Développeur d'intelligence artificielle appliquée

Cours #4

www.impactia.org

Structure de la formation

- **#1: Introduction**
- **#2: Vision**
- **#3: Vision**
- **#4: Vision**
- #5: Renforcement
- #6: Renforcement
- #7: Renforcement
- #8: Langage
- #9: Langage
- #10: Langage
- # 11: Outillage
- #12: Génération d'images
- #13: Génération d'images
- #14: Projet
- #15: Projet

Semaine dernière : Cours #3

- Théorie
  - EfficientNet
  - “Bitter Lesson”
- Pratique
  - Classification d’images
  - Apprentissage par transfert
  - Monitorage de l'entraînement de modèles (Weights and Biases)

Aujourd’hui : Cours #4

- Théorie
  - Augmentation de données
- Pratique
  - EfficientNet en pratique
  - Augmentation de données

Les approches de formations

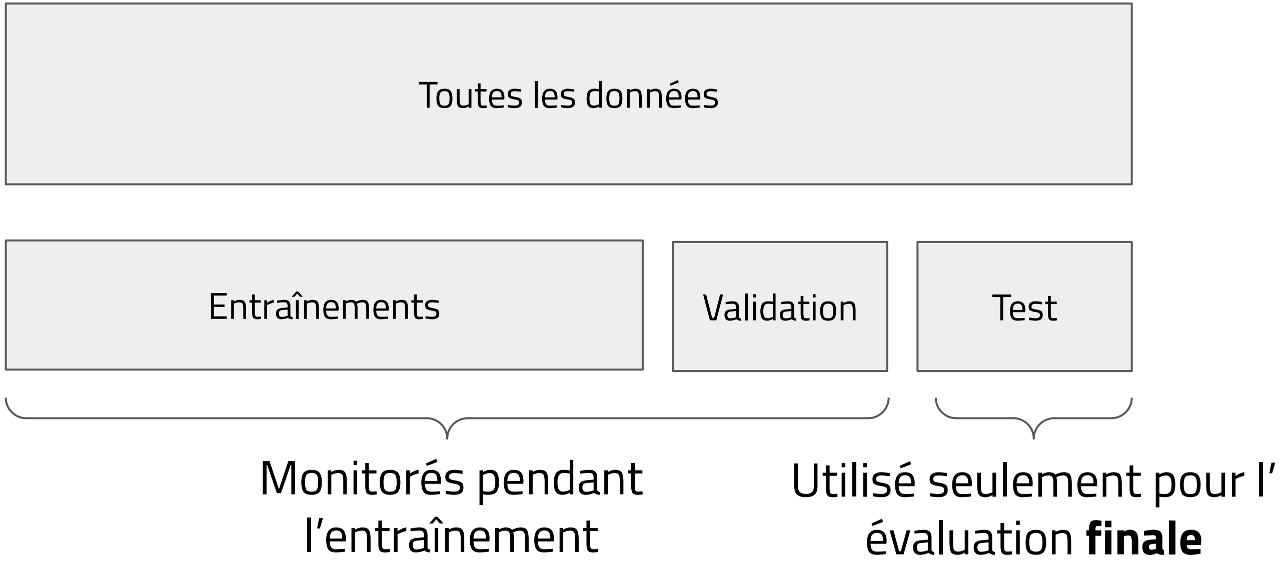
Apprentissage supervisé

Apprentissage par transfert

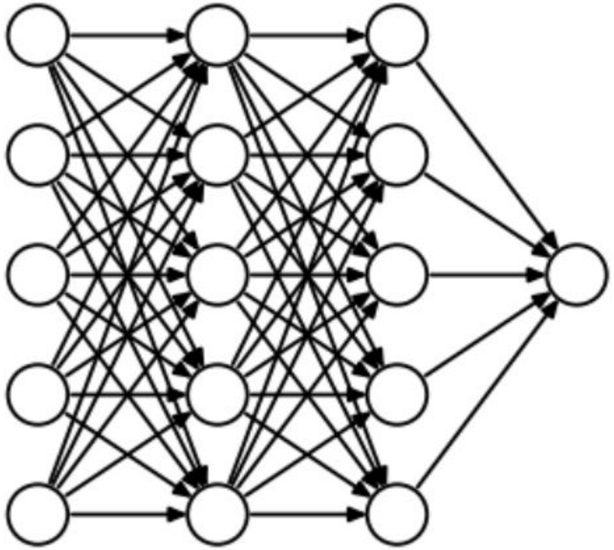
Apprentissage auto-supervisé

Apprentissage par renforcement

Jeux de données : “train”, “validate”, “test”

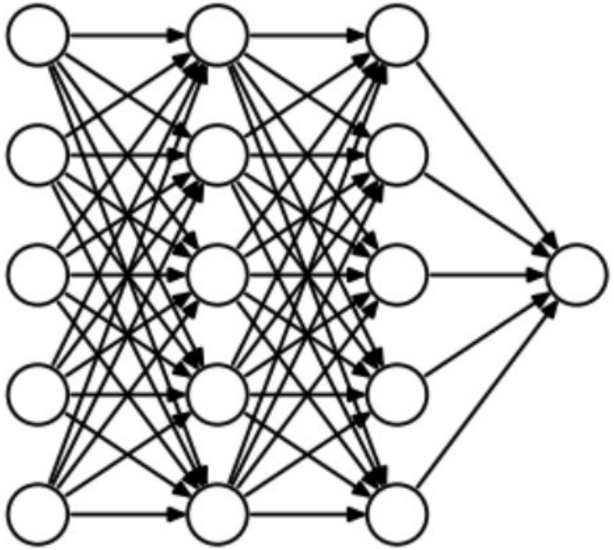

Toutes les données
Entraînements
Validation
Test
Monitorés pendant l’entraînement
Utilisé seulement pour l’ évaluation **finale**

Dropout (“abandon”)

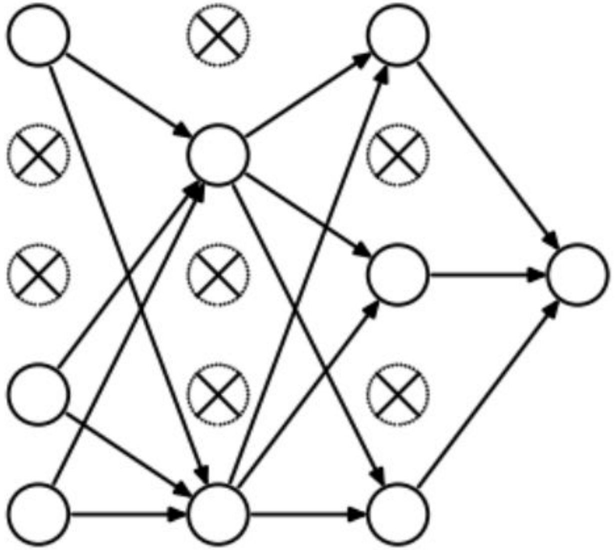


sans
dropout

Dropout (“abandon”)

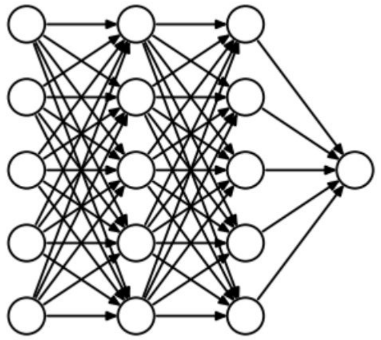


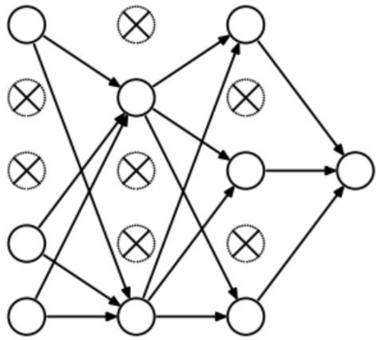
sans dropout



avec dropout

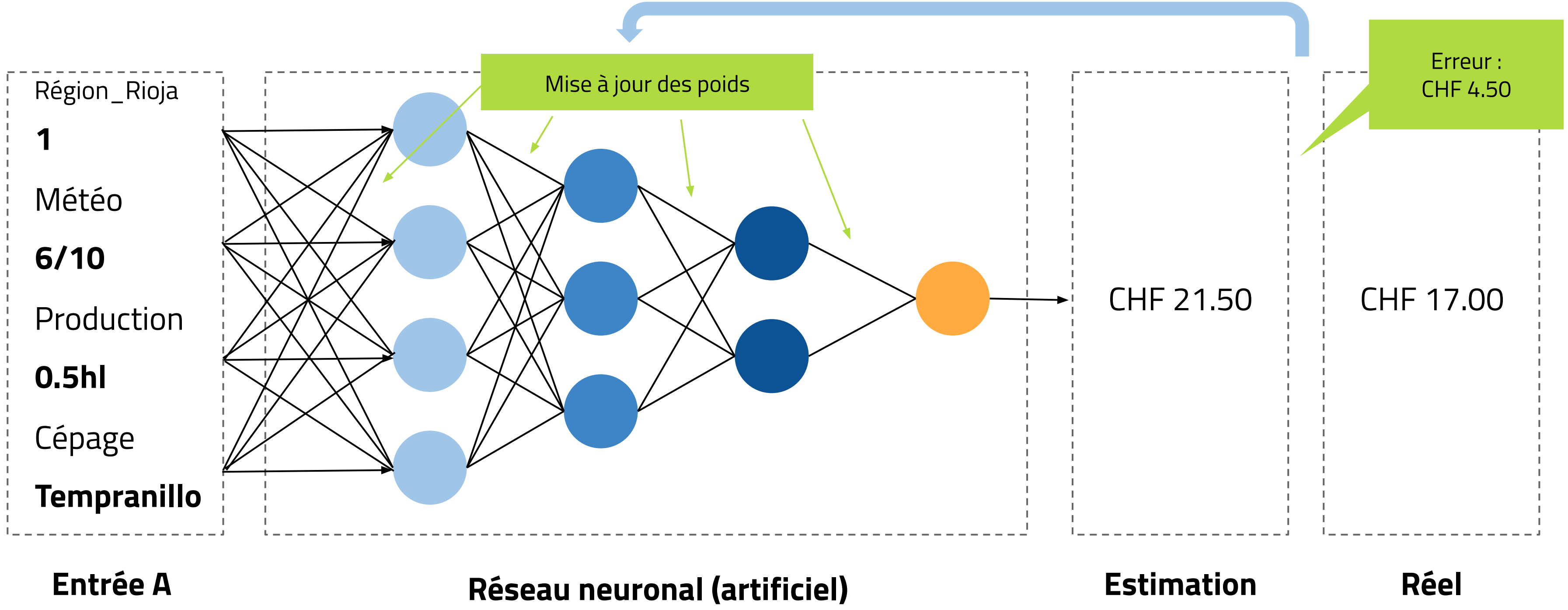
Dropout ("abandon")


sans dropout

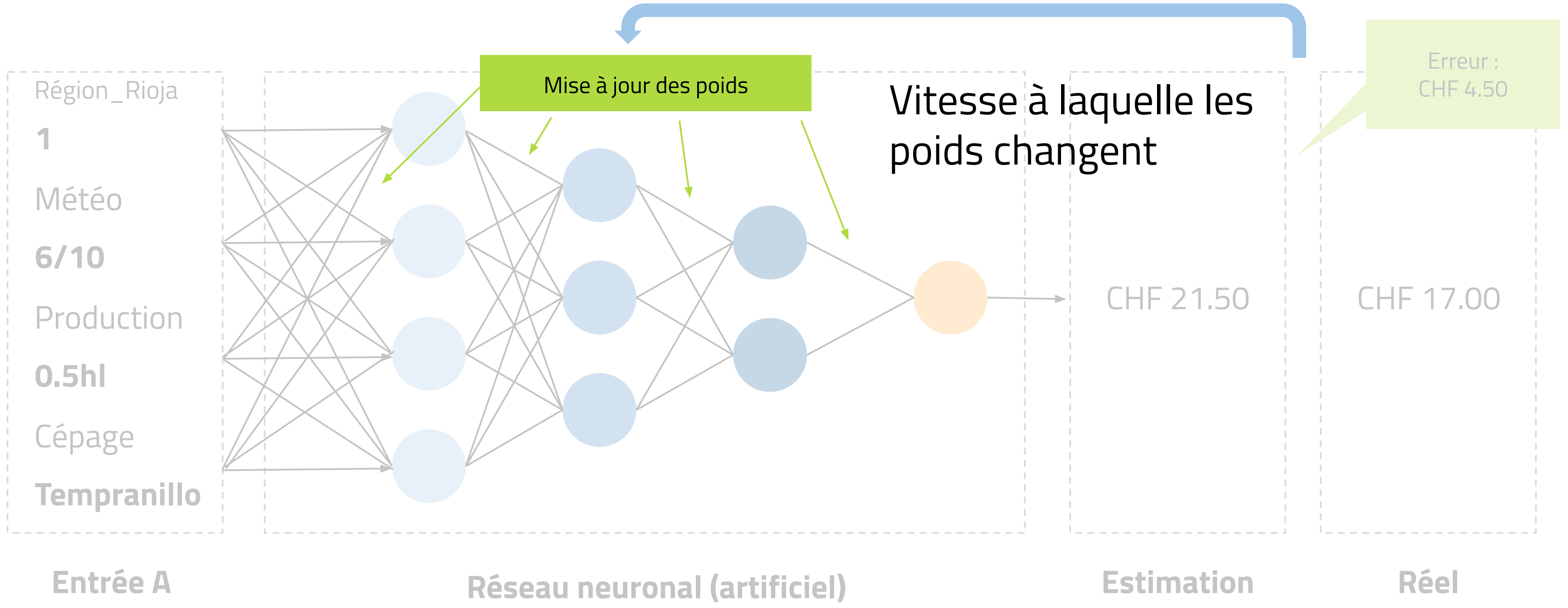

avec dropout

- Dropout rends la précision pendant l'entraînement plus faible, car certaines connections sont désactivées !
- Dropout est utilisé **uniquement** pendant l'entraînement
- Keras désactive Dropout automatique pendant l'évaluation

Learning rate ("taux d'apprentissage")

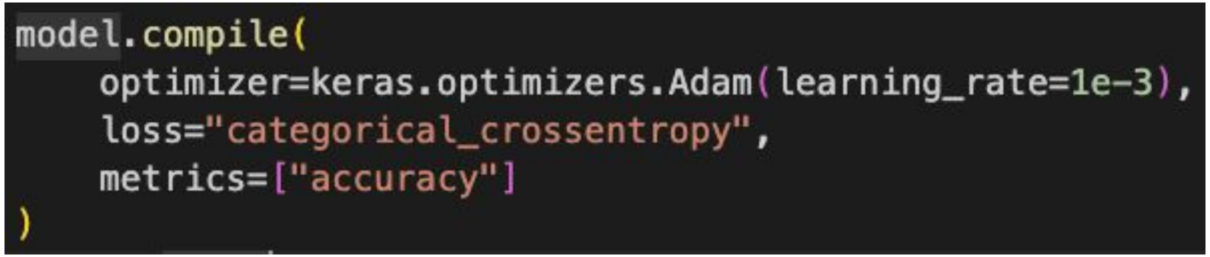


Learning rate ("taux d'apprentissage")



Vitesse à laquelle les poids changent

Learning rate (“taux d’apprentissage”)



```
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"]
)
```

EfficientNet

Optimiser l'architecture des modèles

Contraintes

- Mémoire disponible :
    - téléphone ?
    - ordinateur ?
    - super-ordinateur ?
- Temps nécessaire pour l'utilisation
    - 100ms ? 1s ? 1min ?
- Coûts d'utilisation

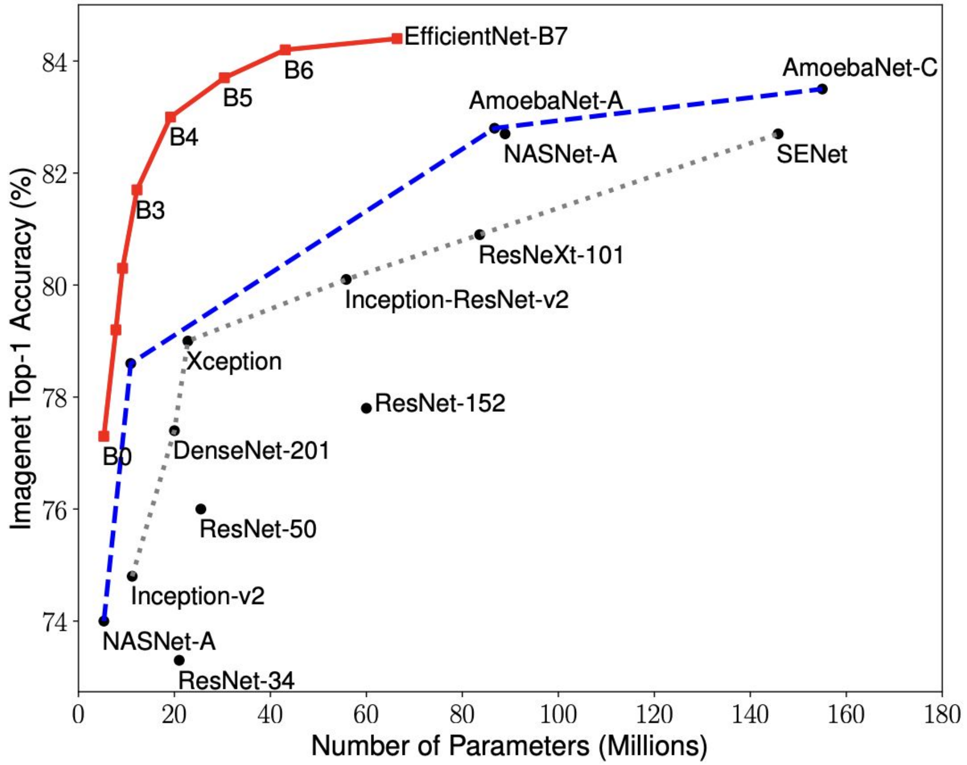
Optimiser l'architecture des modèles

Contraintes

- Mémoire disponible :
  - téléphone ?
  - ordinateur ?
  - super-ordinateur ?
- Temps nécessaire pour l'utilisation
  - 100ms ? 1s ? 1min ?
- Coûts d'utilisation

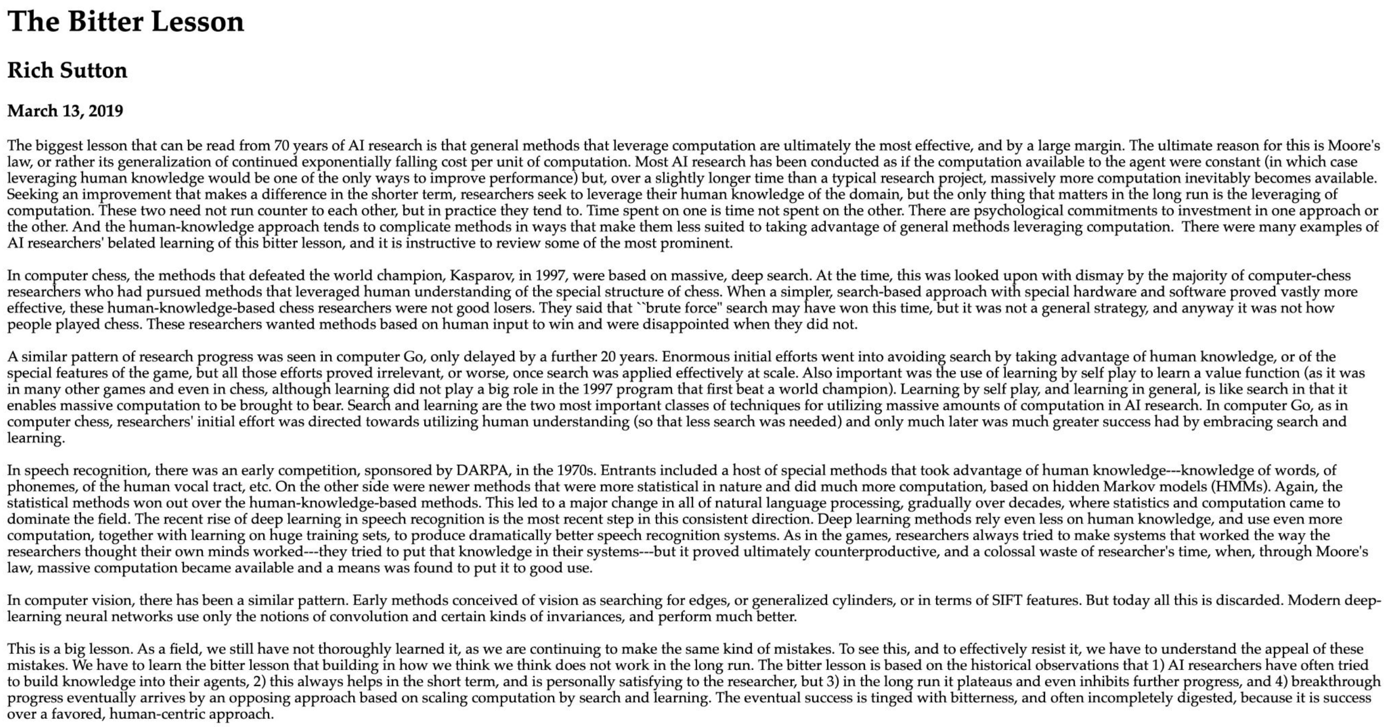
On aimerait trouver le meilleur modèle pour une taille donnée !

Meilleur modèle pour une taille donnée



Modèles EfficientNet!
“Rethinking Model Scaling for Convolutional Neural Networks”
2019, Google

La “leçon amère”



The Bitter Lesson

Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that ``brute force" search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to win and were disappointed when they did not.

A similar pattern of research progress was seen in computer Go, only delayed by a further 20 years. Enormous initial efforts went into avoiding search by taking advantage of human knowledge, or of the special features of the game, but all those efforts proved irrelevant, or worse, once search was applied effectively at scale. Also important was the use of learning by self play to learn a value function (as it was in many other games and even in chess, although learning did not play a big role in the 1997 program that first beat a world champion). Learning by self play, and learning in general, is like search in that it enables massive computation to be brought to bear. Search and learning are the two most important classes of techniques for utilizing massive amounts of computation in AI research. In computer Go, as in computer chess, researchers' initial effort was directed towards utilizing human understanding (so that less search was needed) and only much later was much greater success had by embracing search and learning.

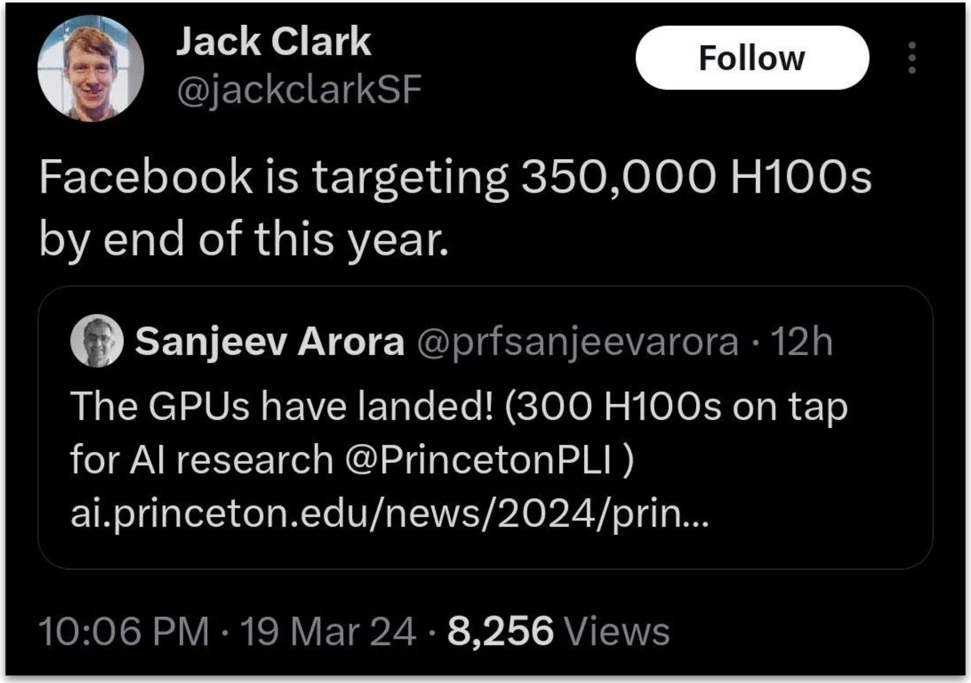
In speech recognition, there was an early competition, sponsored by DARPA, in the 1970s. Entrants included a host of special methods that took advantage of human knowledge---knowledge of words, of phonemes, of the human vocal tract, etc. On the other side were newer methods that were more statistical in nature and did much more computation, based on hidden Markov models (HMMs). Again, the statistical methods won out over the human-knowledge-based methods. This led to a major change in all of natural language processing, gradually over decades, where statistics and computation came to dominate the field. The recent rise of deep learning in speech recognition is the most recent step in this consistent direction. Deep learning methods rely even less on human knowledge, and use even more computation, together with learning on huge training sets, to produce dramatically better speech recognition systems. As in the games, researchers always tried to make systems that worked the way the researchers thought their own minds worked---they tried to put that knowledge in their systems---but it proved ultimately counterproductive, and a colossal waste of researcher's time, when, through Moore's law, massive computation became available and a means was found to put it to good use.

In computer vision, there has been a similar pattern. Early methods conceived of vision as searching for edges, or generalized cylinders, or in terms of SIFT features. But today all this is discarded. Modern deep-learning neural networks use only the notions of convolution and certain kinds of invariances, and perform much better.

This is a big lesson. As a field, we still have not thoroughly learned it, as we are continuing to make the same kind of mistakes. To see this, and to effectively resist it, we have to understand the appeal of these mistakes. We have to learn the bitter lesson that building in how we think we think does not work in the long run. The bitter lesson is based on the historical observations that 1) AI researchers have often tried to build knowledge into their agents, 2) this always helps in the short term, and is personally satisfying to the researcher, but 3) in the long run it plateaus and even inhibits further progress, and 4) breakthrough progress eventually arrives by an opposing approach based on scaling computation by search and learning. The eventual success is tinged with bitterness, and often incompletely digested, because it is success over a favored, human-centric approach.

http://www.incompleteideas.net/IncIdeas/BitterLesson.html

La “leçon amère”


Jack Clark
@jackclarkSF
Follow

Facebook is targeting 350,000 H100s by end of this year.

Sanjeev Arora @prfsanjeevarora · 12h
The GPUs have landed! (300 H100s on tap for AI research @PrincetonPLI)
ai.princeton.edu/news/2024/prin...

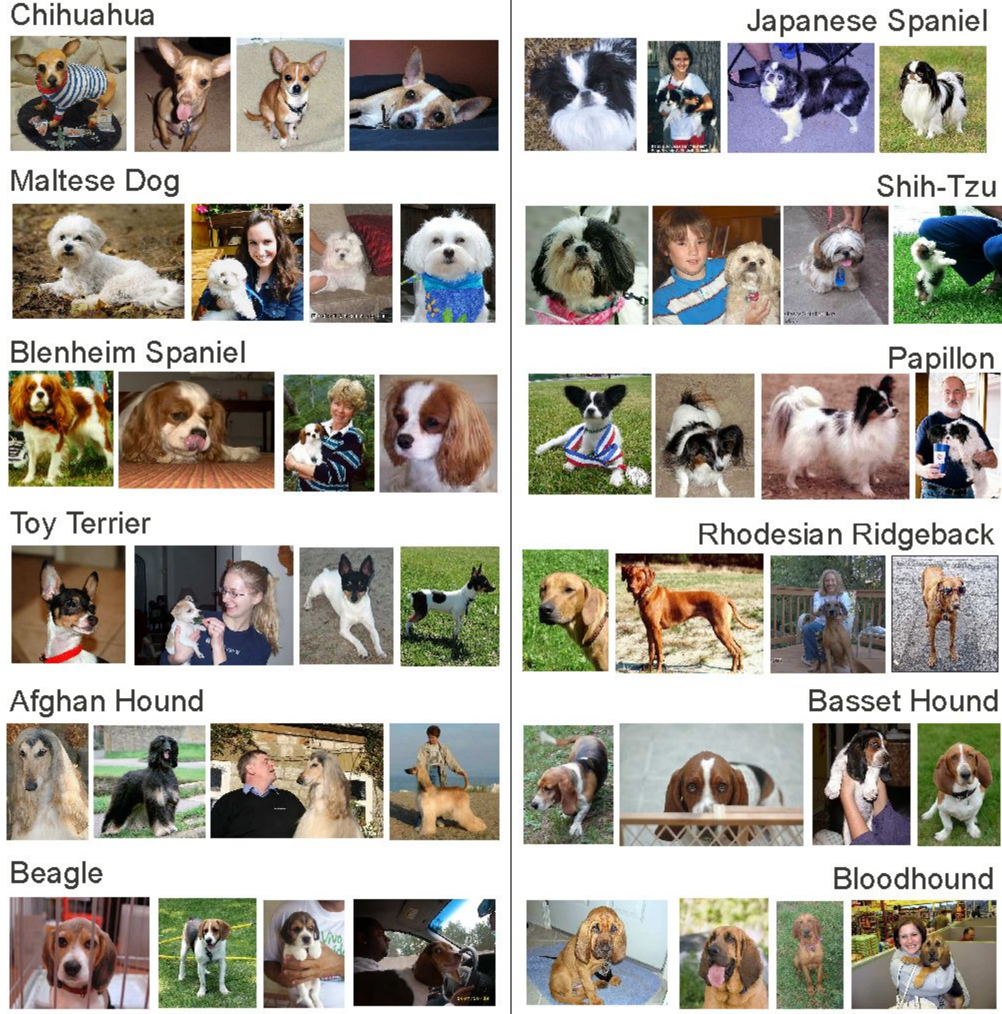
10:06 PM · 19 Mar 24 · 8,256 Views

Stanford Dogs

Sous-ensemble d'Image Net

20,580 images annotées

vision.stanford.edu/aditya86/ImageNetDogs



Mise en pratique

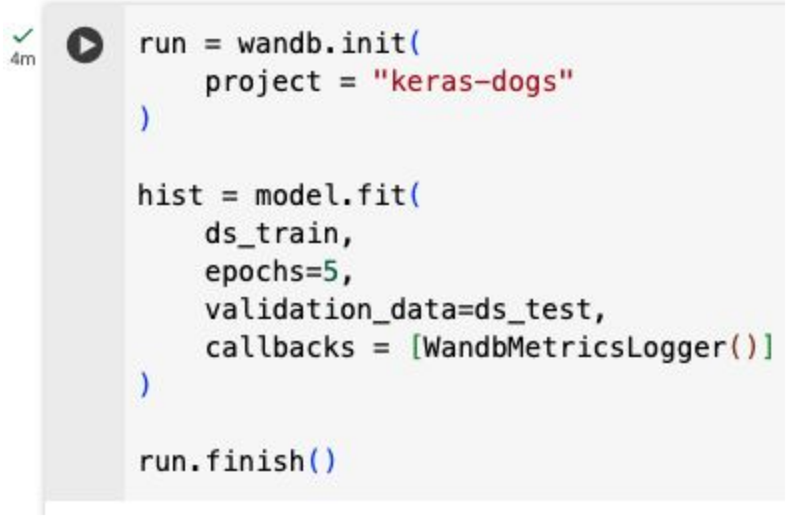
Entraînements de modèles EfficientNet

https://colab.research.google.com/drive/1Jkzcu-Eq91s06mZup6VZ9lEVITje9G8t

Améliorons l'apprentissage par transfert

- Paramètres sur lesquels jouer
  - Learning rate ? Epochs ? Batch size ? dropout ?
  - Entraîner seulements les dernières couches, ou plus ?
- Weights and Biases
  - Ne pas oublier le init/finish
  - Mettre les paramètres dans la "config"
- Plus ambitieux…
  - Changer de modèle EfficientNet ?
  - EfficientNetV2 ?
- Augmentation des données

Weights & Biases



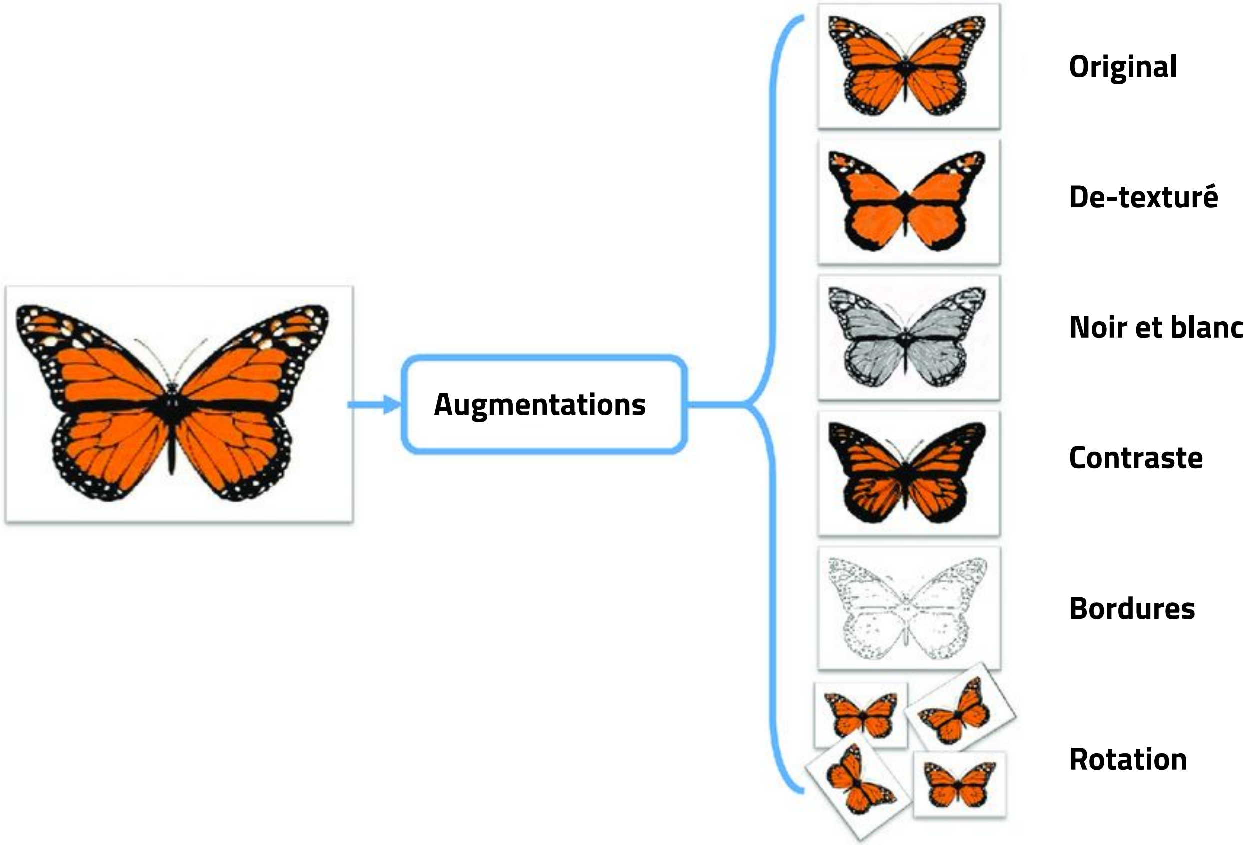
```
run = wandb.init(
    project = "keras-dogs"
)

hist = model.fit(
    ds_train,
    epochs=5,
    validation_data=ds_test,
    callbacks = [WandbMetricsLogger()]
)

run.finish()
```

Augmentation de données

Augmentation des données



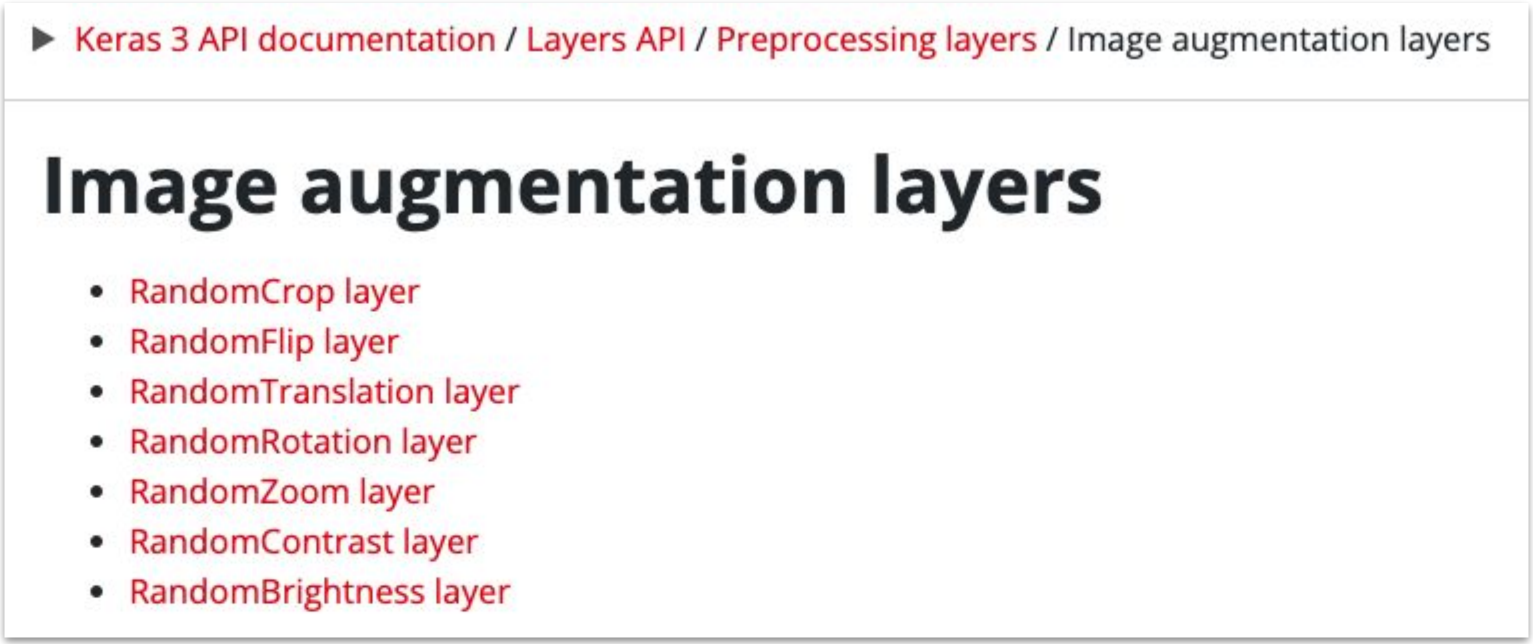

▶ Keras 3 API documentation / Layers API / Preprocessing layers / Image augmentation layers

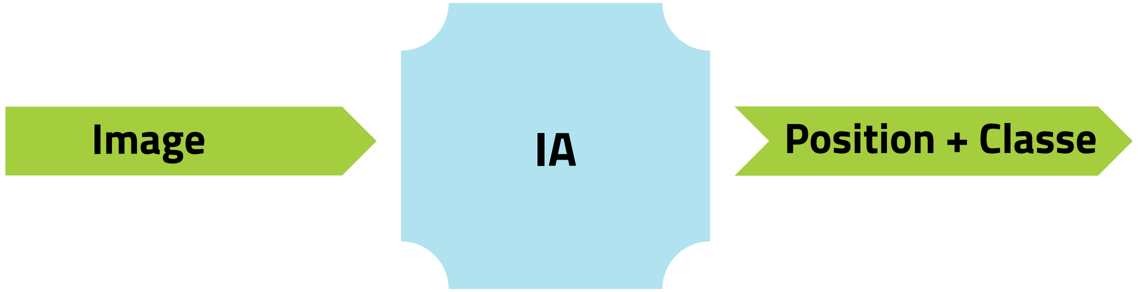
Image augmentation layers

- RandomCrop layer
- RandomFlip layer
- RandomTranslation layer
- RandomRotation layer
- RandomZoom layer
- RandomContrast layer
- RandomBrightness layer

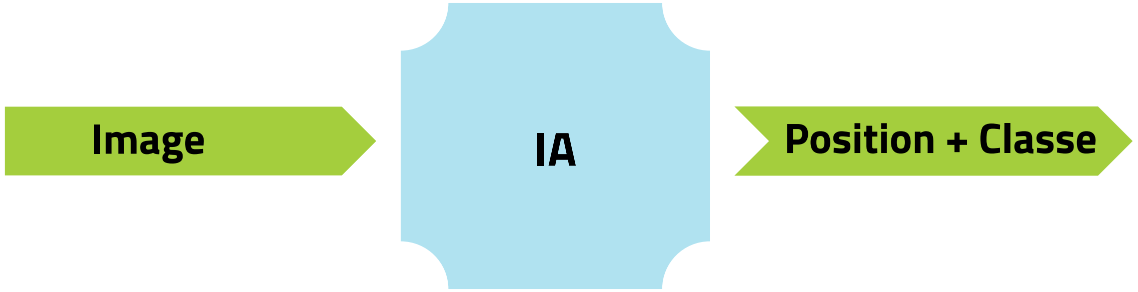
https://keras.io/api/layers/preprocessing_layers/image_augmentation/

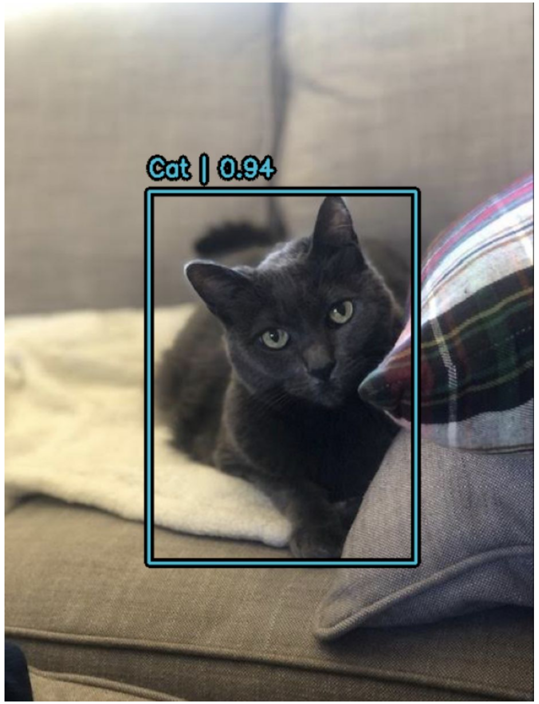
Détection d’objets

Détection d’objets



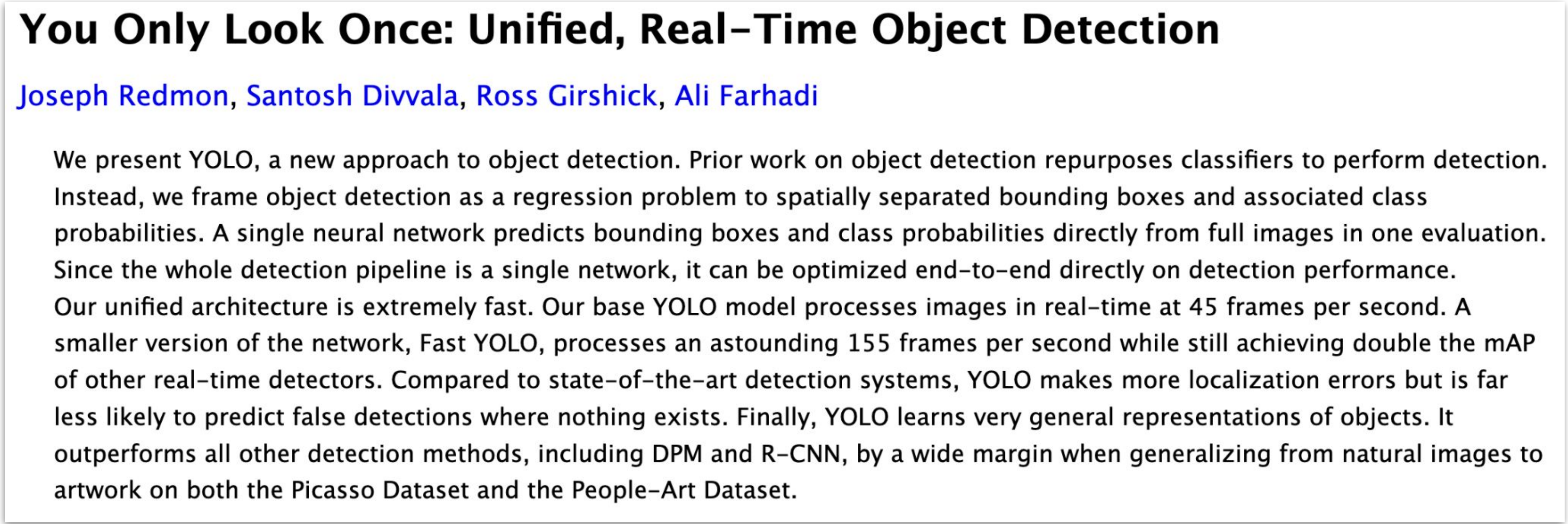
Détection d'objets





YOLO

YOLO

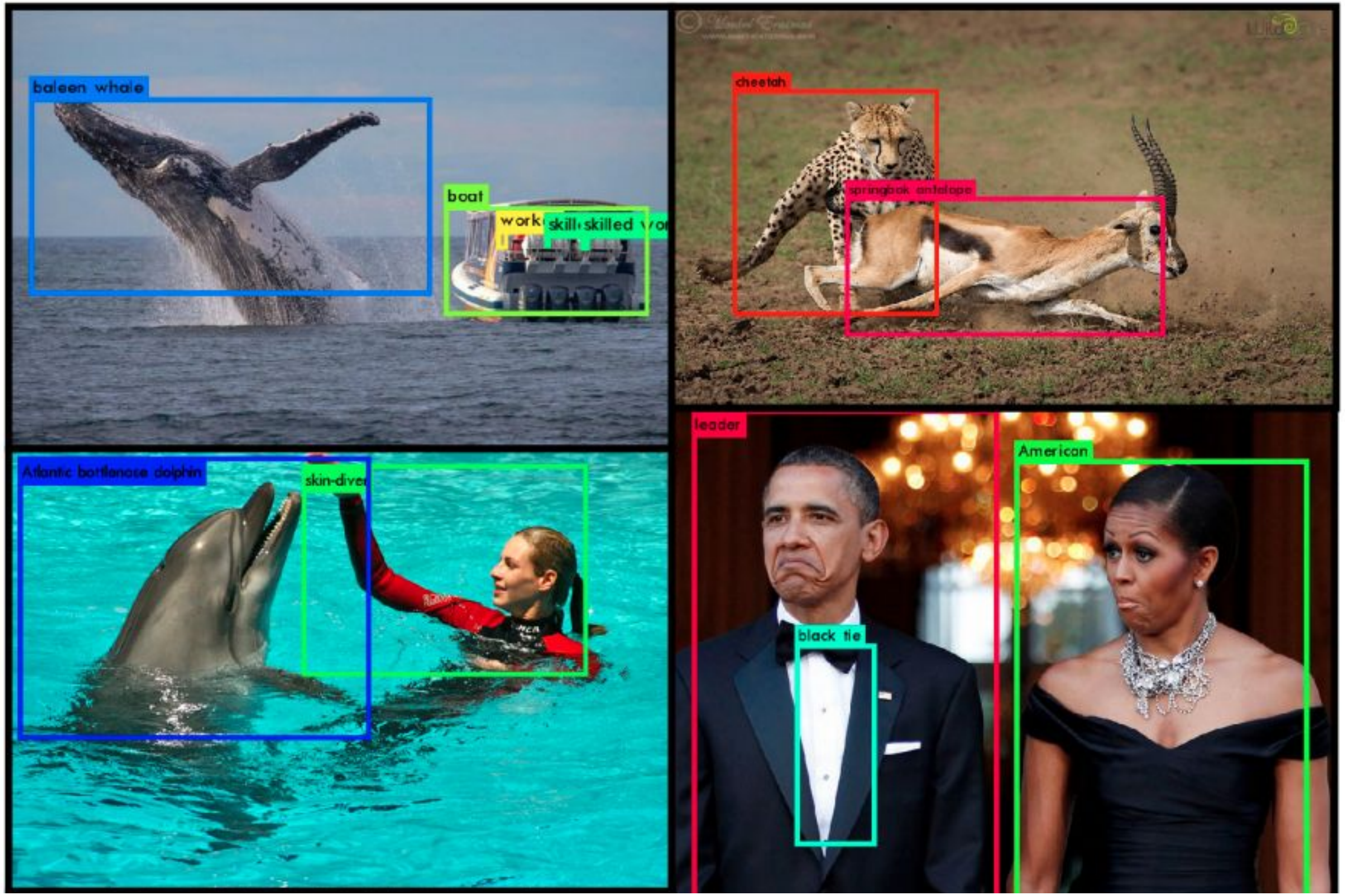


You Only Look Once: Unified, Real-Time Object Detection

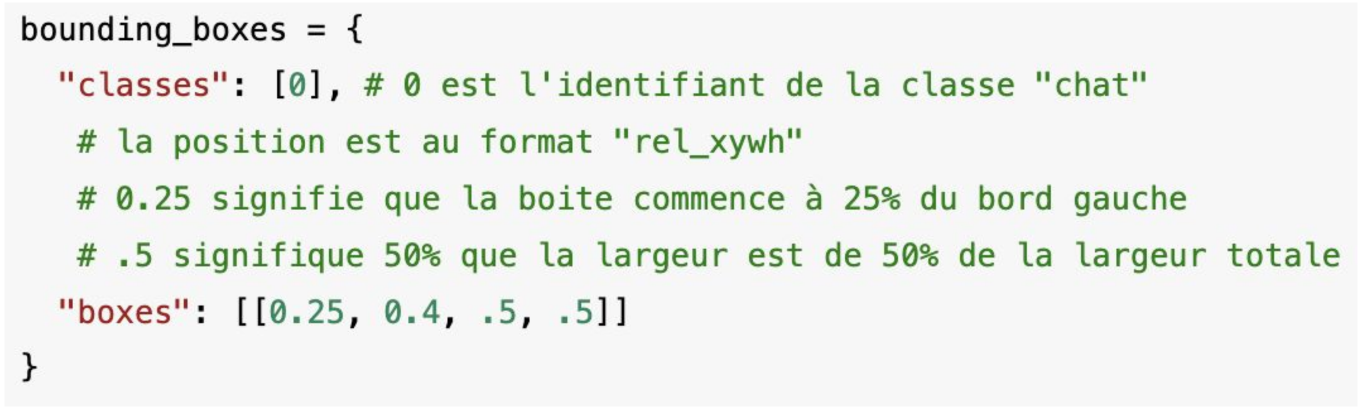
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.
Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is far less likely to predict false detections where nothing exists. Finally, YOLO learns very general representations of objects. It outperforms all other detection methods, including DPM and R-CNN, by a wide margin when generalizing from natural images to artwork on both the Picasso Dataset and the People-Art Dataset.

`https://arxiv.org/abs/1506.02640`

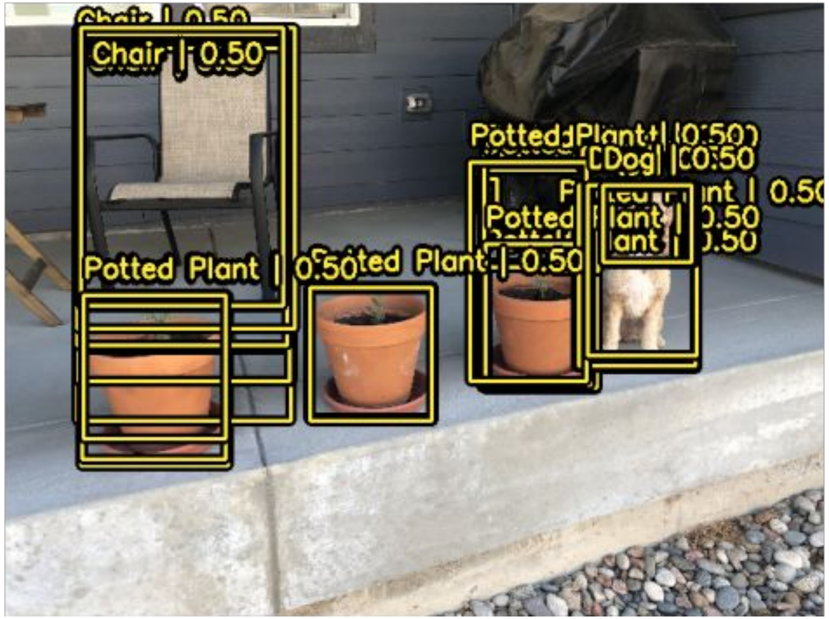

baleen whale
boat
work
skill skilled wor
cheetah
springbok antelope
Atlantic bottlenose dolphin
skin-diver
leader
American
black tie

Détection d’objets

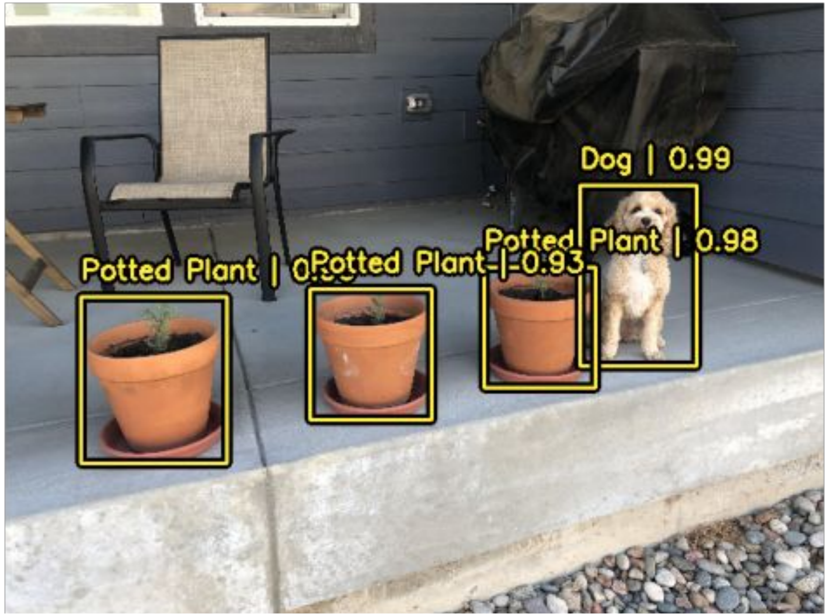


```
bounding_boxes = {
  "classes": [0], # 0 est l'identifiant de la classe "chat"
   # la position est au format "rel_xywh"
   # 0.25 signifie que la boite commence à 25% du bord gauche
   # .5 signifique 50% que la largeur est de 50% de la largeur totale
  "boxes": [[0.25, 0.4, .5, .5]]
}
```

Détection de boîtes, puis “nettoyage” !



Détection de boîtes, puis “nettoyage” !



Mise en pratique

Détection d'objets avec YOLO

https://colab.research.google.com/drive/15wkhqLIDLJsCa2aDvEcPQyPZwY10mIdJ