1. data manipulation

- video to img

- audio

- further manipulation

2. model

- what type of NN

- how many layers

- learning algo

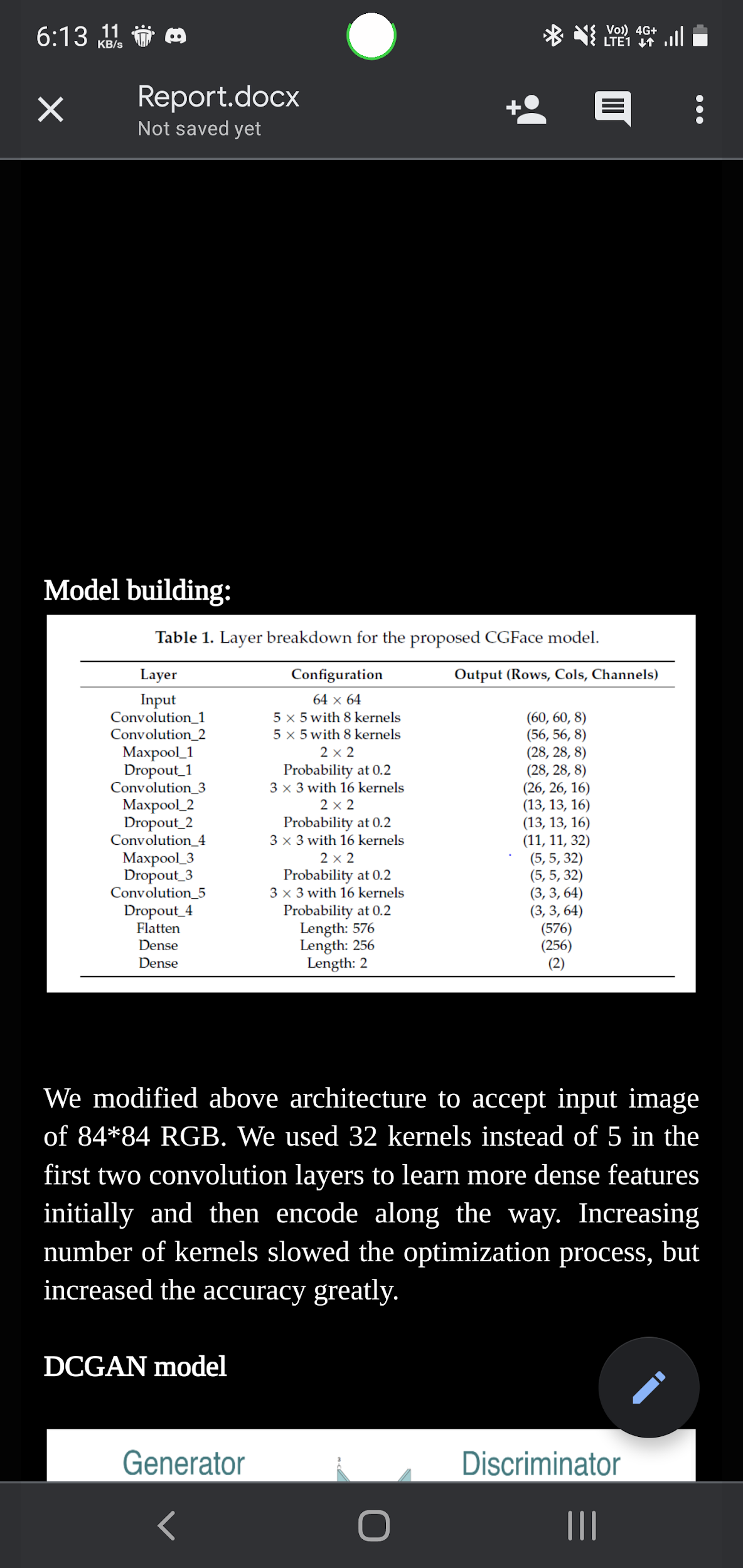
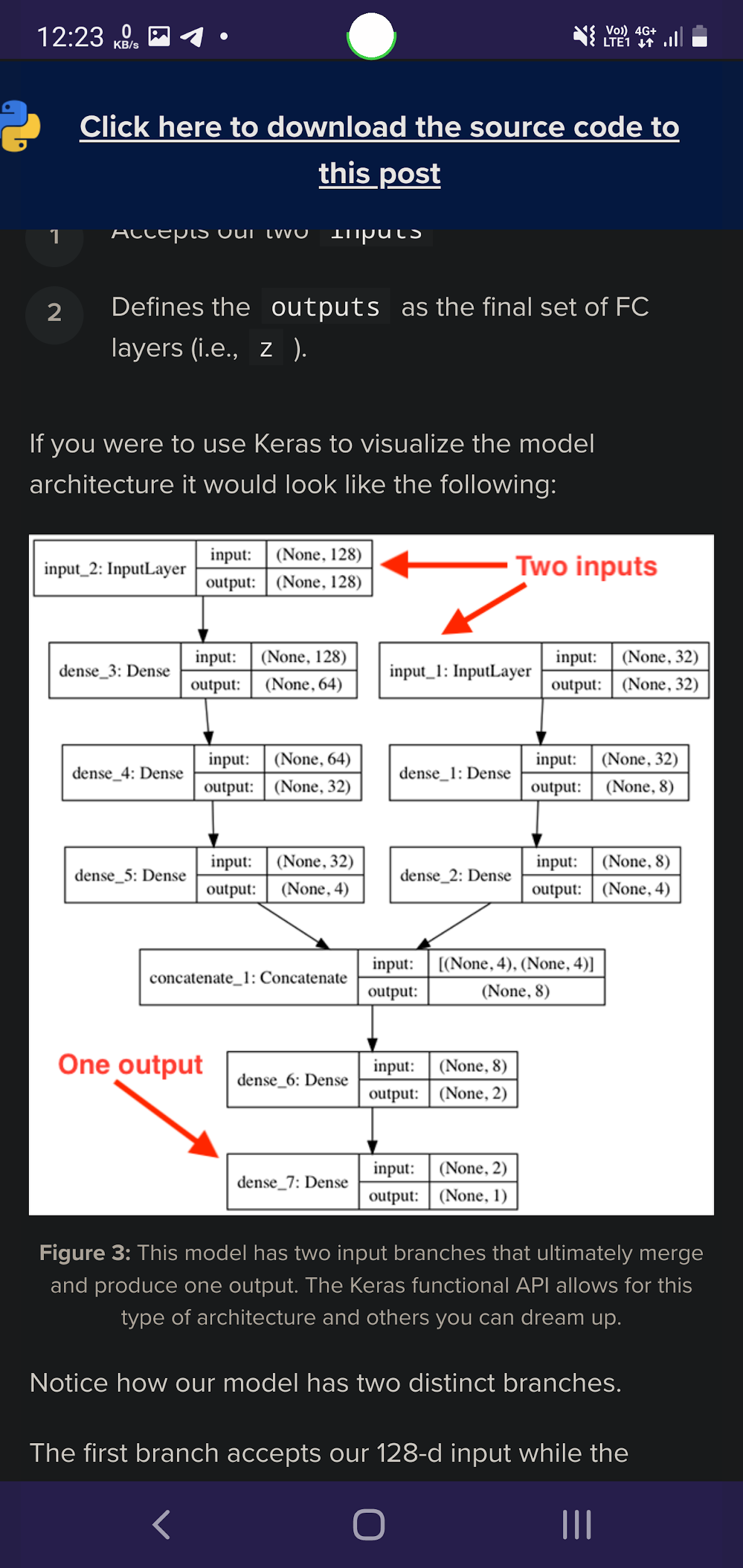
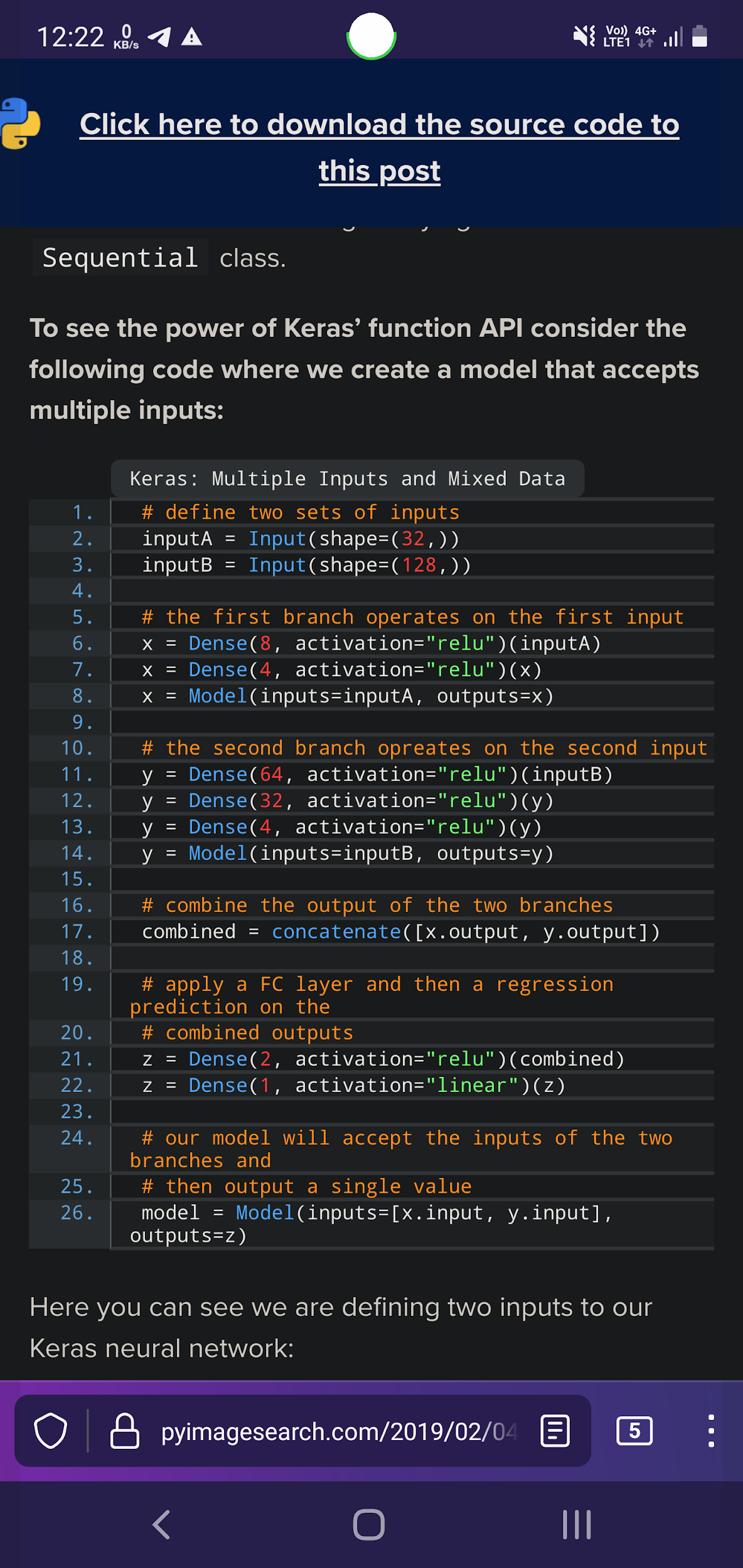
<https://github.com/dessa-oss/DeepFake-Detection>

<https://github.com/swayanshu/Deep-Fake-Detection>

<https://github.com/HongguLiu/Deepfake-Detection>

<https://github.com/topics/deepfake-detection>

<https://www.analyticsvidhya.com/blog/2019/07/learn-build-first-speech-to-text-model-python/#h2_9>



VIDEO EXTRACTION SOFTWARE??

Codes:

MODEL

from keras.models import Model as KerasModel

from keras.layers import Input, Dense, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Dropout, Reshape, Concatenate, LeakyReLU

from keras.optimizers import Adam

IMGWIDTH = 64

class Classifier:

def \_\_init\_\_():

self.model = 0

def predict(self, x):

return self.model.predict(x)

def fit(self, x, y):

return self.model.train\_on\_batch(x, y)

def get\_accuracy(self, x, y):

return self.model.test\_on\_batch(x, y)

def load(self, path):

self.model.load\_weights(path)

class CGFACE(Classifier):

def \_\_init\_\_(self, learning\_rate = 0.0001):

self.model = self.init\_model()

optimizer = Adam(lr = learning\_rate)

self.model.compile(optimizer = optimizer, loss = 'mean\_squared\_error', metrics = ['accuracy'])

def init\_model(self):

x = Input(shape = (IMGWIDTH, IMGWIDTH, 1))

x1 = Conv2D(8, (5, 5), padding='same', activation = 'relu')(x)

x1 = Conv2D(8, (5, 5), padding='same', activation = 'relu')(x)

x1 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x1)

x1 = Dropout(0.2)(x1)

x2 = Conv2D(16,(3, 3), padding='same', activation = 'relu')(x1)

x2 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x2)

x2 = Dropout(0.2)(x2)

x3 = Conv2D(16, (3, 3), padding='same', activation = 'relu')(x2)

#x3 = BatchNormalization()(x3)

x3 = MaxPooling2D(pool\_size=(2, 2), padding='same')(x3)

x3 = Dropout(0.2)(x3)

x4 = Conv2D(16, (3, 3), padding='same', activation = 'relu')(x3)

x4 = Dropout(0.2)(x4)

y = Flatten()(x4)

y = Dense(256)(y)

y = Dense(1, activation = 'sigmoid')(y)

return KerasModel(inputs = x, outputs = y)

IMAGE ADD

from keras.preprocessing.image import ImageDataGenerator, array\_to\_img, img\_to\_array, load\_img

import numpy as np

path = r"D:\2nd\_semester\bigdatascience\project\P1\real\_faces\\"

X = []

Y = []

for i in tqdm(os.listdir(path)):

img = cv2.imread(path + i , cv2.IMREAD\_GRAYSCALE)

norm\_image = cv2.normalize(img, None, alpha=0, beta=1, norm\_type=cv2.NORM\_MINMAX, dtype=cv2.CV\_32F)

X.append(norm\_image)

Y.append(1)

GENERATOR & DISCRIMINATOR

def create\_generator():

gen\_input = Input(shape=(LATENT\_DIM, ))

x = Dense(128 \* 16 \* 16)(gen\_input)

x = LeakyReLU()(x)

x = Reshape((16, 16, 128))(x)

x = Conv2D(256, 5, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2DTranspose(256, 4, strides=2, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2D(512, 5, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2D(512, 5, padding='same')(x)

x = LeakyReLU()(x)

x = Conv2D(CHANNELS, 7, activation='tanh', padding='same')(x)

generator = Model(gen\_input, x)

return generator

def create\_discriminator():

disc\_input = Input(shape=(HEIGHT, WIDTH, CHANNELS))

x = Conv2D(256, 3)(disc\_input)

x = LeakyReLU()(x)

x = Conv2D(256, 4, strides=2)(x)

x = LeakyReLU()(x)

x = Conv2D(256, 4, strides=2)(x)

x = LeakyReLU()(x)

x = Conv2D(256, 4, strides=2)(x)

x = LeakyReLU()(x)

x = Conv2D(256, 4, strides=2)(x)

x = LeakyReLU()(x)

x = Flatten()(x)

x = Dropout(0.4)(x)

x = Dense(1, activation='sigmoid')(x)

discriminator = Model(disc\_input, x)

optimizer = RMSprop(

lr=.0001,

clipvalue=1.0,

decay=1e-8

)

discriminator.compile(

optimizer=optimizer,

loss='binary\_crossentropy'

)

return discriminator

generator = create\_generator()

discriminator = create\_discriminator()

discriminator.trainable = False

gan\_input = Input(shape=(LATENT\_DIM, ))

gan\_output = discriminator(generator(gan\_input))

gan = Model(gan\_input, gan\_output)

optimizer = RMSprop(lr=.0001, clipvalue=1.0, decay=1e-8)

gan.compile(optimizer=optimizer, loss='binary\_crossentropy')