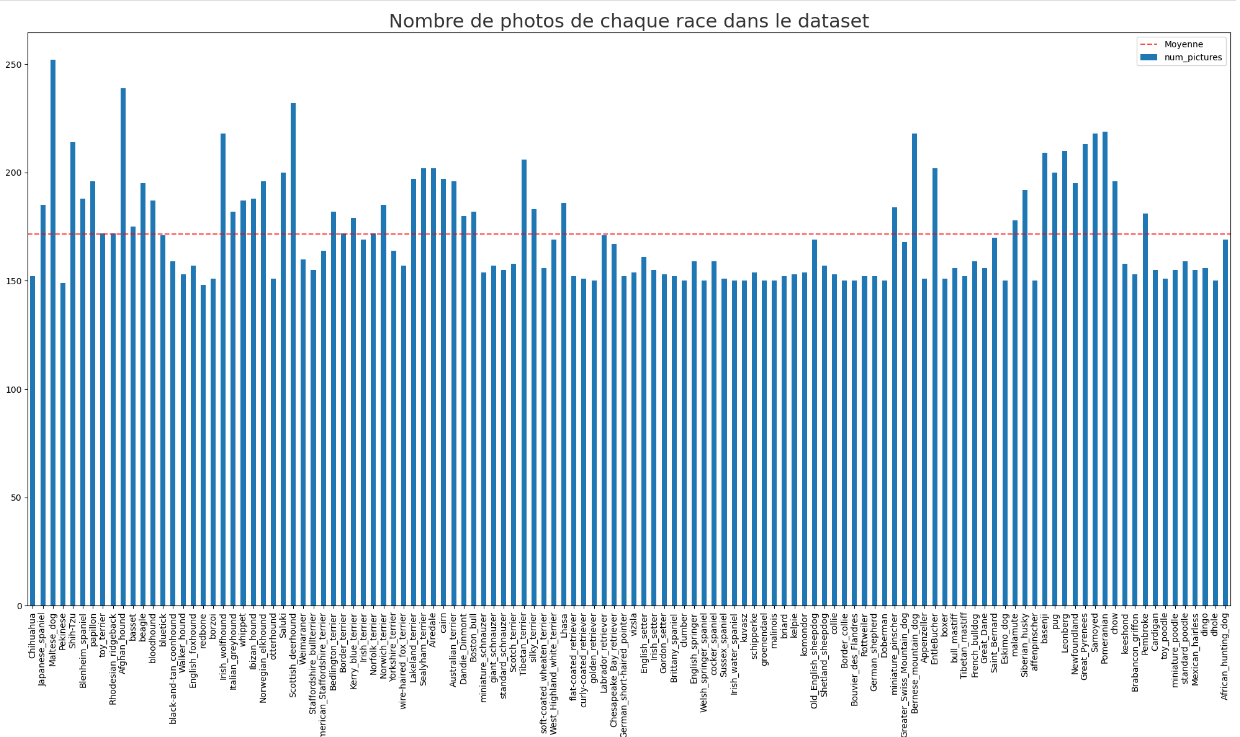
**Note méthodologique : preuve de concept**

## **Dataset retenu**

Nous utilisons le dataset fourni par l’université de Standford nommé “Standford Dogs Dataset”. Il est composé de 20 580 images réparties en 120 races de chien (classes).

Les classes sont relativement bien distribuées (comme nous pouvons le voir sur le graphique ci-dessous) avec une moyenne de 220 images par race.



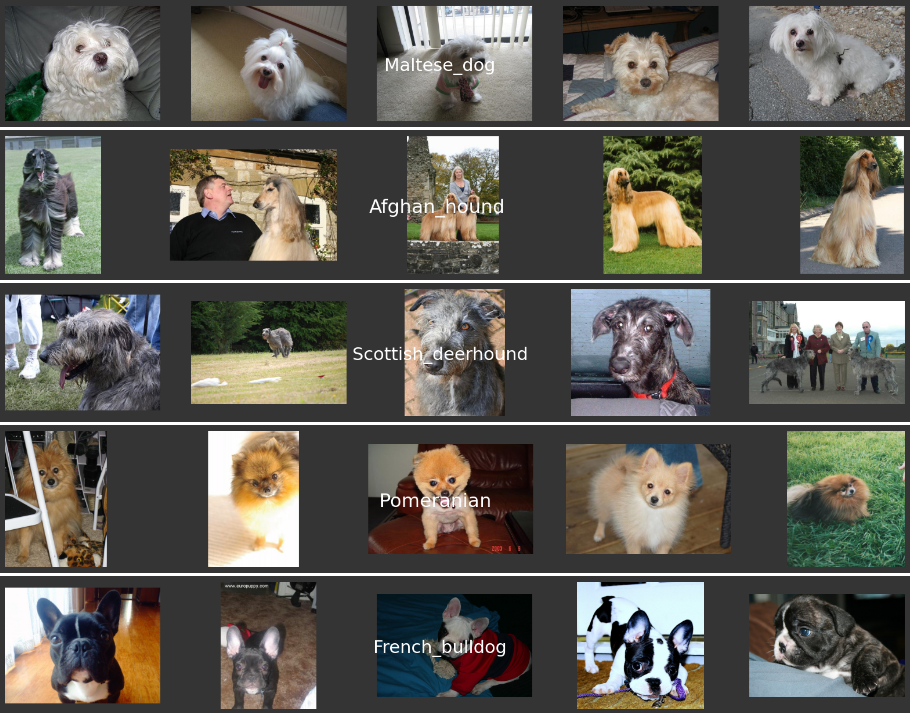
## 

## 

## 

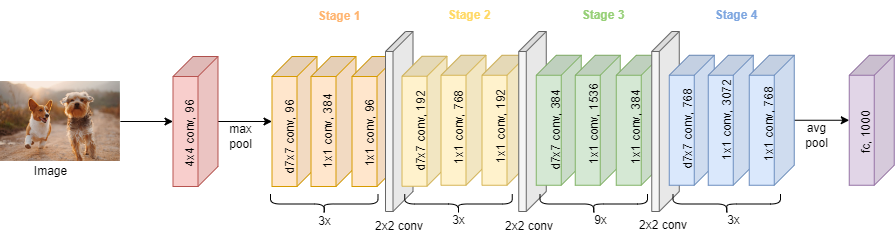
## 

Les images présentes dans ce dataset sont un échantillon de la base ImageNet.

Voici un exemple d’images présentes dans le dataset : 

## **Les concepts de l’algorithme récent**

Voici l’architecture détaillée du modèle ConvNeXt que nous allons tester :



(source : <https://medium.com/@atakanerdogan305/convnext-next-generation-of-convolutional-networks-325607a08c46>)

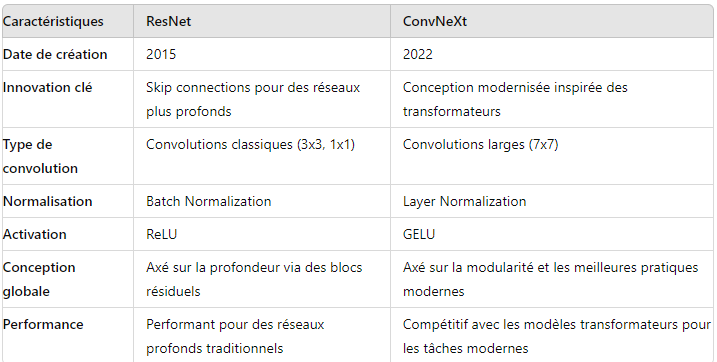
L'architecture de ConvNeXt est similaire à celle des réseaux CNN classiques, comme ResNet, mais avec des améliorations spécifiques qui optimisent l’efficacité des convolutions et permettent de traiter les images à grande échelle.

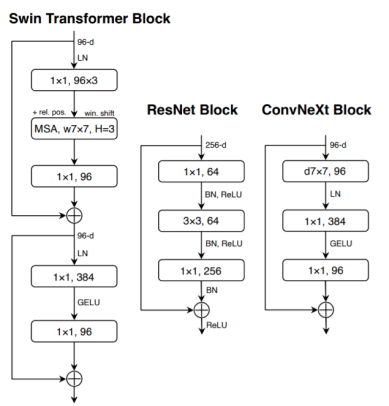
Les principales modifications sont :

* convolutions plus larges (7x7 contre 3x3 pour Resnet)
* activation GELU
* Normalisation en LayerNorm (et non Batch)
* pas de couches de pooling pour la réduction de dimension

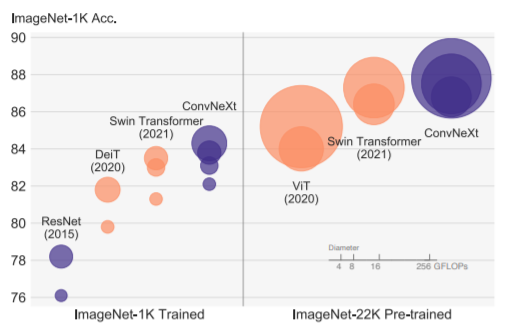
L’objectif de ConvNeXt est d’atteindre les performances des Transformers en vision tout en maintenant la simplicité et les avantages de calcul des CNN traditionnels.

Voici un résumé des principales différences entre ResNet et ConvNeXt :





Ainsi ConvNeXt semble pouvoir être un bon candidat pour permettre de meilleurs résultats que ResNet50 notamment lorsque l’on observe les résultats d’accuracy de différents modèles testés sur la base ImageNet (dont les images de notre dataset sont issues) :



Nos CNN sont tous pré entraînés sur la base ImageNet-1K (1,2 million d'images réparties en 1 000 classes d'objets)

## **La modélisation**

Dans un premier temps, nous calibrons nos 2 algorithmes sur un modèle de classification à 5 classes (races) en recherchant la meilleure performance de chacun.

Nous réalisons les mêmes étapes préalables :

* preprocessing
* séparation en jeu d’entrainement, de validation et de test (75% train / 15% validation / 10% test)
* hyperparamètres : nombre d’epochs (30), batch size (32) et earlystopping (val loss / patience à 5), optimizer (RMSprop)
* utilisation des modèles keras pré entraînés sur ImageNet 1K
* seules les couches des fin sont entraînées sur nos images

Nous adaptons l’architecture de chacun de nos algorithmes (ajout de couche(s) dense) :

* ResNet 50 : Average pooling + une seule couche à activation softmax
* ConvNeXt : Average pooling + ajout d’une couche dense de dimension 128 avant la couche à activation softmax

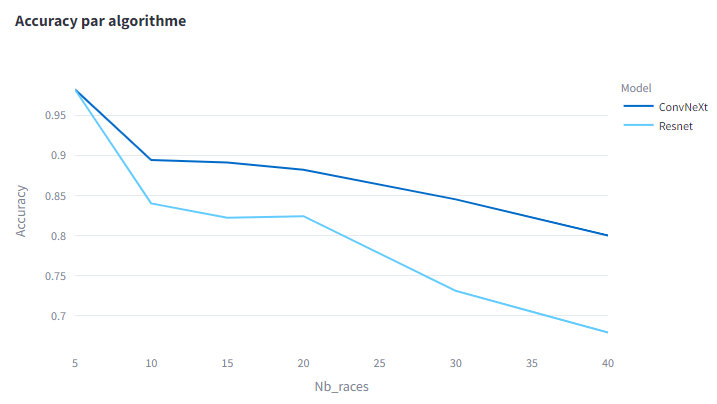
Sur la base de ce calibrage nous augmentons progressivement les nombre de classes recherchées et comparons les résultats de nos 2 algorithmes :

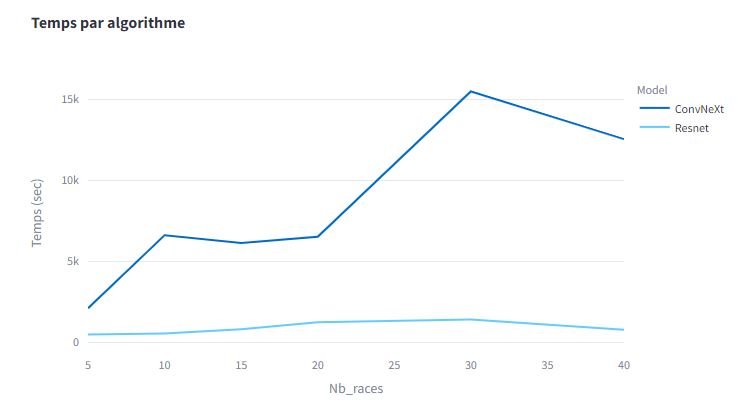
* 10 races
* 15 races
* 20 races
* 30 races
* 40 races

## **Une synthèse des résultats**

Nous avons stocké l’ensemble de nos résultats dans MLFlow et nous conservons le meilleur résultat d’**accuracy** par test et les observons sur le graphique ci-dessous.

En effet l’accuracy (pourcentage de prédictions correcte) est facile à interpréter et pertinente dans le cadre d’un jeu de données équilibré (ce qui est notre cas)





Sur 5 classes, ResNet présente des performances équivalentes à ConvNeXt.

Cependant dès que nous augmentons le nombre de classes, ses performances décroissent alors que ConvNeXt maintient une certaine stabilité.

Pour garantir les performances de ConvNeXt nous avons ajouté une couche dense (à filtre 128) avant la couche softmax et cela augmente nettement le nombre de paramètres entrainables et donc le temps de calcul. Toutefois ce temps de calcul se maintient avec l’augmentation de la cardinalité des classes à partir de 10 classes et reste tout à fait acceptable pour ce genre de travaux sans GPU.

## **L’analyse de la feature importance globale et locale du nouveau modèle**

### Feature importance globale avec visualisation des filtres

La visualisation des filtres dans un réseau de neurones convolutif (CNN) permet de mieux comprendre comment le modèle extrait des caractéristiques importantes à partir des données. Cette approche se concentre sur l'importance globale des caractéristiques ou des patterns que le modèle apprend à travers ses différentes couches de convolution.

Au fur et à mesure que l'on progresse dans le réseau (de couches basses à couches plus profondes), les filtres détectent des caractéristiques de plus en plus abstraites.

Visualiser ces filtres nous aide à comprendre quelles caractéristiques sont détectées à chaque niveau du réseau.

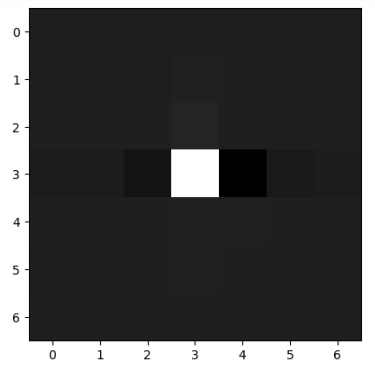
Par exemple :

* Les couches basses peuvent détecter des contours ou des bords simples
* Les couches plus profondes peuvent détecter des caractéristiques complexes comme des formes spécifiques, des textures, ou même des objets entiers.

Cette visualisation est importante pour comprendre comment et où le modèle accorde de l'importance dans les images.

### 

Resnet50



ConvNeXt (couche 4 = convnext\_tiny\_stage\_0\_block\_0\_layernorm)

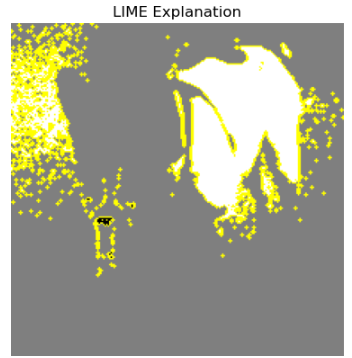
Sur cet exemple, nous analysons comment la 1ère couche de filtre du modèle ConvNeXt s’est comportée.

### Feature importance locale avec Lime

**LIME** (Local Interpretable Model-agnostic Explanations) est une technique utilisée pour expliquer les prédictions d'un modèle de machine learning en fournissant des explications locales et interprétables. Dans le contexte des CNNs pour la classification d'images, LIME fonctionne en procédant par étapes pour déterminer quelles parties de l'image (ou "super-pixels") sont les plus importantes pour la prédiction.

* Étape 1 : Perturbation de l'image originale
* Étape 2 : Prédictions pour les images perturbées
* Étape 3 : Apprentissage d'un modèle linéaire local
* Étape 4 : Calcul des importances locales
* Étape 5 : Interprétation visuelle

Sur cet exemple nous remarquons que les pixels qui semblent avoir eu le plus de poids dans la prédiction sont ceux du “corps” de l’animal (et non sa tête).





## 

## **Les limites et les améliorations possibles**

ConvNeXt s’est montré très performant mais également beaucoup plus consommateur de ressources que ResNet.

ConvNeXt, comme d'autres architectures CNN modernes, est un modèle très puissant mais également complexe et lourd en termes de calcul. Il nécessite une importante puissance de calcul (GPU) pour l'entraînement et peut être difficile à déployer sur des systèmes avec des ressources limitées (ex. appareils mobiles). Cette complexité peut aussi entraîner des temps de prédiction plus lents, ce qui peut être une limitation dans des cas d'utilisation en temps réel.

Il est difficile de comprendre quelles caractéristiques précises de l'image influencent la décision du modèle. Nous observons des “tendances” avec le feature importance locale sans toutefois pouvoir généraliser.

Afin d’améliorer le temps d'exécution du modèle et le nombre de classes testées il serait intéressant de bénéficier d’une GPU (Graphics Processing Unit) pour permettre le test sur 120 classes et gagner sur les temps de calcul.

Afin d’améliorer les performances du modèle nous pourrions optimiser les hyperparamètres à l’aide d’un gridsearch pour l’affiner encore plus (taille des batchs, couches denses …)