Classez des images à l'aide d'algorithmes de Deep Learning

Objectif

- Réaliser un algorithme de détection de la race de chiens

Présentation du projet

Situation : bénévole pour une association de protection des animaux

Problématique : difficulté d'indexer les images des animaux de l'association

Objectif : développer un algorithme de détection de la race d'un chien

Avantage : amélioration de l'efficacité de l'indexation des images

Utilisation des données du Stanford Dogs Dataset pour entraîner le modèle

- http://vision.stanford.edu/aditya86/ImageNetDogs/
- 20580 images de chiens réparties en 120 races

Modèle de base

Sélection de 10 races

```python
# Création d'une partition de 80% des images pour l'entraînement
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset='training',
  seed=420,
  image_size=img_size,
  batch_size=batch_size,
  label_mode='categorical')

# Création d'une partition temporaire de 20% des images
temp_ds = tf.keras.preprocessing.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset='validation',
  seed=420,
  image_size=img_size,
  batch_size=batch_size,
  label_mode='categorical')

# Séparation en deux partitions égales (10%) de la partition temporaire pour la validation et les tests
val_ds = temp_ds.take(len(temp_ds)//2)
test_ds = temp_ds.skip(len(temp_ds)//2)

# Nombre de classes possible de classification
num_classes = len(train_ds.class_names)
```

Générateur

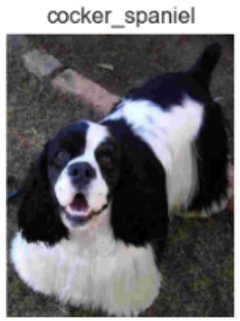
- traitement par répertoire
- séparation train / temp
- redimensionne les images
- chargement par batch
- categorical_crossentropy loss

Séparation val / test

Nombre de classes

Modèle de base

Sélection de 10 races

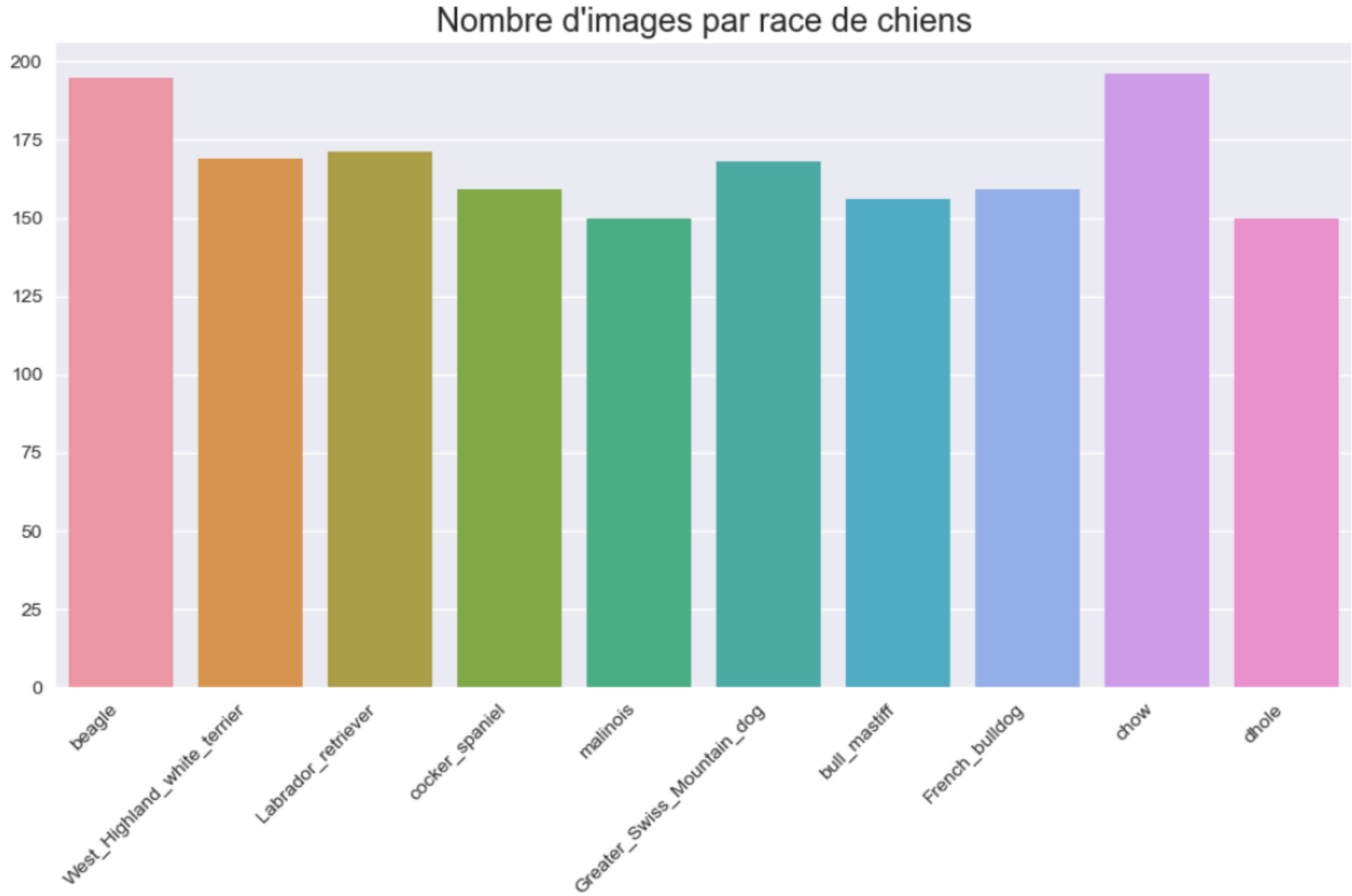


Modèle de base

Sélection de 10 races

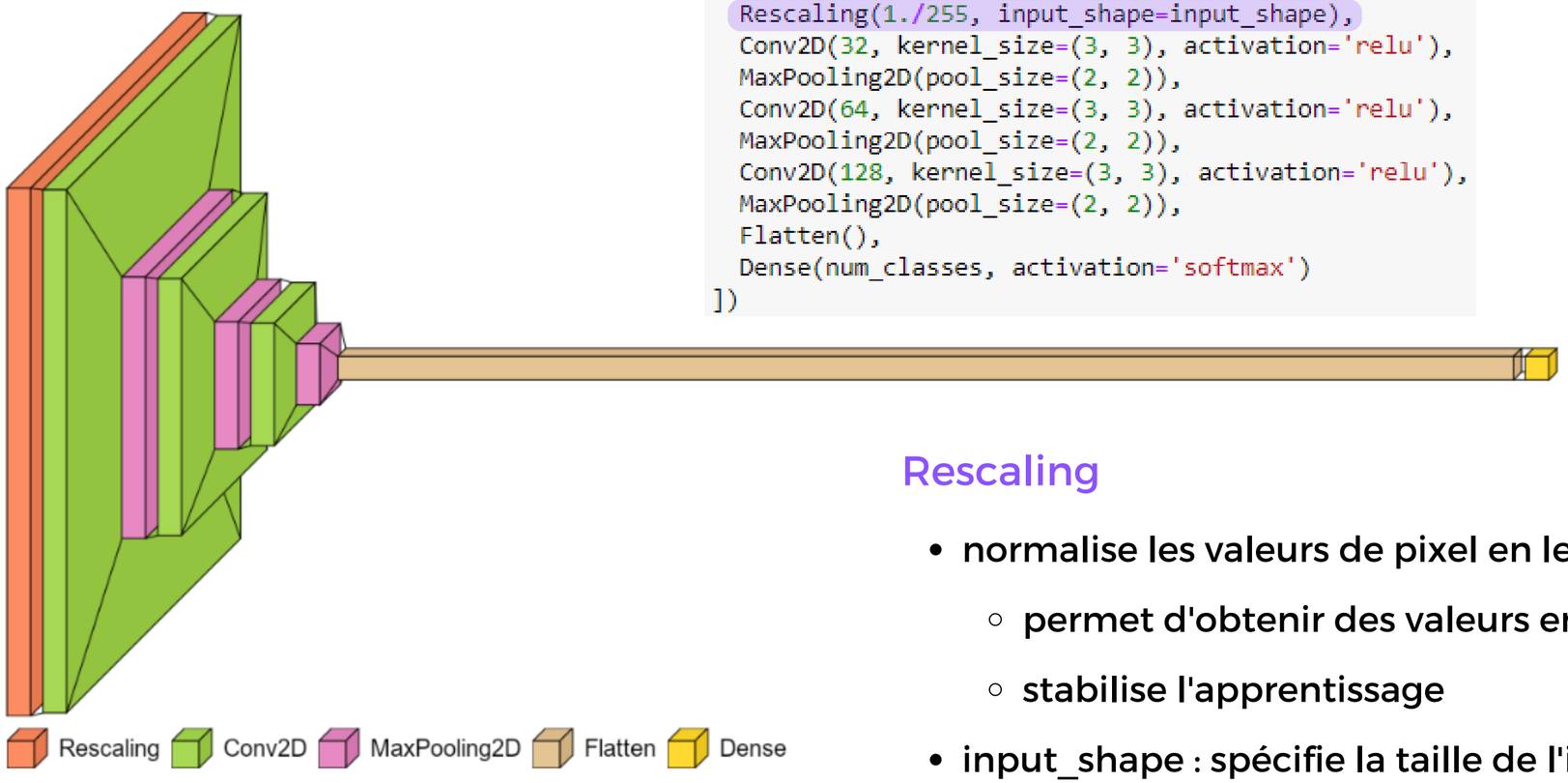
Grande variabilité en termes de taille, de résolution et de qualité

Modèle de base



Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

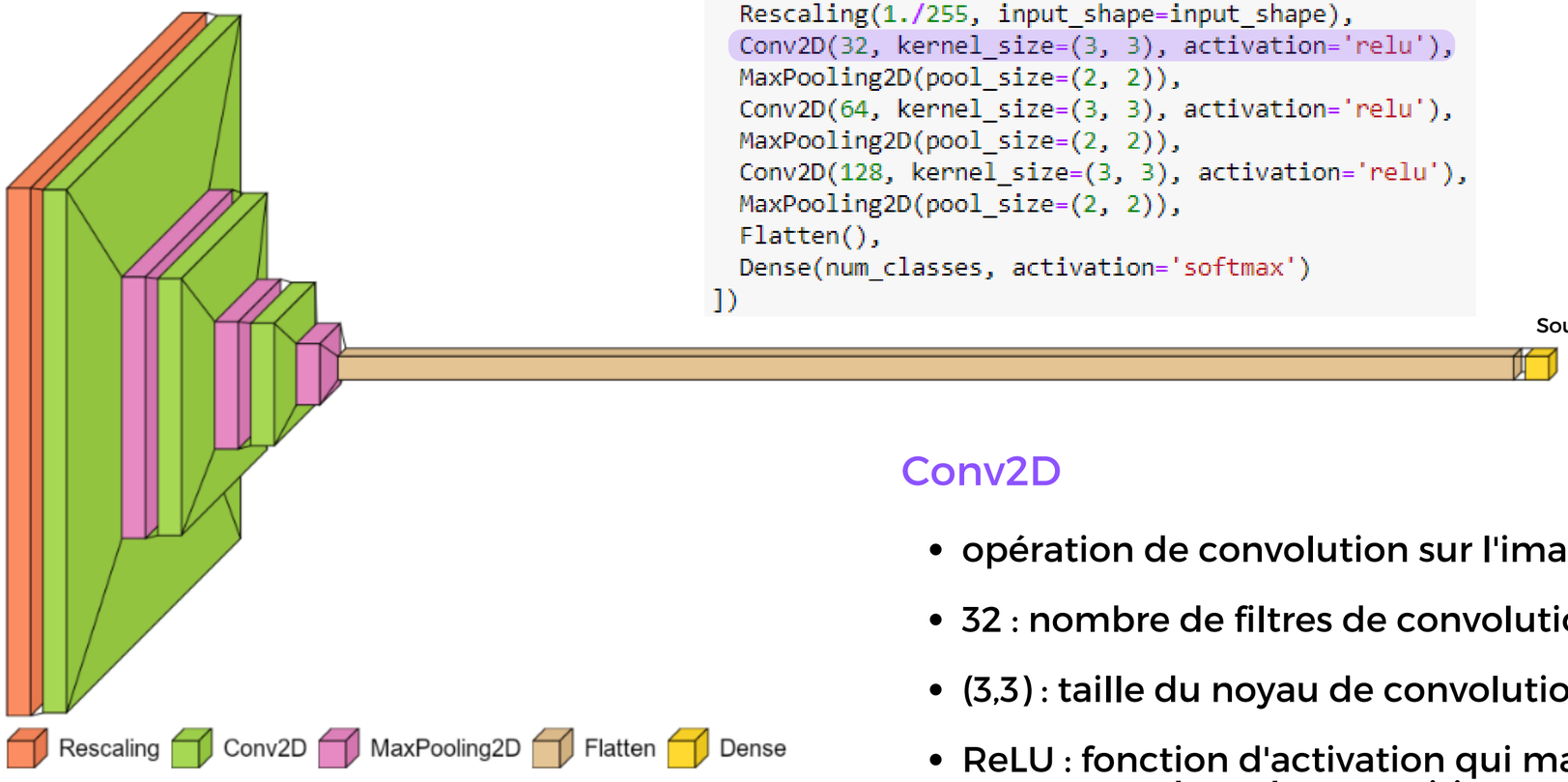


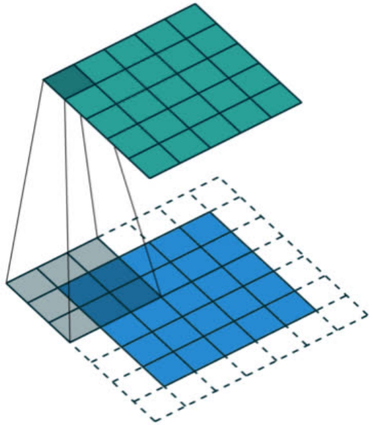
Rescaling

- normalise les valeurs de pixel en les divisant par 255
  - permet d'obtenir des valeurs entre 0 et 1
  - stabilise l'apprentissage
- input_shape : spécifie la taille de l'image d'entrée

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```



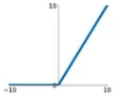


Source : https://github.com/vdumoulin/conv_arithmetic

Conv2D

- opération de convolution sur l'image d'entrée
- 32 : nombre de filtres de convolution appliqués
- (3,3) : taille du noyau de convolution
- ReLU : fonction d'activation qui mappe les valeurs négatives sur 0 et conserve les valeurs positives
- produit terme à terme (produit de Hadamard)

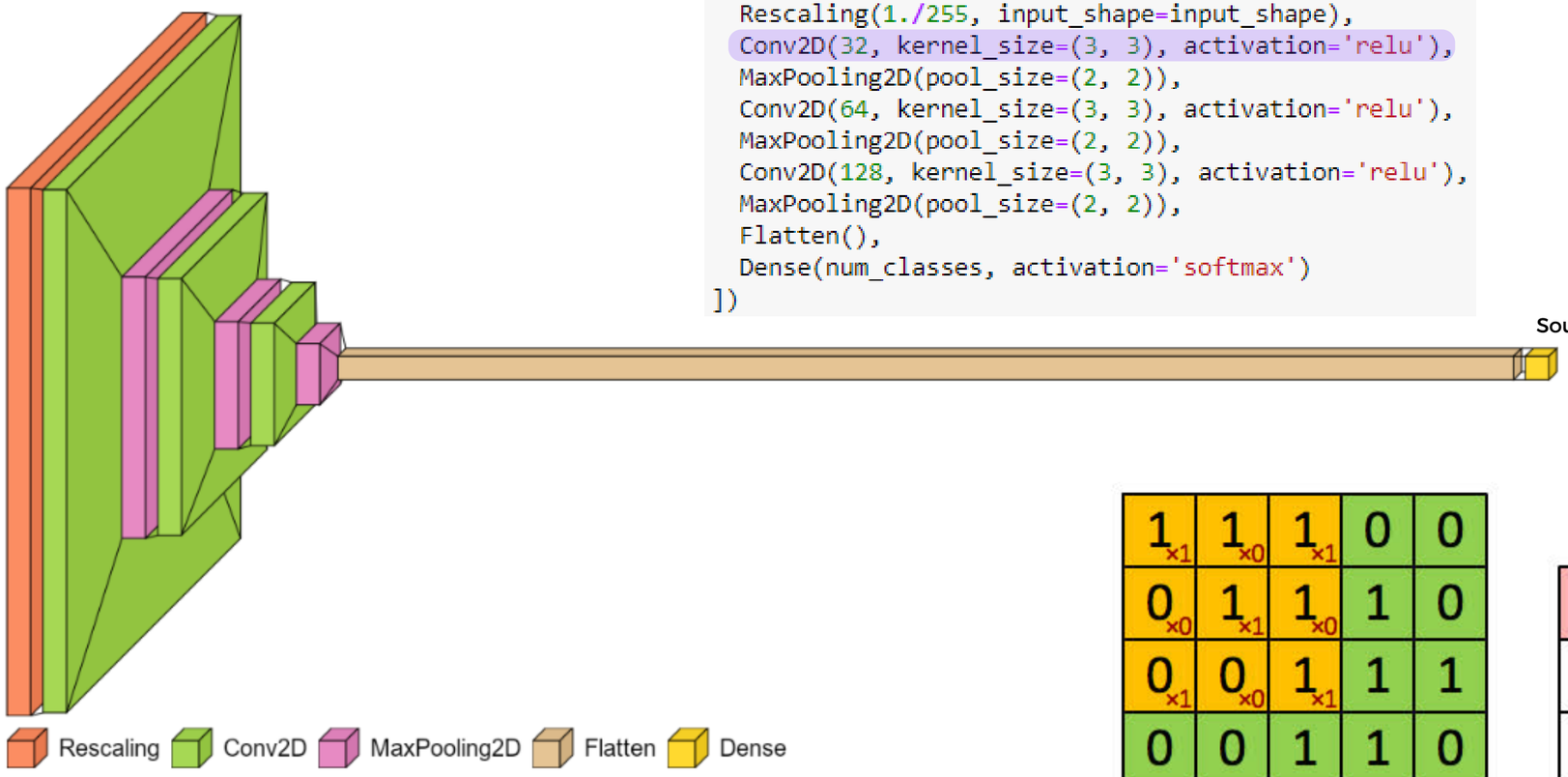
ReLU
$\max(0, x)$

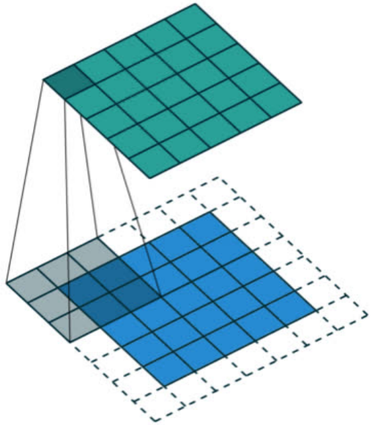


$$\underbrace{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}}_{\text{Une partie de l'image}} \odot \underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{Filtre de convolution}} = \underbrace{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{Partie de l'image convoluee}}$$

Modèle de base

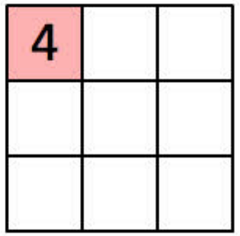
```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```





Source : https://github.com/vdumoulin/conv_arithmetic

Image

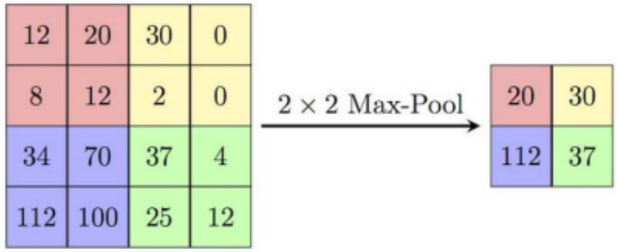


Convolved Feature

Source : IceCream Labs

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

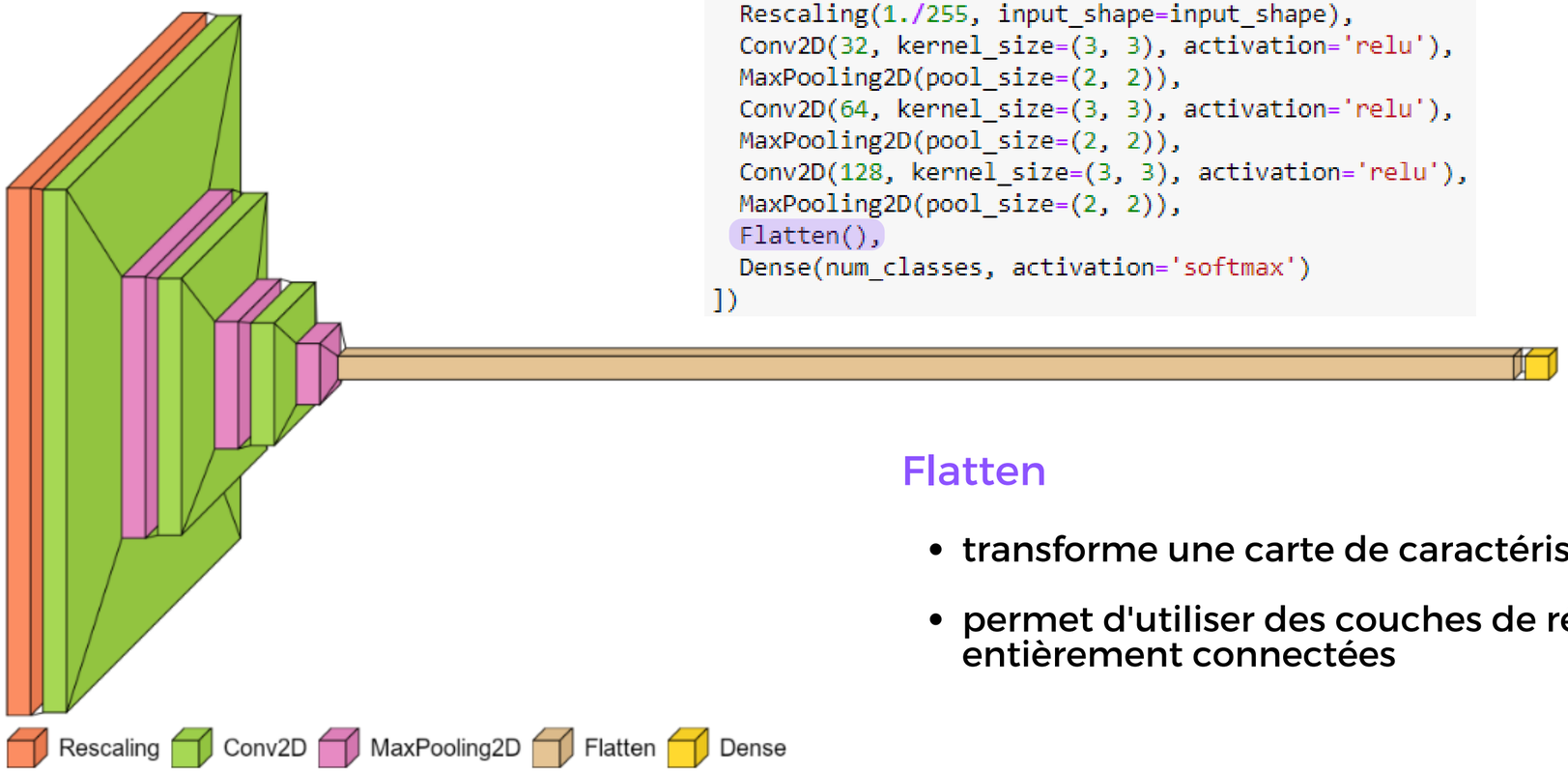


MaxPooling2D

- opération de sous-échantillonnage
- (2,2) : taille de la fenêtre
- permet de réduire la taille spatiale de la carte de caractéristiques tout en préservant les caractéristiques les plus importantes
  - améliore l'efficacité du réseau
  - évite le sur-apprentissage

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

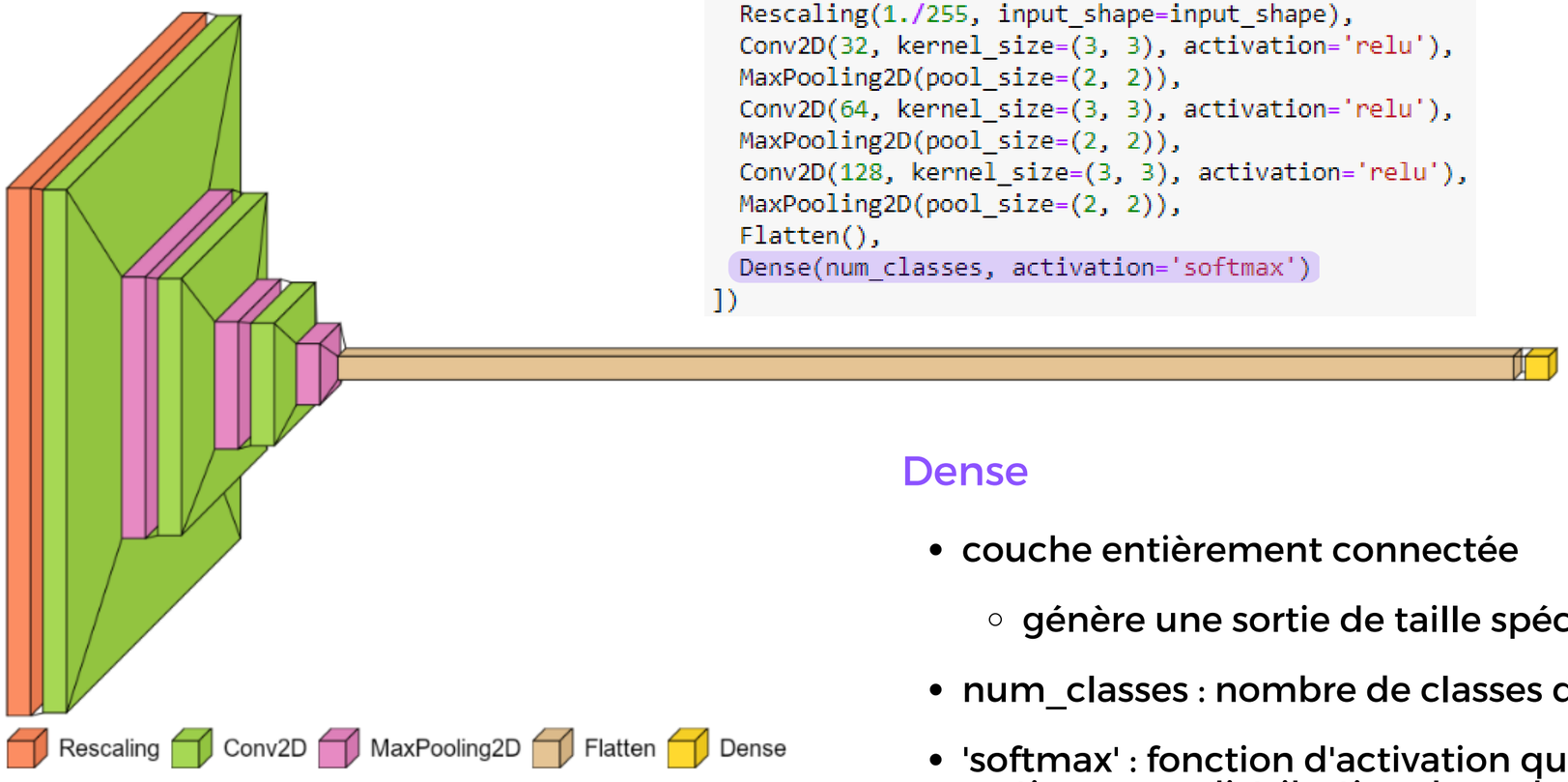


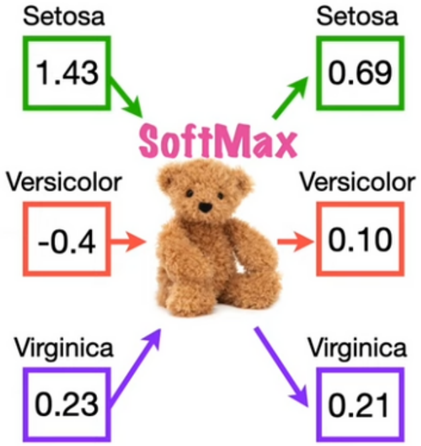
Flatten

- transforme une carte de caractéristiques 2D en un vecteur 1D
- permet d'utiliser des couches de réseau de neurones entièrement connectées

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```



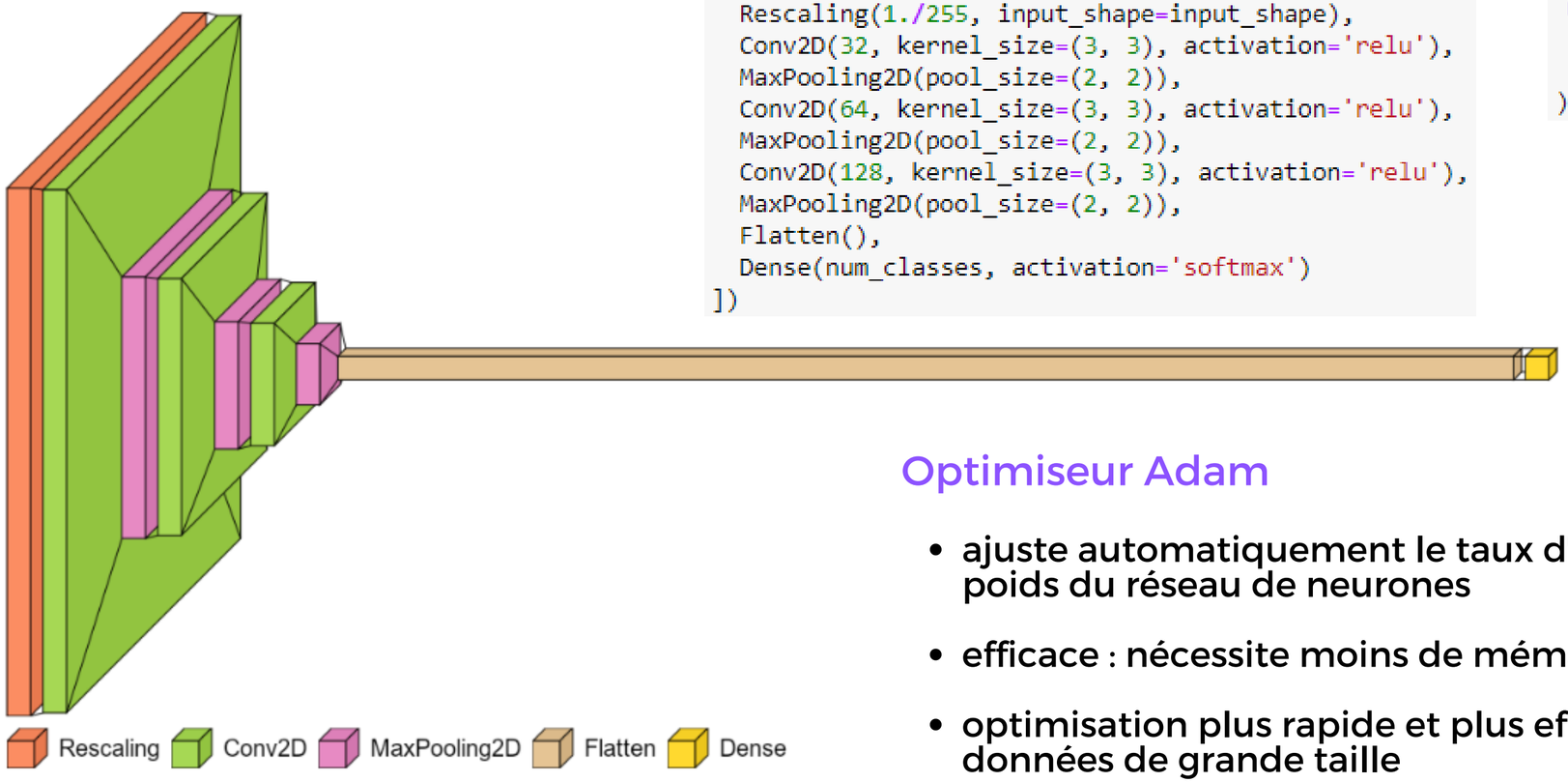


Source : StatQuestwithJoshStarmer (YouTube)

Dense

- couche entièrement connectée
  - génère une sortie de taille spécifiée
- num_classes : nombre de classes de sortie
- 'softmax' : fonction d'activation qui normalise les valeurs de sortie en une distribution de probabilité
  - idéal pour la classification multi-classes
  - $\text{SoftMax}_i(\text{Output Values}) = \frac{e^{\text{Output Value}_i}}{\sum_{j=1}^{k} e^{\text{Output Value}_j}}$

Modèle de base




```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

```
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['categorical_accuracy']
)
```

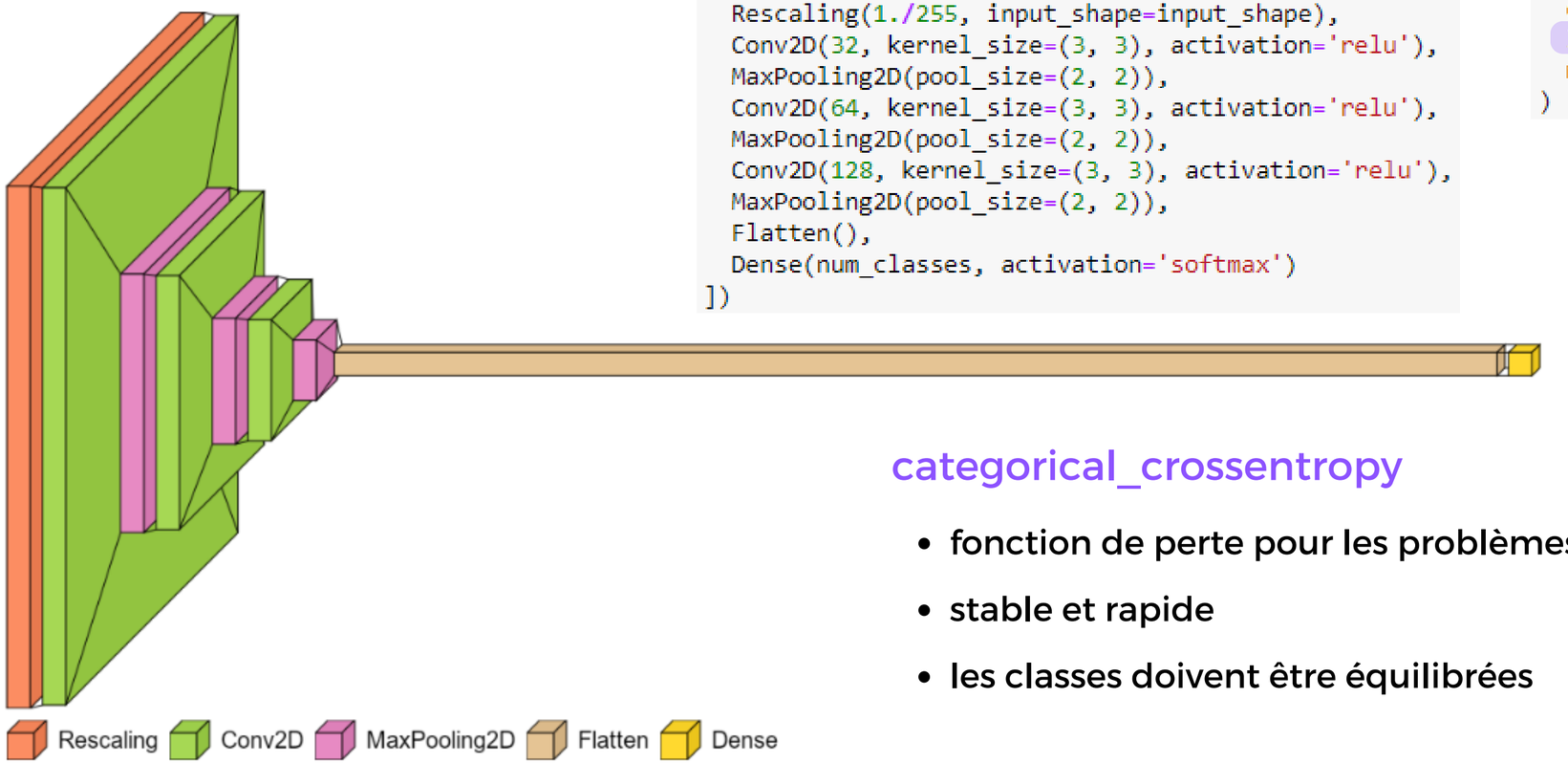
Optimiseur Adam

- ajuste automatiquement le taux d'apprentissage pour chaque poids du réseau de neurones
- efficace : nécessite moins de mémoire
- optimisation plus rapide et plus efficace pour les ensembles de données de grande taille

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

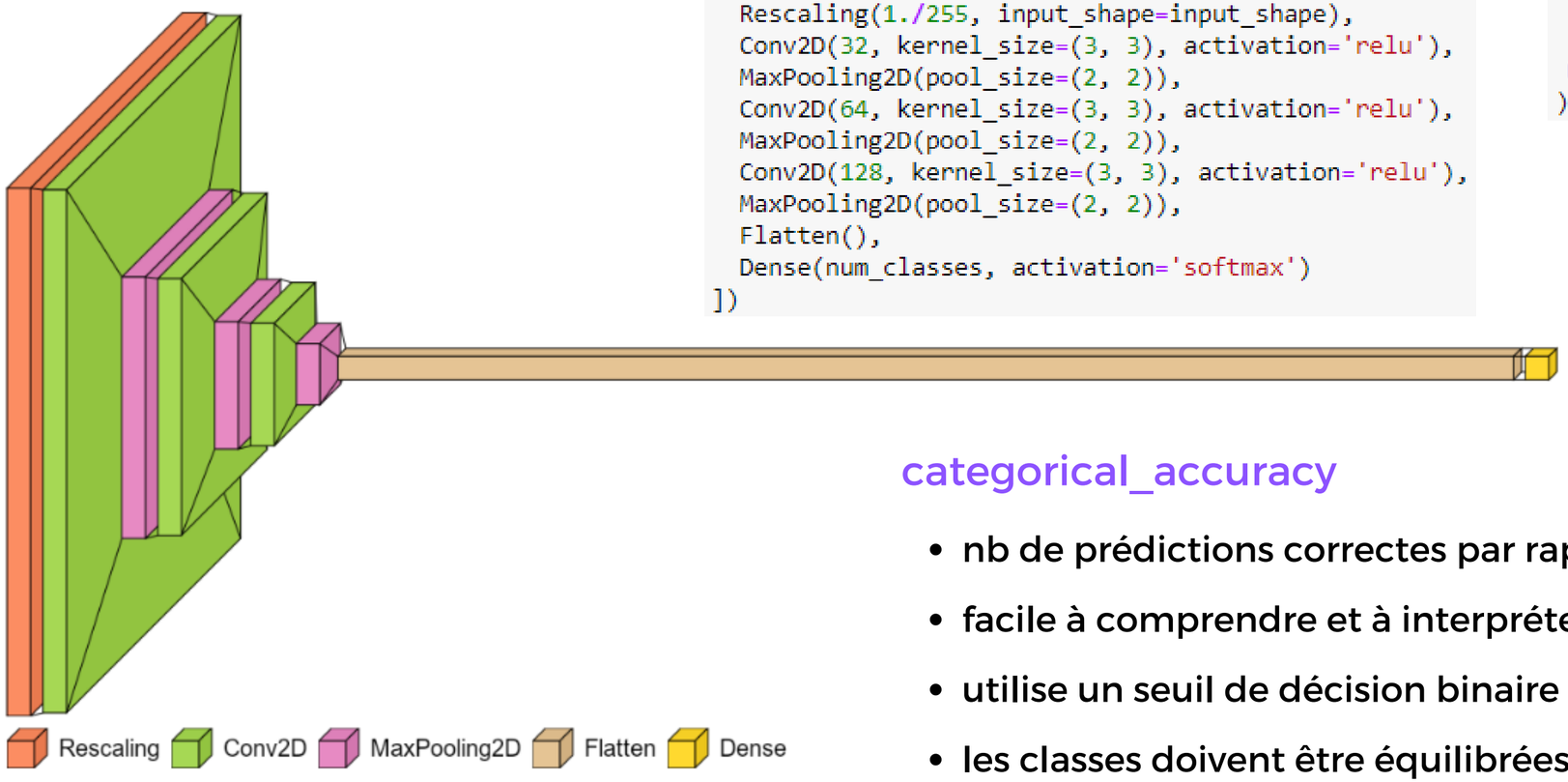
```
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['categorical_accuracy']
)
```



categorical_crossentropy

- fonction de perte pour les problèmes de classification multi-classes
- stable et rapide
- les classes doivent être équilibrées

Modèle de base




```python
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

```python
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['categorical_accuracy']
)
```

categorical_accuracy

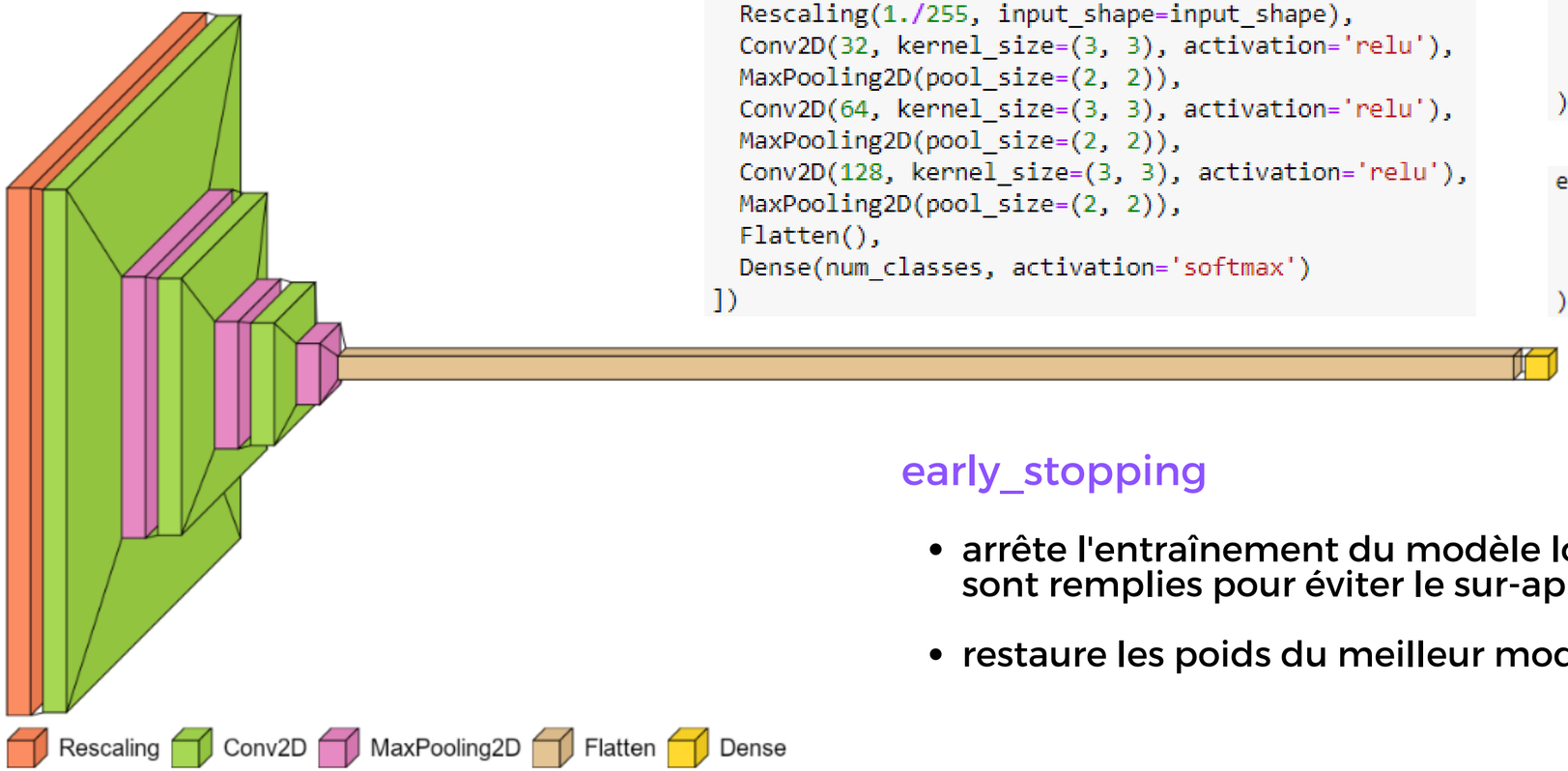
- nb de prédictions correctes par rapport au nb total de prédictions
- facile à comprendre et à interpréter
- utilise un seuil de décision binaire
- les classes doivent être équilibrées

Modèle de base

```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

```
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['categorical_accuracy']
)
```

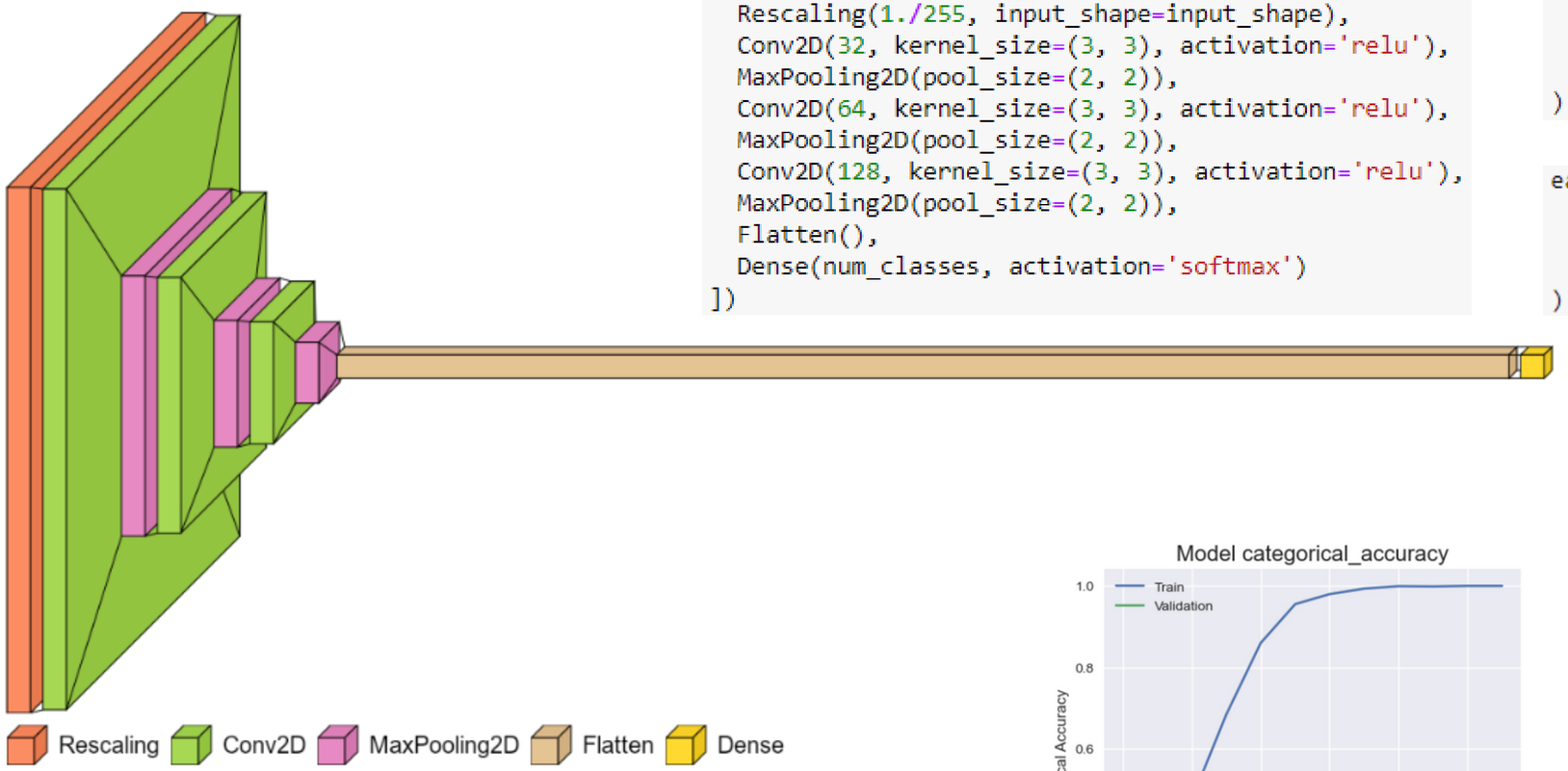
```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```



early_stopping

- arrête l'entraînement du modèle lorsque certaines conditions sont remplies pour éviter le sur-apprentissage.
- restaure les poids du meilleur modèle

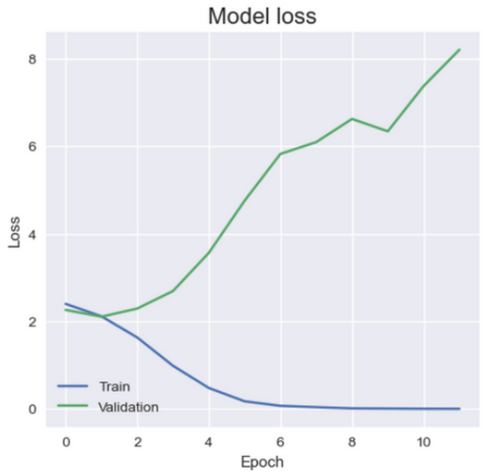
Modèle de base




```
model = Sequential([
  Rescaling(1./255, input_shape=input_shape),
  Conv2D(32, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(64, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Conv2D(128, kernel_size=(3, 3), activation='relu'),
  MaxPooling2D(pool_size=(2, 2)),
  Flatten(),
  Dense(num_classes, activation='softmax')
])
```

```
model.compile(
  optimizer='adam',
  loss='categorical_crossentropy',
  metrics=['categorical_accuracy']
)
```

```
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)
```

Test accuracy: 0.24137930572032928

Modèle de base

Preprossessing des images

- **Redimensionnement et normalisation des images**

Objectif : améliore la qualité et l'efficacité de l'entraînement des algos

Modèle de base

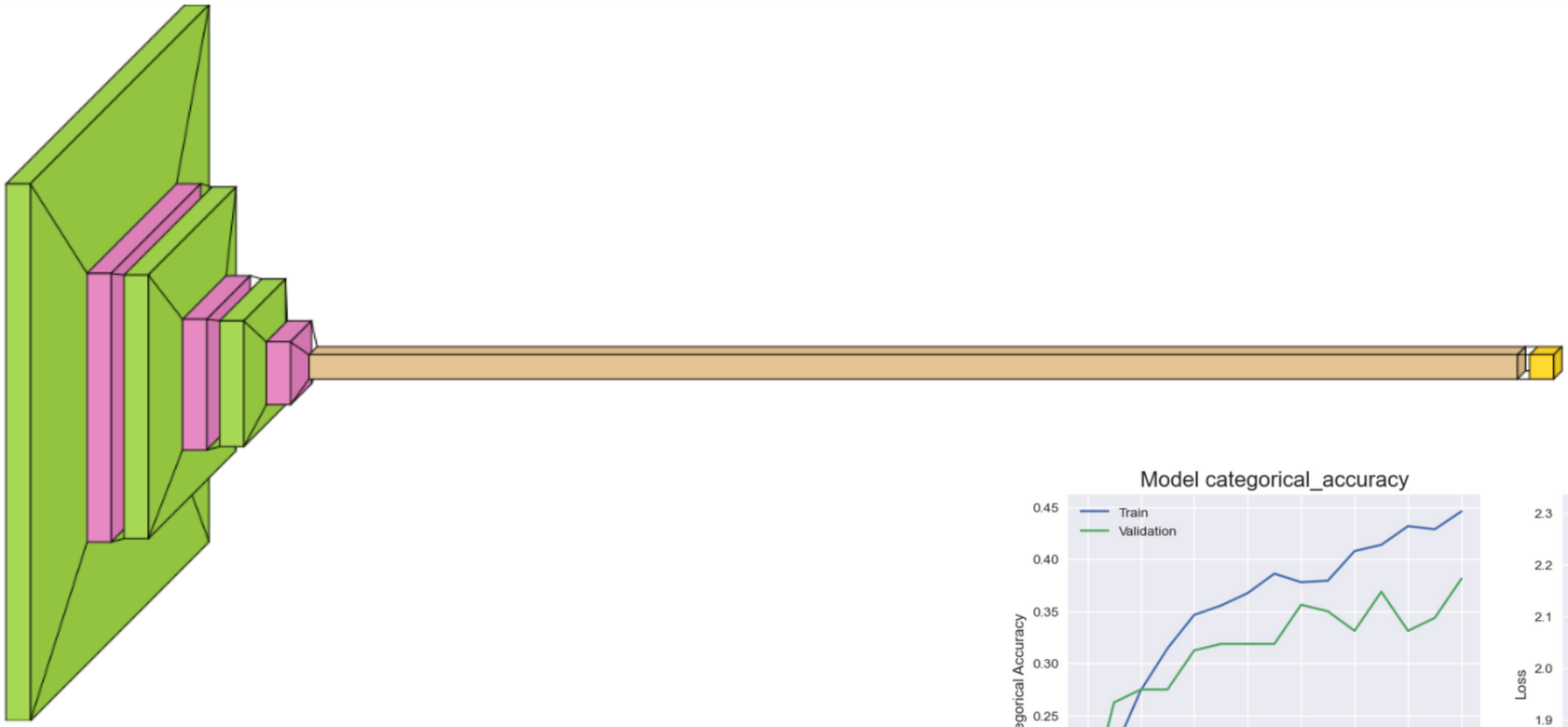
Preprossessing des images

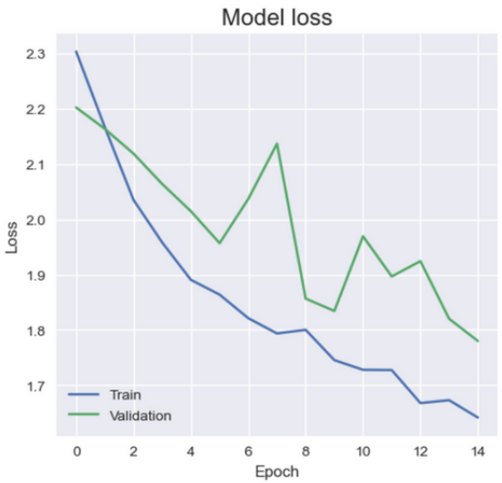
- **Data Augmentation**

```
RandomFlip("horizontal"),
RandomRotation(0.1),
RandomZoom(0.1),
RandomContrast(0.1),
RandomCrop(img_size[0], img_size[1]),
RandomTranslation(height_factor=0.1, width_factor=0.1),
```

Objectif : apporte davantage de diversité au dataset

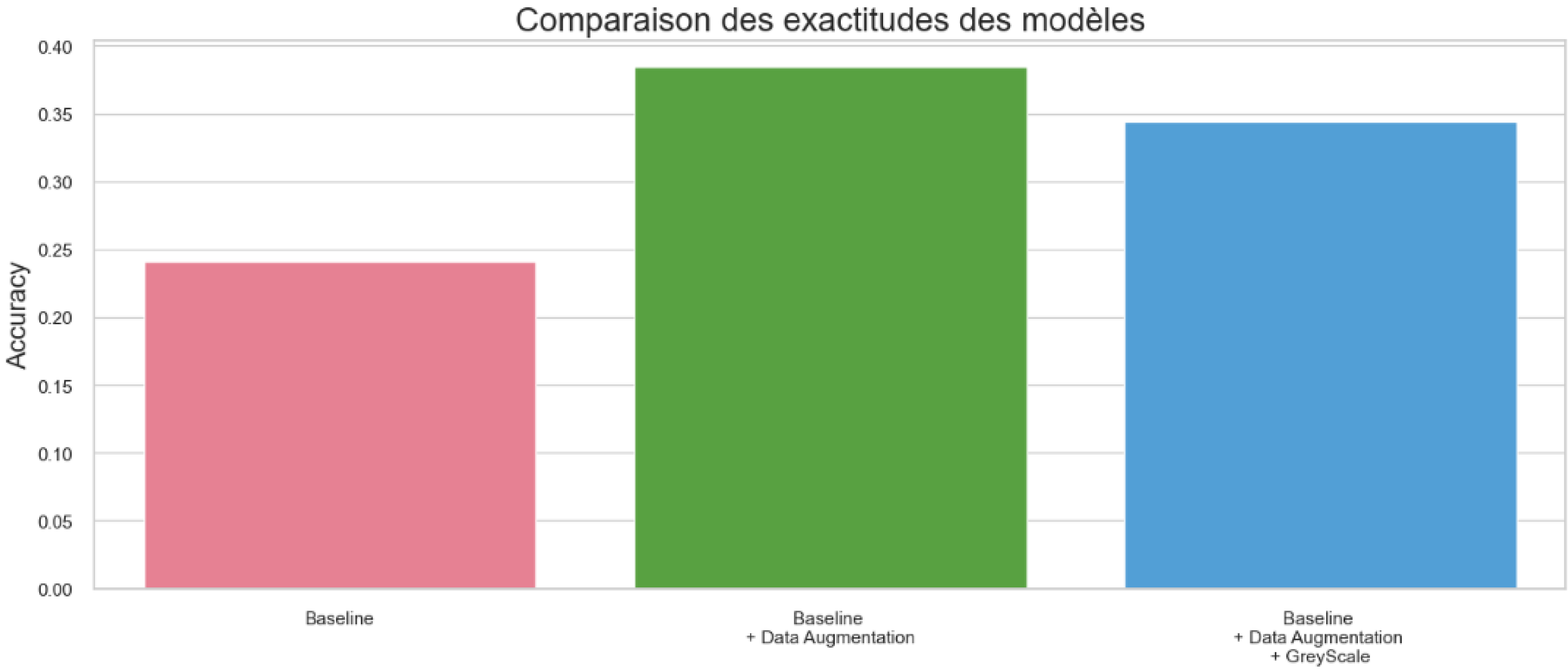
Modèle de base





Test accuracy: 0.3850574791431427

Modèle de base

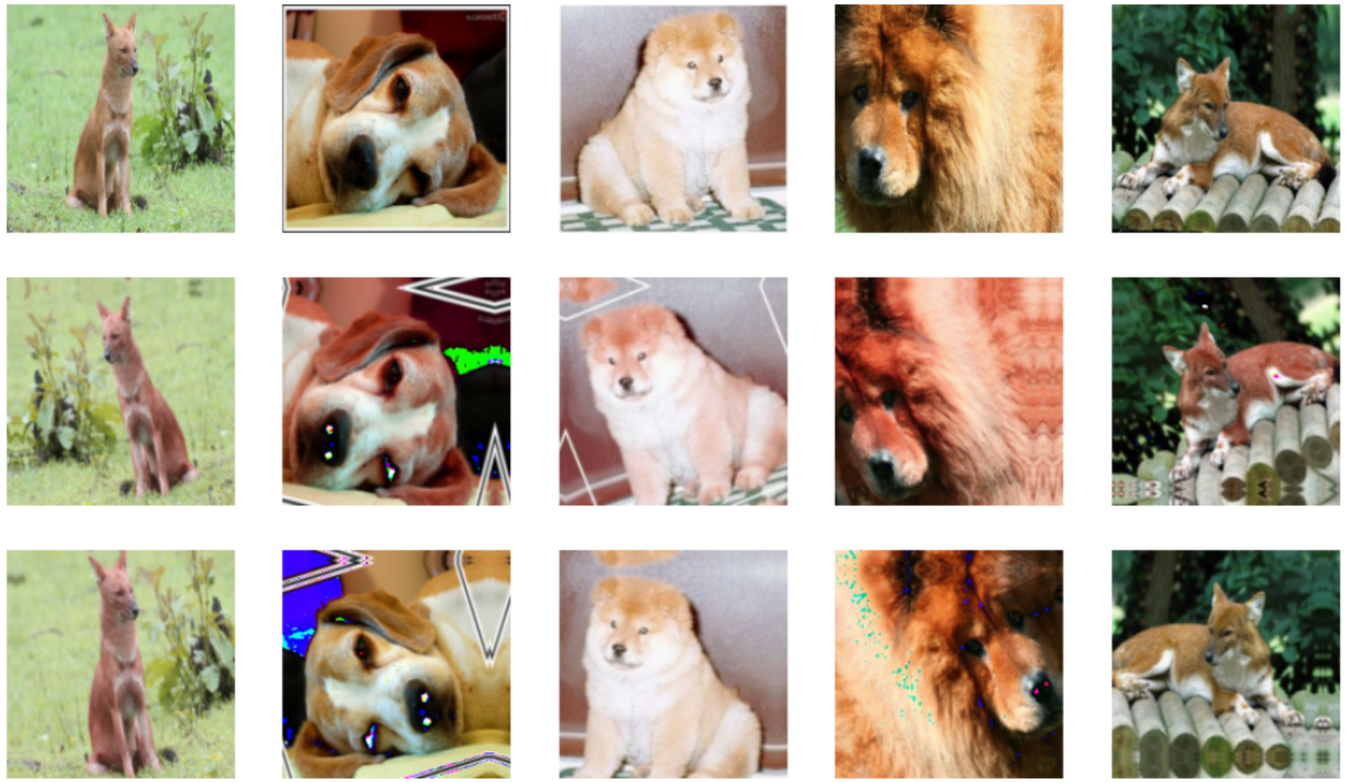


Création d'un CNN "from scratch"

Data Augmentation modifiée

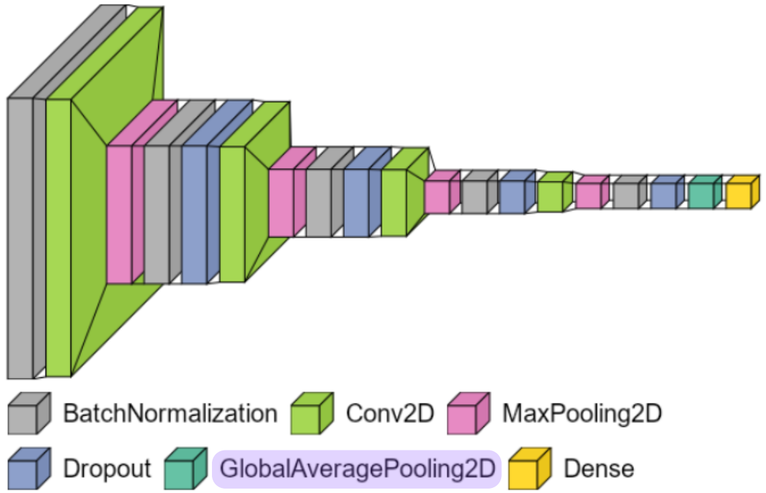
```
data_augmentation = Sequential(
    [
        RandomFlip("horizontal"),
        RandomRotation(0.1),
        RandomZoom(0.1),
        RandomContrast(0.05),
        RandomCrop(img_size[0], img_size[1]),
        RandomTranslation(height_factor=0.1, width_factor=0.1),
    ]
)
```

```
def preprocess_image_with_augmentation(image, label):
    image = tf.image.resize(image, img_size)
    image = tf.image.random_brightness(image, 1.0)
    image = tf.image.random_hue(image, 0.05)
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_saturation(image, 0.95, 1.05)
    image = data_augmentation(image)
    image = tf.cast(image, tf.float32) / 255.0
    return image, label
```



Création d'un CNN "from scratch"

Hyperparamétrage à la main à partir du modèle AlexNet



BatchNormalization

- normalise les activations de chaque couche
- aider à stabiliser le processus d'apprentissage

Dropout

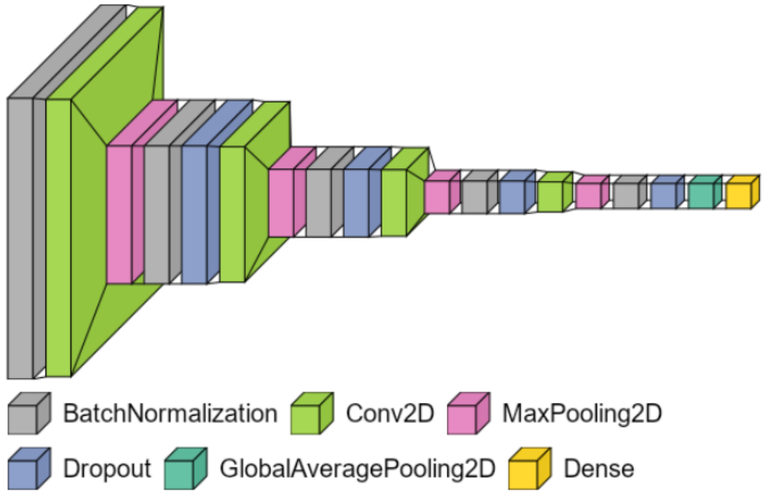
- désactive aléatoirement certains neurones
- réduire le sur-apprentissage en réduisant les interdépendances entre les neurones.

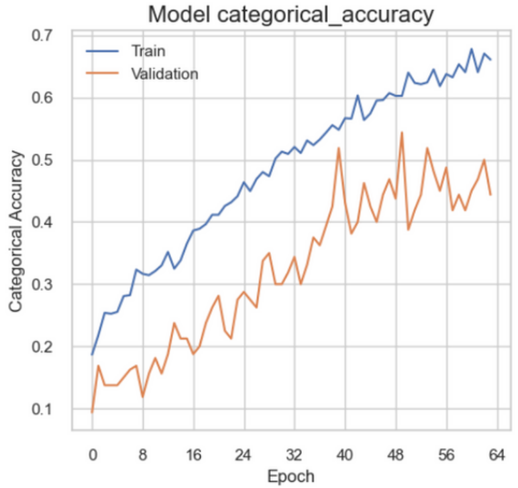
GlobalAveragePooling2D

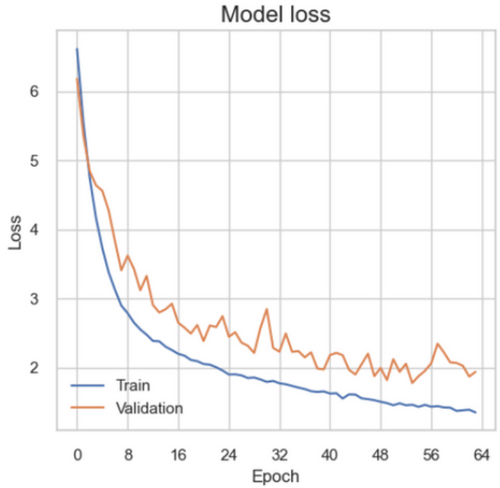
- calcule la moyenne des activations de chaque carte de caractéristiques
- permet de réduire la nb de paramètres (contre l'overfitting)

Création d'un CNN "from scratch"

Hyperparamétrage à la main à partir du modèle AlexNet







Definition des hyperparamètres à rechercher

```
param_grid = {
    'num_filters': [[16, 32, 64, 128], [32, 64, 128, 256]],
    'k_sizes': [[5, 3, 3, 3], [4, 3, 2, 3], [5, 3, 3, 2], [3, 3, 3, 3]],
    'k_init': ['he_normal', 'glorot_uniform', 'random_normal'],
    'l2_value': [0.01, 0.02, 0.03],
    'pool_sizes': [[(2, 2), (2, 2), (2, 2), (2, 2)], [(3, 3), (2, 2), (2, 2), (2, 2)]],
    'dropout': [0.20, 0.25, 0.5, 0.75]
}
```

Nombre de combinaisons : 576

Utilisation GPU ~40 heures

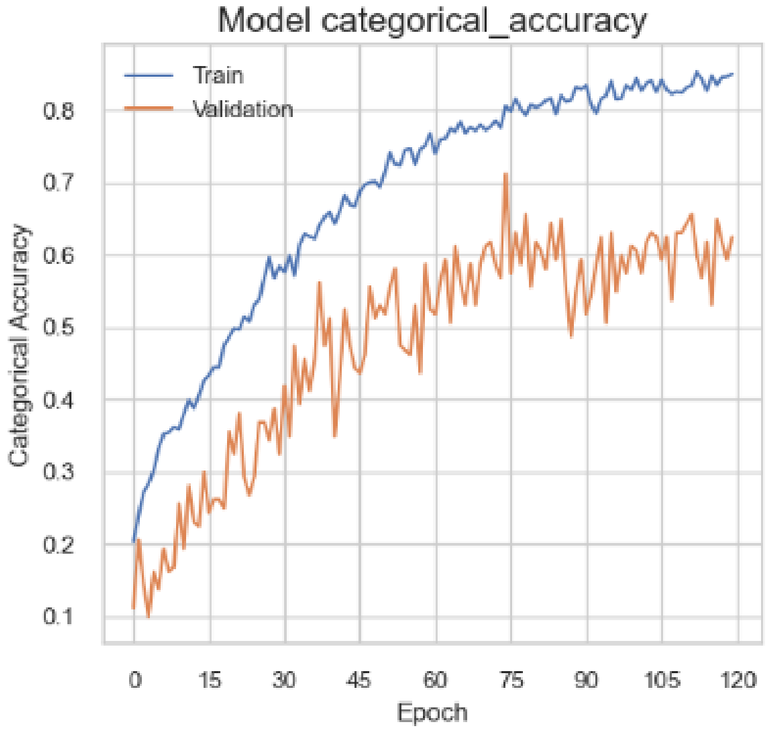
Création d'un CNN "from scratch"

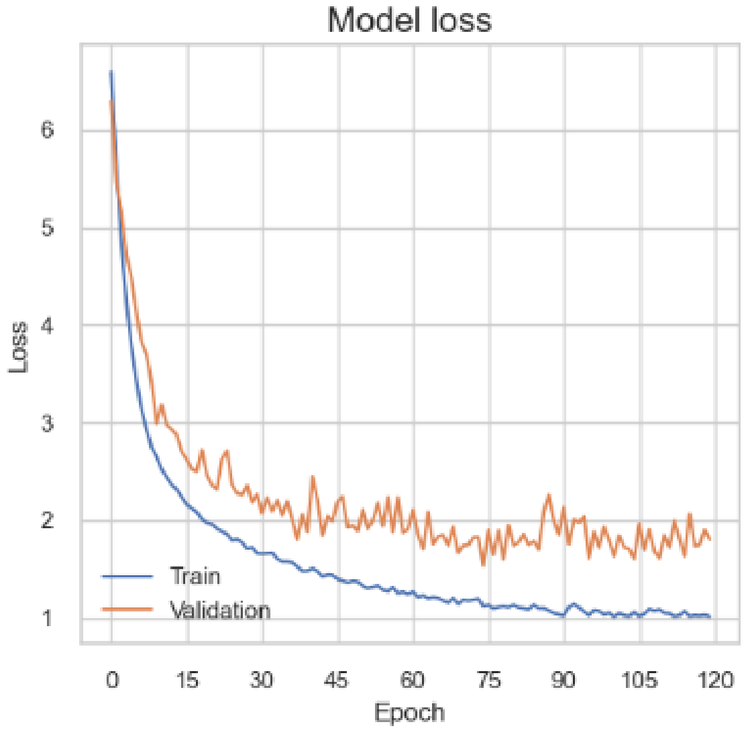
Meilleurs hyperparamètres

```
Best hyperparameters: ([16, 32, 64, 128], [5, 3, 3, 3], 'he_normal', 0.01, [(3, 3), (2, 2), (2, 2), (2, 2)], 0.25)
Best validation accuracy: 0.59
```

Layer (type)	Output Shape	Param #
batch_normalization_115 (BatchNormalization)	(None, 224, 224, 3)	12
conv2d_101 (Conv2D)	(None, 220, 220, 16)	1216
max_pooling2d_101 (MaxPooling2D)	(None, 73, 73, 16)	0
batch_normalization_116 (BatchNormalization)	(None, 73, 73, 16)	64
dropout_92 (Dropout)	(None, 73, 73, 16)	0
conv2d_102 (Conv2D)	(None, 71, 71, 32)	4640
max_pooling2d_102 (MaxPooling2D)	(None, 35, 35, 32)	0
batch_normalization_117 (BatchNormalization)	(None, 35, 35, 32)	128
dropout_93 (Dropout)	(None, 35, 35, 32)	0
conv2d_103 (Conv2D)	(None, 33, 33, 64)	18496
max_pooling2d_103 (MaxPooling2D)	(None, 16, 16, 64)	0
batch_normalization_118 (BatchNormalization)	(None, 16, 16, 64)	256
dropout_94 (Dropout)	(None, 16, 16, 64)	0
conv2d_104 (Conv2D)	(None, 14, 14, 128)	73856
max_pooling2d_104 (MaxPooling2D)	(None, 7, 7, 128)	0
batch_normalization_119 (BatchNormalization)	(None, 7, 7, 128)	512
dropout_95 (Dropout)	(None, 7, 7, 128)	0
global_average_pooling2d_23 (GlobalAveragePooling2D)	(None, 128)	0
dense_26 (Dense)	(None, 10)	1290

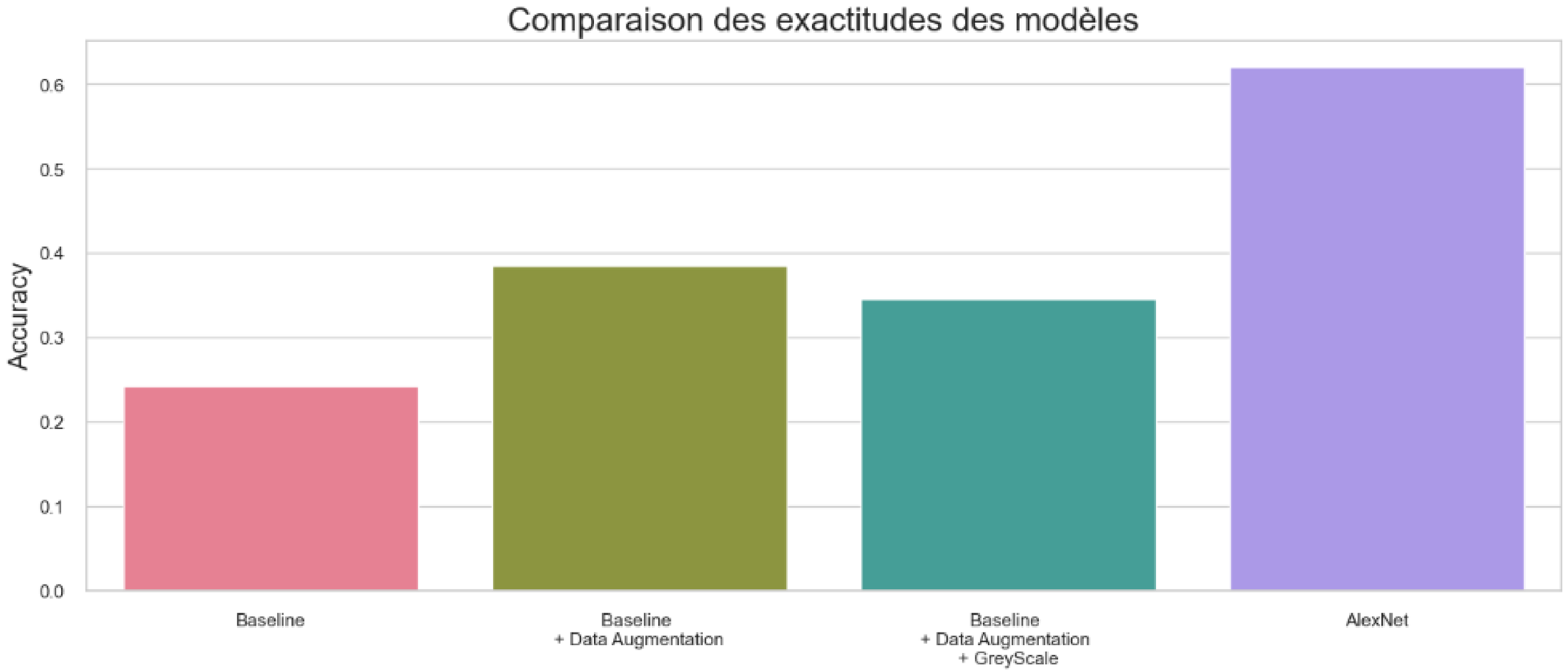
Total params: 100,470
Trainable params: 99,984
Non-trainable params: 486






```
Test accuracy: 0.6206896305084229
```

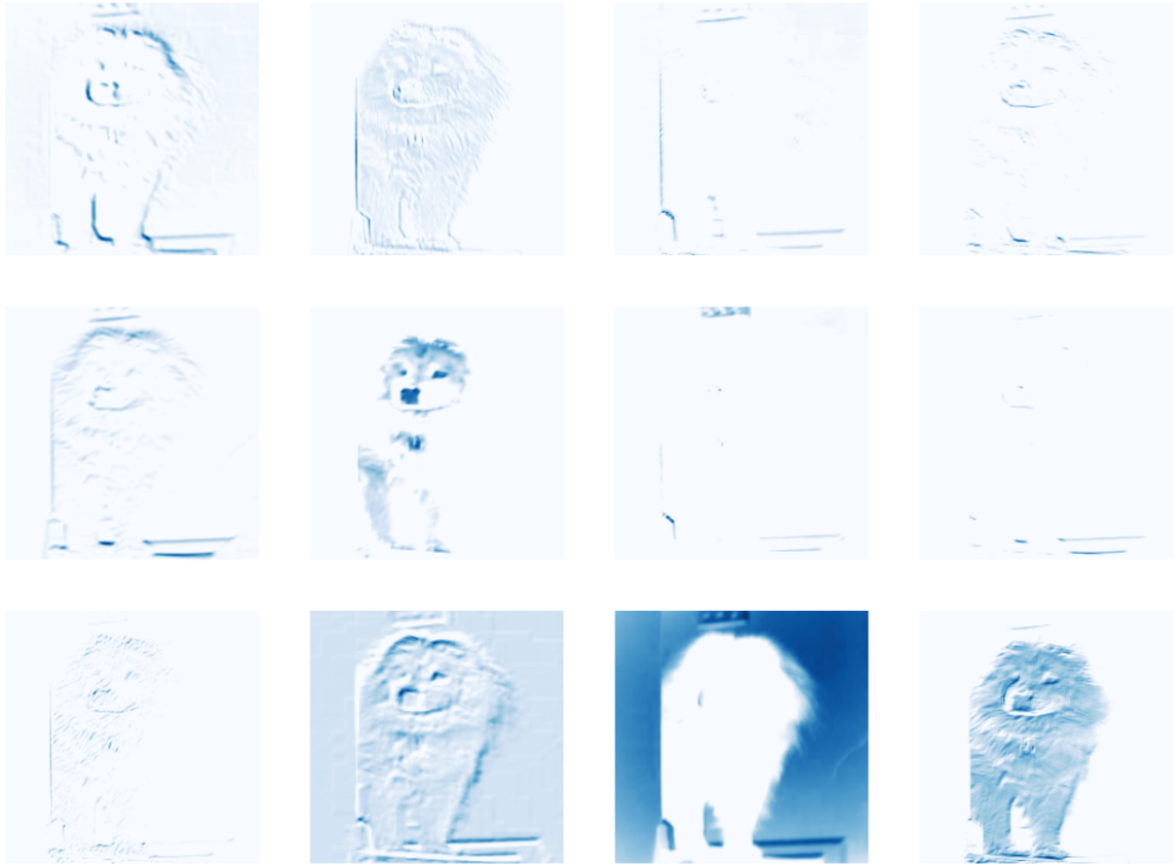
Création d'un CNN "from scratch"



Création d'un CNN "from scratch"

Observation de 12 Features Maps (cartes de caractéristiques)

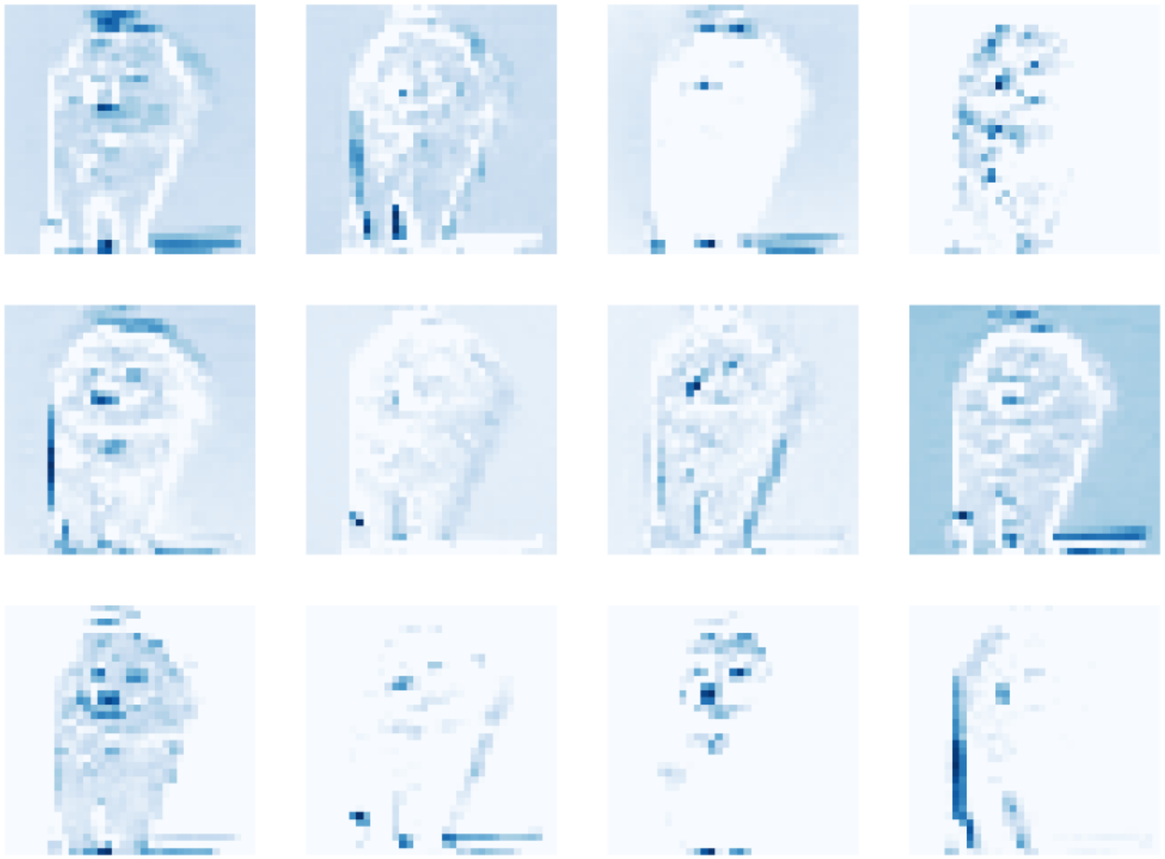
- **1ère convolution (contours, bords)**



Création d'un CNN "from scratch"

Observation de 12 Features Maps (cartes de caractéristiques)

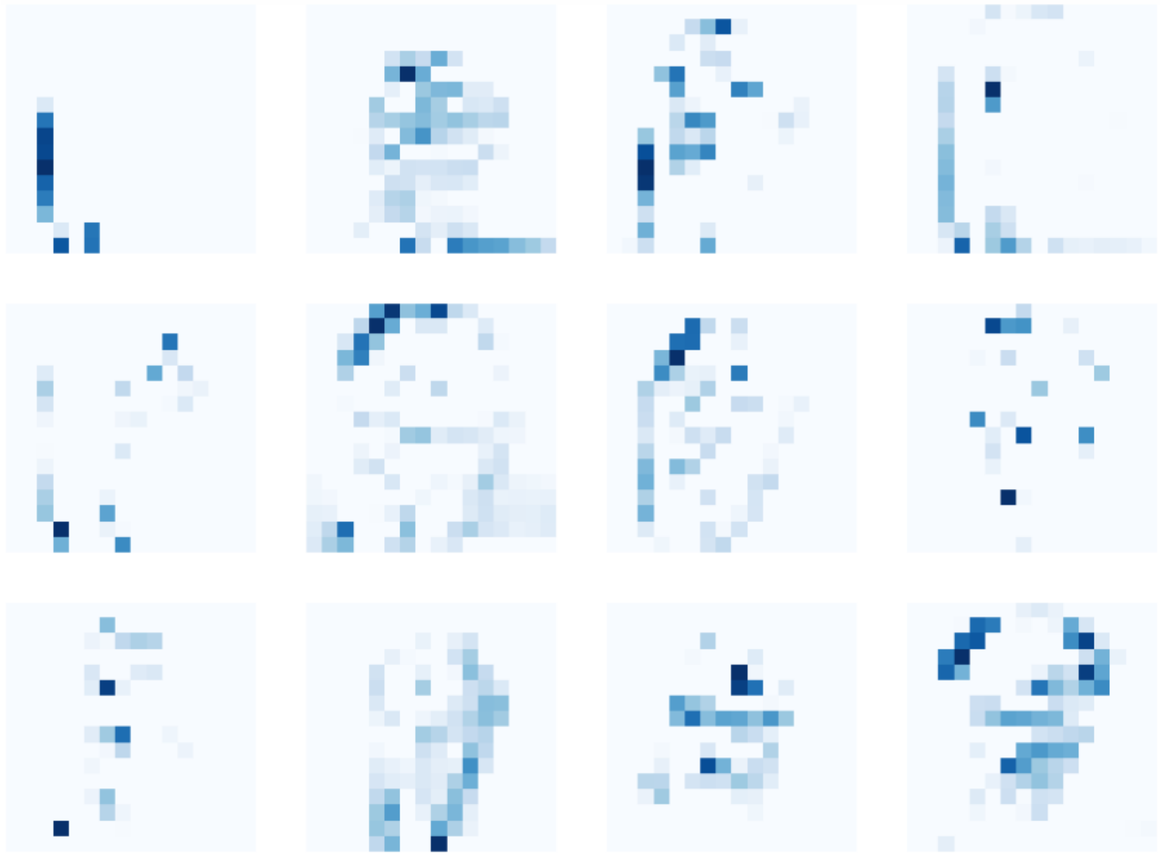
- **2ème convolution**



Création d'un CNN "from scratch"

Observation de 12 Features Maps (cartes de caractéristiques)

- **3ème convolution (plus abstrait)**

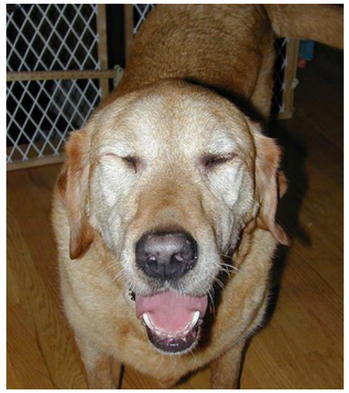


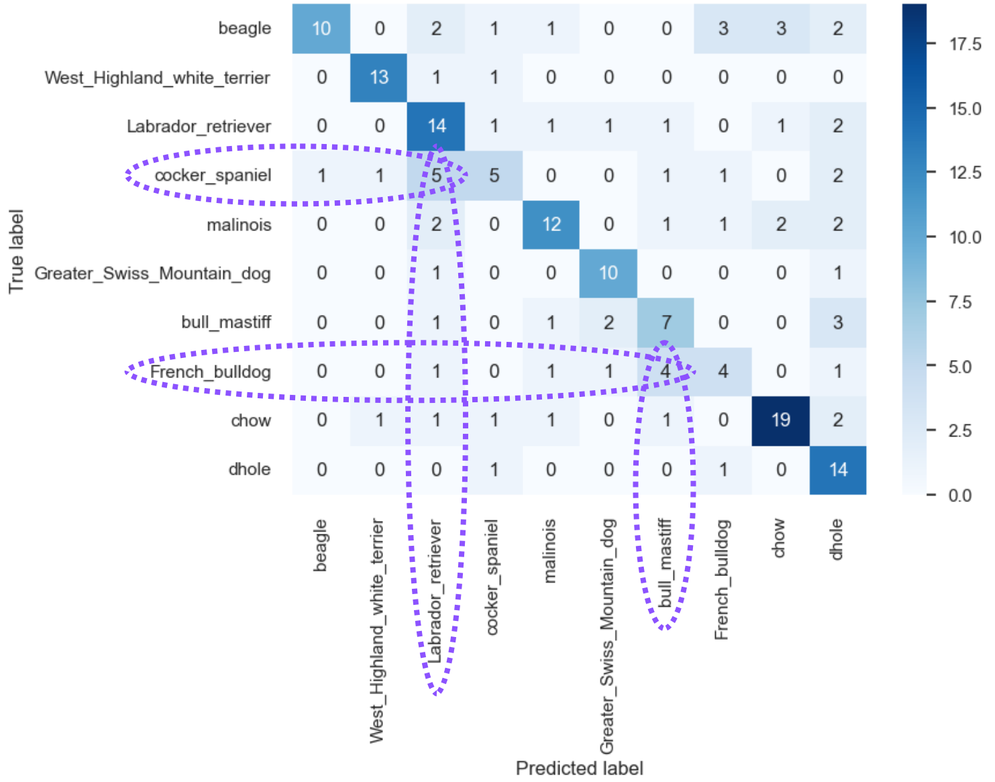
Création d'un CNN "from scratch"

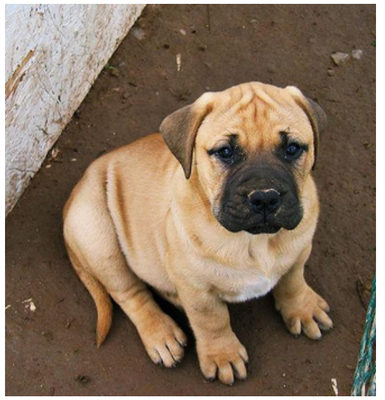
Analyse des résultats

Création d'un CNN "from scratch"

Analyse des résultats

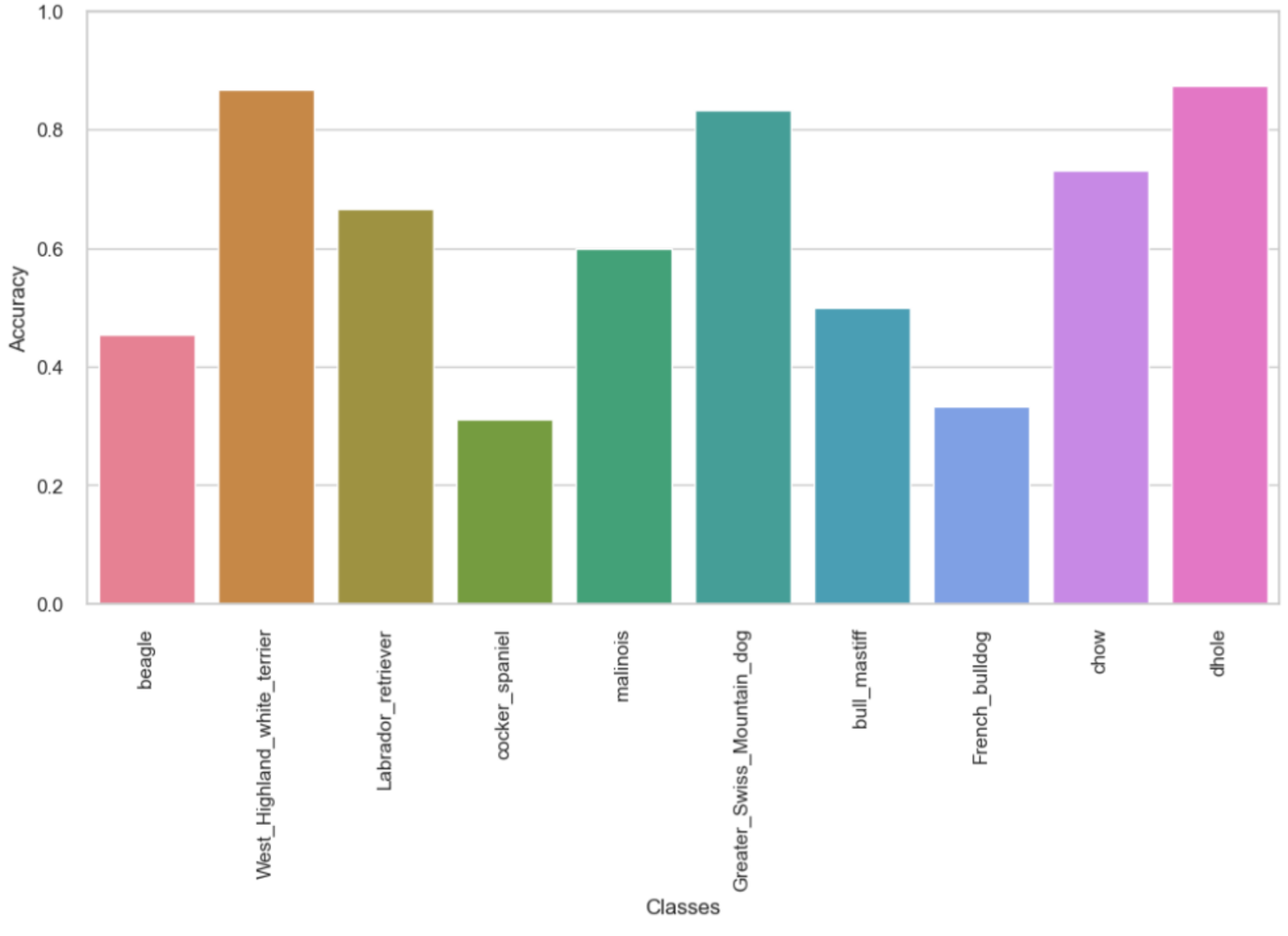






Création d'un CNN "from scratch"

Analyse des résultats



Modèles avec Transfer Learning

Xception (Google - 2017)

Test accuracy: 0.9712643623352051

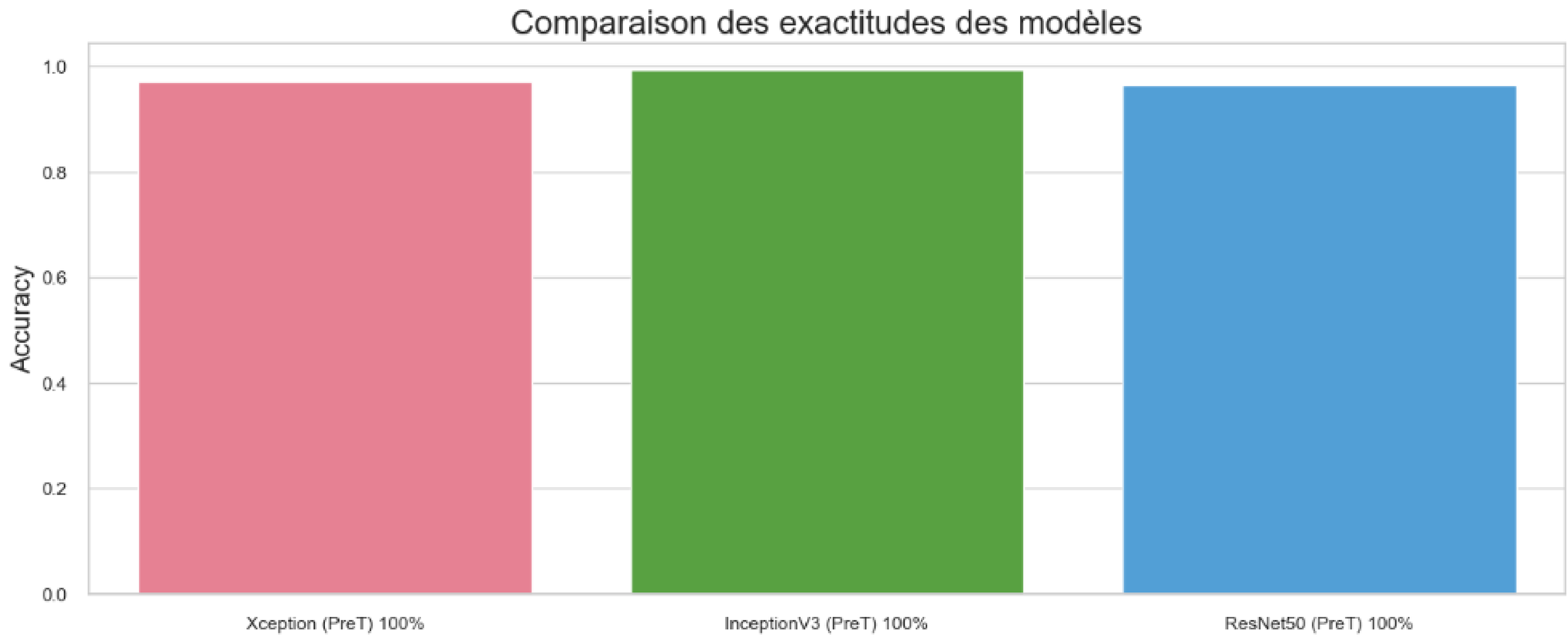
InceptionV3 (Google - 2015)

Test accuracy: 0.9942528605461121

ResNet50 (Microsoft - 2015)

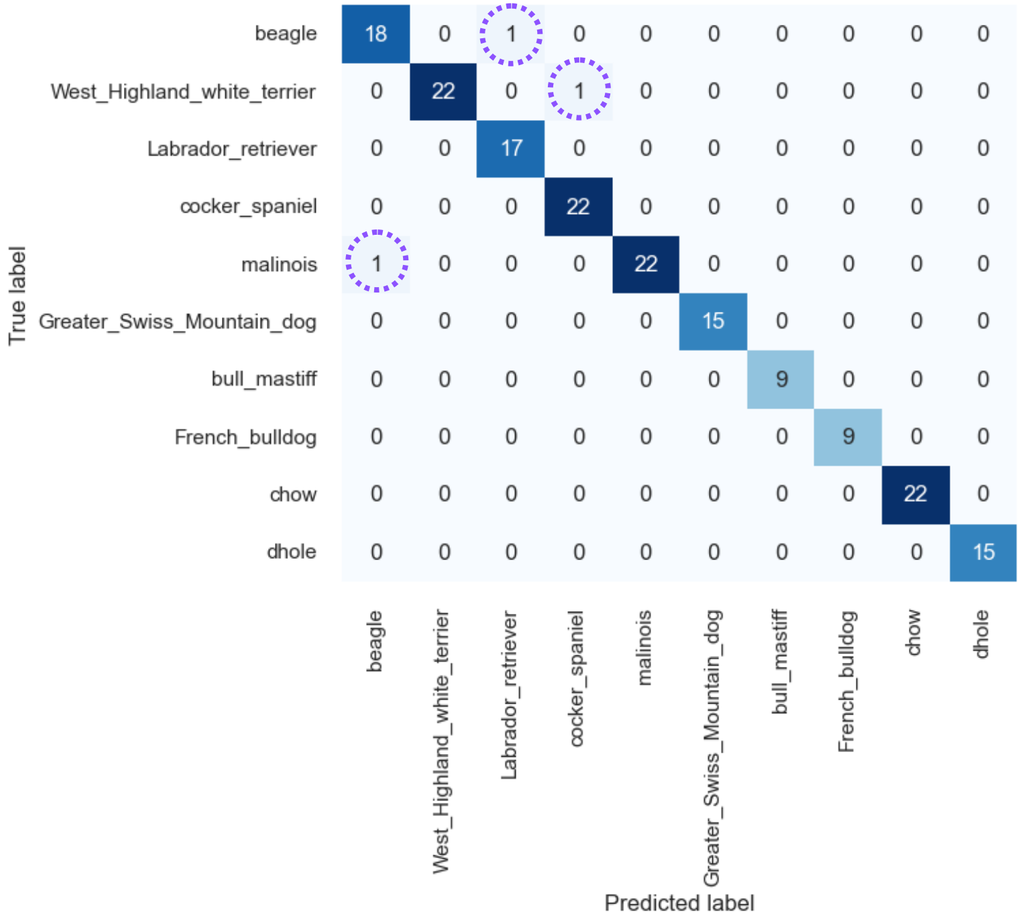
Test accuracy: 0.9655172228813171

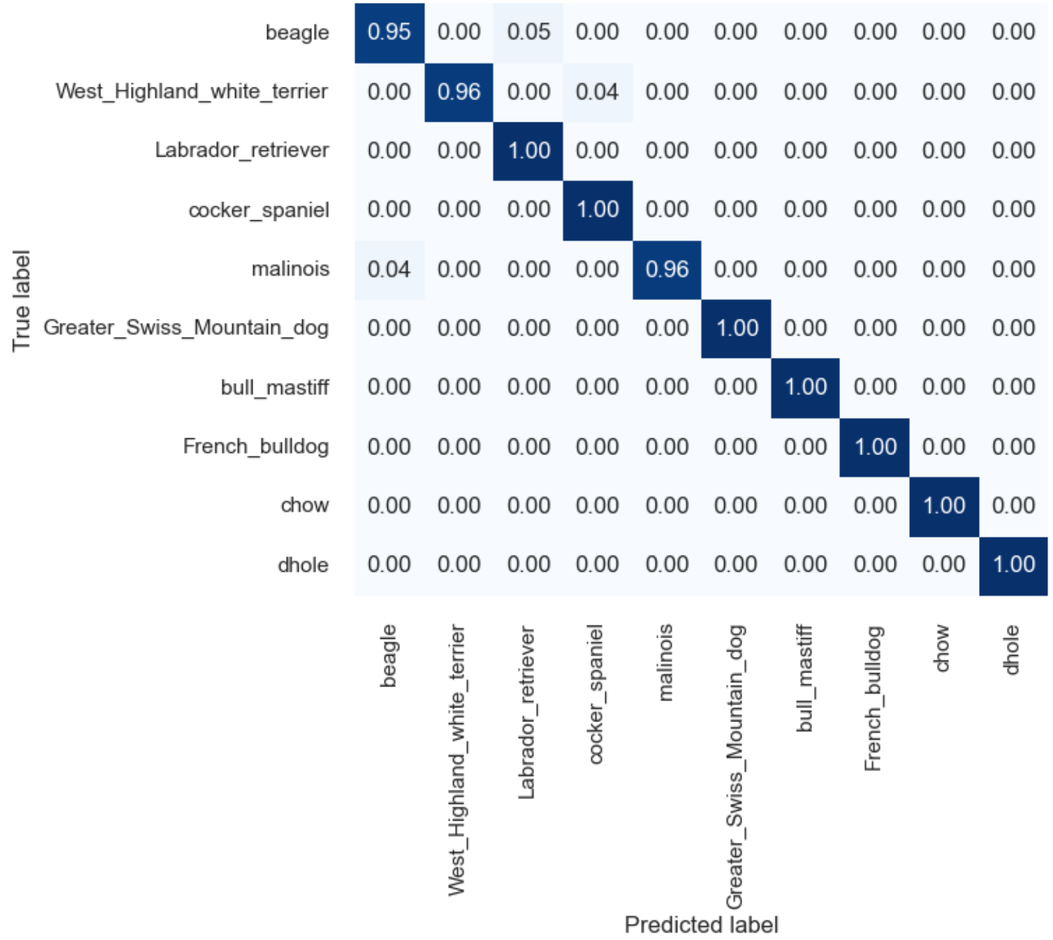
Modèles avec Transfer Learning



Modèles avec Transfer Learning

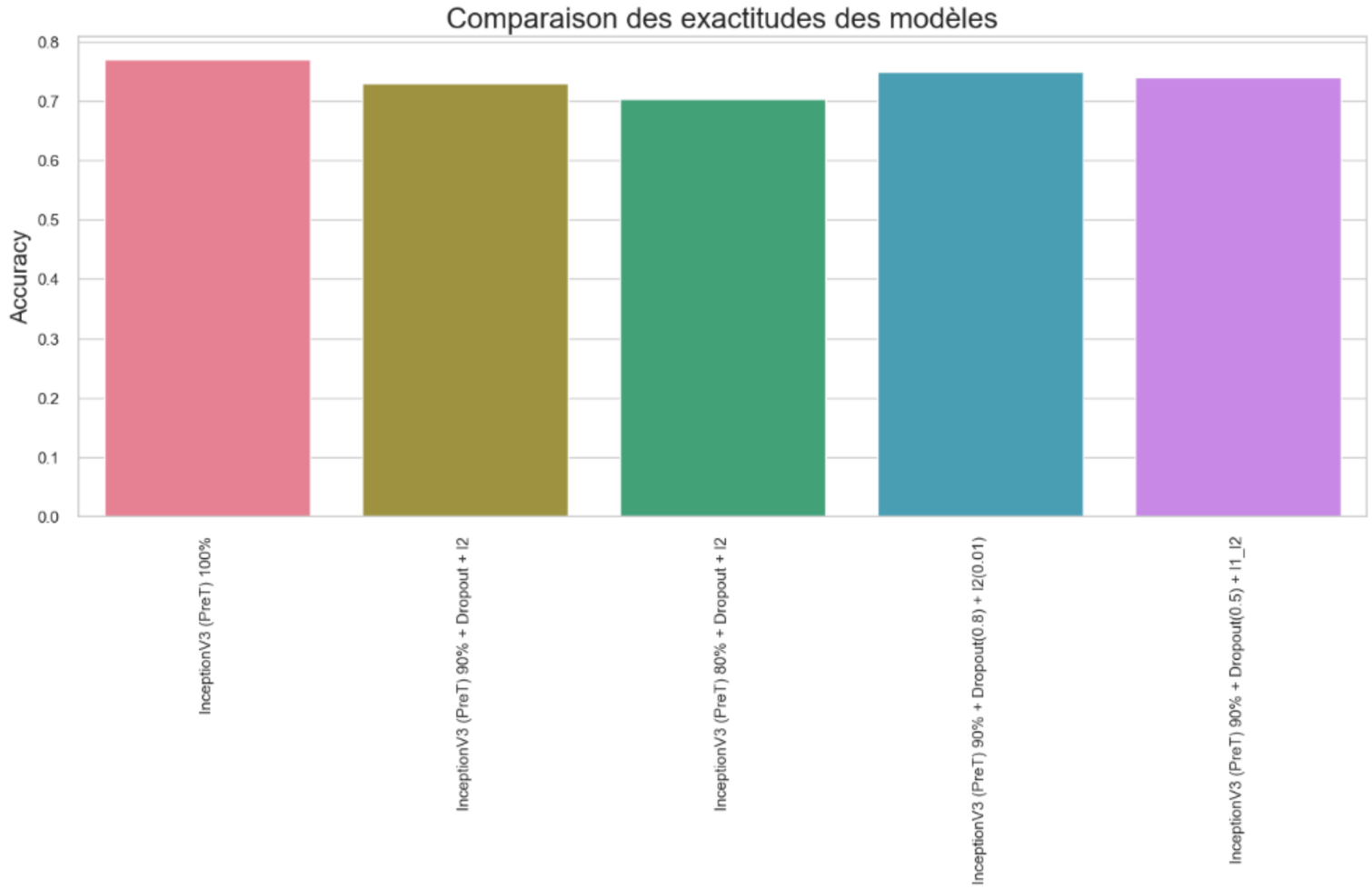
InceptionV3 sur 10 races





Modèles avec Transfer Learning

InceptionV3 sur 120 races

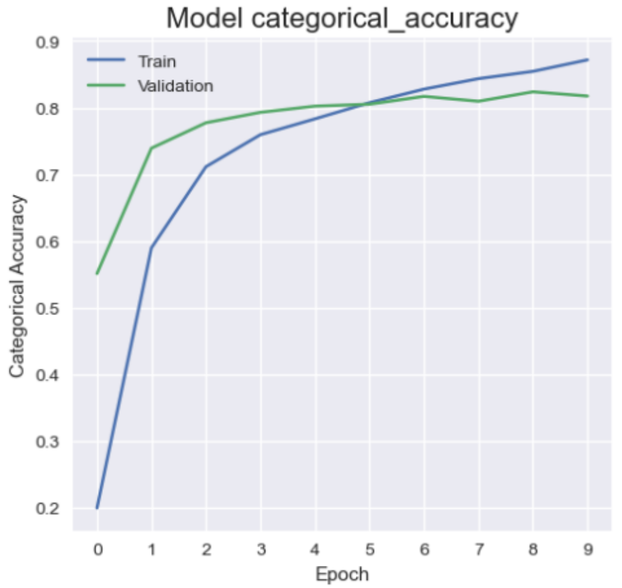


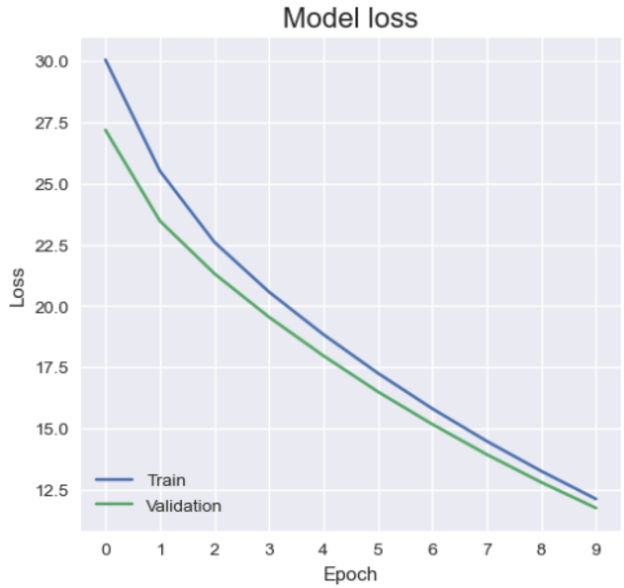
Modèles avec Transfer Learning

InceptionV3 sur 120 races

```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
for layer in base_model.layers:
    layer.trainable = True
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(2048, activation='relu', kernel_regularizer=l2(0.01))(x)
x = Dropout(0.1)(x)
x = Dense(1024, activation='relu', kernel_regularizer=l2(0.005))(x)
x = Dropout(0.05)(x)

predictions = Dense(num_classes120, activation='softmax')(x)

model_120 = Model(inputs=base_model.inputs, outputs=predictions)

model_120.compile(
    optimizer=Adam(lr=0.000008),
    loss='categorical_crossentropy',
    metrics=['categorical_accuracy']
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

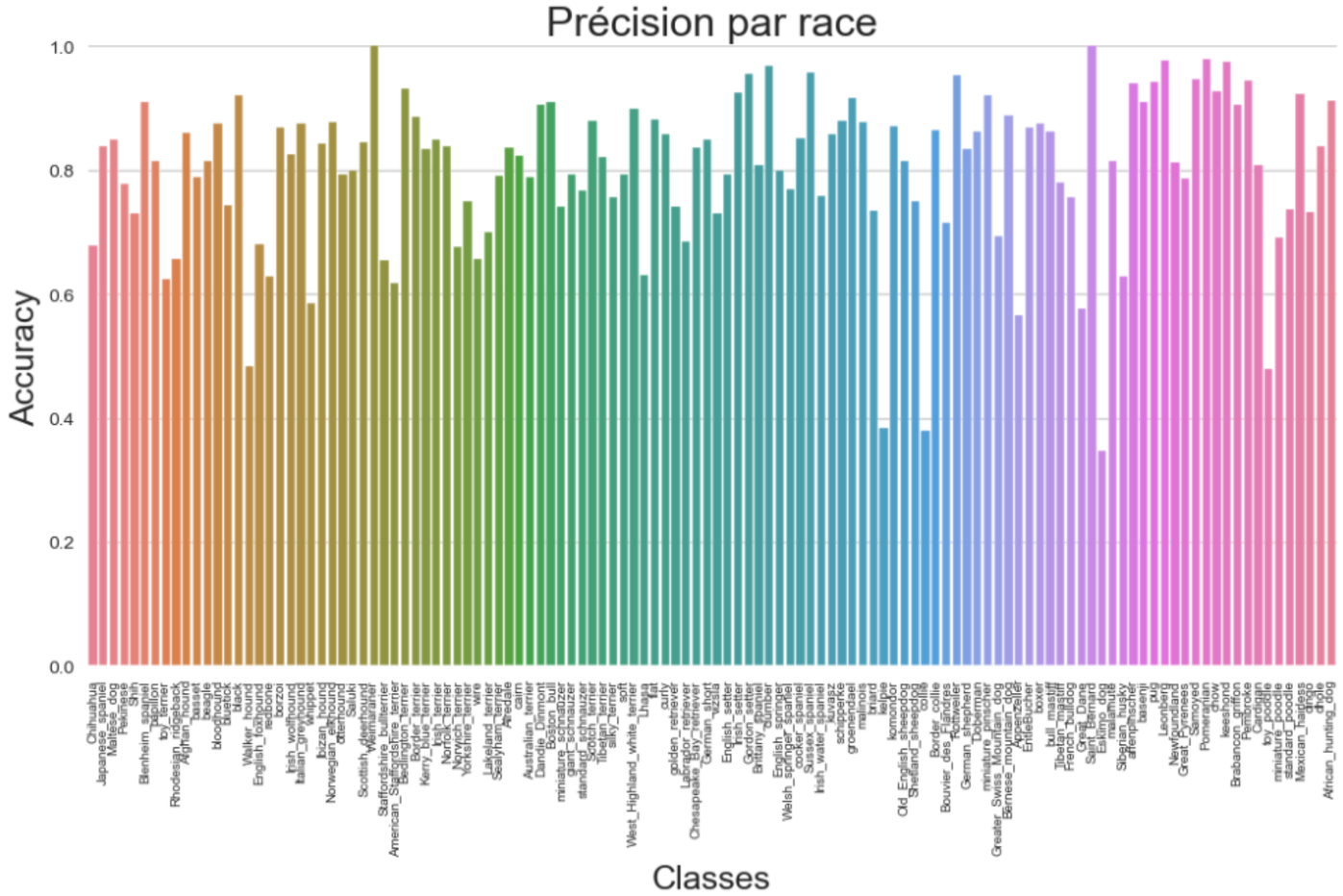





```
Test accuracy: 0.813829779624939
```

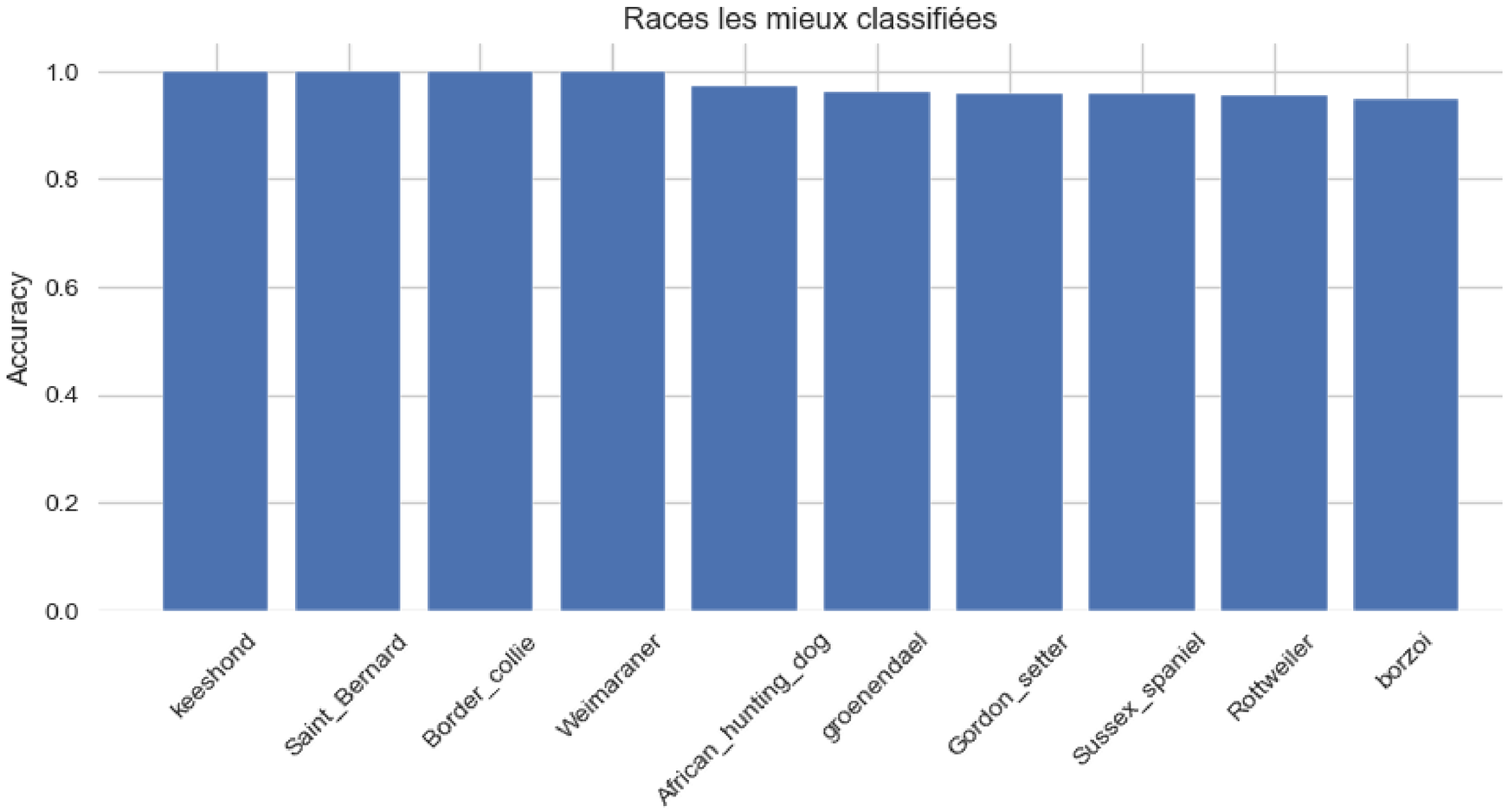
Modèles avec Transfer Learning

InceptionV3 sur 120 races



Modèles avec Transfer Learning

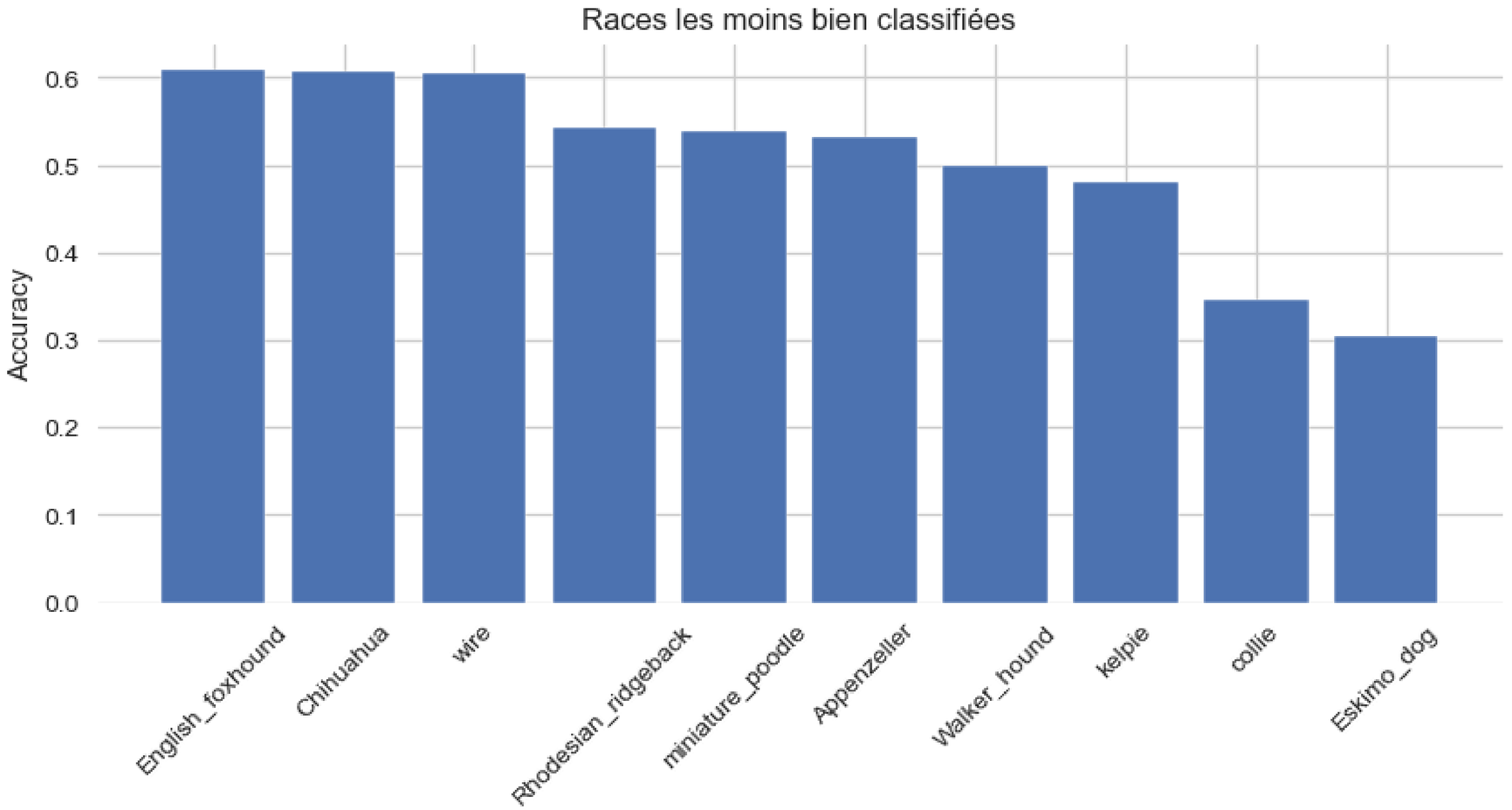
InceptionV3 sur 120 races



Modèles avec Transfer Learning

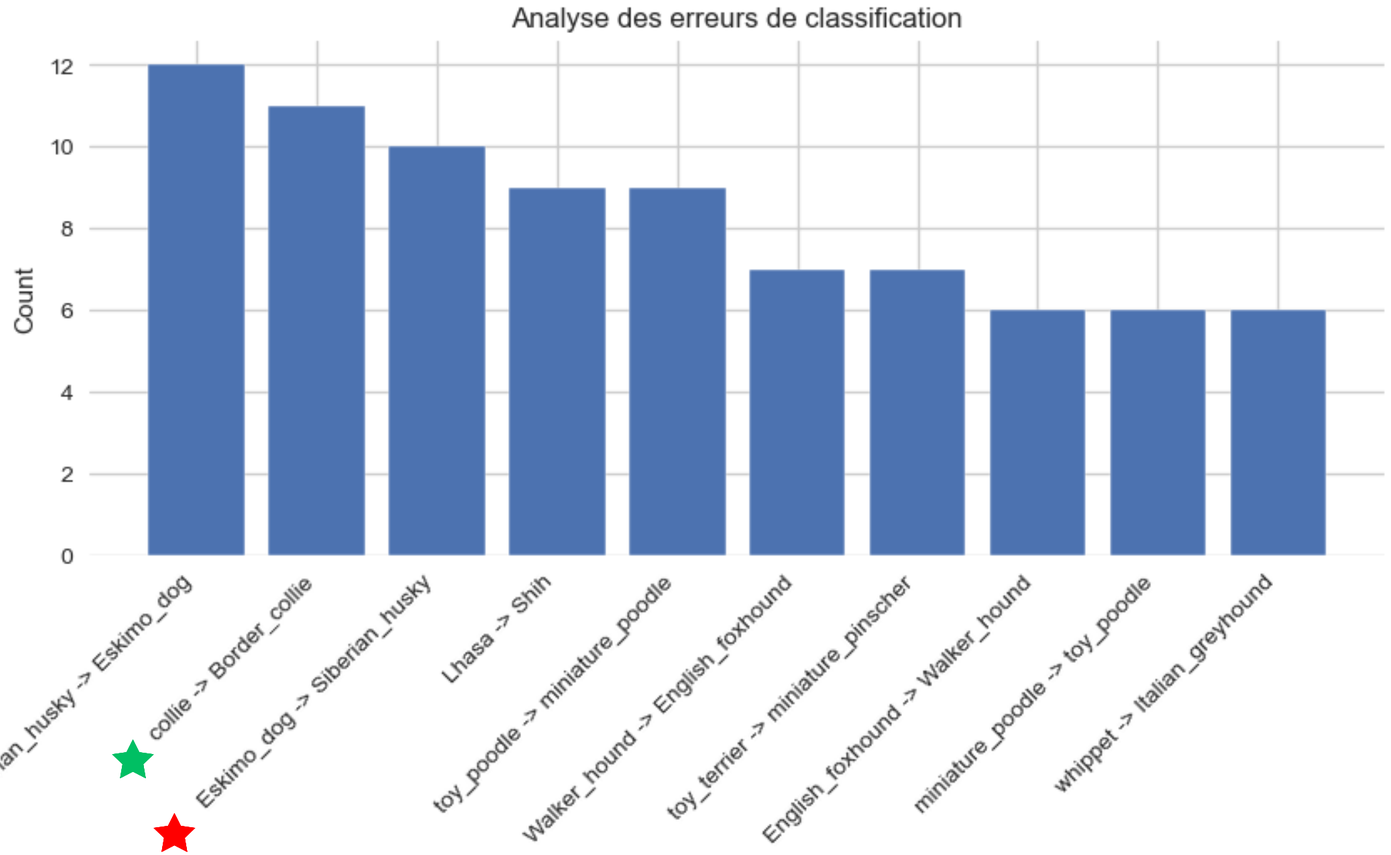
InceptionV3 sur 120 races

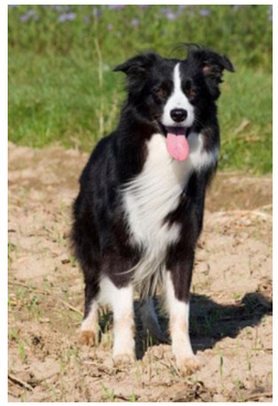
Misclassification error: 19.46%



Modèles avec Transfer Learning

InceptionV3 sur 120 races





Démonstration de l'API

https://sdmwd-p6-iml-detecteur-de-race-de-chien.streamlit.app

Prédicteur de race de chien

Importez votre image

Drag and drop file here
Limit 200MB per file • JPG
Browse files

Sans titre.jpg 1.1MB

Race prédite : **Labrador_retriever**

Précision de la prédiction : **83.6%**

Rappel de l'objectif

- Réaliser un algorithme de détection de la race de chiens

Conclusion

- Modèle à partir d'InceptionV3 (Transfer Learning + Fine Tuning)
- Modèle mis en application en ligne via Streamlit
- Erreur de classification très faible