Eye_Disease_Detection

Eye_Disease_Detection Using Transfer Learning and TensorFlow 2.0

In this project we're going to be using machine learning to help us identify Glaucoma.

To do this, we'll be using data from the Kaggle competition. It consists of a collection of 10,000+ labelled images of 120 different dog breeds.

This kind of problem is called multi-class image classification. It's multi-class because we're trying to classify mutliple different breeds of dog. If we were only trying to classify dogs versus cats, it would be called binary classification (one thing versus another).

Multi-class image classification is an important problem because it's the same kind of technology Tesla uses in their self-driving cars or Airbnb uses in atuomatically adding information to their listings.

Since the most important step in a deep learng problem is getting the data ready (turning it into numbers), that's what we're going to start with.

We're going to go through the following TensorFlow/Deep Learning workflow:

1. Get data ready (download from Kaggle, store, import).
2. Prepare the data (preprocessing, the 3 sets, X & y).
3. Choose and fit/train a model (TensorFlow Hub, `tf.keras.applications` , TensorBoard, EarlyStopping).
4. Evaluating a model (making predictions, comparing them with the ground truth labels).
5. Improve the model through experimentation (start with 1000 images, make sure it works, increase the number of images).
6. Save, sharing and reloading your model (once you're happy with the results).

For preprocessing our data, we're going to use TensorFlow 2.x. The whole premise here is to get our data into Tensors (arrays of numbers which can be run on GPUs) and then allow a machine learning model to find patterns between them.

For our machine learning model, we're going to be using a pretrained deep learning model from TensorFlow Hub.

The process of using a pretrained model and adapting it to your own problem is called **transfer learning**. We do this because rather than train our own model from scratch (could be timely and expensive), we leverage the patterns of another model which has been trained to classify images.

Getting our workspace ready

Before we get started, since we'll be using TensorFlow 2.x and TensorFlow Hub (TensorFlow Hub), let's import them.

NOTE: Don't run the cell below if you're already using TF 2.x.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import ipywidgets as widgets
```

In [2]:
```
import tensorflow as tf
import tensorflow_hub as hub
print("TF Version:",tf.__version__)
print("TF Hub Version:",hub.__version__)
```

```
TF Version: 2.10.0
TF Hub Version: 0.8.0
```

In [3]:
```
if tf.config.list_physical_devices('GPU'):
  print("GPU allocated")
else:
  print("Allocate GPU")
```

```
Allocate GPU
```

Accessing the data

Now the data files we're working with are available on our Google Drive, we can start to check it out.

In [4]:
```
# ! unzip "drive/MyDrive/Data_Science_Project/Glaucoma_Detection/archive.zip" -d "drive/MyDrive/Data_Science_Project/Glaucoma_Det
# print("Completed")
```

In [5]:
```
print("Training Data")
print("-------------")
print("CNV : ",len(os.listdir("Data/train/CNV")))
print("DME : ",len(os.listdir("Data/train/DME")))
print("DRUSEN : ",len(os.listdir("Data/train/DRUSEN")))
print("NORMAL : ",len(os.listdir("Data/train/NORMAL")))
```

```
Training Data
-------------
CNV :  37205
DME :  11348
DRUSEN :  8616
NORMAL :  26315
```

In [6]:

```
print("Test Data")
print("-----------")
print("CNV : ",len(os.listdir("Data/test/CNV")))
print("DME : ",len(os.listdir("Data/test/DME")))
print("DRUSEN : ",len(os.listdir("Data/test/DRUSEN")))
print("NORMAL : ",len(os.listdir("Data/test/NORMAL")))
```

```
Test Data
-----------
CNV :  239
DME :  241
DRUSEN :  240
NORMAL :  241
```

Creating Dataset

Creating Training Dataset

In [7]:

```
#NORMAL
filename = ["Data/train/NORMAL/"+ filename  for filename in os.listdir("Data/train/NORMAL")]
target = [ 0 for x in range(len(os.listdir("Data/train/NORMAL")))]
```

In [8]:

```
#CNV
filename = filename + ["Data/train/CNV/"+ filename  for filename in os.listdir("Data/train/CNV")]
target = target + [ 1 for x in range(len(os.listdir("Data/train/CNV")))]
```

In [9]:

```
#DME
filename = filename + ["Data/train/DME/"+ filename  for filename in os.listdir("Data/train/DME")]
target = target + [ 2 for x in range(len(os.listdir("Data/train/DME")))]
```

In [10]:

```
#DRUSEN
filename = filename + ["Data/train/DRUSEN/"+ filename  for filename in os.listdir("Data/train/DRUSEN")]
target = target + [ 3 for x in range(len(os.listdir("Data/train/DRUSEN")))]
```

In [11]:

```
train_df = pd.DataFrame(data = {
    "filename": filename,
    "target": target
})
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 83484 entries, 0 to 83483
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   filename  83484 non-null  object
 1   target    83484 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 1.3+ MB
```

In [12]:

```
train_df.head(5)
```

Out[12]:

	filename	target
0	Data/train/NORMAL/NORMAL-1001666-1.jpeg	0
1	Data/train/NORMAL/NORMAL-1001772-1.jpeg	0
2	Data/train/NORMAL/NORMAL-1001772-2.jpeg	0
3	Data/train/NORMAL/NORMAL-1001772-3.jpeg	0
4	Data/train/NORMAL/NORMAL-1001772-4.jpeg	0

In [13]:

```
train_df.tail(5)
```

Out[13]:

	filename	target
83479	Data/train/DRUSEN/DRUSEN-995513-9.jpeg	3
83480	Data/train/DRUSEN/DRUSEN-9961809-1.jpeg	3
83481	Data/train/DRUSEN/DRUSEN-997131-1.jpeg	3
83482	Data/train/DRUSEN/DRUSEN-997131-2.jpeg	3
83483	Data/train/DRUSEN/DRUSEN-997131-3.jpeg	3

In [14]:

```
train_df["target"].value_counts()
```

Out[14]:

```
1    37205
0    26315
2    11348
3     8616
Name: target, dtype: int64
```

Creating Test Dataset

In [15]:

```
#NORMAL
filename = ["Data/test/NORMAL/"+ filename  for filename in os.listdir("Data/test/NORMAL")]
target = [ 0 for x in range(len(os.listdir("Data/test/NORMAL")))]
```

In [16]:

```
#CNV
filename = filename + ["Data/test/CNV/"+ filename  for filename in os.listdir("Data/test/CNV")]
target = target + [ 1 for x in range(len(os.listdir("Data/test/CNV")))]
```

In [17]:

```
#DME
filename = filename + ["Data/test/DME/"+ filename  for filename in os.listdir("Data/test/DME")]
target = target + [ 2 for x in range(len(os.listdir("Data/test/DME")))]
```

In [18]:

```
#DRUSEN
filename = filename + ["Data/test/DRUSEN/"+ filename  for filename in os.listdir("Data/test/DRUSEN")]
target = target + [ 3 for x in range(len(os.listdir("Data/test/DRUSEN")))]
```

In [19]:

```
test_df = pd.DataFrame(data = {
    "filename": filename,
    "target": target
})
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 961 entries, 0 to 960
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   filename  961 non-null    object
 1   target    961 non-null    int64 
dtypes: int64(1), object(1)
memory usage: 15.1+ KB
```

In [20]:

```
test_df.head(5)
```

Out[20]:

	filename	target
0	Data/test/NORMAL/NORMAL-1017237-1.jpeg	0
1	Data/test/NORMAL/NORMAL-101880-1.jpeg	0
2	Data/test/NORMAL/NORMAL-1025847-1.jpeg	0
3	Data/test/NORMAL/NORMAL-1038998-1.jpeg	0
4	Data/test/NORMAL/NORMAL-1042462-1.jpeg	0

In [21]:

```
test_df.tail(5)
```

Out[21]:

	filename	target
956	Data/test/DRUSEN/DRUSEN-9689334-1.jpeg	3
957	Data/test/DRUSEN/DRUSEN-9734808-1.jpeg	3
958	Data/test/DRUSEN/DRUSEN-9734808-2.jpeg	3
959	Data/test/DRUSEN/DRUSEN-9800172-1.jpeg	3
960	Data/test/DRUSEN/DRUSEN-987193-1.jpeg	3

In [22]:

```
test_df.target.value_counts()
```

Out[22]:

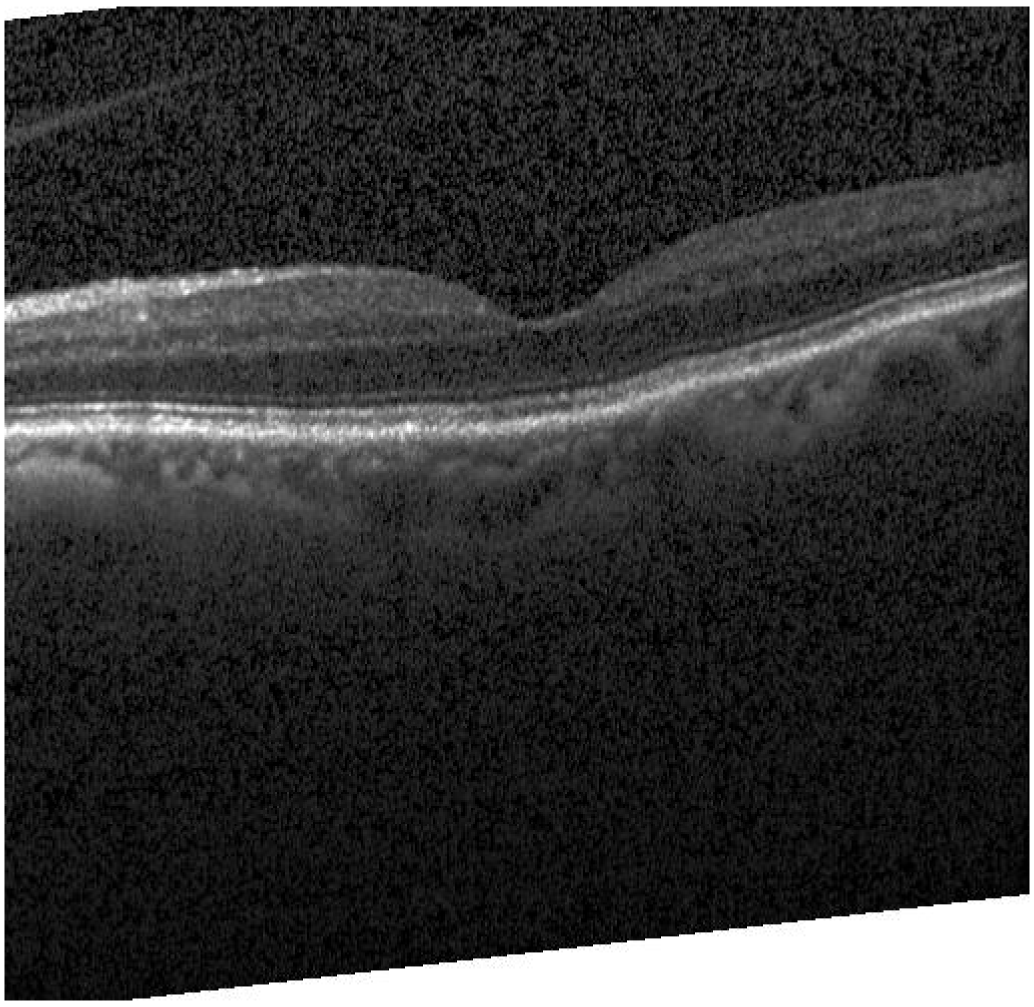
```
0    241
2    241
3    240
1    239
Name: target, dtype: int64
```

To Read images

In [23]:

```
from IPython.display import display, Image
Image(train_df["filename"][0])
```

Out[23]:



In [24]:

```
from matplotlib.pyplot import imread
image = imread(filename[24])
image.shape
```

Out[24]:

```
(496, 512)
```

Getting images and their labels

Since we've got the image ID's and their labels in a DataFrame (`train_df`), we'll use it to create:

- A list a filepaths to training images
- An array of all labels
- An array of all unique labels

We'll only create a list of filepaths to images rather than importing them all to begin with. This is because working with filepaths (strings) is much efficient than working with images.

In [25]:

```
shuffle_train_df = train_df.sample(frac=1)
shuffle_train_df[500:510]
```

Out[25]:

	filename	target
19873	Data/train/NORMAL/NORMAL-7732521-47.jpeg	0
11146	Data/train/NORMAL/NORMAL-4299338-6.jpeg	0
44567	Data/train/CNV/CNV-5823679-31.jpeg	1
6463	Data/train/NORMAL/NORMAL-297192-3.jpeg	0
12275	Data/train/NORMAL/NORMAL-4561631-1.jpeg	0
70389	Data/train/DME/DME-6737988-50.jpeg	2
39698	Data/train/CNV/CNV-4464785-34.jpeg	1
38762	Data/train/CNV/CNV-417468-52.jpeg	1
24008	Data/train/NORMAL/NORMAL-9054040-5.jpeg	0
16355	Data/train/NORMAL/NORMAL-605310-8.jpeg	0

In [26]:

```
target = shuffle_train_df.target

unique_target = shuffle_train_df.target.to_numpy()
unique_target = np.unique(target)
unique_target
```

Out[26]:

```
array([0, 1, 2, 3], dtype=int64)
```

In [27]:

```
bool_labels = [target == unique_target for target in shuffle_train_df.target]
bool_labels[15500]
```

Out[27]:

```
array([False, False,  True, False])
```

In [28]:

```
len(bool_labels)
```

Out[28]:

```
83484
```

In [29]:

```
#Example turning boolean array into integers
print(shuffle_train_df.target[0])
print(np.where(shuffle_train_df.target[0]==unique_target))
print(bool_labels[0].argmax())
print(bool_labels[0].astype(int))
```

```
0
(array([0], dtype=int64),)
0
[1 0 0 0]
```

In [30]:

```
X = shuffle_train_df.filename
Y = bool_labels
```

In [31]:

```
# widgets.IntSlider(
#     NUM_TRAIN=10000,
#     min=1000,
#     max=85000,
#     step=1000,
#     description='NUM_TRAIN:',
#     disabled=False,
#     continuous_update=False,
#     orientation='horizontal',
#     readout=True,
#     readout_format='d'
# )
```

In [32]:

```
NUM_TRAIN = len(train_df["target"])
NUM_TRAIN
```

Out[32]:

```
83484
```

In [33]:

```
#Splitting our data into train and test
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = train_test_split(X[:NUM_TRAIN], Y[:NUM_TRAIN], test_size=0.2, random_state=42,shuffle=False)
```

In [34]:

```
len(x_train),len( y_train),len(x_val),len(y_val)
```

Out[34]:

```
(66787, 66787, 16697, 16697)
```

Preprocessing images (turning images into Tensors)

Our labels are in numeric format but our images are still just file paths.

To preprocess our images into Tensors we're going to write a function which does a few things:

1. Takes an image filename as input.
2. Uses TensorFlow to read the file and save it to a variable, `image`.
3. Turn our `image` (a jpeg file) into Tensors.
4. Resize the `image` to be of shape (224, 224).
5. Return the modified `image`.

TensorFlow documentation on loading images.

In [35]:

```
from matplotlib.pyplot import imread
image = imread(train_df.filename[5])
image
```

Out[35]:

```
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255],
       ...,
       [ 17,  14,   6, ...,   3, 252, 252],
       [ 10,  11,   8, ...,   1, 252, 254],
       [ 10,  14,  15, ...,   0, 251, 255]], dtype=uint8)
```

In [36]:

```
image.shape
```

Out[36]:

```
(496, 512)
```

In [37]:

```
tf.constant(image)[:2]
```

Out[37]:

```
<tf.Tensor: shape=(2, 512), dtype=uint8, numpy=
array([[255, 255, 255, ..., 255, 255, 255],
       [255, 255, 255, ..., 255, 255, 255]], dtype=uint8)>
```

Creating Function

In [38]:

```
IMG_SIZE = 224

def process_image(image_path):
  image = tf.io.read_file(image_path)
  image = tf.image.decode_jpeg(image,channels= 3)
  image = tf.image.convert_image_dtype(image,tf.float32)
  image = tf.image.resize(image, size=[IMG_SIZE, IMG_SIZE ])
  return image
```

In [39]:

```
def get_image_label(image_path,label):
  image = process_image(image_path)
  return image, label
```

In [40]:

```
#Demo
(get_image_label(X[0], tf.constant(Y[0])))
```

Out[40]:

```
(<tf.Tensor: shape=(224, 224, 3), dtype=float32, numpy=
 array([[[0.9968188 , 0.9968188 , 0.9968188 ],
         [0.98693484, 0.98693484, 0.98693484],
         [0.9842637 , 0.9842637 , 0.9842637 ],
         ...,
         [0.06917714, 0.06917714, 0.06917714],
         [0.06863876, 0.06863876, 0.06863876],
         [0.0861598 , 0.0861598 , 0.0861598 ]],

        [[0.98444384, 0.98444384, 0.98444384],
         [0.9965487 , 0.9965487 , 0.9965487 ],
         [0.9952481 , 0.9952481 , 0.9952481 ],
         ...,
         [0.15137047, 0.15137047, 0.15137047],
         [0.04035784, 0.04035784, 0.04035784],
         [0.11231919, 0.11231919, 0.11231919]],

        [[0.97476   , 0.97476   , 0.97476   ],
         [0.9939476 , 0.9939476 , 0.9939476 ],
         [0.96473604, 0.96473604, 0.96473604],
         ...,
         [0.04322843, 0.04322843, 0.04322843],
         [0.14423677, 0.14423677, 0.14423677],
         [0.02772959, 0.02772959, 0.02772959]],

        ...,

        [[0.00170086, 0.00170086, 0.00170086],
         [0.03725491, 0.03725491, 0.03725491],
         [0.02530001, 0.02530001, 0.02530001],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.03359316, 0.03359316, 0.03359316],
         [0.04697878, 0.04697878, 0.04697878],
         [0.        , 0.        , 0.        ],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]],

        [[0.05254126, 0.05254126, 0.05254126],
         [0.04117674, 0.04117674, 0.04117674],
         [0.00671236, 0.00671236, 0.00671236],
         ...,
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ],
         [1.        , 1.        , 1.        ]]], dtype=float32)>,
 <tf.Tensor: shape=(4,), dtype=bool, numpy=array([ True, False, False, False])>)
```

In [41]:

```
BATCH_SIZE = 32

def create_batch_data(x, y = None, batch_size=BATCH_SIZE, Valid_Data = False, Test_Data = False):
  if Test_Data:
     print("Creating test data batches...")
     data = tf.data.Dataset.from_tensor_slices((tf.constant(x)))
     data_batch = data.map(process_image).batch(batch_size)
     return data_batch

  elif Valid_Data:
    print("Creating validation data batches...")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(x), # filepaths
                                               tf.constant(y))) # labels
    data_batch = data.map(get_image_label).batch(BATCH_SIZE)
    return data_batch

  else:
    print("Creating training data batches...")
    data = tf.data.Dataset.from_tensor_slices((tf.constant(x), # filepaths
                                              tf.constant(y))) # labels
    data = data.shuffle(buffer_size=len(x))
    data = data.map(get_image_label)
    data_batch = data.batch(BATCH_SIZE)
    return data_batch
```

In [42]:

```
train_data = create_batch_data(x_train, y_train)
valid_data = create_batch_data(x_val, y_val, Valid_Data = True)
```

```
Creating training data batches...
Creating validation data batches...
```

In [43]:

```python
# Check out the different attributes of our data batches
train_data.element_spec, valid_data.element_spec
```

Out[43]:

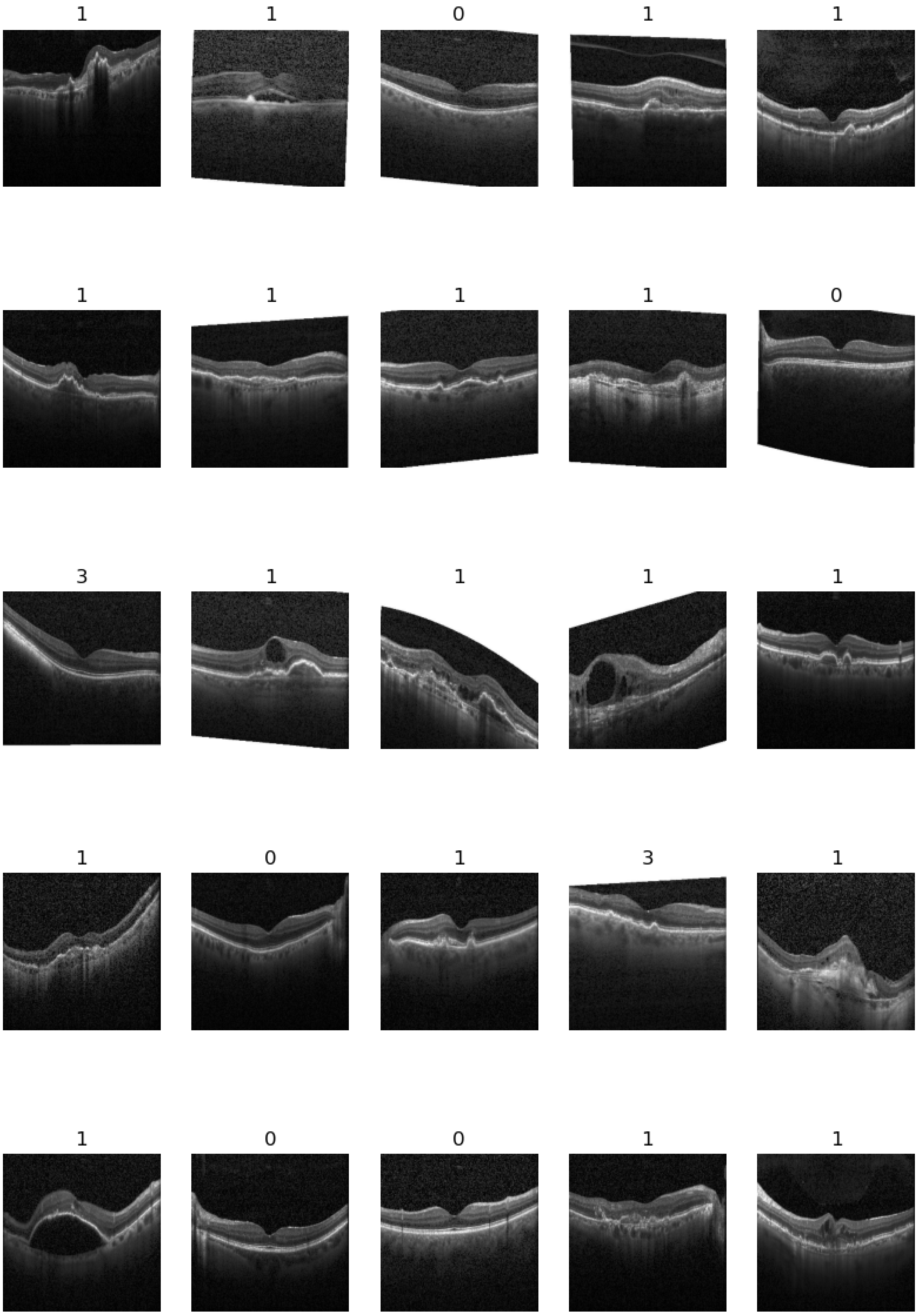
```
((TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 4), dtype=tf.bool, name=None)),
 (TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 4), dtype=tf.bool, name=None)))
```

In [44]:

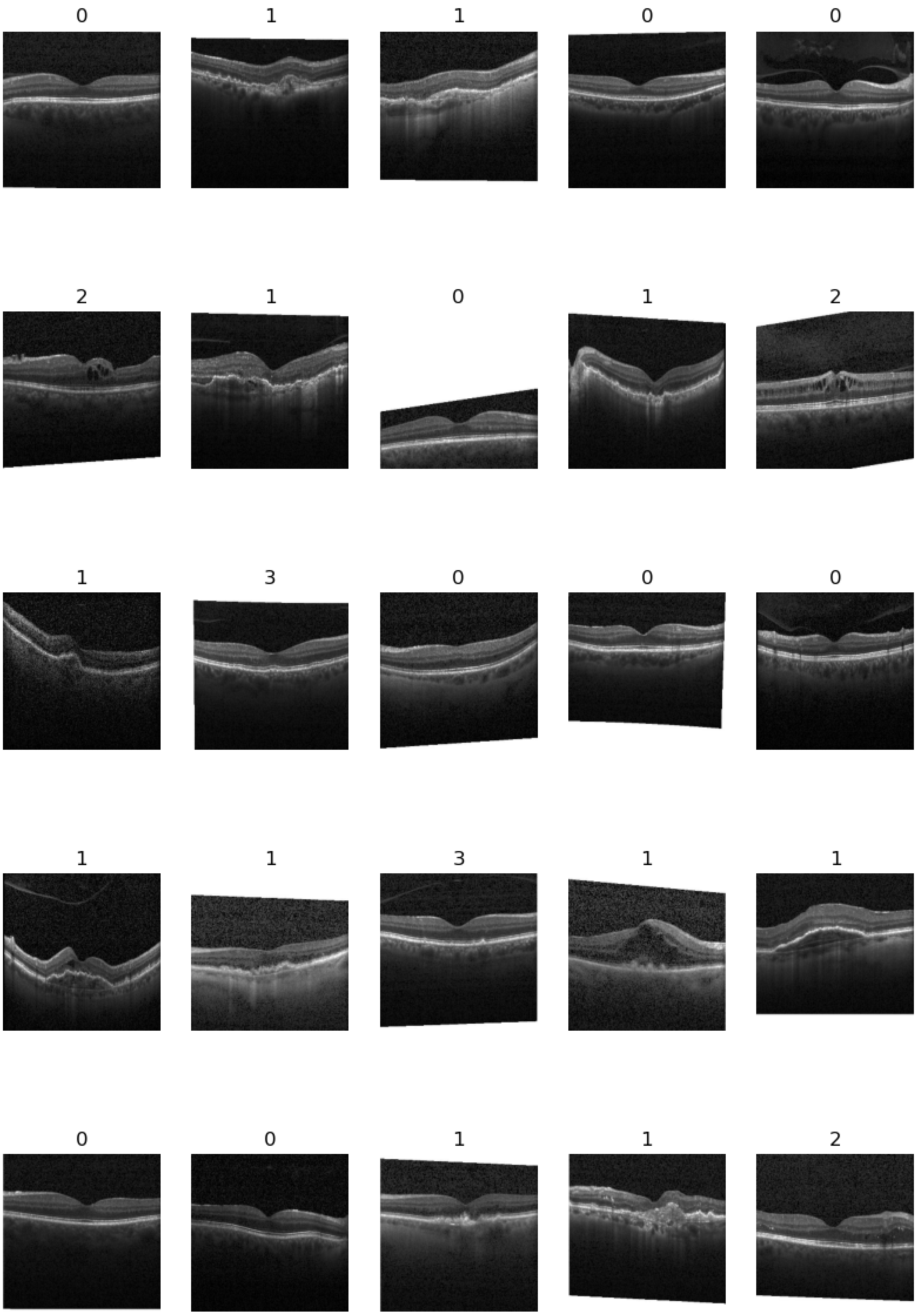
```python
#import matplotlib.pyplot as plt
def show_25_images(images, labels):
  plt.figure(figsize=(10, 15))
  # Loop through 25 (for displaying 25 images)
  for i in range(25):
    # Create subplots (5 rows, 5 columns)
    ax = plt.subplot(5, 5, i+1)
    # Display an image
    plt.imshow(images[i])
    # Add the image label as the title
    plt.title(unique_target[labels[i].argmax()])
    plt.axis("off")
```

In [45]:

```python
# Visualize training images from the training data batch
train_images, train_labels = next(train_data.as_numpy_iterator())
show_25_images(train_images, train_labels)
```



In [46]:
```
# Visualize validation images from the validation data batch
valid_images, valid_labels = next(valid_data.as_numpy_iterator())
show_25_images(valid_images, valid_labels)
```



In [47]:
```
IMG_SIZE
```

Out[47]:
```
224
```

In [48]:
```
# Setup input shape to the model
INPUT_SHAPE = [None, IMG_SIZE, IMG_SIZE, 3] # batch, height, width, colour channels

# Setup output shape of the model
OUTPUT_SHAPE = len(unique_target) # number of unique labels

# Setup model URL from TensorFlow Hub
MODEL_URL = "https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4"
```

In [49]:
```
INPUT_SHAPE
```

Out[49]:
```
[None, 224, 224, 3]
```

In [50]:
```
# Create a function which builds a Keras model
def create_model(input_shape=INPUT_SHAPE, output_shape=OUTPUT_SHAPE, model_url=MODEL_URL):
  print("Building model with:", MODEL_URL)
  # Setup the model layers
  model = tf.keras.Sequential([
    hub.KerasLayer(MODEL_URL), # Layer 1 (input layer)
    tf.keras.layers.Dense(units=OUTPUT_SHAPE,
                          activation="softmax") # Layer 2 (output layer)
  ])

  # Compile the model
  model.compile(
      loss=tf.keras.losses.CategoricalCrossentropy(),
      metrics=["accuracy"] # We'd like this to go up
  )

  # Build the model
  model.build(INPUT_SHAPE) # Let the model know what kind of inputs it'll be getting

  return model
```

In [51]:

```
# Create a model and check its details
model = create_model()
model.summary()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4
WARNING:tensorflow:Please fix your imports. Module tensorflow.python.training.tracking.data_structures has been moved to tensorf
low.python.trackable.data_structures. The old module will be deleted in version 2.11.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 keras_layer (KerasLayer)    (None, 1001)              5432713   
                                                                 
 dense (Dense)               (None, 4)                 4008      
                                                                 
=================================================================
Total params: 5,436,721
Trainable params: 4,008
Non-trainable params: 5,432,713
_________________________________________________________________
```

In [52]:

```
import datetime
def create_tensorboard_callback():
  logdir = os.path.join("logs/",
                        datetime.datetime.now( ).strftime("%Y%m%d - %H%M%S"))
  return tf.keras.callbacks.TensorBoard(logdir)
```

In [53]:

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor="val_accuracy",patience=3)
```

In [54]:

```
NUM_EPOCHS = 20 #@param {type:"slider", min:10, max:100,step:10}
```

In [55]:

```
if tf.config.list_physical_devices('GPU'):
  print("GPU available")
```

In [56]:

```
from keras.callbacks import TensorBoard
def train_model():
  model = create_model()
  tensorboard = create_tensorboard_callback()

  model.fit(x=train_data,
            epochs=NUM_EPOCHS,
            validation_data=valid_data,
            validation_freq=1,
            callbacks=[tensorboard,early_stopping])

  return model
```

In [57]:

```
# model = train_model()
```

In [97]:

```
%reload_ext tensorboard
%tensorboard --logdir logs/
```

```
Reusing TensorBoard on port 6006 (pid 3572), started 8:33:19 ago. (Use '!kill 3572' to kill it.)
```

Saving and Loading the Model

In [59]:

```
def save_model(model, suffix=None):
  """
  Saves a given model in a models directory and appends a suffix (str)
  for clarity and reuse.
  """
  # Create model directory with current time
  modeldir = os.path.join("models",
                          datetime.datetime.now().strftime("%Y%m-%H%M"))
  model_path = modeldir + "-" + suffix + ".h5" # save format of model
  print(f"Saving model to: {model_path}...")
  model.save(model_path)
  return model_path
```

In [60]:

```
def load_model(model_path):
  """
  Loads a saved model from a specified path.
  """
  print(f"Loading saved model from: {model_path}")
  model = tf.keras.models.load_model(model_path,
                                     custom_objects={"KerasLayer":hub.KerasLayer})
  return model
```

In [61]:

```
# Save our trained model
# save_model(model, suffix="images-Adam")
```

In [62]:

```
# Load our model trained on 1000 images
loaded_model = load_model('models/202308-0926-all-images-Adam.h5')
```

```
Loading saved model from: models/202308-0926-all-images-Adam.h5
```

In [63]:

```
predictions = loaded_model.predict(valid_data,verbose=1)
```

```
522/522 [==============================] - 263s 502ms/step
```

In [64]:

```
predictions
```

```
Out[64]: array([[9.80344415e-01, 1.21012445e-05, 9.32080089e-04, 1.87111720e-02],
                [1.81610528e-06, 9.90327716e-01, 5.08109770e-05, 9.61972587e-03],
                [4.83842008e-03, 9.68322873e-01, 1.44412729e-03, 2.53946334e-02],
                ...,
                [9.63350654e-01, 9.09883156e-03, 1.91987790e-02, 8.35176371e-03],
                [2.93361808e-07, 9.99124408e-01, 6.80856116e-04, 1.94455977e-04],
                [4.64177987e-07, 9.82805550e-01, 2.53882299e-05, 1.71685871e-02]],
               dtype=float32)
```

```
In [65]: print(f"Predicted result :{np.argmax(predictions[0])}")
```

```
Predicted result :0
```

Making prediction

```
In [66]: predictions[1],np.sum(predictions[1])
```

```
Out[66]: (array([1.8161053e-06, 9.9032772e-01, 5.0810977e-05, 9.6197259e-03],
                dtype=float32),
          1.0)
```

```
In [67]: print(f"Predicted result:{np.argmax(predictions[1])}")
```

```
Predicted result:1
```

```
In [68]: def get_pred_labels(predictions,pred_data = True):
             if pred_data :
                 if (np.argmax(predictions) == 0):
                     return "No Diseases"
                 elif (np.argmax(predictions) == 1):
                     return 'CNV'
                 elif (np.argmax(predictions) == 2):
                     return "DME"
                 else:
                     return 'DRUSEN'

             else:
                 if (predictions) == 0:
                     return "No Diseases"
                 elif (predictions) == 1:
                     return 'CNV'
                 elif (predictions) == 2:
                     return "DME"
                 else:
                     return 'DRUSEN'
```

```
In [69]: get_pred = get_pred_labels(predictions[1])
         get_pred
```

```
Out[69]: 'CNV'
```

Unbatch Data

```
In [70]: def unbatchify(data):
             images = []
             labels = []

             for image, label in data.unbatch().as_numpy_iterator():
                 images.append(image)
                 labels.append(get_pred_labels(np.argmax(label),False))

             return images,labels
```

```
In [71]: val_images, val_labels = unbatchify(valid_data)
```

```
In [72]: val_images[0], val_labels[0]
```

Out[72]:

```
(array([[[[0.17310925, 0.17310925, 0.17310925],
         [0.12945177, 0.12945177, 0.12945177],
         [0.06216488, 0.06216488, 0.06216488],
         ...,
         [0.99081653, 0.99081653, 0.99081653],
         [0.98239326, 0.98239326, 0.98239326],
         [0.99021643, 0.99021643, 0.99021643]],

        [[0.1414766 , 0.1414766 , 0.1414766 ],
         [0.11884753, 0.11884753, 0.11884753],
         [0.13977592, 0.13977592, 0.13977592],
         ...,
         [0.9957387 , 0.9957387 , 0.9957387 ],
         [0.99837935, 0.99837935, 0.99837935],
         [0.98881495, 0.98881495, 0.98881495]],

        [[0.18469389, 0.18469389, 0.18469389],
         [0.03035212, 0.03035212, 0.03035212],
         [0.16450581, 0.16450581, 0.16450581],
         ...,
         [0.09269815, 0.09269815, 0.09269815],
         [0.0544613 , 0.0544613 , 0.0544613 ],
         [0.00576213, 0.00576213, 0.00576213]],

        ...,

        [[0.01454576, 0.01454576, 0.01454576],
         [0.03467353, 0.03467353, 0.03467353],
         [0.04193664, 0.04193664, 0.04193664],
         ...,
         [0.03873551, 0.03873551, 0.03873551],
         [0.03097257, 0.03097257, 0.03097257],
         [0.01404497, 0.01404497, 0.01404497]],

        [[0.15967563, 0.15967563, 0.15967563],
         [0.14422964, 0.14422964, 0.14422964],
         [0.16011584, 0.16011584, 0.16011584],
         ...,
         [0.02959197, 0.02959197, 0.02959197],
         [0.01200484, 0.01200484, 0.01200484],
         [0.00990339, 0.00990339, 0.00990339]],

        [[0.988296  , 0.988296  , 0.988296  ],
         [0.98833585, 0.98833585, 0.98833585],
         [0.98937637, 0.98937637, 0.98937637],
         ...,
         [0.0262903 , 0.0262903 , 0.0262903 ],
         [0.00642276, 0.00642276, 0.00642276],
         [0.00252087, 0.00252087, 0.00252087]]], dtype=float32),
 'No Diseases')
```

Making functions to help visual your models results are really helpful in understanding how your model is doing.

Since we're working with a multi-class problem , it would also be good to see what other guesses our model is making. Let's build a function to demonstrate. The function will:

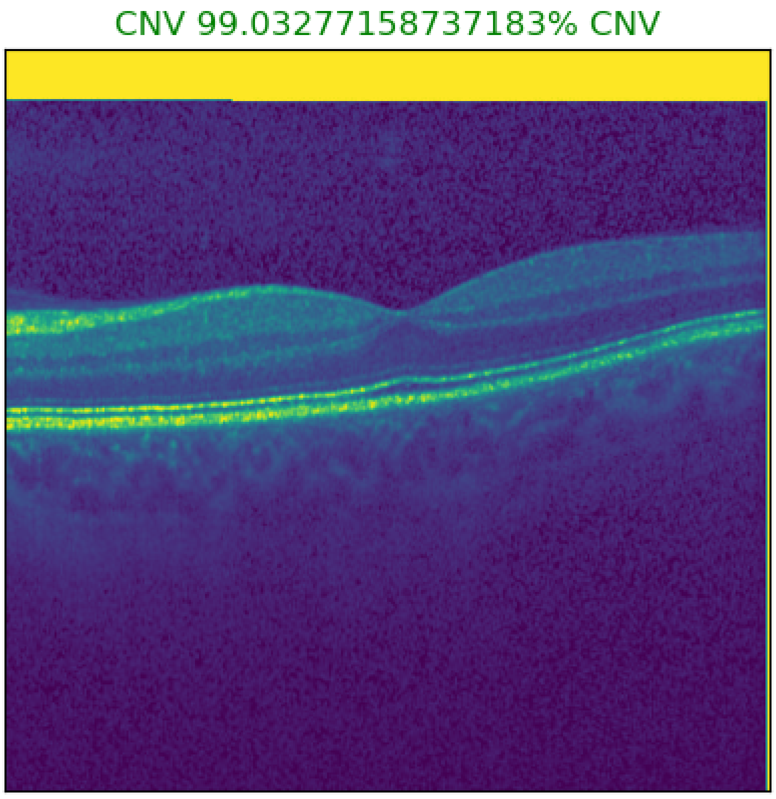
- Prediction probabilities indexes
- Prediction probabilities values
- Prediction labels

In [73]:

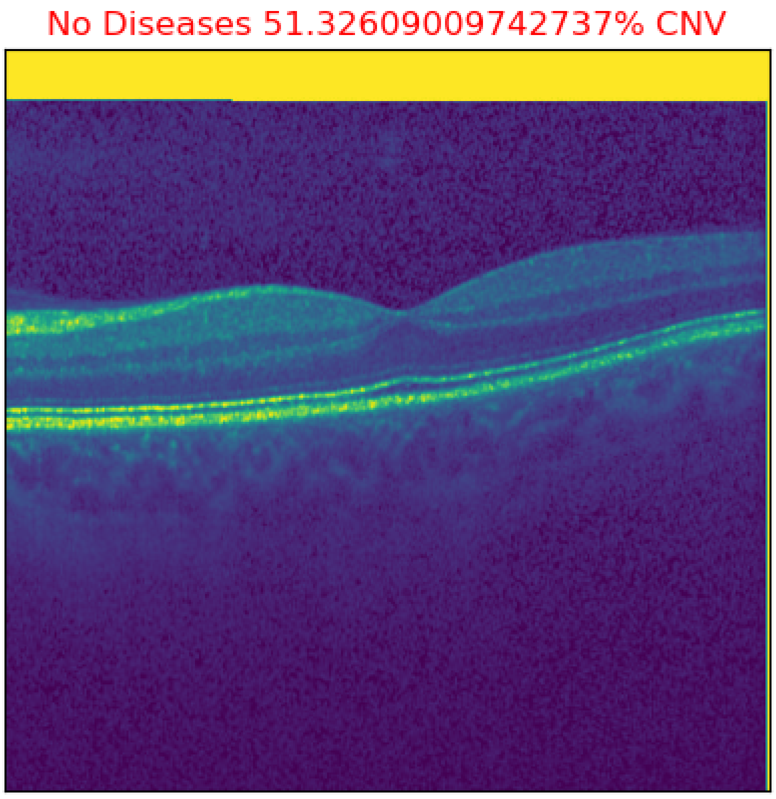
```python
def plot_pred(predictions, images, labels, n=1):
    pred, images, labels = predictions[n], images[n], labels[n]
    pred = get_pred_labels(pred)

    plt.imshow(image)
    plt.xticks([])
    plt.yticks([])

    if pred == labels:
        color = "green"
    else:
        color = "red"
    plt.title(f"{pred} {np.max(predictions[n]) *100}% {labels}",color = color)
```

In [74]:

```python
plot_pred(predictions,
          labels=val_labels,
          images=val_images)
```



In [75]:
```
plot_pred(predictions,
          labels=val_labels,
          images=val_images,
          n = 1984)
```



Training on full Dataset

In [76]:
```
len(X), len(Y)
```

Out[76]:
```
(83484, 83484)
```

In [77]:
```
full_data = create_batch_data(X, Y)
```

```
Creating training data batches...
```

In [78]:
```
full_model = create_model()
```

```
Building model with: https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4
```

In [79]:
```
# Create full model callbacks

# TensorBoard callback
full_model_tensorboard = create_tensorboard_callback()

# Early stopping callback
full_model_early_stopping = tf.keras.callbacks.EarlyStopping(monitor="accuracy",
                                                             patience=3)
```

In [96]:
```
%tensorboard --logdir logs
```

In [ ]:

```
full_model.fit(x=full_data,
               epochs=NUM_EPOCHS,
               callbacks=[full_model_early_stopping,full_model_tensorboard]
               )
```

In [ ]:

```
# Save model to file
save_model(full_model, suffix="all-images-Adam")
```

Making Prediction on Images

In [81]:

```
# Load our model trained on 1000 images
loaded_model = load_model('models/202308-0926-all-images-Adam.h5')
```

```
Loading saved model from: models/202308-0926-all-images-Adam.h5
```

In [82]:

```
test_path = "Test_Images"
test_data = ["Test_Images/"+filename for filename in os.listdir(test_path)]
```

In [83]:

```
test_data
```

Out[83]:

```
['Test_Images/CNV-103044-3.jpeg',
 'Test_Images/CNV-53018-1.jpeg',
 'Test_Images/DME-15208-1.jpeg',
 'Test_Images/DRUSEN-228939-1.jpeg',
 'Test_Images/DRUSEN-2785977-1.jpeg',
 'Test_Images/DRUSEN-95633-1.jpeg',
 'Test_Images/DRUSEN.jpeg',
 'Test_Images/NORMAL-12494-1.jpeg']
```

In [84]:

```
predict_on_image = create_batch_data(test_data,Test_Data=True)
predict_on_image
```

```
Creating test data batches...
```

Out[84]:

```
<BatchDataset element_spec=TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32, name=None)>
```

In [85]:

```
result_on_predict = loaded_model.predict(predict_on_image)
```

```
1/1 [==============================] - 3s 3s/step
```

In [86]:

```
result_on_predict
```

Out[86]:

```
array([[9.6321382e-05, 9.4418913e-01, 2.3347270e-02, 3.2367088e-02],
       [1.0415556e-05, 9.7640085e-01, 1.2555565e-02, 1.1033261e-02],
       [1.8989193e-01, 4.8011364e-03, 7.9961771e-01, 5.6890687e-03],
       [5.7779218e-04, 3.5061803e-02, 1.5632395e-02, 9.4872802e-01],
       [2.7510736e-04, 2.8731588e-01, 7.7225739e-04, 7.1163672e-01],
       [1.2587233e-02, 9.2202432e-02, 1.5852426e-03, 8.9362508e-01],
       [2.1627273e-05, 9.7924125e-01, 3.2396081e-05, 2.0704634e-02],
       [6.7182243e-01, 1.7066358e-02, 6.8528697e-02, 2.4258256e-01]],
      dtype=float32)
```

In [87]:

```
result_label = [ get_pred_labels(label) for label in result_on_predict]
```

In [88]:

```
result_label
```

Out[88]:

```
['CNV', 'CNV', 'DME', 'DRUSEN', 'DRUSEN', 'DRUSEN', 'CNV', 'No Diseases']
```

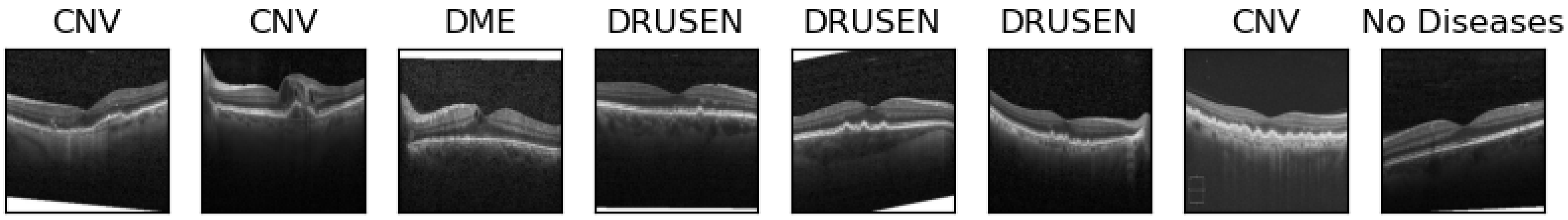
In [89]:

```
result_images = []
for image in predict_on_image.unbatch().as_numpy_iterator():
    result_images.append(image)

result_images
```

```
Out[89]: [array([[[0.43738493, 0.43738493, 0.43738493],
                  [0.21456584, 0.21456584, 0.21456584],
                  [0.06952782, 0.06952782, 0.06952782],
                  ...,
                  [0.86883044, 0.86883044, 0.86883044],
                  [0.99583817, 0.99583817, 0.99583817],
                  [0.97312945, 0.97312945, 0.97312945]],

                 [[0.36709684, 0.36709684, 0.36709684],
                  [0.1284014 , 0.1284014 , 0.1284014 ],
                  [0.01515606, 0.01515606, 0.01515606],
                  ...,
                  [0.03843442, 0.03843442, 0.03843442],
                  [0.14439599, 0.14439599, 0.14439599],
                  [0.00365169, 0.00365169, 0.00365169]],

                 [[0.41971785, 0.41971785, 0.41971785],
                  [0.05440174, 0.05440174, 0.05440174],
                  [0.18753503, 0.18753503, 0.18753503],
                  ...,
                  [0.14147034, 0.14147034, 0.14147034],
                  [0.15354097, 0.15354097, 0.15354097],
                  [0.06795572, 0.06795572, 0.06795572]],

                 ...,

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [0.01070483, 0.01070483, 0.01070483],
                  [0.07568958, 0.07568958, 0.07568958],
                  [0.00549217, 0.00549217, 0.00549217]],

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [0.01086379, 0.01086379, 0.01086379],
                  [0.05681236, 0.05681236, 0.05681236],
                  [0.02255988, 0.02255988, 0.02255988]],

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [0.04388751, 0.04388751, 0.04388751],
                  [0.03668524, 0.03668524, 0.03668524],
                  [0.00396102, 0.00396102, 0.00396102]]], dtype=float32),
          array([[[1.07673071e-01, 1.07673071e-01, 1.07673071e-01],
                  [8.13425407e-02, 8.13425407e-02, 8.13425407e-02],
                  [1.48499414e-01, 1.48499414e-01, 1.48499414e-01],
                  ...,
                  [3.29437517e-02, 3.29437517e-02, 3.29437517e-02],
                  [1.45755047e-02, 1.45755047e-02, 1.45755047e-02],
                  [0.00000000e+00, 0.00000000e+00, 0.00000000e+00]],

                 [[2.38035217e-01, 2.38035217e-01, 2.38035217e-01],
                  [9.94297788e-02, 9.94297788e-02, 9.94297788e-02],
                  [1.27981201e-01, 1.27981201e-01, 1.27981201e-01],
                  ...,
                  [1.92176849e-02, 1.92176849e-02, 1.92176849e-02],
                  [1.95884164e-02, 1.95884164e-02, 1.95884164e-02],
                  [1.57964323e-02, 1.57964323e-02, 1.57964323e-02]],

                 [[3.16456586e-01, 3.16456586e-01, 3.16456586e-01],
                  [1.65196091e-01, 1.65196091e-01, 1.65196091e-01],
                  [1.62344947e-01, 1.62344947e-01, 1.62344947e-01],
                  ...,
                  [2.25041807e-03, 2.25041807e-03, 2.25041807e-03],
                  [1.56670623e-02, 1.56670623e-02, 1.56670623e-02],
                  [1.34656215e-02, 1.34656215e-02, 1.34656215e-02]],

                 ...,

                 [[2.20222399e-04, 2.20222399e-04, 2.20222399e-04],
                  [2.45996416e-02, 2.45996416e-02, 2.45996416e-02],
                  [1.14946058e-02, 1.14946058e-02, 1.14946058e-02],
                  ...,
                  [1.14746550e-02, 1.14746550e-02, 1.14746550e-02],
                  [1.27449799e-02, 1.27449799e-02, 1.27449799e-02],
                  [5.40210493e-03, 5.40210493e-03, 5.40210493e-03]],

                 [[1.08941346e-02, 1.08941346e-02, 1.08941346e-02],
                  [1.44558521e-02, 1.44558521e-02, 1.44558521e-02],
                  [1.72267407e-02, 1.72267407e-02, 1.72267407e-02],
                  ...,
                  [9.66402050e-03, 9.66402050e-03, 9.66402050e-03],
                  [2.80139409e-03, 2.80139409e-03, 2.80139409e-03],
                  [3.37139540e-03, 3.37139540e-03, 3.37139540e-03]],

                 [[2.91417073e-02, 2.91417073e-02, 2.91417073e-02],
                  [2.19984390e-02, 2.19984390e-02, 2.19984390e-02],
                  [1.83477532e-02, 1.83477532e-02, 1.83477532e-02],
                  ...,
                  [3.30097391e-03, 3.30097391e-03, 3.30097391e-03],
                  [4.08203341e-03, 4.08203341e-03, 4.08203341e-03],
                  [1.35859475e-02, 1.35859475e-02, 1.35859475e-02]]], dtype=float32),
          array([[[1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  ...,
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00]],

                 [[1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  ...,
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00]],

                 [[1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  ...,
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00],
                  [1.0000000e+00, 1.0000000e+00, 1.0000000e+00]],

                 ...,

                 [[0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [6.1524468e-03, 6.1524468e-03, 6.1524468e-03],
                  ...,
                  [2.2148490e-02, 2.2148490e-02, 2.2148490e-02],
                  [6.3684866e-02, 6.3684866e-02, 6.3684866e-02],
                  [1.5906494e-02, 1.5906494e-02, 1.5906494e-02]],

                 [[0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [8.4033300e-04, 8.4033300e-04, 8.4033300e-04],
                  ...,
                  [5.0004117e-05, 5.0004117e-05, 5.0004117e-05],
                  [4.3046489e-02, 4.3046489e-02, 4.3046489e-02],
                  [3.9475188e-02, 3.9475188e-02, 3.9475188e-02]],

                 [[0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [0.0000000e+00, 0.0000000e+00, 0.0000000e+00],
                  [7.5127594e-03, 7.5127594e-03, 7.5127594e-03],
                  ...,
                  [3.2552280e-02, 3.2552280e-02, 3.2552280e-02],
                  [7.3947810e-02, 7.3947810e-02, 7.3947810e-02],
                  [3.7695363e-02, 3.7695363e-02, 3.7695363e-02]]], dtype=float32),
          array([[[0.06926771, 0.06926771, 0.06926771],
                  [0.10574232, 0.10574232, 0.10574232],
                  [0.01246499, 0.01246499, 0.01246499],
                  ...,
                  [0.04688953, 0.04688953, 0.04688953],
                  [0.12756054, 0.12756054, 0.12756054],
                  [0.07079794, 0.07079794, 0.07079794]],

                 [[0.02377953, 0.02377953, 0.02377953],
                  [0.        , 0.        , 0.        ],
                  [0.07367948, 0.07367948, 0.07367948],
                  ...,
                  [0.06707813, 0.06707813, 0.06707813],
                  [0.09448744, 0.09448744, 0.09448744],
                  [0.10697365, 0.10697365, 0.10697365]],

                 [[0.0070028 , 0.0070028 , 0.0070028 ],
                  [0.13798524, 0.13798524, 0.13798524],
                  [0.01021409, 0.01021409, 0.01021409],
                  ...,
                  [0.12228928, 0.12228928, 0.12228928],
                  [0.09232648, 0.09232648, 0.09232648],
                  [0.04019525, 0.04019525, 0.04019525]],

                 ...,

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]],

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]],

                 [[1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]]], dtype=float32),
          array([[[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  ...,
                  [1.10486455e-01, 1.10486455e-01, 1.10486455e-01],
                  [8.83358419e-02, 8.83358419e-02, 8.83358419e-02],
                  [9.83380079e-02, 9.83380079e-02, 9.83380079e-02]],

                 [[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  ...,
                  [9.23391506e-02, 9.23391506e-02, 9.23391506e-02],
                  [1.30832046e-01, 1.30832046e-01, 1.30832046e-01],
                  [1.41718566e-01, 1.41718566e-01, 1.41718566e-01]],

                 [[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  ...,
                  [1.90638065e-01, 1.90638065e-01, 1.90638065e-01],
                  [1.29932225e-01, 1.29932225e-01, 1.29932225e-01],
                  [9.36756060e-02, 9.36756060e-02, 9.36756060e-02]],

                 ...,

                 [[9.84783947e-01, 9.84783947e-01, 9.84783947e-01],
                  [6.09541535e-02, 6.09541535e-02, 6.09541535e-02],
                  [6.74968213e-02, 6.74968213e-02, 6.74968213e-02],
                  ...,
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00]],

                 [[9.96448636e-01, 9.96448636e-01, 9.96448636e-01],
                  [8.22734162e-02, 8.22734162e-02, 8.22734162e-02],
                  [5.67529723e-02, 5.67529723e-02, 5.67529723e-02],
                  ...,
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00]],

                 [[9.98299241e-01, 9.98299241e-01, 9.98299241e-01],
                  [1.70077226e-04, 1.70077226e-04, 1.70077226e-04],
                  [3.79547700e-02, 3.79547700e-02, 3.79547700e-02],
                  ...,
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
                  [1.00000000e+00, 1.00000000e+00, 1.00000000e+00]]], dtype=float32),
          array([[[4.6538614e-02, 4.6538614e-02, 4.6538614e-02],
                  [6.2975205e-02, 6.2975205e-02, 6.2975205e-02],
                  [8.1822708e-02, 8.1822708e-02, 8.1822708e-02],
                  ...,
                  [1.0033889e-02, 1.0033889e-02, 1.0033889e-02],
                  [1.7837074e-02, 1.7837074e-02, 1.7837074e-02],
                  [8.7935003e-03, 8.7935003e-03, 8.7935003e-03]],

                 [[3.8985614e-02, 3.8985614e-02, 3.8985614e-02],
                  [2.3289314e-02, 2.3289314e-02, 2.3289314e-02],
                  [3.9335743e-02, 3.9335743e-02, 3.9335743e-02],
                  ...,
                  [1.0001572e-04, 1.0001572e-04, 1.0001572e-04],
                  [5.2971631e-02, 5.2971631e-02, 5.2971631e-02],
                  [1.8717822e-02, 1.8717822e-02, 1.8717822e-02]],

                 [[9.7408958e-02, 9.7408958e-02, 9.7408958e-02],
                  [1.9259706e-01, 1.9259706e-01, 1.9259706e-01],
                  [1.2102842e-01, 1.2102842e-01, 1.2102842e-01],
                  ...,
                  [1.0623341e-01, 1.0623341e-01, 1.0623341e-01],
                  [3.9967015e-02, 3.9967015e-02, 3.9967015e-02],
                  [4.8621446e-03, 4.8621446e-03, 4.8621446e-03]],

                 ...,

                 [[1.0834109e-02, 1.0834109e-02, 1.0834109e-02],
                  [7.5329952e-03, 7.5329952e-03, 7.5329952e-03],
                  [3.4963787e-02, 3.4963787e-02, 3.4963787e-02],
                  ...,
                  [2.8611667e-02, 2.8611667e-02, 2.8611667e-02],
                  [1.0604023e-02, 1.0604023e-02, 1.0604023e-02],
                  [4.3119621e-03, 4.3119621e-03, 4.3119621e-03]],

                 [[9.5239133e-03, 9.5239133e-03, 9.5239133e-03],
                  [3.1502612e-02, 3.1502612e-02, 3.1502612e-02],
                  [1.1504742e-03, 1.1504742e-03, 1.1504742e-03],
                  ...,
                  [1.8907854e-02, 1.8907854e-02, 1.8907854e-02],
                  [2.3507928e-03, 2.3507928e-03, 2.3507928e-03],
                  [4.3567598e-02, 4.3567598e-02, 4.3567598e-02]],

                 [[2.6709810e-03, 2.6709810e-03, 2.6709810e-03],
                  [1.1464763e-02, 1.1464763e-02, 1.1464763e-02],
                  [3.9003488e-02, 3.9003488e-02, 3.9003488e-02],
                  ...,
                  [4.5407861e-02, 4.5407861e-02, 4.5407861e-02],
                  [2.4659449e-02, 2.4659449e-02, 2.4659449e-02],
                  [1.1614488e-02, 1.1614488e-02, 1.1614488e-02]]], dtype=float32),
          array([[[0.14434113, 0.14434113, 0.14434113],
                  [0.12305482, 0.12305482, 0.12305482],
                  [0.13680302, 0.13680302, 0.13680302],
                  ...,
                  [0.13274091, 0.13274091, 0.13274091],
                  [0.14410426, 0.14410426, 0.14410426],
                  [0.13193624, 0.13193624, 0.13193624]],

                 [[0.12504706, 0.12504706, 0.12504706],
                  [0.13159359, 0.13159359, 0.13159359],
                  [0.13033992, 0.13033992, 0.13033992],
                  ...,
                  [0.14582577, 0.14582577, 0.14582577],
                  [0.15402777, 0.15402777, 0.15402777],
                  [0.15626237, 0.15626237, 0.15626237]],

                 [[0.12238955, 0.12238955, 0.12238955],
                  [0.13140456, 0.13140456, 0.13140456],
                  [0.13988014, 0.13988014, 0.13988014],
                  ...,
                  [0.13484672, 0.13484672, 0.13484672],
                  [0.1446642 , 0.1446642 , 0.1446642 ],
                  [0.16097832, 0.16097832, 0.16097832]],

                 ...,

                 [[0.20620131, 0.20620131, 0.20620131],
                  [0.18251134, 0.18251134, 0.18251134],
                  [0.13875039, 0.13875039, 0.13875039],
                  ...,
                  [0.16668236, 0.16668236, 0.16668236],
                  [0.16895512, 0.16895512, 0.16895512],
                  [0.1522144 , 0.1522144 , 0.1522144 ]],

                 [[0.1685361 , 0.1685361 , 0.1685361 ],
                  [0.14796674, 0.14796674, 0.14796674],
                  [0.13973407, 0.13973407, 0.13973407],
                  ...,
                  [0.14954   , 0.14954   , 0.14954   ],
                  [0.1656798 , 0.1656798 , 0.1656798 ],
                  [0.17329411, 0.17329411, 0.17329411]],

                 [[0.1470126 , 0.1470126 , 0.1470126 ],
                  [0.14727575, 0.14727575, 0.14727575],
                  [0.13621324, 0.13621324, 0.13621324],
                  ...,
                  [0.17680998, 0.17680998, 0.17680998],
                  [0.16517133, 0.16517133, 0.16517133],
                  [0.18847956, 0.18847956, 0.18847956]]], dtype=float32),
          array([[[0.98892564, 0.98892564, 0.98892564],
                  [0.99520814, 0.99520814, 0.99520814],
                  [0.99421775, 0.99421775, 0.99421775],
                  ...,
                  [0.10321165, 0.10321165, 0.10321165],
                  [0.0852237 , 0.0852237 , 0.0852237 ],
                  [0.4006325 , 0.4006325 , 0.4006325 ]],

                 [[0.992387  , 0.992387  , 0.992387  ],
                  [0.9809024 , 0.9809024 , 0.9809024 ],
                  [0.99225694, 0.99225694, 0.99225694],
                  ...,
                  [0.09603797, 0.09603797, 0.09603797],
                  [0.10773353, 0.10773353, 0.10773353],
                  [0.43255374, 0.43255374, 0.43255374]],

                 [[0.9648961 , 0.9648961 , 0.9648961 ],
                  [0.96805733, 0.96805733, 0.96805733],
                  [0.96594656, 0.96594656, 0.96594656],
                  ...,
                  [0.08199152, 0.08199152, 0.08199152],
                  [0.03271339, 0.03271339, 0.03271339],
                  [0.38885868, 0.38885868, 0.38885868]],

                 ...,
                 [[0.03384351, 0.03384351, 0.03384351],
                  [0.01275502, 0.01275502, 0.01275502],
                  [0.02563051, 0.02563051, 0.02563051],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]],

                 [[0.04685876, 0.04685876, 0.04685876],
                  [0.03328321, 0.03328321, 0.03328321],
                  [0.01373554, 0.01373554, 0.01373554],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]],

                 [[0.03501428, 0.03501428, 0.03501428],
                  [0.03518409, 0.03518409, 0.03518409],
                  [0.00387158, 0.00387158, 0.00387158],
                  ...,
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ],
                  [1.        , 1.        , 1.        ]]], dtype=float32)]
```

In [90]:

```
plt.figure(figsize=(10,10))
for i,image in enumerate(result_images):
    plt.subplot(1,len(result_on_predict),i+1)
    plt.xticks([])
    plt.yticks([])
    plt.title(result_label[i])
    plt.imshow(image)
```



In [91]:

```
test_data
```

Out[91]:

```
['Test_Images/CNV-103044-3.jpeg',
 'Test_Images/CNV-53018-1.jpeg',
 'Test_Images/DME-15208-1.jpeg',
 'Test_Images/DRUSEN-228939-1.jpeg',
 'Test_Images/DRUSEN-2785977-1.jpeg',
 'Test_Images/DRUSEN-95633-1.jpeg',
 'Test_Images/DRUSEN.jpeg',
 'Test_Images/NORMAL-12494-1.jpeg']
```