Breast cancer classification with Keras and Deep Learning

Import Packages

In [1]:
```
# set the matplotlib backend so figures can be saved in the background
import matplotlib
matplotlib.use("Agg")

# import the necessary packages
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import LearningRateScheduler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.layers import SeparableConv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Activation
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adagrad
from tensorflow.keras import backend as K
from tensorflow.keras.utils import to_categorical
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import random
import shutil
import os
```

Our `Config` class

In [2]:
```
class Config:
    # initialize the path to the *original* input directory of images
    ORIG_INPUT_DATASET = "breast-cancer-classification/data"

    # initialize the base path to the *new* directory that will contain
    # our images after computing the training and testing split
    BASE_PATH = "breast-cancer-classification/data"

    # derive the training, validation, and testing directories
    TRAIN_PATH = os.path.sep.join([BASE_PATH, "training"])
    VAL_PATH = os.path.sep.join([BASE_PATH, "validation"])
    TEST_PATH = os.path.sep.join([BASE_PATH, "testing"])

    # define the amount of data that will be used training
    TRAIN_SPLIT = 0.8

    # the amount of validation data will be a percentage of the
    # *training* data
    VAL_SPLIT = 0.1

# initialize our config class
config = Config()
```

Building the breast cancer image dataset

In [3]:
```
# grab the paths to all input images in the original input directory
# and shuffle them
imagePaths = list(paths.list_images(config.ORIG_INPUT_DATASET))
random.seed(42)
random.shuffle(imagePaths)

# compute the training and testing split
i = int(len(imagePaths) * config.TRAIN_SPLIT)
trainPaths = imagePaths[:i]
testPaths = imagePaths[i:]

# we'll be using part of the training data for validation
i = int(len(trainPaths) * config.VAL_SPLIT)
valPaths = trainPaths[:i]
trainPaths = trainPaths[i:]

# define the datasets that we'll be building
datasets = [
        ("training", trainPaths, config.TRAIN_PATH),
        ("validation", valPaths, config.VAL_PATH),
        ("testing", testPaths, config.TEST_PATH)
]
```

In [4]:
```
# loop over the datasets
for (dType, imagePaths, baseOutput) in datasets:

    # show which data split we are creating
        print("[INFO] building '{}' split".format(dType))

    # if the output base output directory does not exist, create it
        if not os.path.exists(baseOutput):
                print("[INFO] 'creating {}' directory".format(baseOutput))
                os.makedirs(baseOutput)

    # loop over the input image paths
        for inputPath in imagePaths:
                # extract the filename of the input image and extract the
                # class label ("0" for "negative" and "1" for "positive")
                filename = inputPath.split(os.path.sep)[-1]
                label = filename[-5:-4]

        # build the path to the label directory
                labelPath = os.path.sep.join([baseOutput, label])

        # if the label output directory does not exist, create it
                if not os.path.exists(labelPath):
                        print("[INFO] 'creating {}' directory".format(labelPath))
                        os.makedirs(labelPath)

        # construct the path to the destination image and then copy
                # the image itself
                p = os.path.sep.join([labelPath, filename])
                shutil.copy2(inputPath, p)
```

```
[INFO] building 'training' split
---------------------------------------------------------------------------
SameFileError                             Traceback (most recent call last)
<ipython-input-4-e794fee51c6c> in <module>
     28                 # the image itself
     29                 p = os.path.sep.join([labelPath, filename])
---> 30                 shutil.copy2(inputPath, p)

c:\users\leobr\appdata\local\programs\python\python39\lib\shutil.py in copy2(src, dst, follow_symlinks)
    433     if os.path.isdir(dst):
    434         dst = os.path.join(dst, os.path.basename(src))
--> 435     copyfile(src, dst, follow_symlinks=follow_symlinks)
    436     copystat(src, dst, follow_symlinks=follow_symlinks)
    437     return dst

c:\users\leobr\appdata\local\programs\python\python39\lib\shutil.py in copyfile(src, dst, follow_symlinks)
    242
    243     if _samefile(src, dst):
--> 244         raise SameFileError("{!r} and {!r} are the same file".format(src, dst))
    245
    246     file_size = 0

SameFileError: 'breast-cancer-classification/data\\training\\0\\9036_idx5_x551_y2201_class0.png' and 'breast-cancer-classification/data\\training\\0\\9036_idx5_x551_y2201_class0.png' are the same file
```

CancerNet: Our breast cancer prediction CNN

In [5]:
```
class CancerNet:
        @staticmethod
        def build(width, height, depth, classes):
                # initialize the model along with the input shape to be
                # "channels last" and the channels dimension itself
                model = Sequential()
                inputShape = (height, width, depth)
                chanDim = -1

                # if we are using "channels first", update the input shape
                # and channels dimension
                if K.image_data_format() == "channels_first":
                        inputShape = (depth, height, width)
                        chanDim = 1

                # CONV => RELU => POOL
                model.add(SeparableConv2D(32, (3, 3), padding="same",
                        input_shape=inputShape))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(MaxPooling2D(pool_size=(2, 2)))
                model.add(Dropout(0.25))

                # (CONV => RELU => POOL) * 2
                model.add(SeparableConv2D(64, (3, 3), padding="same"))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(SeparableConv2D(64, (3, 3), padding="same"))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(MaxPooling2D(pool_size=(2, 2)))
                model.add(Dropout(0.25))

                # (CONV => RELU => POOL) * 3
                model.add(SeparableConv2D(128, (3, 3), padding="same"))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(SeparableConv2D(128, (3, 3), padding="same"))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(SeparableConv2D(128, (3, 3), padding="same"))
                model.add(Activation("relu"))
                model.add(BatchNormalization(axis=chanDim))
                model.add(MaxPooling2D(pool_size=(2, 2)))
                model.add(Dropout(0.25))

                # first (and only) set of FC => RELU layers
                model.add(Flatten())
                model.add(Dense(256))
                model.add(Activation("relu"))
                model.add(BatchNormalization())
                model.add(Dropout(0.5))

                # softmax classifier
                model.add(Dense(classes))
                model.add(Activation("softmax"))

                # return the constructed network architecture
                return model
```

Our training script

In [6]:
```
# # construct the argument parser and parse the arguments
# ap = argparse.ArgumentParser()
# ap.add_argument("-p", "--plot", type=str, default="plot.png",
#       help="path to output loss/accuracy plot")
# args = vars(ap.parse_args())

# since we are using Jupyter Notebooks we can replace our argument
# parsing code with *hard coded* arguments and values
args = {
        "plot": "plot.png",
}
```

In [7]:
```
# initialize our number of epochs, initial learning rate, and batch
# size
NUM_EPOCHS = 40
INIT_LR = 1e-2
BS = 32

# determine the total number of image paths in training, validation,
# and testing directories
trainPaths = list(paths.list_images(config.TRAIN_PATH))
totalTrain = len(trainPaths)
totalVal = len(list(paths.list_images(config.VAL_PATH)))
totalTest = len(list(paths.list_images(config.TEST_PATH)))

# calculate the total number of training images in each class and
# initialize a dictionary to store the class weights
trainLabels = [int(p.split(os.path.sep)[-2]) for p in trainPaths]
trainLabels = to_categorical(trainLabels)
classTotals = trainLabels.sum(axis=0)
classWeight = dict()

# loop over all classes and calculate the class weight
for i in range(0, len(classTotals)):
        classWeight[i] = classTotals.max() / classTotals[i]
```

In [8]:
```
# initialize the training data augmentation object
trainAug = ImageDataGenerator(
        rescale=1 / 255.0,
        rotation_range=20,
        zoom_range=0.05,
        width_shift_range=0.1,
        height_shift_range=0.1,
        shear_range=0.05,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode="nearest")

# initialize the validation (and testing) data augmentation object
valAug = ImageDataGenerator(rescale=1 / 255.0)
```

In [9]:
```
# initialize the training generator
trainGen = trainAug.flow_from_directory(
        config.TRAIN_PATH,
        class_mode="categorical",
        target_size=(48, 48),
        color_mode="rgb",
        shuffle=True,
        batch_size=BS)

# initialize the validation generator
```

```
valGen = valAug.flow_from_directory(
        config.VAL_PATH,
        class_mode="categorical",
        target_size=(48, 48),
        color_mode="rgb",
        shuffle=False,
        batch_size=BS)

# initialize the testing generator
testGen = valAug.flow_from_directory(
        config.TEST_PATH,
        class_mode="categorical",
        target_size=(48, 48),
        color_mode="rgb",
        shuffle=False,
        batch_size=BS)
```

```
Found 255556 images belonging to 2 classes.
Found 42596 images belonging to 2 classes.
Found 99743 images belonging to 2 classes.
```

In [10]:

```
### initialize our CancerNet model and compile it
model = CancerNet.build(width=48, height=48, depth=3,
        classes=2)
opt = Adagrad(lr=INIT_LR, decay=INIT_LR / NUM_EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# fit the model
H = model.fit(
        x=trainGen,
        steps_per_epoch=totalTrain // BS,
        validation_data=valGen,
        validation_steps=totalVal // BS,
        class_weight=classWeight,
        epochs=NUM_EPOCHS)
```

```
c:\users\leobr\appdata\local\programs\python\python39\lib\site-packages\tensorflow\python\keras\optimizer_v2\optimizer_v2.py:374: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  warnings.warn(
WARNING:tensorflow:From c:\users\leobr\appdata\local\programs\python\python39\lib\site-packages\tensorflow\python\ops\array_ops.py:5043: calling gather (from tensorflow.python.ops.array_ops) with validate_indices is deprecated and will be removed in a future version.
Instructions for updating:
The `validate_indices` argument has no effect. Indices are always validated on CPU and never validated on GPU.
Epoch 1/40
7986/7986 [==============================] - 636s 79ms/step - loss: 0.6054 - accuracy: 0.8206 - val_loss: 0.4922 - val_accuracy: 0.8091
Epoch 2/40
7986/7986 [==============================] - 616s 77ms/step - loss: 0.5556 - accuracy: 0.8333 - val_loss: 0.4122 - val_accuracy: 0.8436
Epoch 3/40
7986/7986 [==============================] - 640s 80ms/step - loss: 0.5451 - accuracy: 0.8354 - val_loss: 0.4163 - val_accuracy: 0.8429
Epoch 4/40
7986/7986 [==============================] - 632s 79ms/step - loss: 0.5415 - accuracy: 0.8376 - val_loss: 0.4265 - val_accuracy: 0.8384
Epoch 5/40
7986/7986 [==============================] - 629s 79ms/step - loss: 0.5367 - accuracy: 0.8393 - val_loss: 0.4181 - val_accuracy: 0.8421
Epoch 6/40
7986/7986 [==============================] - 616s 77ms/step - loss: 0.5361 - accuracy: 0.8388 - val_loss: 0.4124 - val_accuracy: 0.8453
Epoch 7/40
7986/7986 [==============================] - 609s 76ms/step - loss: 0.5349 - accuracy: 0.8392 - val_loss: 0.4161 - val_accuracy: 0.8454
Epoch 8/40
7986/7986 [==============================] - 608s 76ms/step - loss: 0.5333 - accuracy: 0.8397 - val_loss: 0.4219 - val_accuracy: 0.8428
Epoch 9/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5317 - accuracy: 0.8403 - val_loss: 0.4141 - val_accuracy: 0.8452
Epoch 10/40
7986/7986 [==============================] - 609s 76ms/step - loss: 0.5323 - accuracy: 0.8411 - val_loss: 0.4205 - val_accuracy: 0.8437
Epoch 11/40
7986/7986 [==============================] - 610s 76ms/step - loss: 0.5309 - accuracy: 0.8412 - val_loss: 0.4253 - val_accuracy: 0.8422
Epoch 12/40
7986/7986 [==============================] - 611s 77ms/step - loss: 0.5308 - accuracy: 0.8409 - val_loss: 0.4160 - val_accuracy: 0.8442
Epoch 13/40
7986/7986 [==============================] - 621s 78ms/step - loss: 0.5311 - accuracy: 0.8406 - val_loss: 0.4148 - val_accuracy: 0.8440
Epoch 14/40
7986/7986 [==============================] - 609s 76ms/step - loss: 0.5310 - accuracy: 0.8396 - val_loss: 0.4198 - val_accuracy: 0.8430
Epoch 15/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5293 - accuracy: 0.8411 - val_loss: 0.4210 - val_accuracy: 0.8432
Epoch 16/40
7986/7986 [==============================] - 610s 76ms/step - loss: 0.5310 - accuracy: 0.8400 - val_loss: 0.4170 - val_accuracy: 0.8442
Epoch 17/40
7986/7986 [==============================] - 611s 76ms/step - loss: 0.5294 - accuracy: 0.8410 - val_loss: 0.4213 - val_accuracy: 0.8424
Epoch 18/40
7986/7986 [==============================] - 610s 76ms/step - loss: 0.5299 - accuracy: 0.8413 - val_loss: 0.4280 - val_accuracy: 0.8406
Epoch 19/40
7986/7986 [==============================] - 614s 77ms/step - loss: 0.5296 - accuracy: 0.8408 - val_loss: 0.4155 - val_accuracy: 0.8444
Epoch 20/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5292 - accuracy: 0.8411 - val_loss: 0.4182 - val_accuracy: 0.8437
Epoch 21/40
7986/7986 [==============================] - 614s 77ms/step - loss: 0.5277 - accuracy: 0.8419 - val_loss: 0.4195 - val_accuracy: 0.8436
Epoch 22/40
7986/7986 [==============================] - 611s 76ms/step - loss: 0.5287 - accuracy: 0.8413 - val_loss: 0.4196 - val_accuracy: 0.8434
Epoch 23/40
7986/7986 [==============================] - 614s 77ms/step - loss: 0.5285 - accuracy: 0.8413 - val_loss: 0.4212 - val_accuracy: 0.8432
Epoch 24/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5271 - accuracy: 0.8416 - val_loss: 0.4186 - val_accuracy: 0.8433
Epoch 25/40
7986/7986 [==============================] - 615s 77ms/step - loss: 0.5288 - accuracy: 0.8415 - val_loss: 0.4138 - val_accuracy: 0.8449
Epoch 26/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5262 - accuracy: 0.8426 - val_loss: 0.4198 - val_accuracy: 0.8430
Epoch 27/40
7986/7986 [==============================] - 616s 77ms/step - loss: 0.5268 - accuracy: 0.8414 - val_loss: 0.4204 - val_accuracy: 0.8433
Epoch 28/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5284 - accuracy: 0.8412 - val_loss: 0.4202 - val_accuracy: 0.8436
Epoch 29/40
7986/7986 [==============================] - 610s 76ms/step - loss: 0.5275 - accuracy: 0.8413 - val_loss: 0.4172 - val_accuracy: 0.8441
Epoch 30/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5278 - accuracy: 0.8402 - val_loss: 0.4125 - val_accuracy: 0.8455
Epoch 31/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5266 - accuracy: 0.8417 - val_loss: 0.4188 - val_accuracy: 0.8439
Epoch 32/40
7986/7986 [==============================] - 614s 77ms/step - loss: 0.5281 - accuracy: 0.8413 - val_loss: 0.4181 - val_accuracy: 0.8434
Epoch 33/40
7986/7986 [==============================] - 621s 78ms/step - loss: 0.5270 - accuracy: 0.8414 - val_loss: 0.4197 - val_accuracy: 0.8438
Epoch 34/40
7986/7986 [==============================] - 636s 80ms/step - loss: 0.5264 - accuracy: 0.8418 - val_loss: 0.4149 - val_accuracy: 0.8442
Epoch 35/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5276 - accuracy: 0.8418 - val_loss: 0.4203 - val_accuracy: 0.8433
Epoch 36/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5272 - accuracy: 0.8419 - val_loss: 0.4170 - val_accuracy: 0.8441
Epoch 37/40
7986/7986 [==============================] - 613s 77ms/step - loss: 0.5273 - accuracy: 0.8414 - val_loss: 0.4152 - val_accuracy: 0.8439
Epoch 38/40
7986/7986 [==============================] - 618s 77ms/step - loss: 0.5259 - accuracy: 0.8428 - val_loss: 0.4180 - val_accuracy: 0.8435
Epoch 39/40
7986/7986 [==============================] - 612s 77ms/step - loss: 0.5278 - accuracy: 0.8414 - val_loss: 0.4139 - val_accuracy: 0.8443
Epoch 40/40
7986/7986 [==============================] - 610s 76ms/step - loss: 0.5270 - accuracy: 0.8417 - val_loss: 0.4160 - val_accuracy: 0.8439
```

In [17]:

```
import sklearn
import sklearn.metrics
```

In [18]:

```
# reset the testing generator and then use our trained model to
# make predictions on the data
print("[INFO] evaluating network...")
testGen.reset()
predIdxs = model.predict(x=testGen, steps=(totalTest // BS) + 1)

# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(sklearn.metrics.classification_report(testGen.classes, predIdxs,
        target_names=testGen.class_indices.keys()))
```

```
[INFO] evaluating network...
              precision    recall  f1-score   support

           0       0.93      0.85      0.89     71295
           1       0.69      0.84      0.76     28448

    accuracy                           0.85     99743
   macro avg       0.81      0.84      0.82     99743
weighted avg       0.86      0.85      0.85     99743
```

In [20]:

```
# compute the confusion matrix and and use it to derive the raw
# accuracy, sensitivity, and specificity
cm = sklearn.metrics.confusion_matrix(testGen.classes, predIdxs)
total = sum(sum(cm))
acc = (cm[0, 0] + cm[1, 1]) / total
sensitivity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
specificity = cm[1, 1] / (cm[1, 0] + cm[1, 1])

# show the confusion matrix, accuracy, sensitivity, and specificity
print(cm)
print("acc: {:.4f}".format(acc))
print("sensitivity: {:.4f}".format(sensitivity))
print("specificity: {:.4f}".format(specificity))
```

```
[[60707 10588]
 [ 4632 23816]]
acc: 0.8474
sensitivity: 0.8515
specificity: 0.8372
```

In [27]:

```
# plot the training loss and accuracy
N = NUM_EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
plt.show()
```

```
<ipython-input-27-d02a2d202664>:14: UserWarning: Matplotlib is currently using agg, which is a non-GUI backend, so cannot show the figure.
  plt.show()
```

Code License Agreement