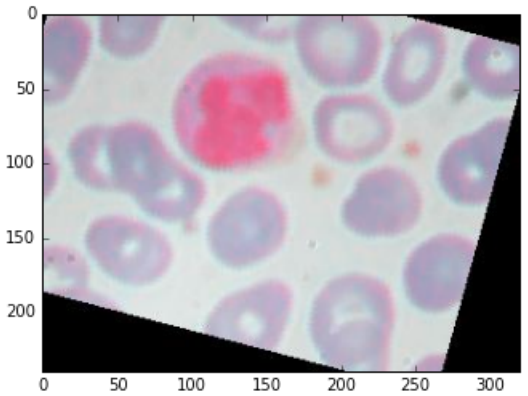
RECONNAISSANCE DE CELLULES SANGUINES

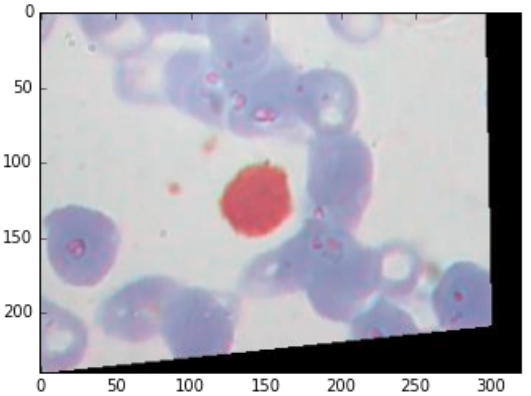
Dataset Kaggle https://www.kaggle.com/paultimothymooney/blood-cells/

Dataset

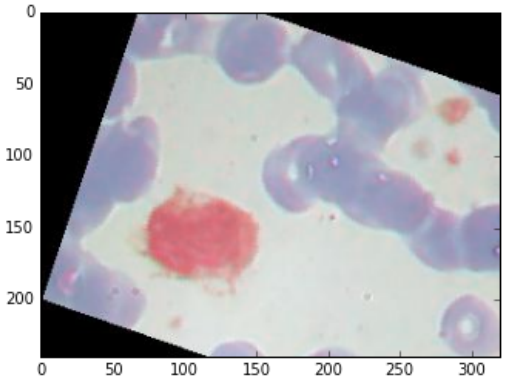
12000 images de cellules sanguines tirées d'un ensemble de 410 images originales par augmentation de données



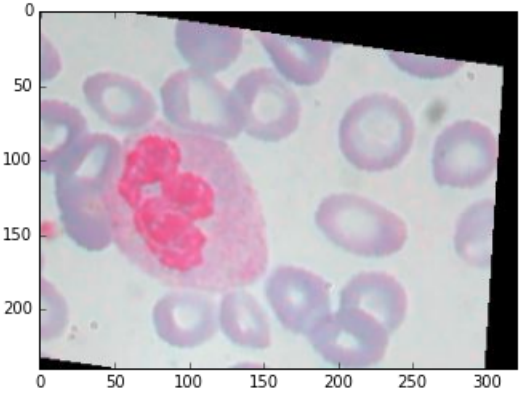
EOSINOPHIL



LYMPHOCYTE



MONOCYTE



NEUTROPHIL

Démarche

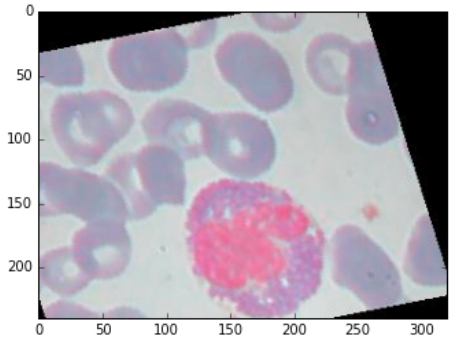
- Approche classique
  - Pretraitement
    - Transformer chaque image en array (120,160)
    - Filtrer pour ne garder que la cellule sanguine
    - Contraster en noir/blanc pour calculer le barycentre
    - Centrer la cellule sanguine
    - Aplatir l'image pour obtenir un array de longueur 120x160
    - Construire X et Y, dataset avec 4 labels
    - Split pour train et validation
  - Utiliser plusieurs algorithmes en binaire (Eosinophil et Lymphocyte) puis multi-classes

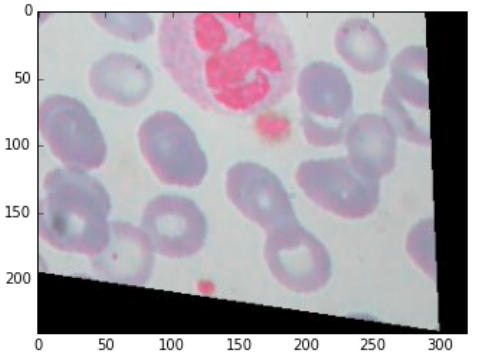
- Approche Deep Learning
  - Split pour train et validation (par directories)
  - D'abord classification binaire (Eosinophil et Lymphocyte) en utilisant un réseau de référence Chiens/Chats (ref. François Chollet- Deep Learning with Python)
  - Idem avec autre réseau obtenu avec modèle pré-entrainé et modification des dernières couches de neurones
  - Puis classification multi-classes avec modèle de base et réseau pré-entrainé

Approche classique

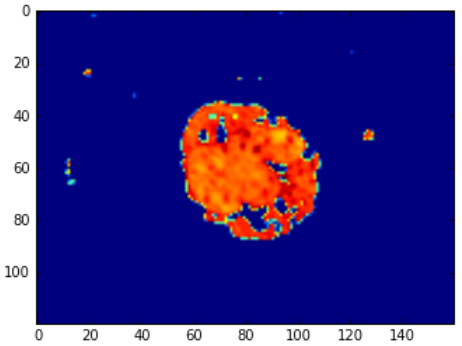
Utilisation d'OPENCV

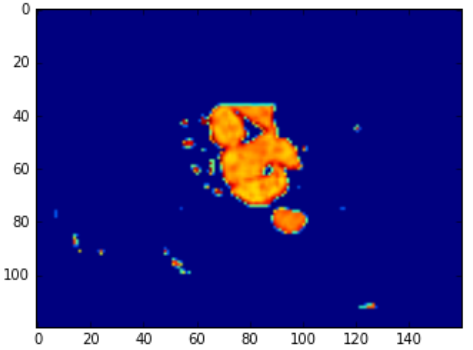
Images originales





Images filtrées centréés





Résultats (classique, binaire et multi-classes)

BINAIRE

LinearSVC() : accuracy= 0.7550

KneighborsClassifier() accuracy: 0.8650

RandomForestClassifier() accuracy: 0.9253

Cross-validation f1-macro : 0.9256

MULTI-CLASSES

LinearSVC() : accuracy= 0.5048

KneighborsClassifier() accuracy: 0.7132

RandomForestClassifier() accuracy: 0.8044

Cross-validation f1-macro :0.7901

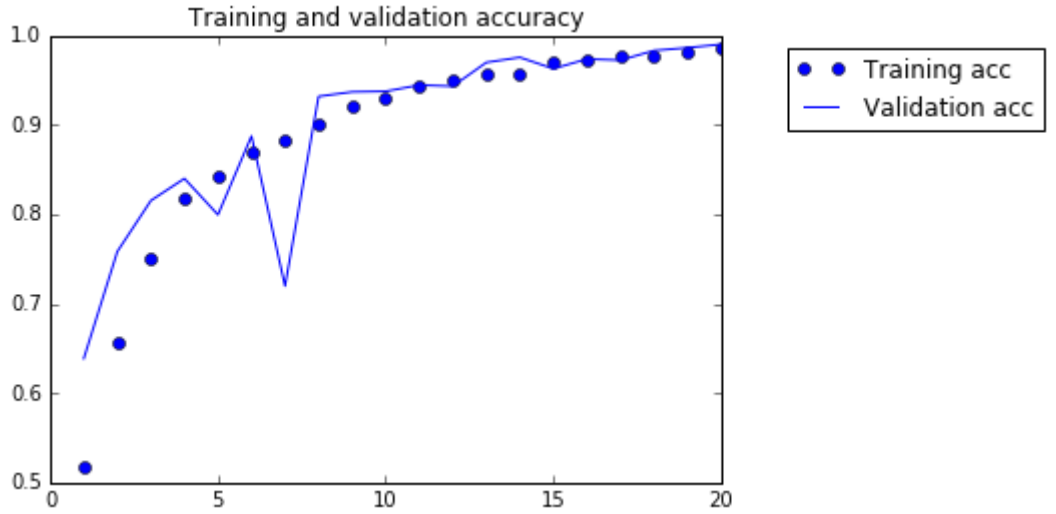
Approche Deep Learning, Binaire

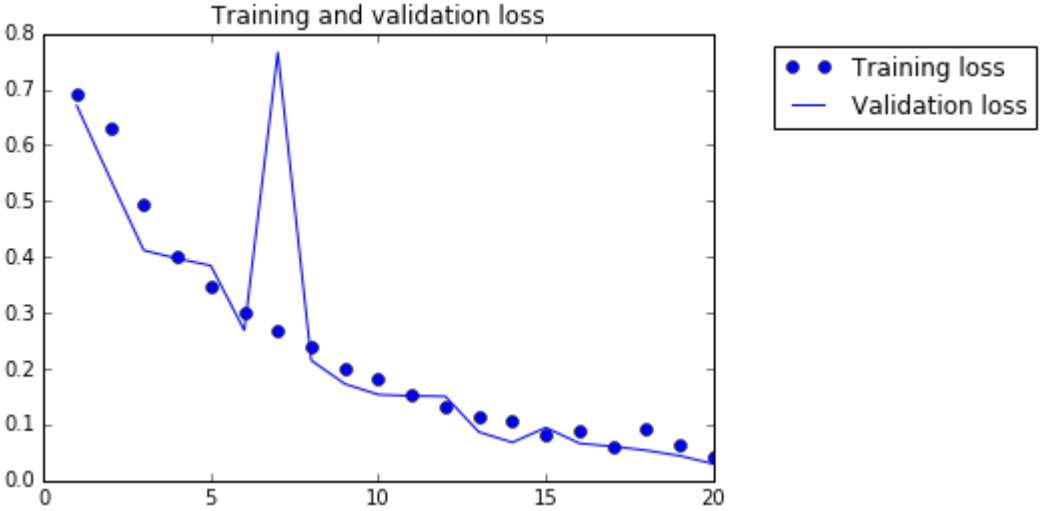
Utilisation de Keras (input= directories)

Réseau :

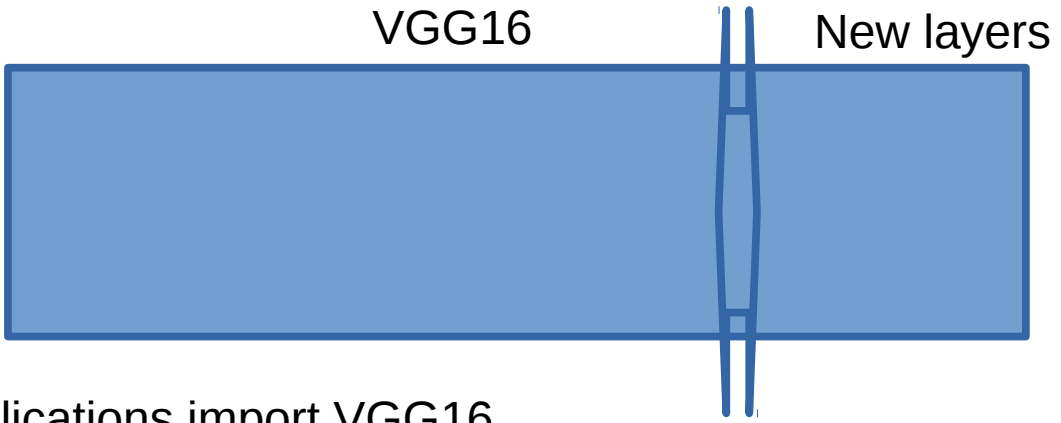
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3),
activation='relu',input_shape=(120, 160, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

Résultats (Keras, binaire)





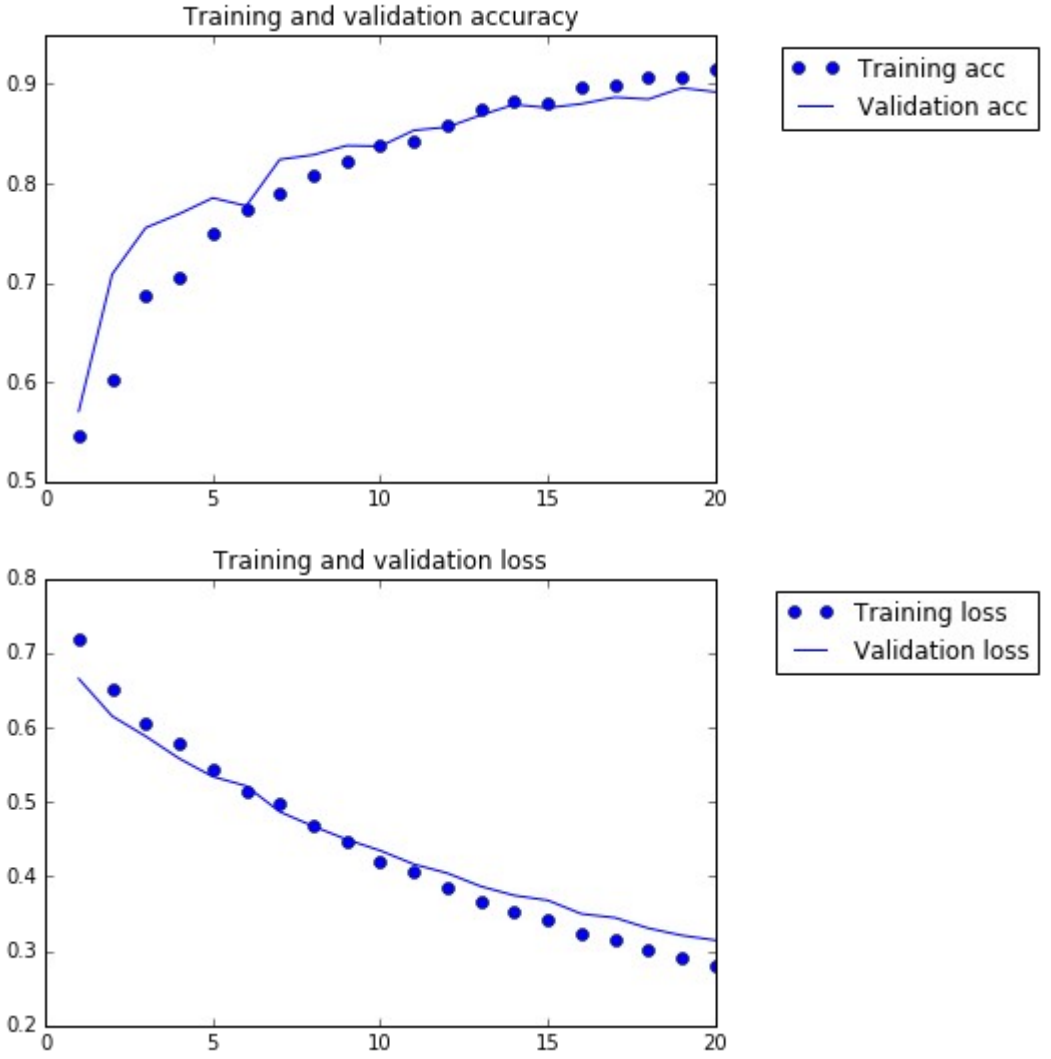
Avec réseau préentrainé




```
from keras.applications import VGG16
conv_base = VGG16(weights='imagenet',
    include_top=False,
    input_shape=(120, 160, 3))
```

```
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=3 * 5 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

Résultats (Keras, binaire, préentrainé)



Deep Learning multi-classes

Modèle :

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=(120, 160, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(4, activation='softmax'))
```

Résultats (Keras, multi-classes)

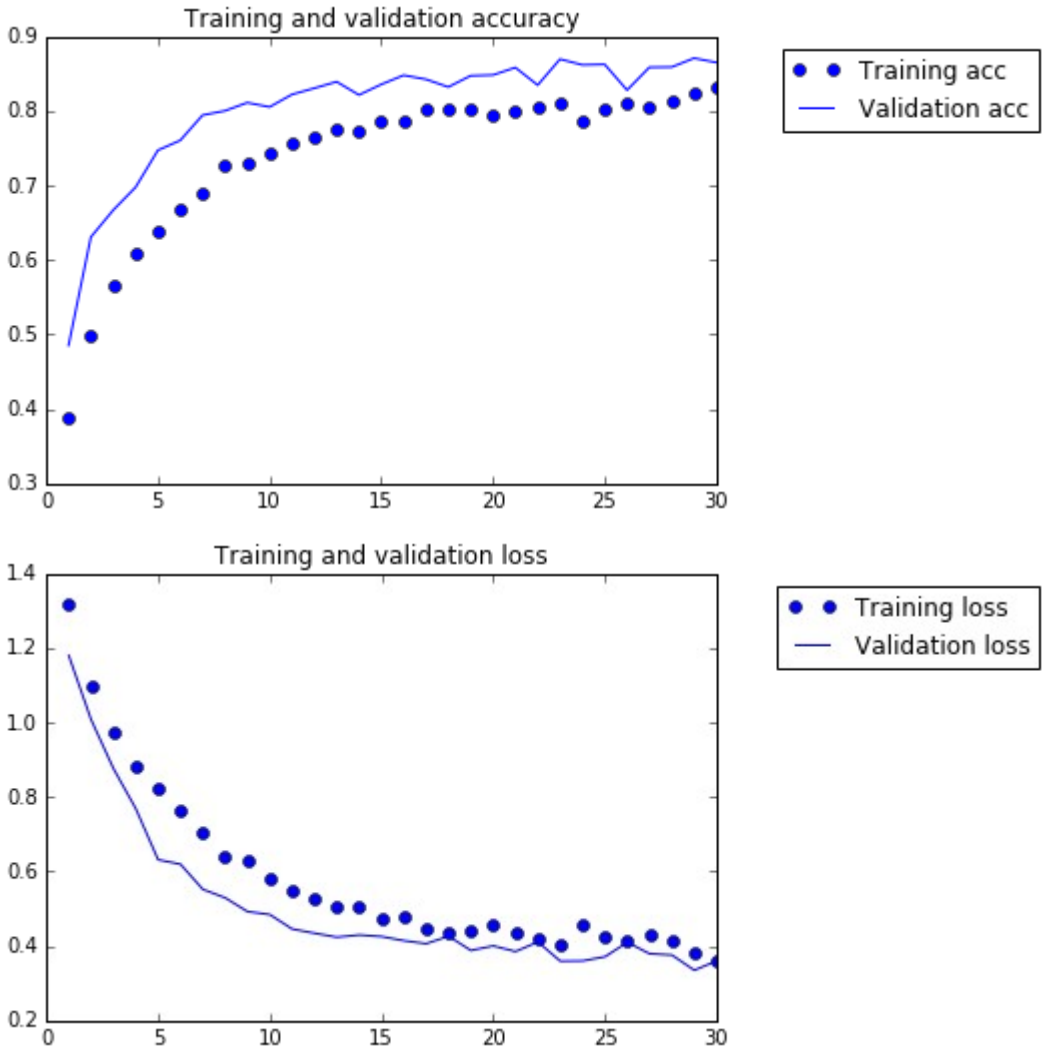
Epoch 18/20
340/340 [==============================] - 164s 483ms/step - loss: 1.3864 - **acc: 0.2454** - val_loss: 1.3863 - val_acc: 0.2531

Epoch 19/20
340/340 [==============================] - 164s 483ms/step - loss: 1.3865 - **acc: 0.2406** - val_loss: 1.3863 - val_acc: 0.2480

Epoch 20/20
340/340 [==============================] - 156s 460ms/step - loss: 1.3864 - **acc: 0.2449** - val_loss: 1.3863 - val_acc: 0.2464

Ne marche pas !!

Résultats (Keras, multi-classes, préentrainé)



Discussion

Approche classique :

En binaire et multi-classes, Random Forest meilleur algo (pas essayé Gradient Boosting…)

Deep Learning :

En binaire le modèle utilisé donne de très bons résultats (sans même réseau pré-entrainé), meilleurs que classique

Multi-classes : echec non expliqué avec modèle nu, bons résultats avec modèle pré-entrainé, meilleurs que classique