# Breast cancer classification with Keras and Deep Learning

## Import Packages

In [1]:
```python
# 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 sklearn
import sklearn.metrics
import os
```

## Our `Config` class

In [2]:
```python
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]:
```python
# 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]:
```python
# 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)
```

## CancerNet: Our breast cancer prediction CNN

In [5]:
```python
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 [7]:
```python
# # 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 [8]:
```python
# initialize our number of epochs, initial learning rate, and batch
# size
NUM_EPOCHS = 1
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 [9]:
```python
# 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 [10]:
```python
# 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 [11]:
```python
### 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.
7986/7986 [==============================] - 686s 86ms/step - loss: 0.7043 - accuracy: 0.7911 - val_loss: 0.6911 - val_accuracy: 0.7217
```

In [12]:
```python
# 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)

# save model
model.save("Breast_cancer")

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

```
[INFO] evaluating network...
INFO:tensorflow:Assets written to: Breast_cancer\assets
              precision    recall  f1-score   support

           0       0.96      0.64      0.77     71295
           1       0.51      0.93      0.66     28448

    accuracy                           0.72     99743
   macro avg       0.73      0.78      0.71     99743
weighted avg       0.83      0.72      0.74     99743
```

In [13]:
```python
# 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))
```

```
[[45715 25580]
 [ 2132 26316]]
acc: 0.7222
sensitivity: 0.6412
specificity: 0.9251
```

In [14]:
```python
# 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"])
```

## Code License Agreement