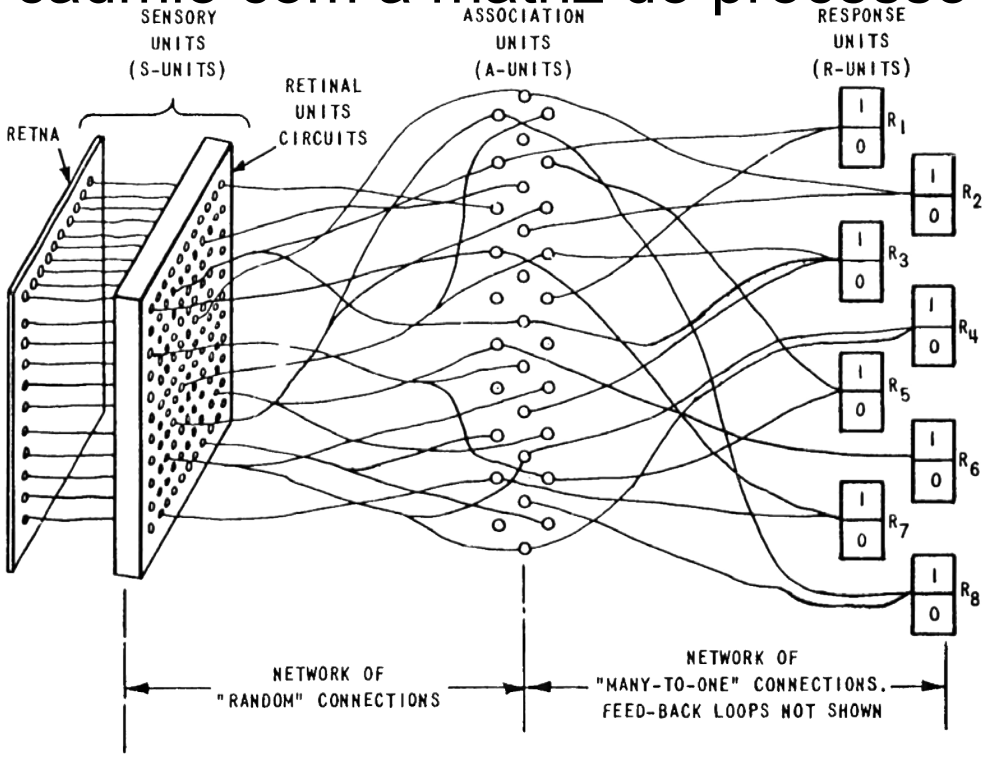
Mark-1 Perceptron, 1957

A ideia era que padrões visuais fossem empregados ao perceptron, para **reconhecimento de imagens**. A implementação orientada à aleatoriedade, no entanto, tornou essa ideia impraticável.

REDES CONVOLUCIONAIS

INSPIRAÇÃO NO MARK-1 PERCEPTRON

A inspiração do neocognitron nessa premissa frustrada do Mark-1 Perceptron **fica evidente ao se comparar os diagramas** dessa matriz de sulfeto de cádmio com a matriz do processo de convolução da rede.



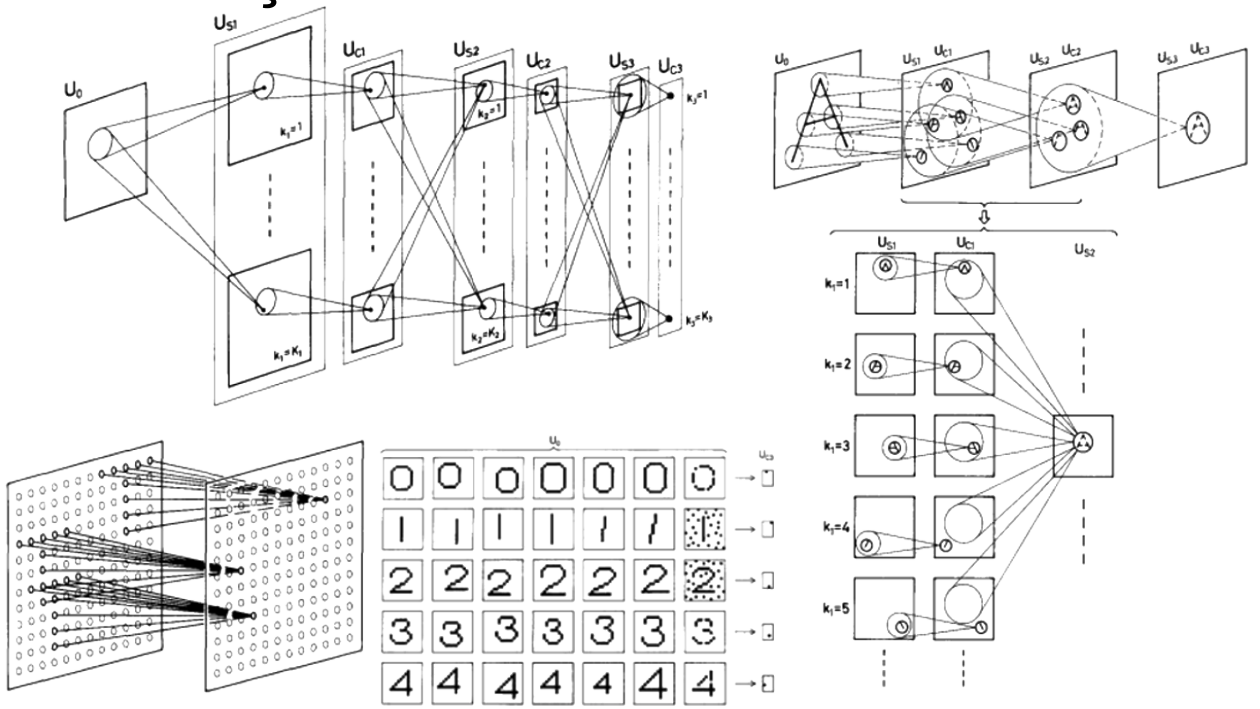


Figure 1 ORGANIZATION OF THE MARK I PERCEPTRON

REDES CONVOLUCIONAIS

O ano da virada para as redes convolucionais foi 1986:

- Inspirado pelo neocognitron, **Yann LeCun apresentou a precursora da LeNet, uma rede convolucional para visão computacional**, aprimorando o neocognitron; e
- ao mesmo tempo, **Waibel, Hanazawa, Hinton, Shikano e Lang apresentaram a rede neural convolucional de atraso temporal**, abreviada como TDNN, **para processamento de linguagem natural**.

Além disso, nesse mesmo ano, **Rumelhart, Hinton e Williams apresentaram o método de aprendizagem por retropropagação**, que alavancou o treinamento das redes multicamadas.

REDES CONVOLUCIONAIS

A NOVA ERA

Ainda assim, 1986 é mais responsável por plantar as sementes da grandiosidade que estava por vir; **os primeiros frutos viriam apenas após dez anos**. Mais especificamente em 1998, com a **LeNet-5** de Yann LeCun.

Com a aurora do século XXI, o paradigma conexionista passou a resolver diversas classes de problemas utilizando principalmente abordagens de **redes de perceptrons multicamadas**, **atualizações da LSTM** e **variações da LeNet** -- embora não se limitando a apenas essas três abordagens.

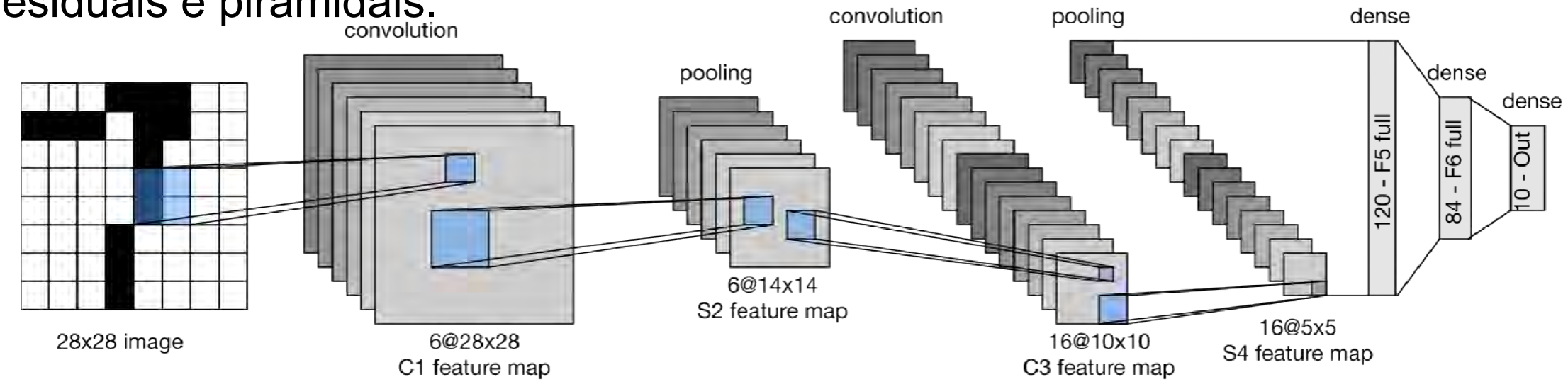
Destaque para o fato de que, desde então, **praticamente todas as grandes redes convolucionais de sucesso são baseadas na LeNet-5**.

REDES CONVOLUCIONAIS

O QUE É UMA REDE CONVOLUCIONAL

Enfim, uma rede convolucional é uma rede neural artificial **caracterizada pela presença de camadas de convolução**. As redes convolucionais para imagens fazem uso de **pooling**, e também se faz necessária a presença de **ao menos uma camada densa** (totalmente conectada) ao fim da rede.

Há também outros tipos de camada aplicáveis e mais inventivas, como as residuais e piramidais.



REDES CONVOLUCIONAIS

O QUE É UMA REDE CONVOLUCIONAL

As redes convolucionais **podem ser aplicadas tanto em vetores unidimensionais** (como é o caso do áudio) **quanto multidimensionais**. No caso de imagens, **as entradas costumam ter duas ou três dimensões**, que podem ser ou não discretizadas ou estratificadas em dimensionalidade fixa.

192	168	0	255	72	43	88	90
188	170	24	255	240	58	90	92
160	177	220	249	243	67	102	100
155	172	230	243	240	80	120	108
152	168	172	220	232	112	132	119
142	140	168	190	199	150	168	122
100	139	153	177	188	177	180	123
80	120	148	165	179	184	200	123

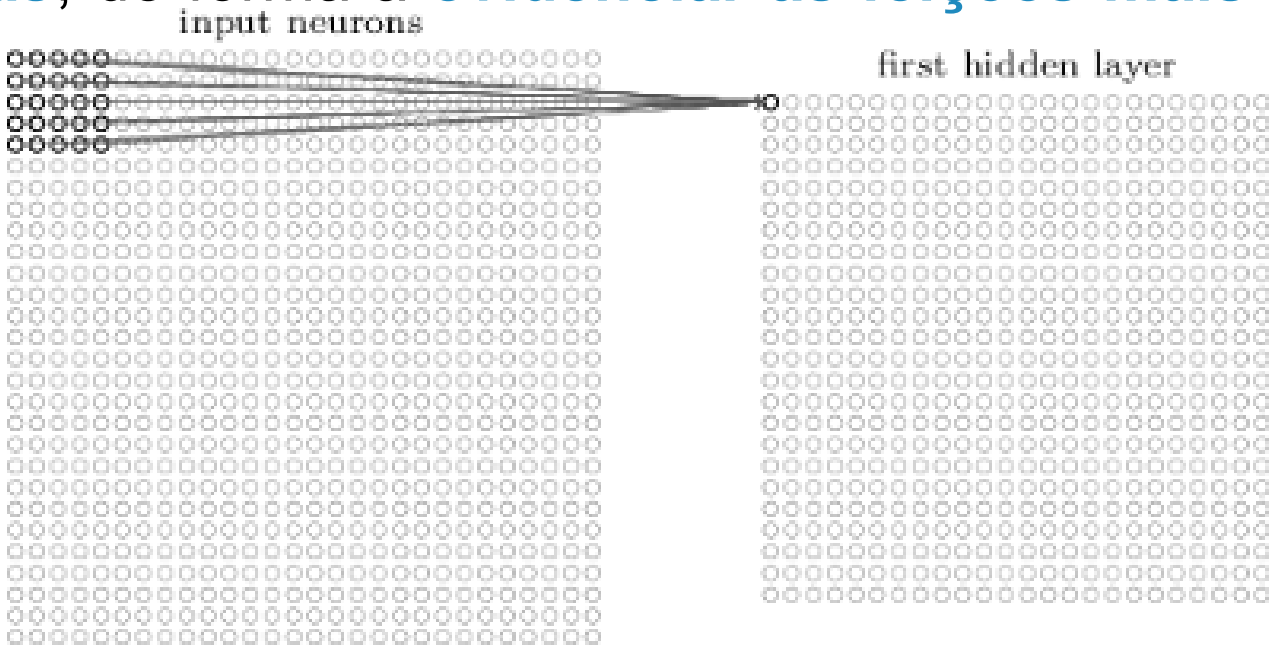
73	76	82	82	81	82	80	24
72	74	80	77	79	79	78	24
70	71	78	202	210	212	213	213
65	68	70	198	207	209	209	210
58	59	172	188	198	205	209	208
42	48	168	190	199	200	202	207
33	40	153	177	188	200	201	204
19	120	148	165	179	184	200	203

52	53	56	60	70	82	88	99
63	64	67	71	81	93	99	110
85	86	89	93	103	115	111	122
118	119	122	126	136	148	144	155
120	121	124	128	138	150	146	157
140	141	144	148	158	170	166	177
145	156	149	153	163	175	171	181
195	196	199	203	213	225	221	231

REDES CONVOLUCIONAIS

O QUE É CONVOLUÇÃO

A **convolução** é um processo de **transformação matricial** muito versátil aplicado, por exemplo, em análises de sinais como operações de funções lineares. É aplicada nas redes convolucionais como um **método de filtragem para a extração das características**, de forma a **evidenciar as feições mais relevantes**. Funciona como uma **varredura de filtros**.



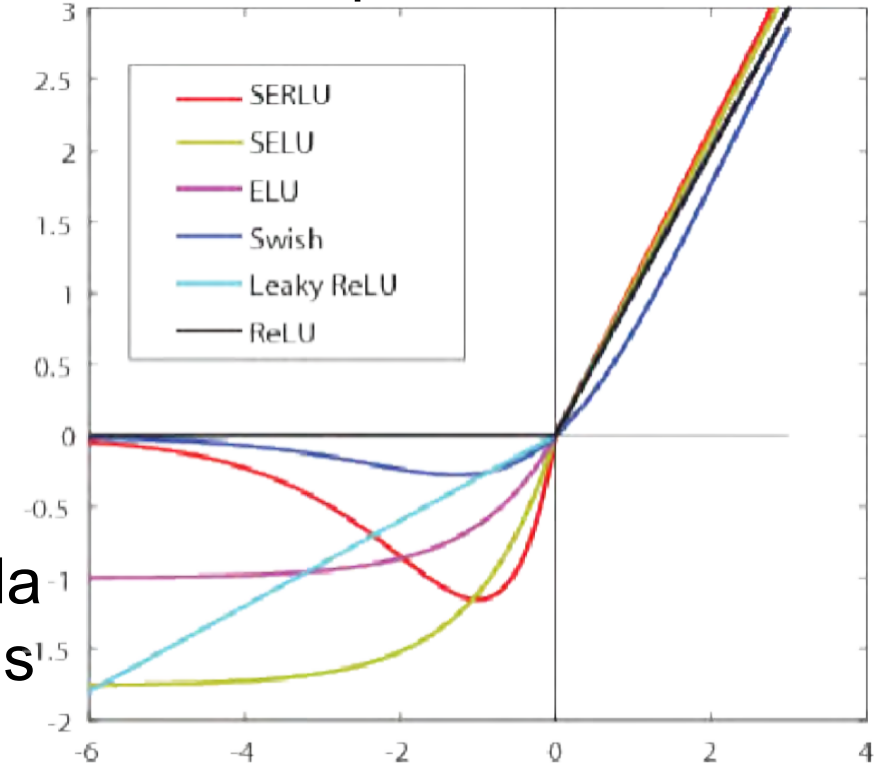
Cada convolução gera uma subcamada na camada posterior, e a quantidade de convoluções é baseada na área de convolução e no valor de saltos na varredura.

REDES CONVOLUCIONAIS

FUNÇÕES DE ATIVAÇÃO

Assim como em qualquer neurônio artificial, **é necessária a aplicação de uma função de ativação e a convolução é justamente isso** aplicado de forma matricial: **os valores dispostos na matriz são transformados a partir da função de ativação**.

A unidade retificadora linear (**ReLU**) **é a função mais comum em redes convolucionais para visão computacional**, mas funções como a unidade exponencial escalar linear (SELU), ReLU vazada, Swish e Mish também recebem atenção da comunidade, não apenas na aplicação em imagens como também em demais aplicações concebíveis.

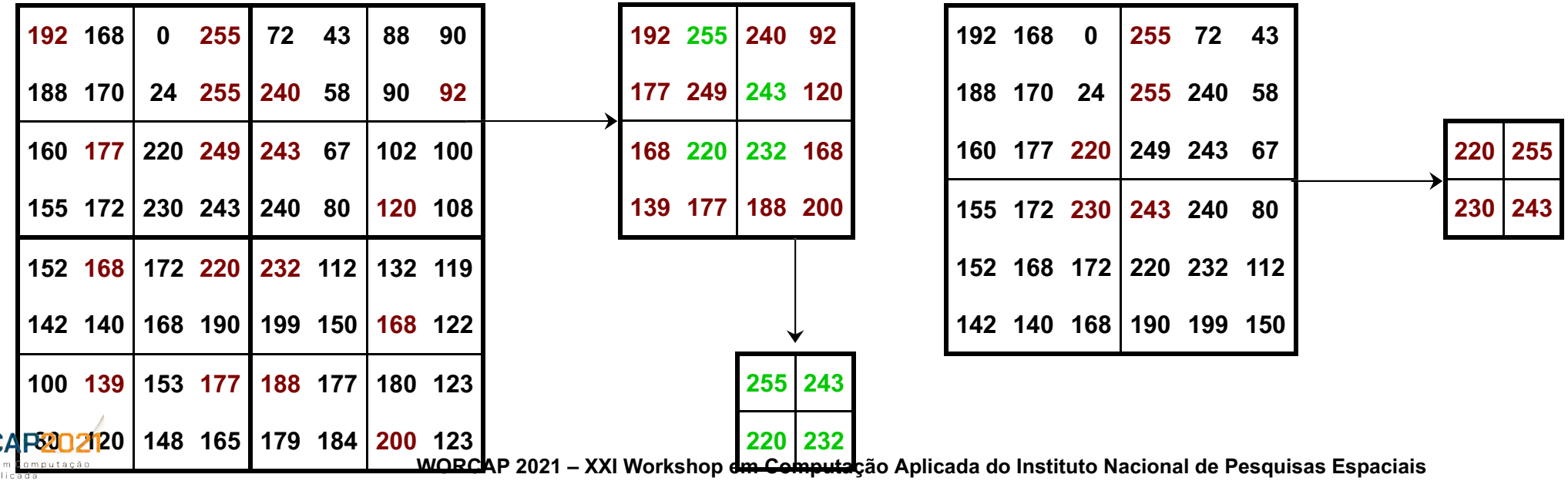


REDES CONVOLUCIONAIS

O QUE É POOLING

O **pooling** é um processo de **simplificação de dados**, inclusive **reduzindo a quantidade de dados trabalhados**, baseado em uma técnica de sumarização que **reduz a quantidade de pesos a serem aprendidos**. Também **ajuda a evitar overfitting**.

Diversas técnicas podem ser aplicadas, das quais se destacam a **média**, **soma**, **produto** e **argumento máximo ou mínimo**. O **MaxPooling faz uso do argumento máximo e é o mais empregado em visão computacional**.

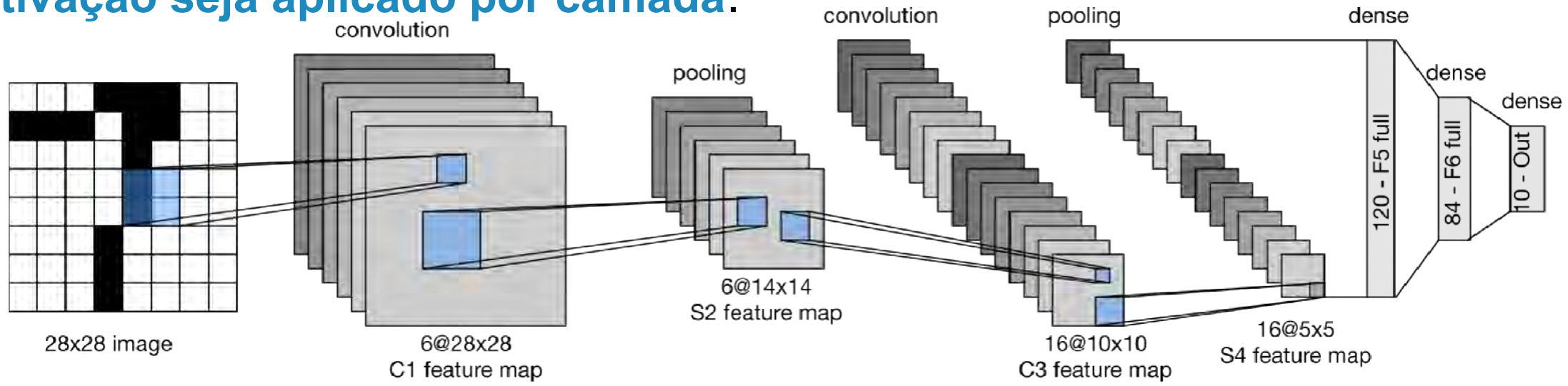


REDES CONVOLUCIONAIS

CAMADAS DE SAÍDA E AFINS

A saída de qualquer rede convolucional é antecipada por uma **camada densa**, onde todos os seus neurônios são ligados a todos os neurônios da camada anterior. As saídas da rede são as classes discrimináveis.

Além disso, é comum e recomendado que **apenas um tipo de função de ativação seja aplicado por camada**.



REDES CONVOLUCIONAIS

PROFUNDIDADE

Outro conceito interessante é a **profundidade**, em que as rede convolucionais divergem das redes multiperceptron convencionais onde 4 camadas são suficiente para separar uma rede rasa de uma profunda.

A **LeNet-5 possui 7 camadas e é considerada uma rede convolucional rasa**, enquanto a **AlexNet possui 8 camadas e é considerada uma rede profunda**. A principal diferença entre essas abordagens, no entanto, é a **quantidade de parâmetros**: a LeNet-5 possui 3246 parâmetros, enquanto a AlexNet tem mais de 70000000 parâmetros.

O conceito de profundidade em uma rede convolucional é, portanto, mais referente à relação entre profundidade e largura, à sua **complexidade**.

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

A detecção e classificação de objetos tende a ser realizada em conjunto. Essa detecção **parte do princípio de uma varredura** (o que é convergente às redes convolucionais), com granularidade progressiva ou regressiva, e a classificação ocorre a cada stride da varredura, de forma que seja **mensurado um valor de probabilidade para cada classe em cada inferência**.

O retorno da rede convolucional (e também outras abordagens aplicáveis) costuma converger a uma estrutura contendo as **coordenadas para localização do objeto** e valores correspondentes **às probabilidades do objeto detectado pertencer a determinadas classes**, sendo um valor para cada classe.

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (**245**, **389**), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de **456** pixels e **altura** de **402** pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, **0.3%**, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, **2%,** 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, **97%**, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, **39%** e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

[473, 590, [0.003, 0.02, 0.97, 0.39, 0.21]]

ou **[245, 389, 701, 791, [0.003, 0.02, 0.97, 0.39, 0.21]]**

ou **[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]**

REDES CONVOLUCIONAIS

DETECÇÃO E CLASSIFICAÇÃO

As coordenadas para localização podem assumir formas diversas, podendo ser um valor de abscissas e ordenadas (coordenada do epicentro do objeto), um par de coordenadas (dois pontos, limites das caixas delimitadoras), coodenada do ponto de origem e altura e largura (um ponto e mais dois valores individuais), etc.

Por exemplo, em um sistema para detectar animais, com cinco diferentes classes, e for detectado um animal nas **coordenadas** (245, 389), com **largura** de 456 pixels e **altura** de 402 pixels, e as **probabilidades para cada classe** forem, respectivamente, 0.3%, 2%, 97%, 39% e 21%, **a estrutura pode ser**:

A REDE YOLO USA ESSA ESTRUTURA

[245, 389, 456, 402, [0.003, 0.02, 0.97, 0.39, 0.21]]

REDES CONVOLUCIONAIS

ESTADO DE INOVAÇÃO

Hoje as redes neurais são populares na sociedade em geral, sendo um conceito cada vez mais conhecido e praticado por classes cada vez mais amplas de pessoas -- **não apenas na área de tecnologia**, consolidando-se como inovação.

Na década de 2010, o **amadurecimento das pesquisas** e **recursos cada vez mais robustos e acessíveis** permitiram a criação de sistemas baseados em redes neurais artificiais capazes de atingir **resultados sobrehumanos em classes cada vez mais diversas de problemas**, e tais sistemas estão pouco a pouco se tornando **acessíveis às massas**.
Não obstante, as redes convolucionais e a visão computacional vêm ganhando terreno e **os últimos 15 anos ditam a maior parte dessa história**.

REDES CONVOLUCIONAIS

AMADURECIMENTO TECNOLÓGICO

A partir da **década de 2000**, começaram a surgir uma série de **redes convolucionais baseadas na LeNet-5**, se aproveitando de **recursos suficientemente robustos**. A história da visão computacional no século XXI passaria a se confundir com a história das redes convolucionais.

O principal alicerce dos avanços da visão computacional passou a ser as **GPGPUs** e **programação paralela**. O pontapé inicial foi a obra de **Chellapilla**, **Puri** e **Simard** em **2006**, que contou com uma **variação da LeNet otimizada para GPGPU** e atingiu resultados **quatro vezes mais ágeis** que em abordagens convencionais.

Essa abordagem logo se tornou uma tendência.

REDES CONVOLUCIONAIS

A ERA DAS REDES CONVOLUCIONAIS PROFUNDAS

As abordagens que aplicam variações da LeNet em GPGPUs e multithreading logo chegou à decada de 2010, quando **a visão computacional entrou em sua era de ouro com as redes convolucionais profundas**. Os pioneiros foram **Cireşan e sua equipe**, que atingiram em 2011 pela primeira vez **resultados sobrehumanos na tarefa de identificação de caracteres manuscritos**, além de um desempenho robusto em classificações mais complexas.

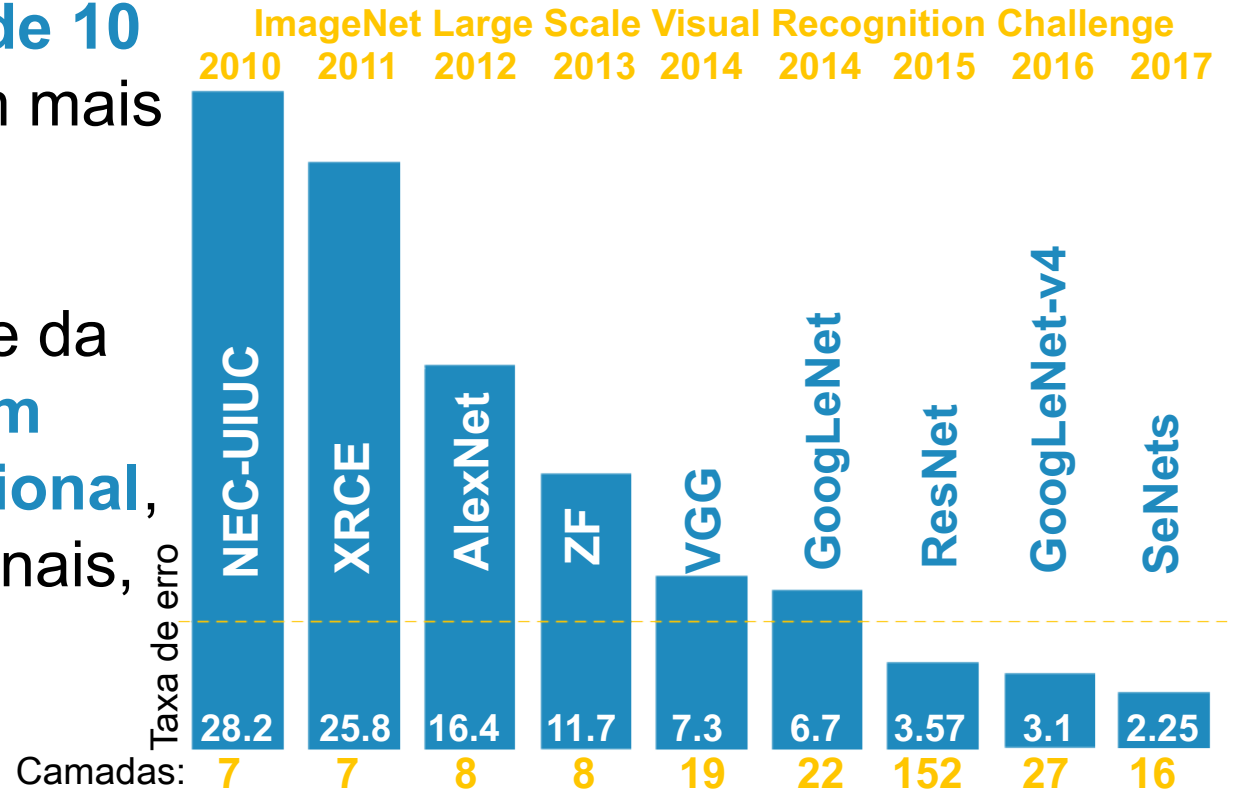
Mas foi a **AlexNet** de **Alex Krizhevsky** que marcou o início dessa era de ouro, em **2011**.

REDES CONVOLUCIONAIS

A ERA DAS REDES CONVOLUCIONAIS PROFUNDAS

Em **2015**, pela primeira vez, um sistema de visão computacional atingiu **resultados sobrehumanos** na tarefa de detectar e discriminar objetos na **LSVRC**, um **benchmark com mais de 10 milhões de imagens** distribuídas em mais de 10 mil categorias.

A partir de então, da segunda metade da década de 2010, **recursos que dizem respeito à área de visão computacional**, ao menos para aplicações convencionais, **passaram a se tornar cotidianos**.



REDES CONVOLUCIONAIS

ESTAGNAÇÃO OU CONSOLIDAÇÃO

Já **no final da década**, os desempenhos de tantas abordagens de referência já beiravam os 100% em benchmarks diversos para detecção e discriminação que **considera-se ter chegado em um ponto de estagnação por excelência**.

A visão computacional hoje se dedica a abordar tarefas mais complexas e condições mais desafiadoras. Portanto, estagnado ou não, o amadurecimento das abordagens de discriminação e classificação servem como **base para abordar tarefas mais complexas**.

Muitos acreditam, no entanto, que os novos grandes avanços só serão novamente possíveis com uma nova mudança de paradigmas, mas isso **não significa que as pesquisas também chegaram a um ponto de inflexão**.

VOLTANDO À PRÁTICA...

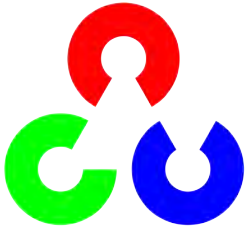
(COM UMA PITADA DE TEORIA)

ALGORITMO DE ATIVAÇÃO DA REDE YOLO

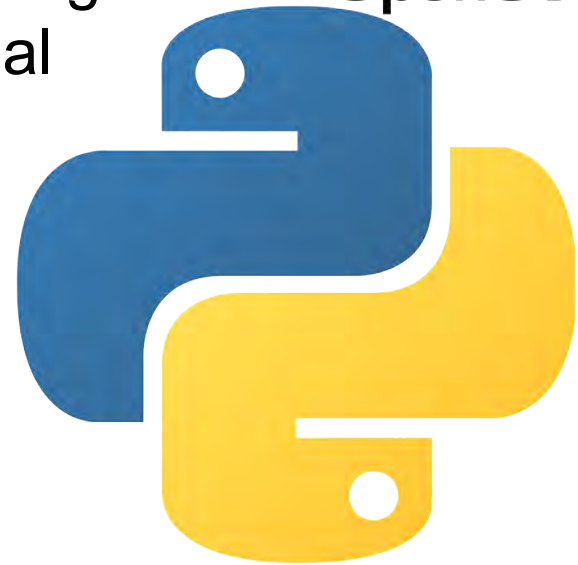
ALGORITMO DE ATIVAÇÃO

CONTEXTUALIZAÇÃO

Depois de fazer tudo funcionar, é hora de ver em mais detalhes **como tudo funciona** e obter ainda mais compreensão sobre **por que funciona**.



O conceito de “ativação” da rede consiste em colocá-la para funcionar em uma aplicação. O algoritmo aqui praticado confia na linguagem de programação **Python** e no framework de visão computacional **OpenCV**.



A versão do Python deve ser Python3 e o OpenCV deve ser 4.4 ou superior.

NOTA: o algoritmo coberto a seguir é o disponível no GitHub.

ALGORITMO DE ATIVAÇÃO

LI E ACEITO OS TERMOS DE USO

AVISO 1: nos códigos disponibilizados no **GitHub** e **Colab**, as funções do OpenCV utilizadas são **chamadas uma a uma** para melhor otimização de desempenho (em especial, memória). **Nestes slides**, o OpenCV é chamado **integralmente** e suas funções apresentam o prefixo `cv2` justamente para tornar mais claro quais as funções do código pertencem ao OpenCV.

```
import cv2
```

SLIDES

```
from cv2 import ...
```

GITHUB / COLAB

AVISO 2: apenas os algoritmos de ativação em uma única imagem e em vídeo serão explorados, uma vez que o código de ativação em todas as imagens do diretório input é **literalmente o mesmo algoritmo** de ativação aplicado em uma única imagem, porém dentro de um loop onde cada iteração toma como quadro uma imagem de tal diretório e sem disposição das imagens resultantes.

ALGORITMO DE ATIVAÇÃO

INICIALIZAÇÃO

0- PRIMEIRO SÃO CARREGADAS AS BIBLIOTECAS:

```
import cv2                                  ← OpenCV*
from numpy import argmax                    ← Extração de argumento máximo (NumPy).
```

1.1- DEPOIS, É CARREGADA A REDE CONVOLUCIONAL:

```
net = cv2.dnn.readNet("networks/model.weights", "networks/model.cfg")
```

Pesos da CNN — Arquitetura da CNN

Arquivo de classes ↓

```
with open("networks/model.names", 'r') as f:    ← Abre o arquivo de classes.
    classes = f.read().splitlines()             ← Extrai as classes, segmentando-as por linha.
```

ALGORITMO DE ATIVAÇÃO

1.2- SÃO CARREGADAS AS IMAGENS:

SE FOR IMAGEM:

```
img = cv2.imread("input.png")
```

Extrator de imagem — Arquivo de imagem

SE FOR VÍDEO:

Ponteiro de leitura — Arquivo de vídeo

```
cap = cv2.VideoCapture("input.mp4")
out = cv2.VideoWriter("output.avi", cv2.VideoWriter_fourcc(*"mp4v"), 20,(int(cap.get(3)), int(cap.get(4))))
```

Ponteiro de escrita — Nome do arquivo — Codec — MP4 — Taxa de FPS — Altura — Largura

(SE FOR VÍDEO) INICIA A VARREDURA DAS IMAGENS:

```
while (1 < 2):          ← Inicia o loop.*
    _, img = cap.read()  ← Extrai o quadro do vídeo.
```

* se tratando de vídeo, a partir desse ponto, as instruções serão inclusas dentro desse loop.

ALGORITMO DE ATIVAÇÃO

DEFINIÇÃO DE VARIÁVEIS E PONTEIROS

1.3- EXTRAI E DEFINE VARIÁVEIS E PONTEIROS:

SE FOR IMAGEM:

```
height, width, _ = img.shape
```

Altura — Largura — Dimensões

SE FOR VÍDEO:

```
try:                                  ← Detecta se há fluxo de vídeo.
    height, width, _ = img.shape
except(AttributeError):               ← Se não houver mais
    break                               imagens, encerra o loop.
```

Imagem de entrada ↓ — Dimensões da rede ↓ — Mantém as proporções ↓

```
net.setInput(cv2.dnn.blobFromImage(img, (1 / 255), (415, 416), (0, 0, 0), swapRB=True, crop=False))
```

- Insere a imagem na CNN
- Converte a imagem em um BLOB*.
- Fator de escala
- Escalar de média
- Transforma BGR em RGB

```
boxes = []          ← Inicia a lista de caixas delimitadoras das detecções.
confidences = []    ← Inicia a lista de taxas de confiança das detecções.
class_ids = []      ← Inicia a lista de identificadores de classe das detecções.
```

* BLOB é um acrônimo para Binary Large Object.

ALGORITMO DE ATIVAÇÃO

DETECÇÃO DOS OBJETOS DE INTERESSE

2- REALIZA AS DETECÇÕES:

Varredura feed-forward ↓ — Extrai os nomes das camadas de saída da CNN ↓

```
for output in net.forward(net.getUnconnectedOutLayersNames()):   ← Realiza a convolução na imagem.
    for detection in output:                                      ← Realiza uma varredura nas detecções.
        scores = detection[5:]                                    ← Extrai os valores de confiança das detecções.
        class_id = argmax(scores)                                 ← Identifica a classe mais provável para cada detecção.
        confidence = scores[class_id]                             ← Extrai valor de confiança da classe mais provável.
        if (confidence > 0.5):                                    ← Filtra as detecções em um limiar de 50% de confiança.
            center_x = int(detection[0] * width)                  ← Extrai a posição da detecção nas abscissas.
            center_y = int(detection[1] * height)                 ← Extrai a posição da detecção nas ordenadas
            w = int(detection[2] * width)                         ← Define limite direito da caixa delimitadora.
            h = int(detection[3] * height)                        ← Define limite do topo da caixa delimitadora.
            x = int(center_x - (w / 2))                           ← Define limite esquerdo da caixa delimitadora.
            y = int(center_y - (h / 2))                           ← Define limite do fundo da caixa delimitadora.
            boxes.append([x, y, w, h])                            ← Armazena as coordenadas da caixa delimitadora.
            confidences.append(float(confidence))                 ← Armazena a taxa de confiança da detecção.
            class_ids.append(class_id)                            ← Armazena a classe da detecção.
```

ALGORITMO DE ATIVAÇÃO

DESENHANDO AS BOUNDING BOXES

Supressão Não-Máxima | Limiar de confiança | Limiar de supressão **3- RENDERIZA AS DETECÇÕES:**

```
indexes = dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)      ← Converge instâncias redundantes.
if (len(indexes) > 0):                                    ← Filtra as detecções válidas.
    for i in indexes.flatten():                           ← Varre as detecções, após uma estratificação.
        x, y, w, h = boxes[i]                             ← Extrai as coordenadas da caixa delimitadora.
        label = str(classes[class_ids[i]])                ← Extrai a classe da detecção.
        confidence = str(round(confidences[i], 2))        ← Extrai o a taxa de confiança da detecção.
        cv2.rectangle(img,                                ← Imagem onde o retângulo é desenhado.
Desenha o                 (x, y),                         ← Coordenada do ponto delimitador máximo.
retângulo                 ((x + w), (y + h)),             ← Coordenada do ponto delimitador mínimo.
da caixa                  (0, 0, 200),                    ← Cor das linhas (BGR).
delimitadora              2)                              ← Espessura da linha (em píxels).
        cv2.putText(img,                                  ← Imagem onde o texto é escrito.
Escreve a                 (label + ' ' + confidence),     ← Conteúdo escrito.
classe e a                (x, (y + 20)),                  ← Posição onde o texto é escrito.
taxa de                   cv2.FONT_HERSHEY_PLAIN,         ← Fonte do texto.
confiança                 1,                              ← Tamanho do texto.
da                        (0, 0, 200),                    ← Cor da fonte (BGR).
detecção                  2)                              ← Expessura da fonte.
```

ALGORITMO DE ATIVAÇÃO

APRESENTANDO E ARMAZENANDO AS IMAGENS

4- DISPÕE A IMAGEM COM A DETECÇÃO:

Nome da janela ↓ Imagem disposta ↓

```
cv2.imshow("DATA", img)
```

5- ARMAZENA A IMAGEM EM UM ARQUIVO DE SAÍDA:

SE FOR IMAGEM:

```
cv2.imwrite("output.png", img)

while (1 < 2):
    key = cv2.waitKey()
    if (key == 27):
        cv2.destroyAllWindows()
        break
```

Se for teclado ESC, encerra a execução do algoritmo.

SE FOR VÍDEO:

```
out.write(img)

key = cv2.waitKey(1)
if (key == 27):
    cv2.destroyAllWindows()
    break
```

6- (JÁ FORA DO LOOP DO VÍDEO) ENCERRA OS PONTEIROS DE LEITURA E ESCRITA:

Encerra o ponteiro de leitura → `cap.release()`

Encerra o ponteiro de escrita → `out.release()`

Encerra a aplicação ao fechar a janela aberta → `cv2.destroyAllWindows()`

INDO MAIS AFUNDO

INTRODUÇÃO AO OPENCV

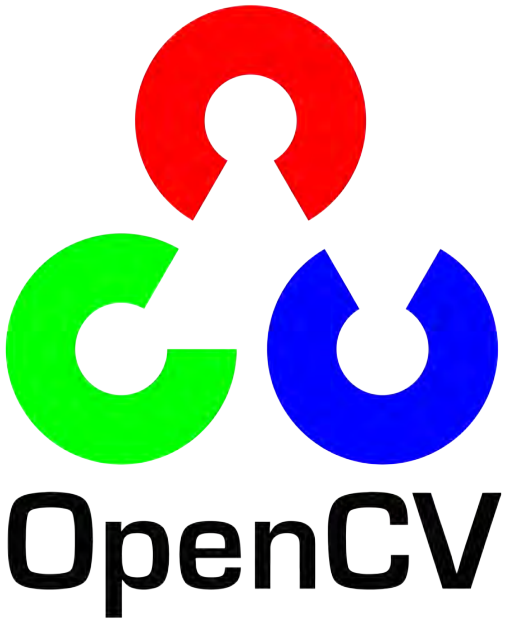
INTRODUÇÃO AO OPENCV

CONTEXTUALIZAÇÃO

Durante a segunda metade do século XX, a visão computacional nasceu, aprendeu a dar seus primeiros passos e, nestes passos curtos e conservadores, formou um alicerce que o sustenta até hoje. Uma das características mais notáveis, no entanto, é que **trazer evoluções à visão computacional era uma tarefa então inerentemente difícil**.

Há um conjunto de fatores que levaram a visão computacional a decolar, e um deles é criação de arcabouços para o desenvolvimento.

O mais proeminente, por sua vez, é o OpenCV.



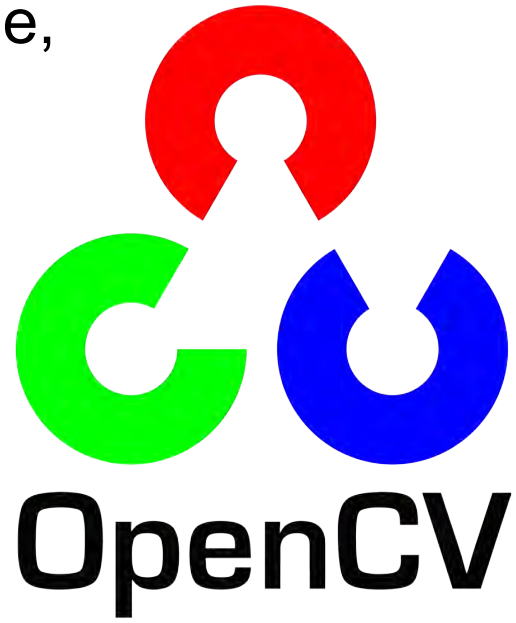
INTRODUÇÃO AO OPENCV

O QUE É O OPENCV

Acrônimo para Open Source Computer Vision Library, trata-se de uma **biblioteca para desenvolvimento de visão computacional**.

Foi desenvolvida, a princípio, como um **projeto da Intel na Rússia no fim da década de 1990**, e logo se tornou um projeto em **código aberto**.
Teve sua primeira versão disponibilizada em meados de 2000 e, após uma "gestação" com anos de alphas e betas, teve sua **versão 1.0 lançada em 2006**.

Se encontra atualmente na versão **4.5.2** (abril de 2021).



INTRODUÇÃO AO OPENCV

O QUE É O OPENCV

O OpenCV foi desenvolvido em **C/C++**, e o código nessa linguagem de programação traz uma série de vantagens. Apesar disso, códigos em C/C++ **não são tão legíveis ou fáceis de manejar** quanto outras disponíveis.

Por conta disso, além de C++, a biblioteca tem interfaces disponíveis também em outras linguagens de programação, das quais se destacam **Python**, **MATLAB**, **Java** e **JavaScript**.

Na medida do possível, o código em C/C++ é mantido sob a interface dessas outras linguagens, o que **garante um ótimo desempenho** mesmo em linguagens de alto nível.

INTRODUÇÃO AO OPENCV

O QUE ESTÁ INCLUSO NO PACOTE

O **OpenCV é uma biblioteca bem extensa e completa**, e se destacam os módulos para:

1) processamento de imagens **[imgproc]**;
2) leitura e escrita de arquivos de imagem e vídeo **[imgcodecs, videoio]**;
3) análise de vídeo **[video]**;
4) calibração de câmera e reconstrução 3D **[calib3d]**;
5) descrição, detecção e categorização de características em 2D **[features2d]**;
6) detecção de objetos **[objdetect]**;
7) redes neurais profundas **[dnn]**;
8) aprendizado de máquina **[ml]**;
9) agrupamento e busca multidimensional **[flann]**;
10) junção de imagens **[stitching]**;
11) fotografia computacional **[photo]**;
12) computação gráfica **[highgui, gapi]**;
13) rastreio de objetos **[tracking]**;
14) detecção e reconhecimento de texto **[text]**;
15) análise de faces **[face]**;
16) superresolução **[dnn_superres]**;

e muitos outros. **Ao todo, são atualmente 70 módulos oficiais, além do principal**.
Pode valer a pena dar uma olhada na documentação.

INTRODUÇÃO AO OPENCV

DOCUMENTAÇÃO

INTRODUÇÃO AO OPENCV

POR ONDE COMEÇAR

Lidando com o OpenCV, portanto, temos **um mar de informações** a respeito de seu aprendizado e possibilidades onde **o céu é o limite**.

Após a instalação*, **pode ser difícil enxergar por onde começar**. Alguns recursos considerados fundamentais e relevantes serão introduzidos, dentre eles:

1) Leitura e escrita;
2) Transformação de imagem;
3) Inferência de informações; e
4) Computação gráfica básica.

*A complexidade envolvida na instalação a levou a ser omitida nessa aula. Os tutoriais de instalação (para Linux, MacOS, Windows e afins) podem ser acessados no último slide.

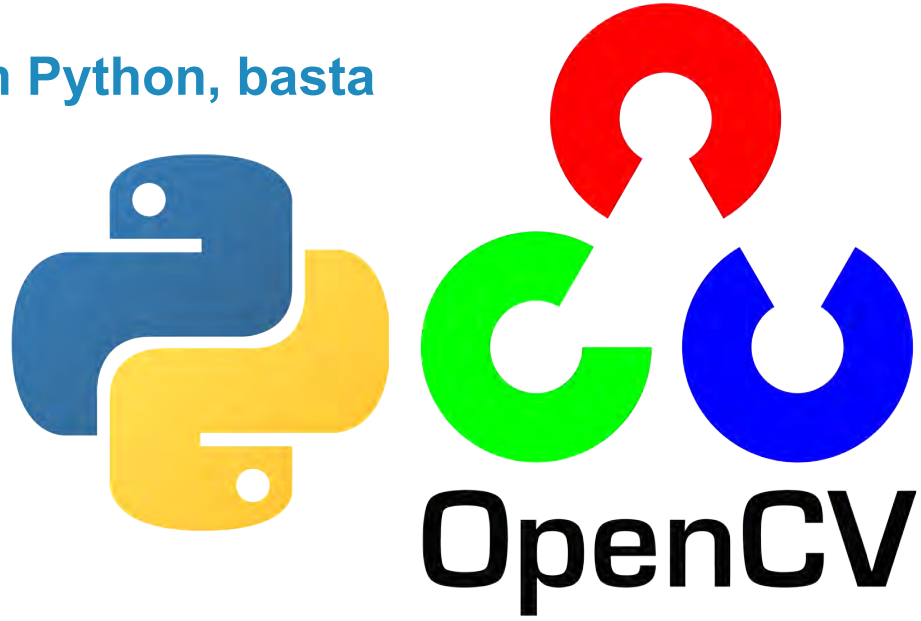
INTRODUÇÃO AO OPENCV

POR ONDE COMEÇAR

Nota: **A documentação do OpenCV não é homgênea entre as linguagens** apresentadas (a maior parte está em C++ e Python, mas há trechos apenas em C++ ou apenas em Python etc). **Todo o código apresentado nessa aula será na linguagem de programação Python**, mas a construção da interface da biblioteca nesta linguagem de programação a confere uma semelhança sintática com o código em C++, de forma que **a leitura da documentação não tenda a ser um problema** mesmo sem a fluência em ambas as linguagens.

De todo o modo, **para carregar a biblioteca OpenCV em Python, basta importá-la** com o seguinte comando:

```
import cv2
```



INTRODUÇÃO AO OPENCV

LEITURA E ESCRITA

- COMO LER IMAGENS

```
img = cv2.imread("input.png")
```

Extrator de imagem — Arquivo de imagem

- COMO INSERIR IMAGENS EM ARQUIVOS

```
cv2.imwrite("output.png", img)
```

Imagem de entrada

Gerador de imagem — Arquivo de saída

Se tratando de imagens individuais, elas **podem ser carregadas individualmente e inseridas em variáveis** como bem se desejar.

INTRODUÇÃO AO OPENCV

LEITURA E ESCRITA

- PONTEIROS DE ESCRITA E LEITURA PARA VÍDEOS

Se tratando de vídeos (grupos de imagens codificados em formato específico), elas **devem ser carregadas e inseridas em um ponteiro de escrita**.

Ponteiro de leitura — Arquivo de vídeo

```
cap = cv2.VideoCapture("input.mp4")
out = cv2.VideoWriter("output.avi", cv2.VideoWriter_fourcc(*"mp4v"), 20,(int(cap.get(3)), int(cap.get(4))))
```

Ponteiro de escrita — Nome do arquivo — Codec — MP4 — Taxa de FPS — Altura — Largura

```
while (1 < 2):
    _, img = cap.read()

    out.write(img)
```

← Inicia o loop.*

← Extrai o quadro do vídeo.

← Insere o quadro do vídeo no arquivo do ponteiro de saída.

* lidando com vídeos, as intruções sobre suas imagens devem ser inclusas dentro desse loop.

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

As imagens carregadas podem ser livremente manipuladas partindo do princípio que as **imagens nada mais são do que matrizes**. Em outras palavras, as imagens são um conjunto de valores matemáticos matricialmente organizados.
O ponto-chave para manipular os dados é entender como são organizadas as matrizes, estas que **possuem vetores de 3 dimensões**:

DIMENSÃO 1: Altura;
DIMENSÃO 2: Largura;
DIMENSÃO 3: Profundidade.

Convencionalmente, cada valor desses vetores é um valor inteiro entre 0 e 255. Esse valor também é denominado **frequência**.

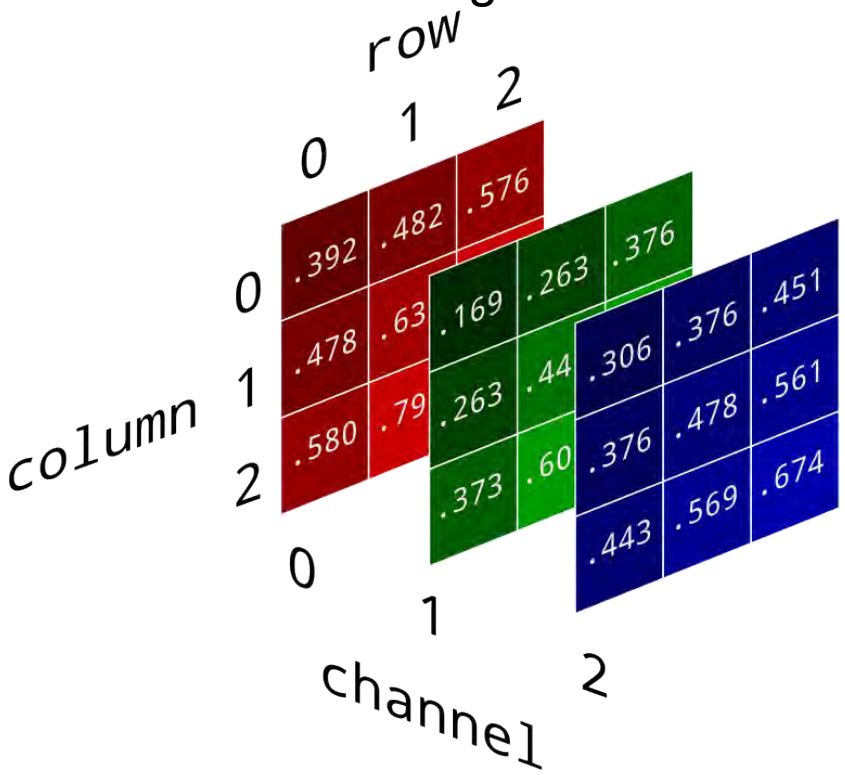
Portanto, em uma imagem BGR com resolução de 1920x1080 possui 1080 vetores contendo 1920 vetores com três valores que são os canais (altura, largura e profundidade, respectivamente). Por exemplo, se for extraído o valor de `img[20][60]` e for retornado um vetor contendo `[180, 240, 3]`, significa que o píxel nas coordenadas cartesianas (60, 20) tem uma frequência de 3, 240 e 180 para as cores vermelho, verde e azul, respectivamente.

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

As imagens carregadas podem ser livremente manipuladas partindo do princípio que as **imagens nada mais são do que matrizes**. Em outras palavras, as imagens são um conjunto de valores matemáticos matricialmente organizados.
O ponto-chave para manipular os dados é entender como são organizadas as matrizes, estas que **possuem vetores de 3 dimensões**:

DIMENSÃO 1: Altura;
DIMENSÃO 2: Largura;
DIMENSÃO 3: Profundidade.



INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- EXTRAÇÃO DE DIMENSÕES:

```
img.shape ← Retorna uma tupla com altura, largura e profundidade.
img.size  ← Retorna a quantidade de píxels (altura X largura X profundidade).
```

Altura e largura são medidos em píxels, enquanto a profundidade é, de fato, a **quantidade de canais**.

- REDIMENSIONAMENTO:

```
img = cv2.resize(img, (720, 480))
```

Redimensionador | Imagem | Largura | Altura

OU

```
scale = 0.5  ← Define um valor para o fator de escala.
img = cv2.resize(img, (int(img.shape[1] * scale), int(img.shape[0] * scale)))
```

O redimensionamento consiste em **definir novos valores para largura e altura da imagem**. Podem ser definidos **valores absolutos** ou usar **valores relativos**.
Os valores devem ser **números inteiros**.

- ESPELHAMENTO DA IMAGEM:

```
img = img[::-1, ::-1]
```

Vertical | Horizontal

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- APARAGEM DE IMAGEM:

```
img = img[0:500, 0:500]
```

Altura — Largura

OU

```
[ Límite superior : Límite inferior,
  Límite direito  : Límite esquerdo ]
```

```
trim = 0.1     ← Define um valor de margem para aparagem.
img = img[int(img.shape[0] * trim) : int(img.shape[0] * (1 - trim)),
          int(img.shape[1] * trim) : int(img.shape[1] * (1 - trim))]
```

- ROTAÇÃO DE IMAGEM:

```
img = cv2.rotate(img, cv.ROTATE_180)  ← Roda em 180º.
```

Rotacionador — Imagem — Graus de rotação

```
img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)          ← Roda em 90º.
img = cv2.rotate(img, cv2.ROTATE_90_COUNTERCLOCKWISE)   ← Roda em 270º.
```

Existem métodos mais complexos e eficientes para rotacionar imagens, mas este é um método básico embutido no OpenCV.

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- SEPARAÇÃO E FUSÃO DE CANAIS:

```
B, G, R = cv2.split(img)
```
← Separa os canais da imagem (azul, verde e vermelho).

```
img = cv2.merge((B, G, R))
```
← Funde os canais (azul, verde, vermelho e, possivelmente, alfa).

- ALTERAÇÃO DE CANAIS:

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```
← Transforma BGR em RGB.

Imagem de entrada

Método

- ESTRATIFICAÇÕES DE CANAIS:

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```
← Transforma BGR em escala de cinza.

```
img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
```
← Distribui um canal de escala de cinza em BGR.

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- TRANSFORMADAS DE FREQUÊNCIAS:

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
```

Imagem de entrada | Limiar | Valor transformado | Método

← Transforma frequências em valores mínimos e máximos de acordo com um limar.

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
```

← Transforma frequências em valores máximos em mínimos de acordo com um limiar.

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
```

Imagem de entrada | Limiar | Limiar superior | Método

← Trunca as frequências abaixo de um limiar.

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
```

← Transforma as frequências que estiverem abaixo de um limiar para um valor mínimo.

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)
```

← Transforma as frequências que estiverem acima de um limiar para um valor mínimo.

INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- TRANSFORMADAS DE FREQUÊNCIAS:

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
```

Imagem de entrada | Limiar | Valor transformado | Método

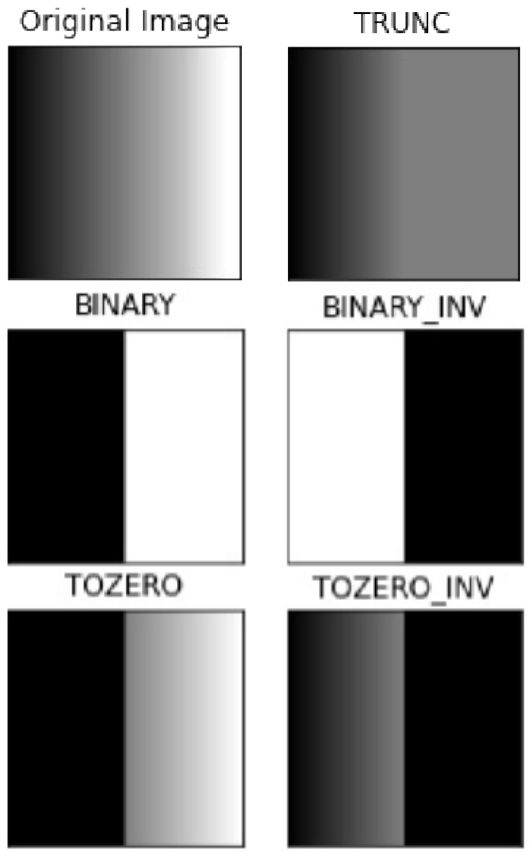
```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
```

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
```

Imagem de entrada | Limiar | Limiar superior | Método

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
```

```
_, img = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)
```



INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- EXTRAÇÃO DE BORDAS:

```
img = cv2.Canny(img, 120, 255, 3)
```

← Infere e extrai bordas com o algorítmo Canny.

Imagem de entrada | Limiar | Limiar | Abertura

- SOBREPOSIÇÃO DE IMAGENS:

```
mask = cv2.bitwise_not(cv2.Canny(img, 120, 255, 3))
```

← Definição de máscara com bordas.

Operador binário not | Extração das bordas com o algoritmo Canny.

```
img = cv2.bitwise_and(img, img, mask=mask)
```

← Fusão da imagem pura com a máscara de bordas.

Operador binário and | Entrada1 | Entrada2 | Máscara

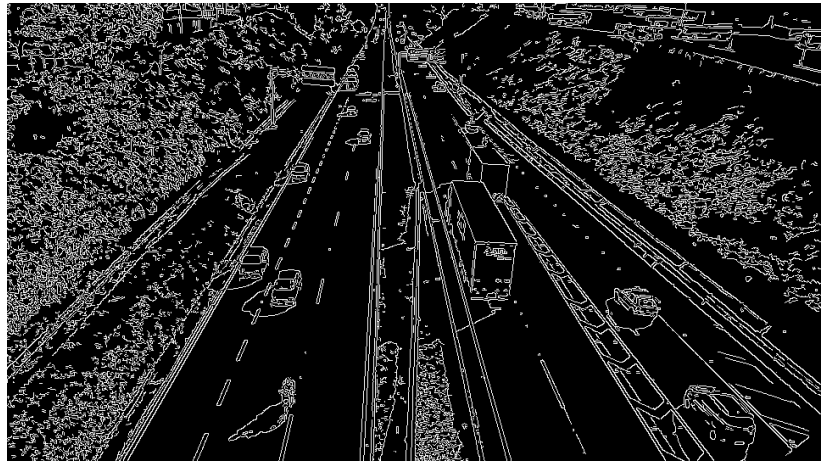
INTRODUÇÃO AO OPENCV

TRANSFORMAÇÃO DE DADOS

- EXTRAÇÃO DE BORDAS:

```
img = cv2.Canny(img, 120, 255, 3)
```

Imagem de entrada | Limiar | Limiar | Abertura



- SOBREPOSIÇÃO DE IMAGENS:

```
mask = cv2.bitwise_not(cv2.Canny(img, 120, 255, 3))
```

```
img = cv2.bitwise_and(img, img, mask=mask)
```

Operador binário and | Entrada1 | Entrada2 | Máscara

INTRODUÇÃO AO OPENCV

INFERÊNCIA DE INFORMAÇÕES

Além de tratar e manipular as imagens, o OpenCV é uma ótima ferramenta para **inferência de informações em imagens**.

Já foi exposto que as imagens nada mais são do que vetores. Não obstante a isso, elas são **campos vetoriais**.

Para se extrair informações destes campos vetoriais, precisamos entender seu sistema de coordenadas:

Diferente do comum, esse campo vetorial tem a origem de um **4º quadrante cartesiano**, onde o eixo das ordenadas é invertido. Portanto:

- valores menores de X estão mais à esquerda;
- valores maiores de X estão mais à direita;
- valores menores de Y estão mais acima;
- valores maiores de Y estão mais abaixo.

Enfim, **é possível inferir informações de objetos detectados a partir de sua posição** no campo vetorial.

INTRODUÇÃO AO OPENCV

INFERÊNCIA DE INFORMAÇÕES

- DETECÇÃO DO OBJETO* EM UMA REGIÃO ESPECÍFICA:

```
if (obj.x > int(img.shape[1] / 2)):          ← Detecta se está na região direita da imagem.
    right += 1                               ← Se sim, incrementa um contador para a direita.
else:                                        ← Se não, estando na região esquerda
    left += 1                                ← incrementa um contador para a esquerda.

if ((200 > obj.x > 600) and (650 > obj.y > 700)):← Detecta se o objeto está dentro de
    in_position += 1                             coordenadas específicas e incrementa um contador.
```

- DISTÂNCIA DE UM OBJETO* EM RELAÇÃO A OUTRO:

```
dist = math.sqrt(((obj1.x - obj2.x) ** 2) + ((obj1.y - obj2.y) ** 2))
hipo = math.sqrt((img.shape[0] ** 2) + (img.shape[1] ** 2))
if (dist < int(hipo * 0.05)):      ← Detecta se a quantidade de píxels entre dois objetos é menor que
    close += 1                           5% da imagem e, se for o caso, incrementa em um contador.
```

* as definições dos objetos foram simplificadas como protótipos contendo a posição de seus centróides (x, y) e tamanho da bounding box (size), concentrando-se aqui apenas na exposição das ideias.

INTRODUÇÃO AO OPENCV

INFERÊNCIA DE INFORMAÇÕES

- CLASSIFICAÇÃO DE OBJETOS* POR TAMANHO:

```
if (obj.size >= int(hipo ** 0.5)):          ← Detecta se ocupa pelo menos metade da imagem.
    huge += 1                               ← Se sim, incrementa um contador para objetos imensos.
elif (obj.size >= int(hipo ** 0.25)):       ← Detecta se ocupa pelo menos 1/4 da imagem.
    large += 1                              ← Se sim, incrementa um contador para objetos grandes.
elif (obj.size >= int(hipo ** 0.125)):      ← Detecta se ocupa pelo menos 1/8 da imagem.
    medium += 1                             ← Se sim, incrementa um contador para objetos médios.
elif (obj.size >= int(hipo ** 0.06)):       ← Detecta se ocupa pelo menos 6% da imagem.
    smol += 1                               ← Se sim, incrementa um contador para objetos pequenos.
else:                                       ← Se não, ocupando menos que 6% da imagem,
    tiny += 1                               ← incrementa um contador para objetos minúsculos.
```

As possibilidades ficam limitadas à então criatividade e conhecimento técnico do programador. Outra ferramenta possível para inferência é o **histograma de frequências**, capaz de inferir informações como **cores e luminosidade**.

* as definições dos objetos foram simplificadas como protótipos contendo a posição de seus centróides (x, y) e tamanho da bounding box (size), concentrando-se aqui apenas na exposição das ideias.

INTRODUÇÃO AO OPENCV

INFERÊNCIA DE INFORMAÇÕES

- DICA: Biblioteca Shapely

Existe também uma biblioteca disponível para o Python chamada **Shapely**, desenvolvida para **análise de objetos geométricos em planos cartesianos** e é muito usada para lidar com mapas e dados geoespaciais. Também **funciona muito bem para visão computacional**.

Essa biblioteca é **fácil de instalar**, **simples de usar** e nos poupa da implementação de classes e funções de **matemática vetorial** para definir objetos geométricos e extrair informações tanto deles quanto da relação com outros objetos e o próprio espaço cartesiano. Apesar de seu uso simples, **suas capacidades são bem extensas** e consumiriam bastante tempo por aqui. Diferente do OpenCV, no entanto, é fácil encontrar conteúdo tanto oficial quanto extraoficial sobre por onde começar com a biblioteca Shapely.

A equipe de Sensoriamento Remoto do INPE fez um ótimo trabalho com esse material:
https://prog-geo.github.io/vetorial/shapely.html

INTRODUÇÃO AO OPENCV

Apesar de não ser tão eficiente quanto um editor de imagem ou vídeo própriamente dito, o OpenCV **conta com ferramentas de computação gráfica para "escrever e desenhar" nas imagens**. De fato, **o campo vetorial pode ser utilizado como um canvas** seguindo os mesmos princípios.

De forma mais básica, **os píxels podem ser acessados arbitrariamente de forma isolada para serem modificados**, realizando transformações puramente matemáticas, mas se destacam as ferramentas para:

- desenho de **retas**;
- desenho de **retângulos**;
- desenho de **círculos**;
- desenho de **curvas**;
- desenho de **demais polígonos**;
- escrita de **texto**.

Além dessas ferramentas, claro, o OpenCV permite a **disposição das imagens em tempo de execução**:

```
cv2.imshow("Imagem", img)
```

Nome da janela — Imagem disposta

INTRODUÇÃO AO OPENCV

- PONTOS:

```
img[42][1337] = [0, 0, 255]
```

Y X Azul Verde Vermelho

A partir desse **princípio simples,** instruções mais complexas podem ser programadas para se **"desenhar" qualquer coisa.**

- RETAS:

```
cv2.line(img,
         (10, 300),
         (800, 300),
         (0, 255, 0),
          2)
```

← Imagem onde a reta é desenhada.
← Coordenada do ponto de início.
← Coordenada do ponto de fim.
← Cor da linha (BGR).
← Espessura da linha (em píxels).

- RETÂNGULOS:

```
cv2.rectangle(img,
              (420, 332),
              (113, 420),
              (122, 255, 122),
               2)
```

← Imagem onde o retângulo é desenhado.
← Coordenada do ponto delimitador máximo.
← Coordenada do ponto delimitador mínimo.
← Cor das linhas (BGR).
← Espessura das linhas (em píxels).

INTRODUÇÃO AO OPENCV

- CÍRCULOS:

```
cv2.circle(img,
           (923, 190),
           250,
           (255, 255, 255),
            -1)
```

← Imagem onde o círculo é desenhado.
← Coordenada do epicentro do círculo.
← Raio do círculo.
← Cor da linha (BGR).
← Espessura da linha (em píxels).

- ELÍPSES:

```
cv2.ellipse(img,
           (923, 190),
           (100, 150),
            45,
            25,
            360,
           (0, 0, 0),
            -1)
```

← Imagem onde o círculo é desenhado.
← Coordenada do epicentro da elípse.
← Largura e altura da elípse.
← Ângulo de rotação.
← Unidade de corte (em graus).
← Unidade de preenchimento (em graus).
← Cor da elípse (BGR).
← Espessura da linha (em píxels).

INTRODUÇÃO AO OPENCV

- FORMAS LIVRES:

```
points = [(800, 300), (750, 240), (250, 400), (1005, 975)]   ← Pontos que definem a linha.
cv2.polylines(img,                                            ← Imagem onde a reta é desenhada.
              np.array([points]),                             ← Lista de coordenada percorridas.
              True,                                           ← Define se a primeira e última coordenada são ligadas, como um polígono
              (0, 255, 0),                                    ← Cor da linha (BGR).
               2)                                             ← Espessura da linha (em píxels).
```

- TEXTO:

```
cv2.putText(img,                                  ← Imagem onde o texto é escrito.
            ("TEXTO"),                            ← Conteúdo escrito.
            (20, 20),                             ← Posição onde o texto é escrito.
            cv2.FONT_HERSHEY_PLAIN,               ← Fonte do texto.
             1,                                   ← Tamanho do texto.
            (200, 0, 180),                        ← Cor da fonte (RGB).
             2)                                   ← Expessura da fonte.
```

A função **polylines pode ser usada para gerar linhas curvas** complexas, se for gerada uma função linear para preencher uma lista de coordenadas, que pode ser passada como parâmetro do polylines.

INTRODUÇÃO AO OPENCV

O QUE MAIS É POSSÍVEL

Seria muito simples e estimulante dizer que, com essas capacidades e recursos do OpenCV, **qualquer coisa é possível**.
Há um lado negativo e um lado positivo nessa frase:

Lado negativo: não é exatamente verdade;
Lado positivo: é uma questão de tempo até que seja.

O estudo do OpenCV aliado a conhecimentos e competências como **programação** e demais recursos, além de uma certa criatividade, é uma ótima forma de atingir **avanços significativos na área de visão computacional** e, mais do que isso, **tornar esse mundo um lugar melhor**.

E VOCÊS?

PERGUNTAS?

WORCAP2021
Workshop em Computação Aplicada

COMPUTAÇÃO APLICADA

INPE
UNIDADE DE PESQUISA DO MCTI

MUITO OBRIGADO

FOI UMA HONRA

Contato: rafael.andrade23@fatec.sp.gov.br
LAC - sala 14.