Ocular disease prediction - AlexNet

Claudio Pella - 5006427

I used AlexNet to train a model to classify images in order to predict what disease the eye has. It is a Multiclass classification problem (despite what I said during the interview), because each eye has exactly one label, not several. The data needs to be cleaned: when I have 2 labels in a single lines, it simply means that the patient has overall 1 label per eye. The dataset doesn't not distinguish from right and left eye: for example, if patient #3 has a Normal Eye and a Miopia on the other one, both lines will show the labels N and M. It is a typo from older versions of the dataset.

Dataset: https://www.kaggle.com/andrewmvd/ocular-disease-recognition-odir5k

Libraries

In [43]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount
("/content/drive", force_remount=True).
```

In [44]:

```
import pandas as pd
import plotly.offline as pyo
pyo.init_notebook_mode()
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import cv2
from plotly.subplots import make_subplots
import plotly.graph_objects as go
from sklearn import preprocessing
import random
import tensorflow as tf
import warnings
warnings.filterwarnings("ignore")
!pip install visualkeras

%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
from glob import glob
import seaborn as sns
from PIL import Image
np.random.seed(123)
from sklearn.preprocessing import label_binarize
from sklearn.metrics import confusion_matrix
import itertools

import keras
from keras.utils.np_utils import to_categorical # used for converting labels to one-hot-e
ncoding
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D
from keras import backend as K
import itertools
from keras.layers import Convolution2D
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import (
    BatchNormalization, SeparableConv2D, MaxPooling2D, Activation, Flatten, Dropout, Den
se
)
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding

from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
```

```
Requirement already satisfied: visualkeras in /usr/local/lib/python3.7/dist-packages (0.0
.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-packages (f
rom visualkeras) (7.1.2)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.7/dist-packages (f
rom visualkeras) (1.19.5)
Requirement already satisfied: aggdraw>=1.3.11 in /usr/local/lib/python3.7/dist-packages
(from visualkeras) (1.3.12)
```

Data Loading and cleaning

After uploading the dataset, I renamed the values in the column filename so that it would match the exact location of the file. I uploaded the dataset on my drive in order to use GPU accellerator in Colab. The dataset made a distinction between left and right eyes (since they belong to the same patient): to increase the number of cases, I decided to treat them as different, therefore I modified the two columns with the diagnostics, since for each line we had diagnostics for both eyes, keeping only the one to which the description is referring to.

In [63]:

```
df = pd.read_csv('/content/drive/MyDrive/Oculus/full_df.csv')
df['filename']='/content/drive/MyDrive/Oculus/preprocessed_images/'+df['filename']
df['Diagnostic Keywords'] = df['Left-Diagnostic Keywords']
df['Diagnostic Keywords'][0:3193]=df['Right-Diagnostic Keywords'][0:3193]

df['labels'].value_counts()
```

Out[63]:

```
['N']    2873
['D']    1608
['O']     708
['C']     293
['G']     284
['A']     266
['M']     232
['H']     128
Name: labels, dtype: int64
```

As we can see, the dataset is very unbalanced. We have almost 3000 normal eyes and only 128 with Hypertension. Therefore I decided to group up the small labels into a single one: CHAMG stands for Cataract, Hypertesion, Age related macular degeneration, Miopia and Glaucoma. To do so I implemented a dictionary.

In [65]:

```
cat_dict1 = {
**dict.fromkeys(["['N']"],'Normal'),
**dict.fromkeys( ["['D']"],'Diabetes'),
**dict.fromkeys( ["['O']"],'Other'),
**dict.fromkeys( ["['G']","['M']","['A']","['C']","['H']"],'C-H-A-M-G'),
}

df['label']=df['labels'].map(cat_dict1)
num_classes = 4
```

Still, the Normal category has too many cases, therefore I decided to drop 1000 of them, to make the dataset more balanced.

In [66]:

```
label_list = df['label'].value_counts().index.tolist()

df_balanced = df.loc[df['label'] == 'Normal' , : ].sample(1700).copy()
for label in label_list:
    if label != 'Normal':
        sample_df = df.loc[df['label'] == label , : ].sample(frac=1).copy()
        df_balanced = pd.concat([ df_balanced , sample_df],axis = 0 , ignore_index=True)

    # Shuffle data
df_balanced = df_balanced.sample(frac = 1).reset_index(drop = True)
```

In [67]:

```
df = df_balanced
df.head()
```

Out[67]:

	ID	Patient Age	Patient Sex	Left-Fundus	Right-Fundus	Left-Diagnostic Keywords	Right-Diagnostic Keywords	N	D	G	C	A	H	M	O	filepath	la
0	2356	51	Male	2356_left.jpg	2356_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	
1	199	50	Female	199_left.jpg	199_right.jpg	branch retinal vein occlusion	moderate non proliferative retinopathy	0	1	0	0	0	0	0	1	../input/ocular-disease-recognition-odir5k/ODI...	
2	3300	60	Female	3300_left.jpg	3300_right.jpg	normal fundus , lens dust	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	
3	3034	48	Male	3034_left.jpg	3034_right.jpg	normal fundus	normal fundus	1	0	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	
4	4210	62	Female	4210_left.jpg	4210_right.jpg	moderate non proliferative retinopathy	moderate non proliferative retinopathy	0	1	0	0	0	0	0	0	../input/ocular-disease-recognition-odir5k/ODI...	

In [68]:

```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df['labels']= label_encoder.fit_transform(df['label'])

# How many cases do we have for each category?
df['label'].value_counts()
```

Out[68]:

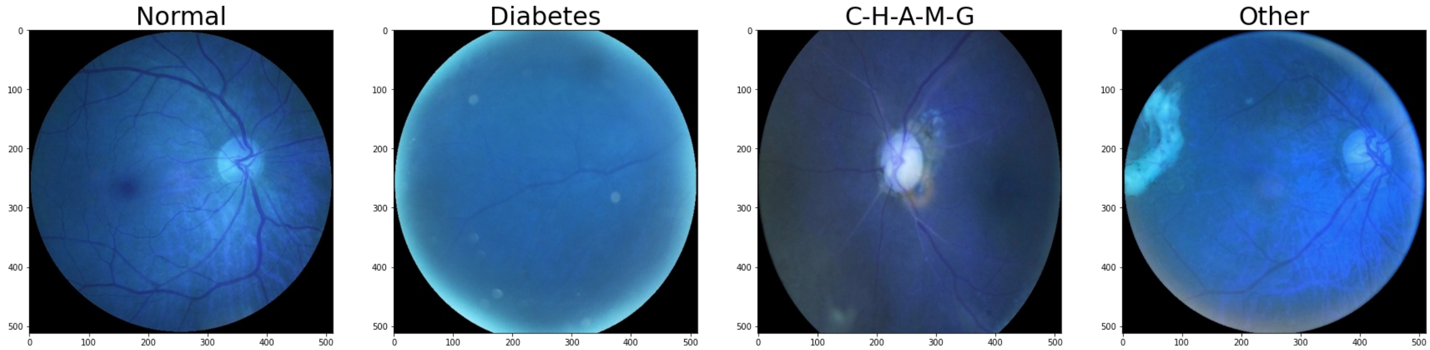
```
Normal       1700
Diabetes     1608
C-H-A-M-G    1203
Other         708
Name: label, dtype: int64
```

Now, let's see a few images:

In [69]:

```
count=1
```

```
f = plt.figure(figsize=(30,20))
for Class in label_list:
    seg = df[df['label']==Class]
    address = seg.sample().iloc[0]['filename']
    img = cv2.imread(address)
    ax = f.add_subplot(1, 4,count)
    ax = plt.imshow(img)
    count += 1
    ax = plt.title(Class,fontsize= 30)
plt.show()
```



Let's drop some irrelevant (for my analysis) columns:

In [70]:

```
df = df.drop(['filepath','Patient Sex','Patient Age', 'Left-Fundus','Right-Fundus','Righ
t-Diagnostic Keywords','Left-Diagnostic Keywords'], axis=1)
```

Pre-processing

Let's pre-process the images. The dataset provides images of 512x512, so I decided to resize them (by exactly half).

In [71]:

```
df['image'] = df['filename'].map(lambda x: np.asarray(Image.open(x).resize((256,256))))
```

In [72]:

```
df['image'].map(lambda x: x.shape) #size of the input
```

Out[72]:

```
0        (256, 256, 3)
1        (256, 256, 3)
2        (256, 256, 3)
3        (256, 256, 3)
4        (256, 256, 3)
             ...
5214     (256, 256, 3)
5215     (256, 256, 3)
5216     (256, 256, 3)
5217     (256, 256, 3)
5218     (256, 256, 3)
Name: image, Length: 5219, dtype: object
```

Implementation of the Neural Network

I used AlexNet with standard parameters. Since it is a multiclass classification, i used a softmax function.

In [73]:

```
import tensorflow as tf
tf.config.run_functions_eagerly(True)
```

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten , Conv1D
from tensorflow.keras.layers import concatenate
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D,MaxPooling1D
from tensorflow.keras.utils import plot_model

input_shape = (256, 256, 3)

#Initialization
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=input_shape, kernel_size=(11,11), strides=(4,
4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2), padding='same'))

#Flattening
AlexNet.add(Flatten())

# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.5))

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.5))

#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(Dropout(0.5))

#Output Layer
AlexNet.add(Dense(num_classes))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))

#Model Summary
AlexNet.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_15 (Conv2D)          (None, 64, 64, 96)        34944     
                                                                 
 batch_normalization_27 (Bat  (None, 64, 64, 96)       384       
 chNormalization)                                                
                                                                 
 activation_27 (Activation)  (None, 64, 64, 96)        0         
                                                                 
 max_pooling2d_9 (MaxPooling  (None, 32, 32, 96)       0         
 2D)                                                             
                                                                 
 conv2d_16 (Conv2D)          (None, 32, 32, 256)       614656    
                                                                 
 batch_normalization_28 (Bat  (None, 32, 32, 256)      1024      
 chNormalization)                                                
                                                                 
 activation_28 (Activation)  (None, 32, 32, 256)       0         
                                                                 
 max_pooling2d_10 (MaxPoolin  (None, 16, 16, 256)      0         
 g2D)                                                            
                                                                 
 conv2d_17 (Conv2D)          (None, 16, 16, 384)       885120    
                                                                 
 batch_normalization_29 (Bat  (None, 16, 16, 384)      1536      
 chNormalization)                                                
                                                                 
 activation_29 (Activation)  (None, 16, 16, 384)       0         
                                                                 
 conv2d_18 (Conv2D)          (None, 16, 16, 384)       1327488   
                                                                 
 batch_normalization_30 (Bat  (None, 16, 16, 384)      1536      
 chNormalization)                                                
                                                                 
 activation_30 (Activation)  (None, 16, 16, 384)       0         
                                                                 
 conv2d_19 (Conv2D)          (None, 16, 16, 256)       884992    
                                                                 
 batch_normalization_31 (Bat  (None, 16, 16, 256)      1024      
 chNormalization)                                                
                                                                 
 activation_31 (Activation)  (None, 16, 16, 256)       0         
                                                                 
 max_pooling2d_11 (MaxPoolin  (None, 8, 8, 256)        0         
 g2D)                                                            
                                                                 
 flatten_3 (Flatten)         (None, 16384)             0         
                                                                 
 dense_12 (Dense)            (None, 4096)              67112960  
                                                                 
 batch_normalization_32 (Bat  (None, 4096)             16384     
 chNormalization)                                                
                                                                 
 activation_32 (Activation)  (None, 4096)              0         
                                                                 
 dropout_9 (Dropout)         (None, 4096)              0         
                                                                 
 dense_13 (Dense)            (None, 4096)              16781312  
                                                                 
 batch_normalization_33 (Bat  (None, 4096)             16384     
 chNormalization)                                                
                                                                 
 activation_33 (Activation)  (None, 4096)              0         
                                                                 
 dropout_10 (Dropout)        (None, 4096)              0         
                                                                 
 dense_14 (Dense)            (None, 1000)              4097000   
                                                                 
 batch_normalization_34 (Bat  (None, 1000)             4000      
 chNormalization)                                                
```

```
 activation_34 (Activation)   (None, 1000)              0

 dropout_11 (Dropout)         (None, 1000)              0

 dense_15 (Dense)             (None, 4)                 4004

 batch_normalization_35 (Bat  (None, 4)                 16
 chNormalization)

 activation_35 (Activation)   (None, 4)                 0

=================================================================
Total params: 91,784,764
Trainable params: 91,763,620
Non-trainable params: 21,144
_________________________________________________________________
```

A brief representation of the neural network.

As metrics, I decided to use both accuracy and f1. I re-balanced the dataset in order to be able to use accuracy, but it is still not perfectly balanced and therefore I'll computer F1 score, which is done by having precision and recall.

In [75]:

```
# Compile the model
METRICS = [
                'accuracy',
                tf.keras.metrics.Precision(),
                tf.keras.metrics.Recall(),
        ]


AlexNet.compile(optimizer = "Adam" , loss = "categorical_crossentropy", metrics=METRICS)

learning_rate_reduction = ReduceLROnPlateau(min_lr=0.00001)
```

While trying to improve the goodness of the model, I tried to set a decreasing learning rate.

Train-Test splitting

In [76]:

```
features=df['image']
target=df['labels']

x_train_o, x_test_o, y_train_o, y_test_o = train_test_split(features, target, test_size=0
.15,random_state=1234)

x_train = np.asarray(x_train_o.tolist())
x_test = np.asarray(x_test_o.tolist())
```

In [77]:

```
y_train = to_categorical(y_train_o, num_classes = num_classes)
y_test = to_categorical(y_test_o, num_classes = num_classes)

x_train, x_validate, y_train, y_validate = train_test_split(x_train, y_train, test_size
= 0.15, random_state = 2)

len(x_train)
```

Out[77]:

```
3770
```

Training

In [78]:

```
# Fit the model
epochs = 50
history = AlexNet.fit(x_train,y_train, batch_size=32,
                              epochs = epochs, validation_data = (x_validate,y_validate)
,
                              verbose = 1
                               , callbacks=[learning_rate_reduction])
```

```
Epoch 1/50
118/118 [==============================] - 28s 235ms/step - loss: 1.4374 - accuracy: 0.32
79 - precision_4: 0.3488 - recall_4: 0.0279 - val_loss: 7.7386 - val_accuracy: 0.3033 - v
al_precision_4: 0.3040 - val_recall_4: 0.2853 - lr: 0.0010
Epoch 2/50
118/118 [==============================] - 27s 227ms/step - loss: 1.3478 - accuracy: 0.34
69 - precision_4: 0.4054 - recall_4: 0.0040 - val_loss: 1.4594 - val_accuracy: 0.3228 - v
al_precision_4: 0.3025 - val_recall_4: 0.0736 - lr: 0.0010
Epoch 3/50
118/118 [==============================] - 28s 236ms/step - loss: 1.3234 - accuracy: 0.38
75 - precision_4: 0.3766 - recall_4: 0.0077 - val_loss: 1.4017 - val_accuracy: 0.3333 - v
al_precision_4: 0.3661 - val_recall_4: 0.0616 - lr: 0.0010
Epoch 4/50
118/118 [==============================] - 28s 237ms/step - loss: 1.3102 - accuracy: 0.36
84 - precision_4: 0.5055 - recall_4: 0.0122 - val_loss: 1.3235 - val_accuracy: 0.3438 - v
al_precision_4: 1.0000 - val_recall_4: 0.0015 - lr: 0.0010
Epoch 5/50
118/118 [==============================] - 28s 237ms/step - loss: 1.2951 - accuracy: 0.38
70 - precision_4: 0.5645 - recall_4: 0.0279 - val_loss: 1.3913 - val_accuracy: 0.3544 - v
al_precision_4: 0.4409 - val_recall_4: 0.0616 - lr: 0.0010
Epoch 6/50
118/118 [==============================] - 27s 227ms/step - loss: 1.2892 - accuracy: 0.39
52 - precision_4: 0.5722 - recall_4: 0.0294 - val_loss: 1.3227 - val_accuracy: 0.3634 - v
al_precision_4: 0.5200 - val_recall_4: 0.0390 - lr: 0.0010
Epoch 7/50
118/118 [==============================] - 28s 236ms/step - loss: 1.2744 - accuracy: 0.41
30 - precision_4: 0.6082 - recall_4: 0.0515 - val_loss: 1.3794 - val_accuracy: 0.3559 - v
al_precision_4: 0.4000 - val_recall_4: 0.1051 - lr: 0.0010
Epoch 8/50
118/118 [==============================] - 28s 236ms/step - loss: 1.2645 - accuracy: 0.41
30 - precision_4: 0.5958 - recall_4: 0.0602 - val_loss: 1.3025 - val_accuracy: 0.3634 - v
al_precision_4: 0.5765 - val_recall_4: 0.0736 - lr: 0.0010
Epoch 9/50
118/118 [==============================] - 28s 237ms/step - loss: 1.2480 - accuracy: 0.42
28 - precision_4: 0.6111 - recall_4: 0.0759 - val_loss: 1.3323 - val_accuracy: 0.3934 - v
al_precision_4: 0.5161 - val_recall_4: 0.1201 - lr: 0.0010
Epoch 10/50
118/118 [==============================] - 28s 236ms/step - loss: 1.2351 - accuracy: 0.42
94 - precision_4: 0.5744 - recall_4: 0.0952 - val_loss: 1.3262 - val_accuracy: 0.3739 - v
al_precision_4: 0.5893 - val_recall_4: 0.0495 - lr: 0.0010
Epoch 11/50
118/118 [==============================] - 28s 238ms/step - loss: 1.2274 - accuracy: 0.43
66 - precision_4: 0.6254 - recall_4: 0.1151 - val_loss: 1.3873 - val_accuracy: 0.3498 - v
al_precision_4: 0.4286 - val_recall_4: 0.0991 - lr: 0.0010
Epoch 12/50
118/118 [==============================] - 27s 229ms/step - loss: 1.2214 - accuracy: 0.44
96 - precision_4: 0.5954 - recall_4: 0.1233 - val_loss: 1.3849 - val_accuracy: 0.3664 - v
al_precision_4: 0.3955 - val_recall_4: 0.1592 - lr: 0.0010
Epoch 13/50
118/118 [==============================] - 27s 228ms/step - loss: 1.2081 - accuracy: 0.45
46 - precision_4: 0.6090 - recall_4: 0.1363 - val_loss: 1.3185 - val_accuracy: 0.3874 - v
al_precision_4: 0.4342 - val_recall_4: 0.0991 - lr: 0.0010
Epoch 14/50
118/118 [==============================] - 27s 228ms/step - loss: 1.1871 - accuracy: 0.46
21 - precision_4: 0.6010 - recall_4: 0.1674 - val_loss: 1.4283 - val_accuracy: 0.3348 - v
al_precision_4: 0.3808 - val_recall_4: 0.1486 - lr: 0.0010
Epoch 15/50
118/118 [==============================] - 28s 237ms/step - loss: 1.1789 - accuracy: 0.47
24 - precision_4: 0.6039 - recall_4: 0.1889 - val_loss: 1.4264 - val_accuracy: 0.3168 - v
al_precision_4: 0.3491 - val_recall_4: 0.1216 - lr: 0.0010
```

```
Epoch 16/50
118/118 [==============================] - 28s 237ms/step - loss: 1.1603 - accuracy: 0.48
70 - precision_4: 0.5992 - recall_4: 0.1979 - val_loss: 1.3550 - val_accuracy: 0.3799 - v
al_precision_4: 0.3972 - val_recall_4: 0.1682 - lr: 0.0010
Epoch 17/50
118/118 [==============================] - 27s 228ms/step - loss: 1.1358 - accuracy: 0.48
83 - precision_4: 0.6157 - recall_4: 0.2265 - val_loss: 1.3013 - val_accuracy: 0.3919 - v
al_precision_4: 0.4948 - val_recall_4: 0.1441 - lr: 0.0010
Epoch 18/50
118/118 [==============================] - 27s 229ms/step - loss: 1.1082 - accuracy: 0.50
11 - precision_4: 0.6348 - recall_4: 0.2716 - val_loss: 1.3577 - val_accuracy: 0.3949 - v
al_precision_4: 0.4398 - val_recall_4: 0.2688 - lr: 0.0010
Epoch 19/50
118/118 [==============================] - 27s 227ms/step - loss: 1.0810 - accuracy: 0.52
57 - precision_4: 0.6482 - recall_4: 0.2952 - val_loss: 1.3274 - val_accuracy: 0.4054 - v
al_precision_4: 0.4789 - val_recall_4: 0.2042 - lr: 0.0010
Epoch 20/50
118/118 [==============================] - 28s 234ms/step - loss: 1.0536 - accuracy: 0.54
01 - precision_4: 0.6504 - recall_4: 0.3103 - val_loss: 1.3066 - val_accuracy: 0.4174 - v
al_precision_4: 0.4734 - val_recall_4: 0.2538 - lr: 0.0010
Epoch 21/50
118/118 [==============================] - 28s 235ms/step - loss: 1.0305 - accuracy: 0.56
31 - precision_4: 0.6667 - recall_4: 0.3650 - val_loss: 1.3591 - val_accuracy: 0.4249 - v
al_precision_4: 0.4741 - val_recall_4: 0.2748 - lr: 0.0010
Epoch 22/50
118/118 [==============================] - 26s 225ms/step - loss: 0.9810 - accuracy: 0.57
53 - precision_4: 0.6926 - recall_4: 0.3997 - val_loss: 1.4275 - val_accuracy: 0.3994 - v
al_precision_4: 0.4282 - val_recall_4: 0.2823 - lr: 0.0010
Epoch 23/50
118/118 [==============================] - 28s 233ms/step - loss: 0.9302 - accuracy: 0.60
85 - precision_4: 0.7000 - recall_4: 0.4387 - val_loss: 1.4261 - val_accuracy: 0.3904 - v
al_precision_4: 0.4115 - val_recall_4: 0.2372 - lr: 0.0010
Epoch 24/50
118/118 [==============================] - 27s 226ms/step - loss: 0.9078 - accuracy: 0.62
20 - precision_4: 0.7236 - recall_4: 0.4618 - val_loss: 1.4330 - val_accuracy: 0.4384 - v
al_precision_4: 0.4843 - val_recall_4: 0.3709 - lr: 0.0010
Epoch 25/50
118/118 [==============================] - 27s 233ms/step - loss: 0.8495 - accuracy: 0.65
89 - precision_4: 0.7548 - recall_4: 0.5103 - val_loss: 1.4784 - val_accuracy: 0.3694 - v
al_precision_4: 0.4286 - val_recall_4: 0.2432 - lr: 0.0010
Epoch 26/50
118/118 [==============================] - 28s 234ms/step - loss: 0.8071 - accuracy: 0.68
20 - precision_4: 0.7652 - recall_4: 0.5507 - val_loss: 1.6624 - val_accuracy: 0.4069 - v
al_precision_4: 0.4214 - val_recall_4: 0.3544 - lr: 0.0010
Epoch 27/50
118/118 [==============================] - 27s 225ms/step - loss: 0.7678 - accuracy: 0.69
47 - precision_4: 0.7805 - recall_4: 0.5875 - val_loss: 1.5827 - val_accuracy: 0.4039 - v
al_precision_4: 0.4205 - val_recall_4: 0.3574 - lr: 0.0010
Epoch 28/50
118/118 [==============================] - 27s 225ms/step - loss: 0.5817 - accuracy: 0.81
46 - precision_4: 0.8849 - recall_4: 0.6973 - val_loss: 1.3908 - val_accuracy: 0.4354 - v
al_precision_4: 0.4839 - val_recall_4: 0.3378 - lr: 1.0000e-04
Epoch 29/50
118/118 [==============================] - 28s 234ms/step - loss: 0.5311 - accuracy: 0.83
69 - precision_4: 0.9067 - recall_4: 0.7347 - val_loss: 1.3788 - val_accuracy: 0.4429 - v
al_precision_4: 0.4791 - val_recall_4: 0.3438 - lr: 1.0000e-04
Epoch 30/50
118/118 [==============================] - 28s 233ms/step - loss: 0.5119 - accuracy: 0.84
46 - precision_4: 0.9107 - recall_4: 0.7462 - val_loss: 1.3837 - val_accuracy: 0.4384 - v
al_precision_4: 0.4924 - val_recall_4: 0.3408 - lr: 1.0000e-04
Epoch 31/50
118/118 [==============================] - 28s 236ms/step - loss: 0.4800 - accuracy: 0.86
90 - precision_4: 0.9297 - recall_4: 0.7682 - val_loss: 1.3825 - val_accuracy: 0.4474 - v
al_precision_4: 0.4837 - val_recall_4: 0.3559 - lr: 1.0000e-04
Epoch 32/50
118/118 [==============================] - 28s 235ms/step - loss: 0.4651 - accuracy: 0.87
14 - precision_4: 0.9279 - recall_4: 0.7822 - val_loss: 1.4007 - val_accuracy: 0.4489 - v
al_precision_4: 0.4657 - val_recall_4: 0.3363 - lr: 1.0000e-04
Epoch 33/50
118/118 [==============================] - 28s 235ms/step - loss: 0.4371 - accuracy: 0.88
67 - precision_4: 0.9364 - recall_4: 0.8082 - val_loss: 1.4102 - val_accuracy: 0.4399 - v
al_precision_4: 0.4688 - val_recall_4: 0.3273 - lr: 1.0000e-04
```

```
Epoch 34/50
118/118 [==============================] - 28s 235ms/step - loss: 0.4264 - accuracy: 0.88
41 - precision_4: 0.9336 - recall_4: 0.8162 - val_loss: 1.3977 - val_accuracy: 0.4339 - v
al_precision_4: 0.4919 - val_recall_4: 0.3634 - lr: 1.0000e-04
Epoch 35/50
118/118 [==============================] - 27s 226ms/step - loss: 0.4088 - accuracy: 0.89
55 - precision_4: 0.9473 - recall_4: 0.8302 - val_loss: 1.4115 - val_accuracy: 0.4474 - v
al_precision_4: 0.4850 - val_recall_4: 0.3393 - lr: 1.0000e-04
Epoch 36/50
118/118 [==============================] - 28s 234ms/step - loss: 0.3931 - accuracy: 0.90
24 - precision_4: 0.9477 - recall_4: 0.8358 - val_loss: 1.4131 - val_accuracy: 0.4535 - v
al_precision_4: 0.4920 - val_recall_4: 0.3709 - lr: 1.0000e-04
Epoch 37/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3786 - accuracy: 0.91
64 - precision_4: 0.9515 - recall_4: 0.8533 - val_loss: 1.4171 - val_accuracy: 0.4489 - v
al_precision_4: 0.4911 - val_recall_4: 0.3739 - lr: 1.0000e-04
Epoch 38/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3531 - accuracy: 0.92
52 - precision_4: 0.9599 - recall_4: 0.8695 - val_loss: 1.4096 - val_accuracy: 0.4429 - v
al_precision_4: 0.4910 - val_recall_4: 0.3679 - lr: 1.0000e-05
Epoch 39/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3520 - accuracy: 0.92
49 - precision_4: 0.9630 - recall_4: 0.8690 - val_loss: 1.4085 - val_accuracy: 0.4474 - v
al_precision_4: 0.5052 - val_recall_4: 0.3679 - lr: 1.0000e-05
Epoch 40/50
118/118 [==============================] - 28s 236ms/step - loss: 0.3454 - accuracy: 0.92
97 - precision_4: 0.9622 - recall_4: 0.8772 - val_loss: 1.4075 - val_accuracy: 0.4489 - v
al_precision_4: 0.5000 - val_recall_4: 0.3649 - lr: 1.0000e-05
Epoch 41/50
118/118 [==============================] - 28s 237ms/step - loss: 0.3482 - accuracy: 0.93
29 - precision_4: 0.9635 - recall_4: 0.8690 - val_loss: 1.4073 - val_accuracy: 0.4444 - v
al_precision_4: 0.5020 - val_recall_4: 0.3709 - lr: 1.0000e-05
Epoch 42/50
118/118 [==============================] - 27s 226ms/step - loss: 0.3499 - accuracy: 0.92
55 - precision_4: 0.9612 - recall_4: 0.8684 - val_loss: 1.4107 - val_accuracy: 0.4474 - v
al_precision_4: 0.5020 - val_recall_4: 0.3679 - lr: 1.0000e-05
Epoch 43/50
118/118 [==============================] - 28s 235ms/step - loss: 0.3416 - accuracy: 0.93
00 - precision_4: 0.9642 - recall_4: 0.8796 - val_loss: 1.4090 - val_accuracy: 0.4489 - v
al_precision_4: 0.5061 - val_recall_4: 0.3709 - lr: 1.0000e-05
Epoch 44/50
118/118 [==============================] - 28s 235ms/step - loss: 0.3451 - accuracy: 0.92
89 - precision_4: 0.9622 - recall_4: 0.8772 - val_loss: 1.4091 - val_accuracy: 0.4459 - v
al_precision_4: 0.5062 - val_recall_4: 0.3679 - lr: 1.0000e-05
Epoch 45/50
118/118 [==============================] - 28s 236ms/step - loss: 0.3375 - accuracy: 0.93
42 - precision_4: 0.9696 - recall_4: 0.8796 - val_loss: 1.4154 - val_accuracy: 0.4444 - v
al_precision_4: 0.5010 - val_recall_4: 0.3694 - lr: 1.0000e-05
Epoch 46/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3427 - accuracy: 0.93
29 - precision_4: 0.9652 - recall_4: 0.8767 - val_loss: 1.4146 - val_accuracy: 0.4459 - v
al_precision_4: 0.4990 - val_recall_4: 0.3724 - lr: 1.0000e-05
Epoch 47/50
118/118 [==============================] - 27s 225ms/step - loss: 0.3424 - accuracy: 0.92
63 - precision_4: 0.9564 - recall_4: 0.8785 - val_loss: 1.4173 - val_accuracy: 0.4459 - v
al_precision_4: 0.5000 - val_recall_4: 0.3724 - lr: 1.0000e-05
Epoch 48/50
118/118 [==============================] - 28s 235ms/step - loss: 0.3242 - accuracy: 0.93
82 - precision_4: 0.9679 - recall_4: 0.8886 - val_loss: 1.4168 - val_accuracy: 0.4444 - v
al_precision_4: 0.4970 - val_recall_4: 0.3694 - lr: 1.0000e-05
Epoch 49/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3325 - accuracy: 0.93
40 - precision_4: 0.9629 - recall_4: 0.8891 - val_loss: 1.4186 - val_accuracy: 0.4505 - v
al_precision_4: 0.4940 - val_recall_4: 0.3679 - lr: 1.0000e-05
Epoch 50/50
118/118 [==============================] - 27s 227ms/step - loss: 0.3335 - accuracy: 0.93
77 - precision_4: 0.9690 - recall_4: 0.8873 - val_loss: 1.4213 - val_accuracy: 0.4399 - v
al_precision_4: 0.4960 - val_recall_4: 0.3709 - lr: 1.0000e-05
```

Testing

Now let's test. As I said before, I am going to use the F1 score, along with accuracy, precision and recall.

According to the metrics computed during the training of the last epoch, we can assume that the model is slightly overfitted on the training data.

In [79]:

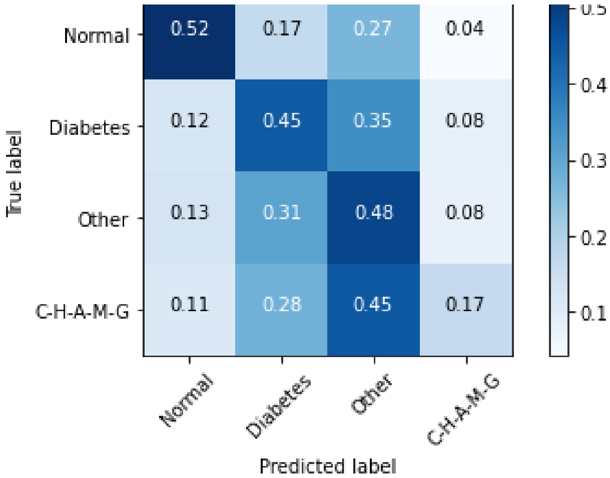
```
loss, accuracy, precision, recall = AlexNet.evaluate(x_test, y_test, verbose=1)
loss_v, accuracy_v, precision_v, recall_v = AlexNet.evaluate(x_validate, y_validate, ver
bose=0)
F1 = 2 * (precision * recall) / (precision + recall)
print("Validation: accuracy = %f  ;  Loss: %f" % (accuracy_v, loss_v))
print("Test: \n accuracy = %f \n precision = %f \n recall = %f \n F1 = %f" % (accuracy,
precision, recall, F1))
```

```
25/25 [==============================] - 2s 80ms/step - loss: 1.3922 - accuracy: 0.4355 -
precision_4: 0.4702 - recall_4: 0.3423
Validation: accuracy = 0.439940  ;  Loss: 1.421298
Test:
 accuracy = 0.435504
 precision = 0.470175
 recall = 0.342273
 F1 = 0.396157
```

Indeed it is.
However, the result on test set is not great: I have a relatively small dataset with unbalanced classes. Let's see the confusion matrix in order to try to understand what went wrong.

In [80]:

```
def plot_confusion_matrix(cm, classes,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)


    thresh = cm.max() / 2. #color threshold, I need it to
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], '.2f'),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

# Predict the values from the validation dataset
y_pred = AlexNet.predict(x_test)
# Convert predictions classes to one hot vectors
y_pred_classes = np.argmax(y_pred,axis = 1)
# Convert validation observations to one hot vectors
y_true = np.argmax(y_test,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(y_true, y_pred_classes, normalize='true')




# plot the confusion matrix
label_names = df['label'].unique()
plot_confusion_matrix(confusion_mtx, classes = label_names)
```



The results are not astonishing, but I have a relatively small dataset.

In [81]:

```
from sklearn.metrics import classification_report

print(classification_report(y_true,y_pred_classes, target_names=label_names))
```

```
              precision    recall  f1-score   support

      Normal       0.53      0.52      0.53       162
    Diabetes       0.47      0.45      0.46       264
       Other       0.39      0.48      0.43       248
   C-H-A-M-G       0.27      0.17      0.20       109

    accuracy                           0.44       783
   macro avg       0.42      0.41      0.41       783
weighted avg       0.43      0.44      0.43       783
```

Results

From the confusion matrix and the metrics, I can tell that the model is sort of capable of identifying Normal and Diabetic eyes, while the rest of categories make a lot of noise. It makes sense: I have no medical expertise and yet I create a new category called CHAMG, composed of subcategories only because of their size. I have no idea if those diseases share similar elements in the fundus of the eye and therefore I was expecting a very low accuracy in that. In fact, my model is completely uncapable of predicting CHAMG: the f1 score is around 20%.