train_model

December 18, 2023

1 Adem BEN JABRIA

1.1 ## Université Côte D'Azur - M2 MIAGE IA2

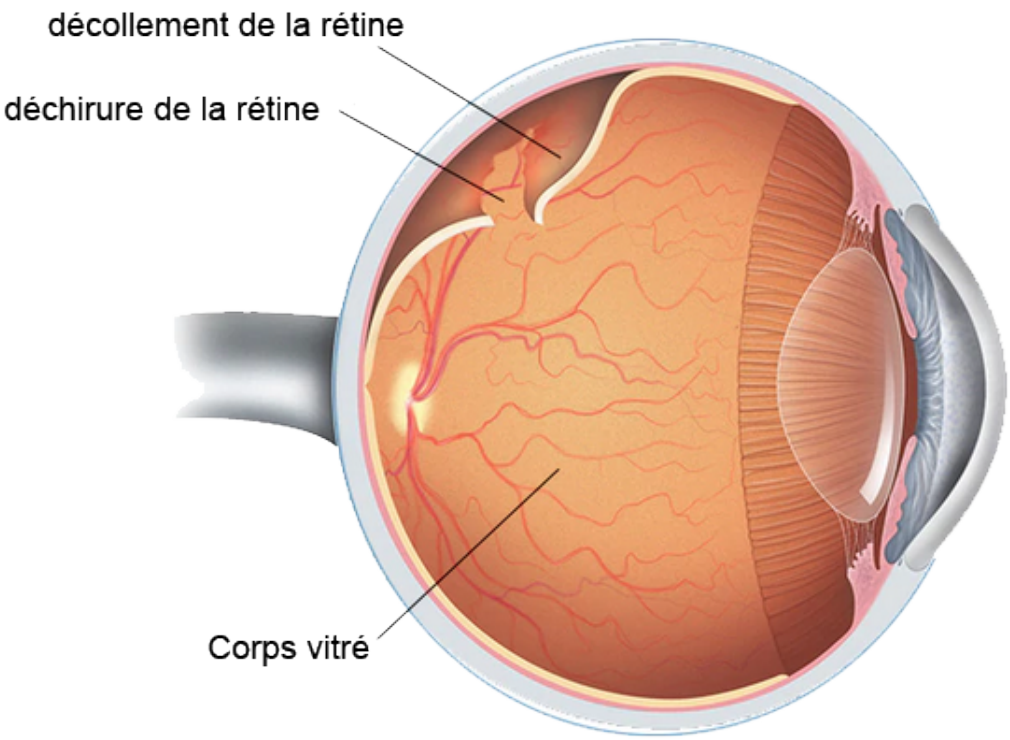
2 La détection de la rétinopathie diabétique

2.0.1 Identifier les signes de rétinopathie diabétique dans les images de l'œil

Ayant de nombreux membres de ma famille atteints de diabète de type 1, ce sujet me tient particulièrement à cœur. C'est cette raison personnelle qui m'a motivé à choisir ce domaine de recherche.

Dans le monde, **537 millions d'adultes** âgés de 20 à 79 ans vivent actuellement avec un diabète, ce qui représente **10,5%** de la population mondiale dans ce groupe d'âge. Ces chiffres sont alarmants et devraient augmenter pour atteindre **643 millions (11,3 %) d'ici 2030** et **783 millions (12,2%) d'ici 2045**. De plus, **212 millions** de personnes vivant avec le diabète ne sont pas diagnostiquées.

En France, la prévalence du diabète de type 2 diagnostiqué est d'environ **4,5%** de la population générale, tandis que le diabète de type 1 concerne environ **10 %** des personnes diabétiques. Avec près de **3,9 millions** de patients diagnostiqués en 2021, dont 10% de type 1, le diabète représente une **épidémie croissante**. Depuis 20 ans, l'incidence augmente de **4% par an**, et il apparaît de plus en plus tôt, notamment chez les enfants de moins de 5 ans.



La **rétinopathie diabétique (DR)**, complication oculaire du diabète, est une préoccupation majeure en France comme dans le reste du monde. Elle peut conduire à la cécité si elle n'est pas détectée et traitée à temps. Actuellement, la détection de la DR est un processus manuel exigeant, réalisé par des cliniciens formés qui examinent des photographies du fond d'œil. Cette méthode est non seulement longue, mais aussi sujette à des retards qui peuvent entraîner des pertes de suivi et des retards de traitement.

Avec la croissance constante du nombre de personnes atteintes de diabète, il est essentiel de développer des méthodes automatisées pour le dépistage de la DR. Ces méthodes, basées sur la classification d'images, la reconnaissance de formes et l'apprentissage automatique, doivent être suffisamment avancées pour être utilisées efficacement en milieu clinique. L'objectif est de créer des modèles automatisés qui peuvent détecter avec précision la DR, réduisant ainsi le besoin d'une expertise et d'un équipement spécialisés, souvent absents dans les régions les plus touchées par le diabète.

3 À propos du jeu de données

3.1 À propos des données

Les images consistent en des scans rétiniens filtrés par un filtre gaussien pour détecter la rétinopathie diabétique. Le jeu de données original est disponible sur *APTOS 2019 Blindness Detection*. Ces images sont redimensionnées en 224x224 pixels afin d'entrainer plus facilement (et plus rapidement) le modèle.

Toutes les images sont déjà sauvegardées dans leurs dossiers respectifs selon la gravité/le stade de

la rétinopathie diabétique. Un fichier `train.csv` est fournis dans le zip du jeu de données.

0 - No_DR

1 - Mild

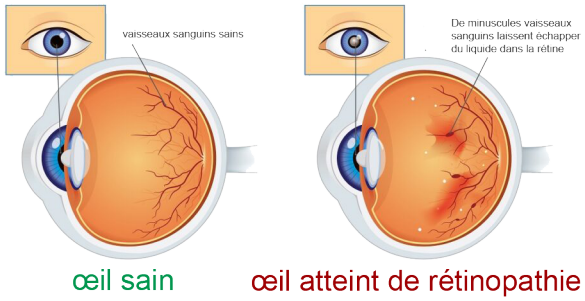
2 - Moderate

3 - Severe

4 - Proliferate_DR

4 Comprendre les stades de la rétinopathie diabétique

Une glycémie, une pression artérielle et des taux de cholestérol élevés, ainsi qu'une augmentation du poids corporel sont associés au diabète non contrôlé et peuvent endommager les délicats vaisseaux sanguins de la rétine, causant une maladie appelée rétinopathie diabétique. Aux premiers stades de la rétinopathie diabétique, la perte de vision peut être prévenue ou limitée ; mais à mesure que la condition progresse, il devient plus difficile de prévenir la perte de vision.



4.1 Rétinopathie Non Proliférative

La rétinopathie diabétique non proliférative est le stade initial et moins grave de la maladie. En raison des dommages causés par des niveaux élevés de sucre dans le sang, les petits vaisseaux sanguins de la rétine commencent à gonfler et à fuir du liquide et du sang. La fuite provoque un gonflement à l'intérieur de la rétine appelé œdème maculaire, qui est une cause courante de déficience visuelle chez les personnes diabétiques.

À mesure que la rétinopathie non proliférative progresse, les vaisseaux sanguins peuvent se bloquer ou se fermer, et de nouveaux vaisseaux sanguins anormaux peuvent commencer à se former dans les mauvaises parties de la rétine. La vision est généralement non affectée aux stades non prolifératifs, mais il est préférable pour l'ophtalmologue de surveiller ces changements de près tous les quelques mois pour s'assurer qu'ils ne s'aggravent pas au stade "prolifératif".

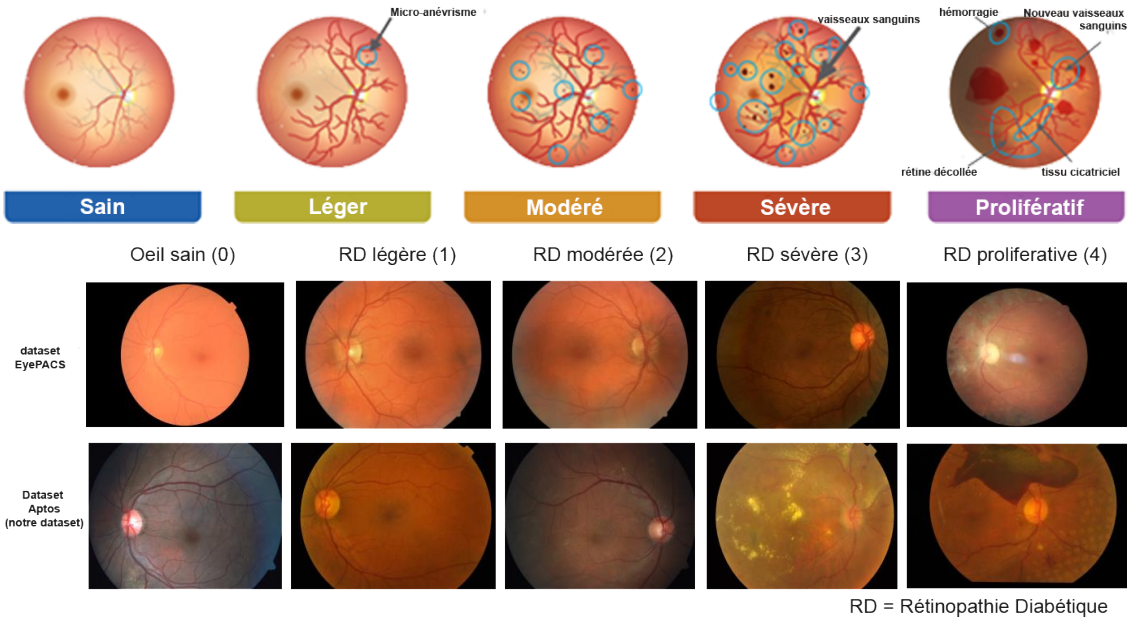
4.2 Rétinopathie Diabétique Proliférative

La forme la plus grave de la maladie, la rétinopathie diabétique proliférative, se caractérise par un processus connu sous le nom de néovascularisation. Lorsque les vaisseaux sanguins de la rétine sont bloqués, la rétine est privée de l'oxygène et des nutriments dont elle a besoin pour prospérer. Sentant sa réserve de sang limitée, la rétine réagit en développant de nouveaux vaisseaux sanguins anormaux dans les mauvaises parties de la rétine. Mais les nouveaux vaisseaux sanguins ne fournissent pas

un flux sanguin adéquat. Ils sont extrêmement fragiles et, à mesure qu'ils grandissent, peuvent fuir du sang dans le corps vitré. Parfois, cette hémorragie provoque des distorsions visuelles appelées corps flottants ; si l'hémorragie est suffisamment grave, elle peut partiellement ou complètement altérer la vision.

Dans certains cas, la formation de ces nouveaux vaisseaux sanguins anormaux conduit au développement de tissu cicatriciel, ou fibrose, dans la rétine. Le tissu cicatriciel peut faire plisser ou "se pincer" la rétine, ou même la détacher de sa position normale le long de l'arrière de l'œil, ce qui est appelé "décollement rétinien tractive". Selon la complication, une perte de la vision centrale peut survenir.

Une autre complication à surveiller est la croissance de ces vaisseaux sanguins anormaux bloquant l'angle de drainage, le mécanisme naturel par lequel le fluide sort de l'œil. Si cela se produit, la pression à l'intérieur de l'œil peut augmenter très rapidement, provoquant un glaucome à angle fermé néovasculaire.



[20]:
```
# Importation des bibliothèques nécessaires.
from tensorflow import lite
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import pandas as pd
import random, os
import shutil
import matplotlib.pyplot as plt
from matplotlib.image import imread
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.metrics import categorical_accuracy
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore') #Pour facilité la lecture et la correction
```

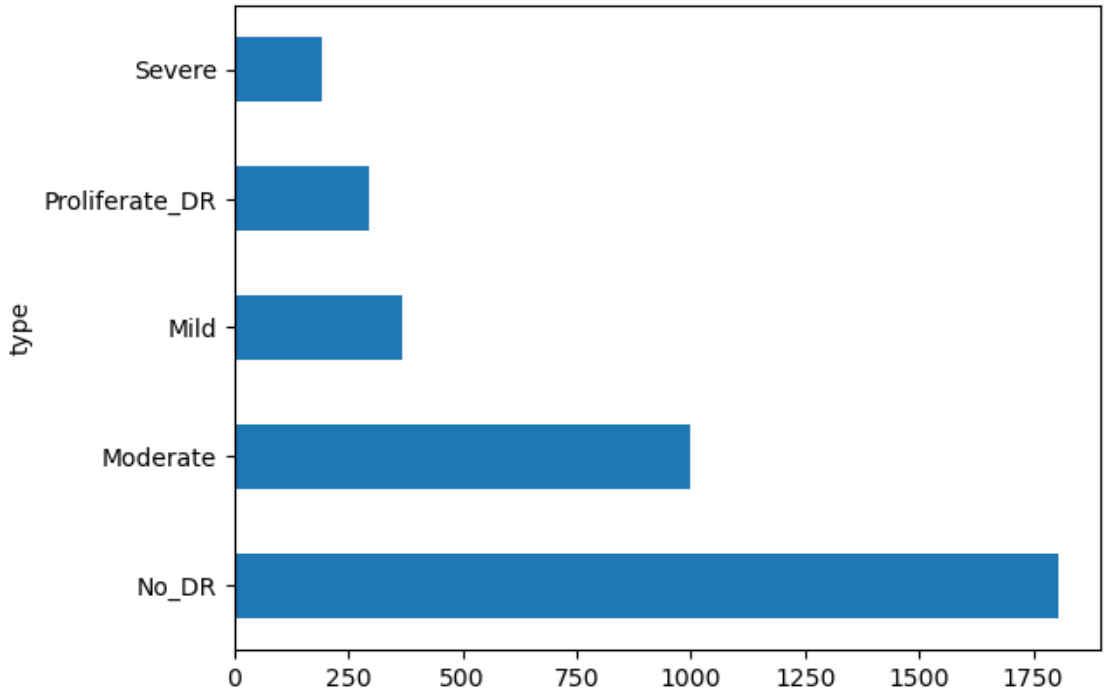
```
# Chemin d'accès au fichier CSV contenant les données de diagnostic.
csv_file_path = r'C:\Users\ademb\Documents\diabetic retinopathy_detection\train.
 ↪csv'

# Lecture du fichier CSV et chargement dans un DataFrame pandas.
df = pd.read_csv(csv_file_path)

# Création de dictionnaires pour mapper les diagnostics à des étiquettes␣
 ↪binaires et détaillées.
diagnosis_dict_binary = {
    0: 'No_DR',
    1: 'DR',
    2: 'DR',
    3: 'DR',
    4: 'DR'
}

diagnosis_dict = {
    0: 'No_DR',
    1: 'Mild',
    2: 'Moderate',
    3: 'Severe',
    4: 'Proliferate_DR',
}

# Application des mappages sur le DataFrame pour créer de nouvelles colonnes␣
 ↪avec les étiquettes.
df['binary_type'] = df['diagnosis'].map(diagnosis_dict_binary.get)
df['type'] = df['diagnosis'].map(diagnosis_dict.get)

df.head()
```

[20]:

	id_code	diagnosis	binary_type	type
0	000c1434d8d7	2	DR	Moderate
1	001639a390f0	4	DR	Proliferate_DR
2	0024cdab0c1e	1	DR	Mild
3	002c21358ce6	0	No_DR	No_DR
4	005b95c28852	0	No_DR	No_DR

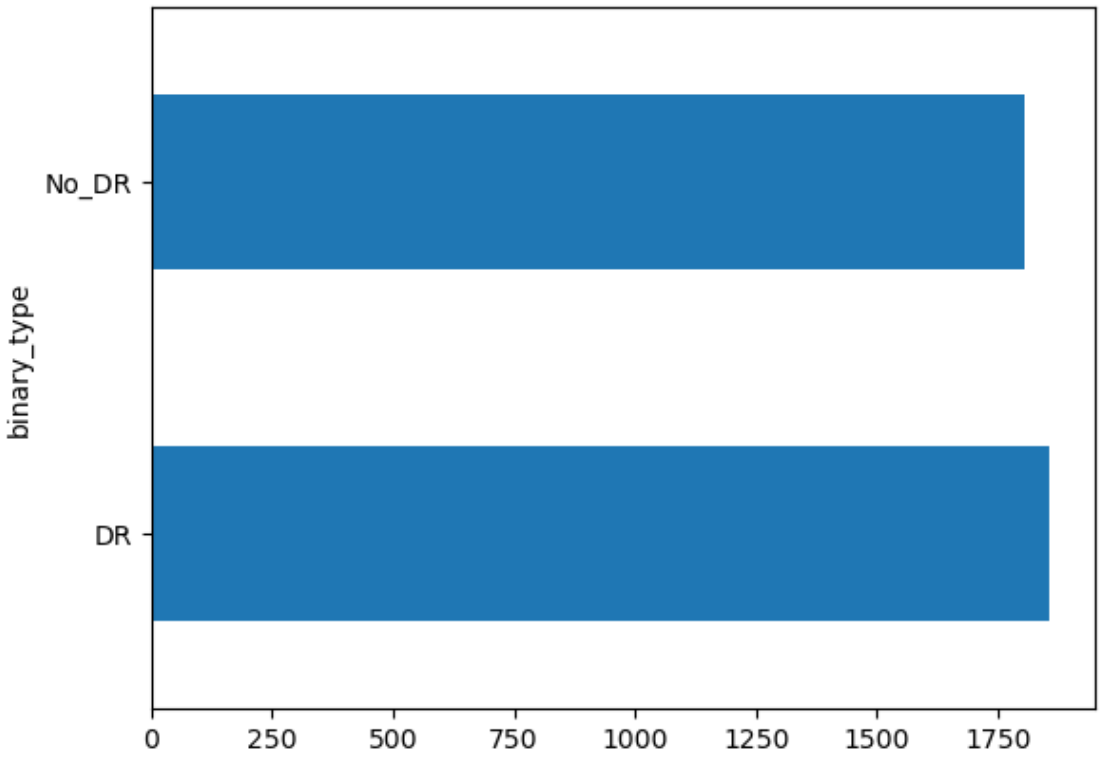
[21]:
```
df['type'].value_counts().plot(kind='barh')
```

[21]:
```
<Axes: ylabel='type'>
```



[22]:
```
df['binary_type'].value_counts().plot(kind='barh')
```

[22]: <Axes: ylabel='binary_type'>



[23]:
```
# Division des données en ensembles d'entraînement, de validation et de test de␣
 ↪manière stratifiée.
train_intermediate, val = train_test_split(df, test_size = 0.15, stratify =␣
 ↪df['type'])
train, test = train_test_split(train_intermediate, test_size = 0.15 / (1 - 0.
 ↪15), stratify = train_intermediate['type'])

# Affichage de la distribution des étiquettes dans chaque ensemble de données.
print(train['type'].value_counts(), '\n')
print(test['type'].value_counts(), '\n')
print(val['type'].value_counts(), '\n')
```

```
type
No_DR             1263
Moderate           699
Mild               258
Proliferate_DR     207
Severe             135
Name: count, dtype: int64

type
No_DR             271
```

```
Moderate          150
Mild               56
Proliferate_DR     44
Severe             29
Name: count, dtype: int64

type
No_DR             271
Moderate          150
Mild               56
Proliferate_DR     44
Severe             29
Name: count, dtype: int64
```

[24]:
```python
# Création des répertoires pour stocker les images triées par ensemble et étiquettes.
base_dir = ''

train_dir = os.path.join(base_dir, 'train')
val_dir = os.path.join(base_dir, 'validation')
test_dir = os.path.join(base_dir, 'test')

# Suppression des anciens répertoires et création de nouveaux si nécessaire.
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)

if os.path.exists(train_dir):
    shutil.rmtree(train_dir)
os.makedirs(train_dir)

if os.path.exists(val_dir):
    shutil.rmtree(val_dir)
os.makedirs(val_dir)

if os.path.exists(test_dir):
    shutil.rmtree(test_dir)
os.makedirs(test_dir)
```

[25]:
```python
# Chemin d'accès au répertoire source des images.
src_dir = r'C:\Users\ademb\Documents\diabetic retinopathy_detection\data\gaussian_filtered_images'

# Copie des images dans les répertoires correspondants selon les ensembles d'entraînement, de validation et de test.
for index, row in train.iterrows():
    diagnosis = row['type']
```

```
    binary_diagnosis = row['binary_type']
    id_code = row['id_code'] + ".png"
    srcfile = os.path.join(src_dir, diagnosis, id_code)
    dstfile = os.path.join(train_dir, binary_diagnosis)
    os.makedirs(dstfile, exist_ok = True)
    shutil.copy(srcfile, dstfile)

for index, row in val.iterrows():
    diagnosis = row['type']
    binary_diagnosis = row['binary_type']
    id_code = row['id_code'] + ".png"
    srcfile = os.path.join(src_dir, diagnosis, id_code)
    dstfile = os.path.join(val_dir, binary_diagnosis)
    os.makedirs(dstfile, exist_ok = True)
    shutil.copy(srcfile, dstfile)

for index, row in test.iterrows():
    diagnosis = row['type']
    binary_diagnosis = row['binary_type']
    id_code = row['id_code'] + ".png"
    srcfile = os.path.join(src_dir, diagnosis, id_code)
    dstfile = os.path.join(test_dir, binary_diagnosis)
    os.makedirs(dstfile, exist_ok = True)
    shutil.copy(srcfile, dstfile)
```

[26]:
```
# Configuration des chemins vers les dossiers contenant les images pour␣
 ↪l'entraînement, la validation et les tests.
train_path = 'train'
val_path = 'validation'
test_path = 'test'

# Configuration du générateur d'images pour l'ensemble d'entraînement.
# ImageDataGenerator est utilisé pour prétraiter les images de manière␣
 ↪automatisée.
# 'rescale=1./255' est utilisé pour normaliser les valeurs de pixel des images␣
 ↪dans la plage [0, 1].
train_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(
    train_path,       # Chemin vers les images d'entraînement.
    target_size=(224,224),  # Redimensionnement des images à 224x224 pixels.
    shuffle = True    # Mélange aléatoire des images pour éviter tout biais␣
 ↪d'ordre lors de l'entraînement.
)

# Configuration du générateur d'images pour l'ensemble de validation avec les␣
 ↪mêmes paramètres que pour l'entraînement.
val_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(
    val_path,         # Chemin vers les images de validation.
```

```
    target_size=(224,224),  # Taille cible identique pour assurer la cohérence␣
 ↪dans le traitement des images.
    shuffle = True    # Mélange des images pour la validation, utile pour␣
 ↪évaluer les performances de manière aléatoire.
)

# Configuration du générateur d'images pour l'ensemble de test.
# 'shuffle = False' est utilisé pour le test pour conserver l'ordre des images,␣
 ↪utile lors de l'évaluation des prédictions.
test_batches = ImageDataGenerator(rescale = 1./255).flow_from_directory(
    test_path,        # Chemin vers les images de test.
    target_size=(224,224),  # Taille cible pour le redimensionnement des images␣
 ↪de test.
    shuffle = False   # Pas de mélange pour l'ensemble de test pour␣
 ↪correspondre les prédictions avec leurs vraies étiquettes.
)
```

```
Found 2562 images belonging to 2 classes.
Found 550 images belonging to 2 classes.
Found 550 images belonging to 2 classes.
```

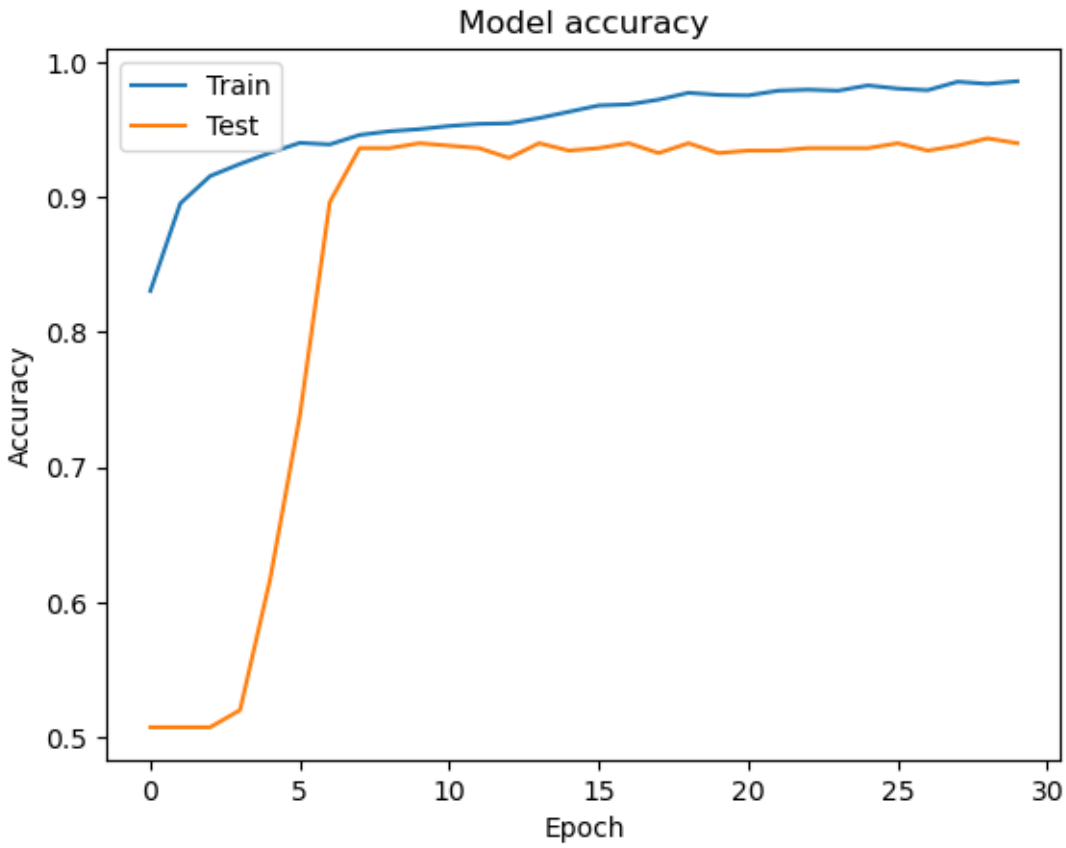
[27]:
```
# Construction du modèle de réseau de neurones convolutifs (CNN) pour la␣
 ↪classification.
model = tf.keras.Sequential([
    layers.Conv2D(8, (3,3), padding="valid", input_shape=(224,224,3),␣
 ↪activation = 'relu'), # Première couche de convolution avec 8 filtres de␣
 ↪taille 3x3, une activation ReLU et un padding 'valid' (sans padding).
    layers.MaxPooling2D(pool_size=(2,2)), # Première couche de MaxPooling pour␣
 ↪réduire la dimension spatiale de la sortie.
    layers.BatchNormalization(), # Couche de normalisation par lots pour␣
 ↪normaliser les activations de la couche précédente.
    layers.Conv2D(16, (3,3), padding="valid", activation = 'relu'), # Deuxième␣
 ↪couche de convolution avec 16 filtres de taille 3x3, une activation ReLU.
    layers.MaxPooling2D(pool_size=(2,2)), # Deuxième couche de MaxPooling.
    layers.BatchNormalization(), # Normalisation par lots.
    layers.Conv2D(32, (4,4), padding="valid", activation = 'relu'), # Troisième␣
 ↪couche de convolution avec 32 filtres de taille 4x4, une activation ReLU.
    layers.MaxPooling2D(pool_size=(2,2)), # Troisième couche de MaxPooling.
    layers.BatchNormalization(), # Normalisation par lots.
    layers.Flatten(), # Aplatir les sorties pour les transformer en vecteur␣
 ↪avant de les passer à la couche dense.
    layers.Dense(32, activation = 'relu'), # Première couche dense avec 32␣
 ↪unités et une activation ReLU.
    layers.Dropout(0.15), # Couche de Dropout pour réduire le surajustement en␣
 ↪désactivant aléatoirement 15% des neurones.
```

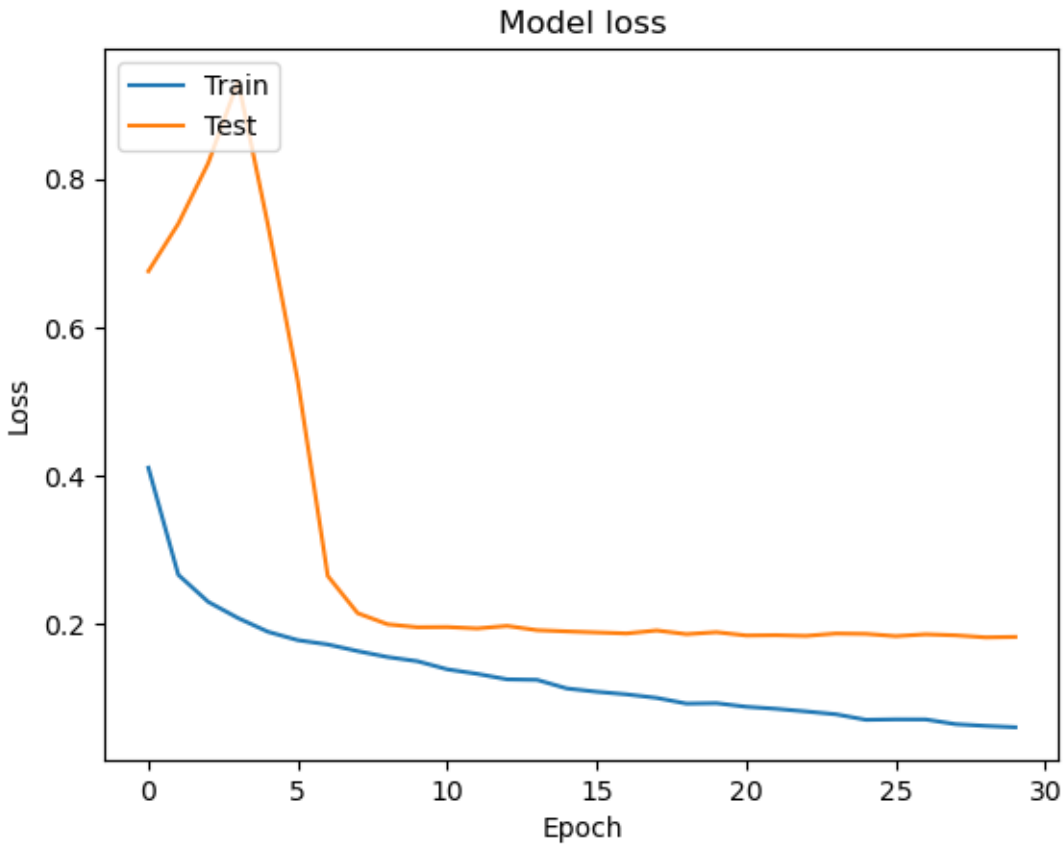
```
    layers.Dense(2, activation = 'softmax') # Couche de sortie avec activation␣
 ↪Softmax pour la classification binaire.
])

# Compilation du modèle.
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate = 1e-5), # Utilisation de␣
 ↪l'optimiseur Adam avec un faible taux d'apprentissage pour une convergence␣
 ↪stable.
    loss=tf.keras.losses.BinaryCrossentropy(), # Utilisation de la fonction de␣
 ↪perte BinaryCrossentropy pour un problème de classification binaire.
    metrics=['acc'] # Suivi de la précision ('acc') comme métrique pour évaluer␣
 ↪la performance du modèle.
)

# Entraînement du modèle sur les données d'entraînement avec validation sur les␣
 ↪données de validation.
history = model.fit(
    train_batches,  # Données d'entraînement.
    epochs=30,      # Nombre d'époques pour l'entraînement.
    validation_data=val_batches  # Données de validation pour évaluer la␣
 ↪performance du modèle après chaque époque.
)


# Affichage des graphiques de l'évolution de la précision et de la perte au␣
 ↪cours des époques.
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```

```
Epoch 1/30
81/81 [==============================] - 15s 182ms/step - loss: 0.4108 - acc:
0.8306 - val_loss: 0.6755 - val_acc: 0.5073
Epoch 2/30
```

```
81/81 [==============================] - 5s 58ms/step - loss: 0.2667 - acc:
0.8954 - val_loss: 0.7394 - val_acc: 0.5073
Epoch 3/30
81/81 [==============================] - 5s 57ms/step - loss: 0.2303 - acc:
0.9157 - val_loss: 0.8209 - val_acc: 0.5073
Epoch 4/30
81/81 [==============================] - 5s 57ms/step - loss: 0.2084 - acc:
0.9247 - val_loss: 0.9312 - val_acc: 0.5200
Epoch 5/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1900 - acc:
0.9329 - val_loss: 0.7377 - val_acc: 0.6164
Epoch 6/30
81/81 [==============================] - 5s 57ms/step - loss: 0.1787 - acc:
0.9403 - val_loss: 0.5271 - val_acc: 0.7382
Epoch 7/30
81/81 [==============================] - 5s 57ms/step - loss: 0.1729 - acc:
0.9391 - val_loss: 0.2652 - val_acc: 0.8964
Epoch 8/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1639 - acc:
0.9461 - val_loss: 0.2150 - val_acc: 0.9364
Epoch 9/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1559 - acc:
0.9489 - val_loss: 0.2001 - val_acc: 0.9364
Epoch 10/30
81/81 [==============================] - 5s 57ms/step - loss: 0.1503 - acc:
0.9504 - val_loss: 0.1961 - val_acc: 0.9400
Epoch 11/30
81/81 [==============================] - 5s 57ms/step - loss: 0.1394 - acc:
0.9528 - val_loss: 0.1963 - val_acc: 0.9382
Epoch 12/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1333 - acc:
0.9543 - val_loss: 0.1945 - val_acc: 0.9364
Epoch 13/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1259 - acc:
0.9547 - val_loss: 0.1979 - val_acc: 0.9291
Epoch 14/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1252 - acc:
0.9586 - val_loss: 0.1921 - val_acc: 0.9400
Epoch 15/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1136 - acc:
0.9633 - val_loss: 0.1905 - val_acc: 0.9345
Epoch 16/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1092 - acc:
0.9680 - val_loss: 0.1891 - val_acc: 0.9364
Epoch 17/30
81/81 [==============================] - 5s 58ms/step - loss: 0.1057 - acc:
0.9688 - val_loss: 0.1877 - val_acc: 0.9400
Epoch 18/30
```

```
81/81 [==============================] - 5s 59ms/step - loss: 0.1009 - acc:
0.9723 - val_loss: 0.1917 - val_acc: 0.9327
Epoch 19/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0933 - acc:
0.9774 - val_loss: 0.1868 - val_acc: 0.9400
Epoch 20/30
81/81 [==============================] - 5s 59ms/step - loss: 0.0938 - acc:
0.9758 - val_loss: 0.1894 - val_acc: 0.9327
Epoch 21/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0890 - acc:
0.9754 - val_loss: 0.1850 - val_acc: 0.9345
Epoch 22/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0862 - acc:
0.9789 - val_loss: 0.1854 - val_acc: 0.9345
Epoch 23/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0827 - acc:
0.9797 - val_loss: 0.1844 - val_acc: 0.9364
Epoch 24/30
81/81 [==============================] - 5s 59ms/step - loss: 0.0789 - acc:
0.9789 - val_loss: 0.1876 - val_acc: 0.9364
Epoch 25/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0715 - acc:
0.9828 - val_loss: 0.1872 - val_acc: 0.9364
Epoch 26/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0720 - acc:
0.9805 - val_loss: 0.1841 - val_acc: 0.9400
Epoch 27/30
81/81 [==============================] - 5s 59ms/step - loss: 0.0719 - acc:
0.9793 - val_loss: 0.1865 - val_acc: 0.9345
Epoch 28/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0656 - acc:
0.9856 - val_loss: 0.1852 - val_acc: 0.9382
Epoch 29/30
81/81 [==============================] - 5s 59ms/step - loss: 0.0632 - acc:
0.9840 - val_loss: 0.1824 - val_acc: 0.9436
Epoch 30/30
81/81 [==============================] - 5s 58ms/step - loss: 0.0614 - acc:
0.9859 - val_loss: 0.1830 - val_acc: 0.9400
```


Model accuracy
Train
Test
Accuracy
Epoch
1.0
0.9
0.8
0.7
0.6
0.5
0
5
10
15
20
25
30



[28]:
```
# Sauvegarde du modèle entraîné.
model.save('64x3-CNN.model')
```

```
INFO:tensorflow:Assets written to: 64x3-CNN.model\assets

INFO:tensorflow:Assets written to: 64x3-CNN.model\assets
```

[29]:
```
# Évaluation du modèle sur l'ensemble de test.
loss, acc = model.evaluate_generator(test_batches, verbose=1)
print("Accuracy: ", acc)
```

```
18/18 [==============================] - 3s 182ms/step - loss: 0.1481 - acc:
0.9564
Accuracy:  0.9563636183738708
```

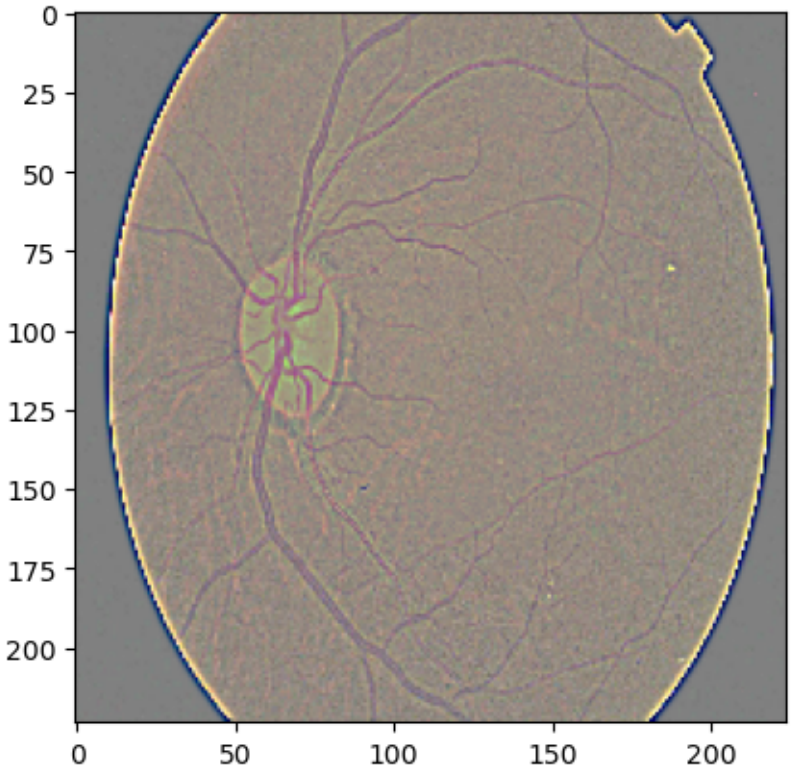
5 Détection de la rétinopathie diabétique

[30]:
```
# Importation des bibliothèques pour la prédiction sur une nouvelle image.
import tensorflow as tf
import cv2
```

```
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

# Fonction pour prédire la classe d'une image (présence ou non de rétinopathie␣
 ↪diabétique).
def predict_class(path):
    img = cv2.imread(path) # Lire l'image à partir du chemin spécifié.
    RGBImg = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Convertir l'image de BGR␣
 ↪(format OpenCV par défaut) en RGB.
    RGBImg = cv2.resize(RGBImg, (224, 224)) # Redimensionner l'image pour␣
 ↪correspondre à la taille d'entrée attendue par le modèle (224x224).
    plt.imshow(RGBImg) # Afficher l'image RGB redimensionnée.
    image = np.array(RGBImg) / 255.0 # Normaliser les valeurs des pixels de␣
 ↪l'image à une plage de [0, 1].
    new_model = tf.keras.models.load_model("64x3-CNN.model") # Charger le␣
 ↪modèle entraîné précédemment sauvegardé.
    predict = new_model.predict(np.array([image])) # Prédire la classe de␣
 ↪l'image en utilisant le modèle. L'image est encapsulée dans une liste pour␣
 ↪correspondre à la forme d'entrée attendue.
    per = np.argmax(predict, axis=1) # Utiliser argmax pour obtenir l'indice de␣
 ↪la classe ayant la plus haute probabilité.
    if per == 1:
        print('No DR') # Si l'indice est 1, écrire 'No DR' (Pas de Rétinopathie␣
 ↪Diabétique).
    else:
        print('DR') # Sinon, écrire 'DR' (Rétinopathie Diabétique).
```

[31]:
```
predict_class(r'C:\Users\ademb\Documents\diabetic␣
 ↪retinopathy_detection\data\gaussian_filtered_images\Mild\0a61bddab956.png')
```

```
1/1 [==============================] - 0s 56ms/step
DR
```


0
25
50
75
100
125
150
175
200
0
50
100
150
200