Московский государственный технический университет им. Н. Э. Баумана
Факультет «Информатика и системы управления»
Кафедра «Системы обработки информации и управления»

Крусовая работа по предмету «Методы машинного обучения»

на тему: «Компьютерное зрение для анализа туберкулез»

ИСПОЛНИТЕЛЬ:

Студент группы ИУ5-33М

Дин Но

Москва – 2021

1. Введение

Благодаря непрерывному развитию и прогрессу технологий медицинской визуализации и компьютерных технологий анализ медицинских изображений стал незаменимым инструментом и техническим средством в медицинских исследованиях, диагностике и лечении клинических заболеваний. В последние годы глубокое обучение (Deep learning, DL), особенно глубокие сверточные нейронные сети (Convolutional neural networks,CNNs), быстро превратилось в актуальную тему в анализе медицинских изображений. оно может автоматически извлекать предполагаемые характеристики диагностики заболеваний из больших данных медицинских изображений.

В области медицинской визуализации, когда врачи или исследователи проводят количественный анализ, мониторинг в режиме реального времени и планирование лечения определенной внутренней ткани и органа, для принятия правильных решений о лечении им обычно необходимо знать некоторую подробную информацию об этом типе ткани и органа. Поэтому биомедицинская визуализация становится незаменимой и все более важной частью диагностики и лечения заболеваний. Как в полной мере использовать искусственный интеллект и методы глубокого обучения для анализа и обработки этих сверхбольших больших данных медицинских изображений, а также предоставлять научные методы для скрининга, диагностики и оценки различных основных заболеваний в клинической медицине, является серьезной научной проблемой, которую необходимо срочно решить в области анализа медицинских изображений и ключевых технологий передовой медицинской визуализации.

2. Выбор сетевой модели для глубокого обучения

2.1 AlexNet

Архитектура AlexNet CNN выиграла конкурс ImageNet ILSVRC 2012 с большим отрывом: она достигла 17 % частоты ошибок в топ-5, в то время как вторая лучшая достигла только 26 %! Он был разработан Алексом Крижевским (отсюда и название), Ильей Суцкевером и Джеффри Хинтоном. Он очень похож на LeNet-5, только намного больше и глубже, и он был первым, кто укладывал сверточные слои непосредственно друг на друга, вместо того, чтобы укладывать объединяющий слой поверх каждого сверточного слоя. В таблице 14-2 представлена эта архитектура.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13×13	3×3	1	SAME	ReLU
C6	Convolution	384	13×13	3×3	1	SAME	ReLU
C5	Convolution	384	13×13	3×3	1	SAME	ReLU
S4	Max Pooling	256	13×13	3×3	2	VALID	–
C3	Convolution	256	27×27	5×5	1	SAME	ReLU
S2	Max Pooling	96	27×27	3×3	2	VALID	–
C1	Convolution	96	55×55	11×11	4	VALID	ReLU
In	Input	3 (RGB)	227×227	–	–	–	–

Чтобы уменьшить переобучение, авторы использовали два метода регуляризации: сначала они применили отсев с коэффициентом отсева 50 % во время обучения к выходам слоев F8 и F9. Во-вторых, они выполнили увеличение данных путем случайного смещения обучающих изображений на различные смещения, переворачивания их по горизонтали и изменения условий освещения.

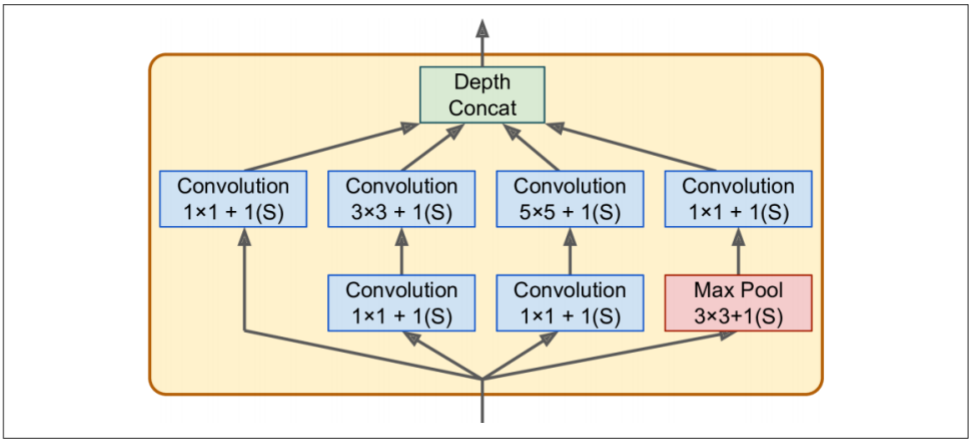
AlexNet также использует этап конкурентной нормализации сразу после этапа ReLU уровней C1 и C3, называемый локальной нормализацией

отклика. Наиболее сильно активированные нейроны подавляют другие нейроны, расположенные в том же положении на соседних картах признаков (такая конкурентная активация наблюдалась в биологических нейронах). Это побуждает различные карты объектов специализироваться, раздвигая их и заставляя их исследовать более широкий спектр функций, в конечном счете улучшая обобщение.

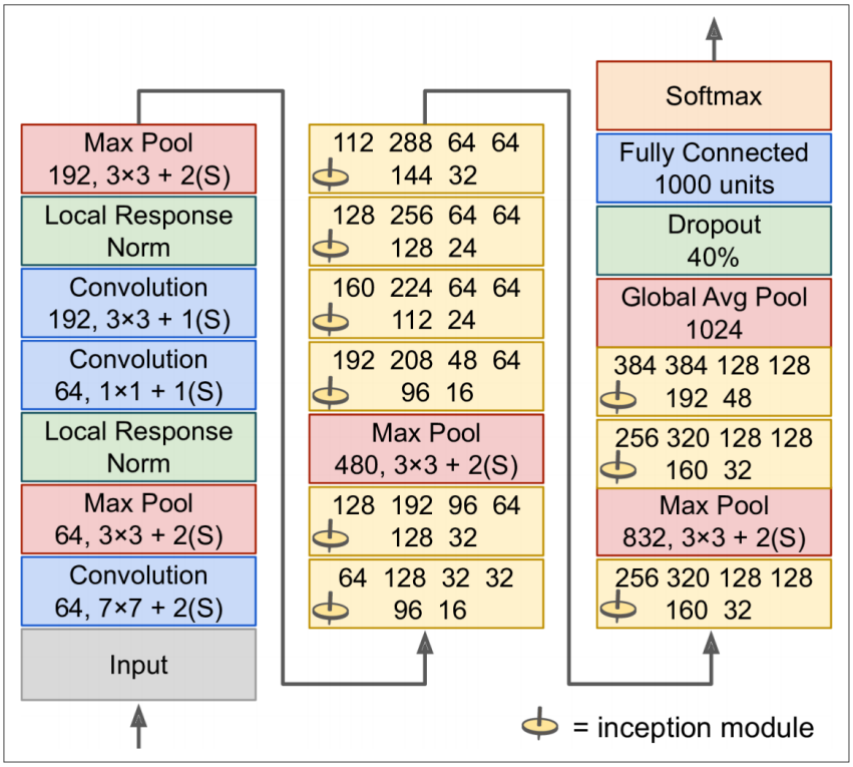
2.2 GoogLeNet

Архитектура GoogLeNet была разработана Кристианом Сегеди и др. из Google Research, и она выиграла конкурс ILSVRC 2014, снизив частоту ошибок в топ-5 ниже 7 %. Эта отличная производительность во многом была обусловлена тем фактом, что сеть была намного глубже, чем предыдущие CNN. Это стало возможным благодаря подсетям, называемым начальными модулями, которые позволяют GoogLeNet использовать параметры гораздо эффективнее, чем предыдущие архитектуры: на самом деле GoogLeNet имеет в 10 раз меньше параметров, чем AlexNet (примерно 6 миллионов вместо 60 миллионов).

На рисунке показана архитектура начального модуля. Обозначение “3 × 3 + 1(Ы)” означает, что слой использует ядро 3 × 3, шаг 1 и ТАКОЕ ЖЕ заполнение. Входной сигнал сначала копируется и подается на четыре разных уровня. Все сверточные слои используют функцию активации ReLUU. Обратите внимание, что во втором наборе сверточных слоев используются ядра разных размеров (1 × 1, 3 × 3 и 5 × 5), что позволяет им захватывать шаблоны в разных масштабах. Также обратите внимание, что каждый отдельный слой использует шаг 1 и ОДИНАКОВОЕ заполнение (даже максимальный слой объединения), поэтому все их выходы имеют ту же высоту и ширину, что и их входы. Это позволяет объединить все выходные данные по измерению глубины в конечном объединяющем слое глубины (т. Е. Сложить карты объектов из всех четырех верхних сверточных слоев). Этот слой объединения может быть реализован в TensorFlow с помощью операции tf.concat() с осью=3 (ось 3 - глубина).



Теперь давайте посмотрим на архитектуру GoogLeNet CNN. Количество карт объектов, выводимых каждым сверточным слоем и каждым объединяющим слоем, показано перед размером ядра. Архитектура настолько глубока, что ее приходится представлять в трех столбцах, но GoogLeNet на самом деле представляет собой один высокий стек, включающий девять начальных модулей (коробки с волчками). Шесть чисел в начальных модулях представляют количество карт объектов, выводимых каждым сверточным слоем в модуле. Обратите внимание, что все сверточные слои используют функцию активации ReLU.



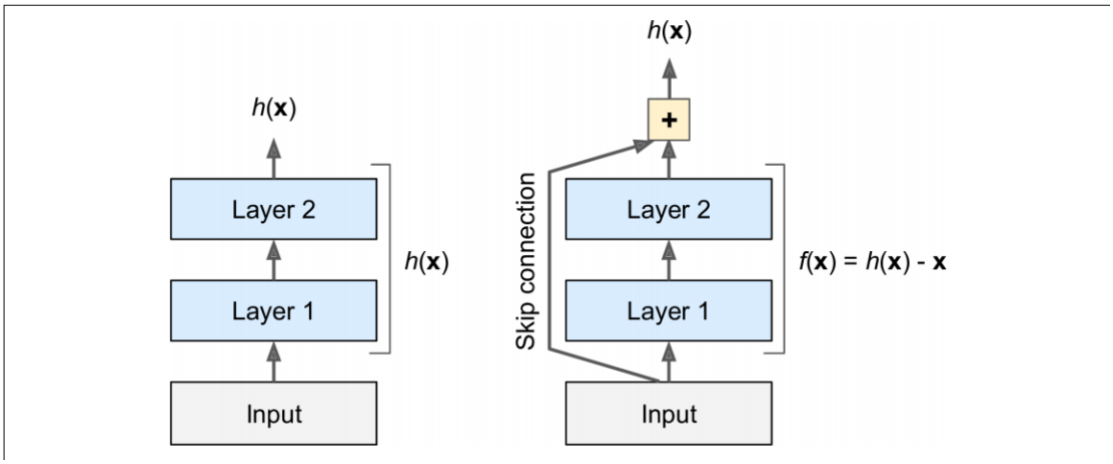
2.3 VGGNet

Второе место в конкурсе ILSVRC 2014 занял VGGNet, разработанный К. Симонян и А.Зиссерман. У него была очень простая и классическая архитектура, с 2 или 3 сверточными слоями, слоем объединения, затем снова 2 или 3 сверточных слоя, слой объединения и т. Д. (Всего всего 16 сверточных слоев), плюс конечная плотная сеть с 2 скрытыми слоями и выходным слоем. Он использовал только 3 = 3 фильтра, но много фильтров.

2.4 ResNet

Вызов ILSVRC 2015 был выигран с использованием остаточной сети (или ResNet), разработанной Kaiming He и др.15, которая обеспечила поразительную частоту ошибок в топ-5 ниже 3,6%, используя чрезвычайно глубокую сеть CNN, состоящую из 152 слоев.

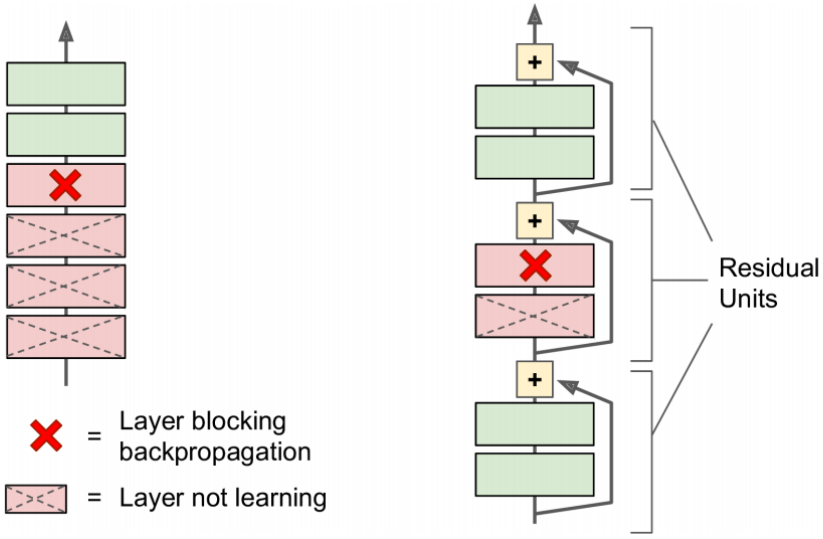
При обучении нейронной сети цель состоит в том, чтобы заставить ее моделировать целевую функцию h(x). Если вы добавите вход x к выходу сети (т. Е. Добавите пропущенное соединение), то сеть будет вынуждена моделировать f(x) = h(x) – x, а не h(x). Это называется остаточным обучением.



Когда вы инициализируете обычную нейронную сеть, ее веса близки к нулю, поэтому сеть просто выводит значения, близкие к нулю. Если вы добавляете пропущенное соединение, результирующая сеть просто выводит копию своих входных данных; другими словами, она изначально моделирует функцию идентификации.

Если целевая функция достаточно близка к функции идентификации (что часто бывает), это значительно ускорит обучение.

Более того, если вы добавите много пропущенных подключений, сеть может начать прогрессировать, даже если несколько уровней еще не начали обучение. Благодаря пропуску соединений сигнал может легко распространяться по всей сети. Глубокую остаточную сеть можно рассматривать как набор остаточных блоков, где каждый остаточный блок представляет собой небольшую нейронную сеть с пропускным соединением.



2.5 DenseNet

DenseNet - это классификационная сеть после ResNet. Изменение метода подключения позволило ей достичь лучших результатов, чем ResNet, на различных больших наборах данных.

Структура сети в начале аналогична ResNet. Сначала выполняется крупномасштабная свертка, за которой следует слой объединения; затем подключается несколько последовательных подмодулей (слой плотного блока и Транзитный слой); и, наконец, подключается объединение и полное соединение.

DenseNet соединяет каждый слой с другими слоями способом прямой связи.В традиционных сверточных нейронных сетях имеется L соединений с сетью L-уровня, в то время как в DenseNet будет L(L+1)/2

соединений.Входные данные каждого слоя поступают с выходных данных всех предыдущих слоев.

Преимущества сети DenseNet：

(1).Уменьшите исчезающий градиент (исчезающий градиент).

(2).Улучшите доставку функций.

(3).Поощряйте повторное использование функций (поощряйте повторное использование функций).

(4).Небольшое количество параметров.

2.6 Выбор

Основная идея DenseNet заключается в установлении связи между различными уровнями, полном использовании функций и дальнейшем решении проблемы исчезновения градиентов. Углубление сети не является проблемой, и эффект обучения очень хороший. Кроме того, использование уровня узких мест, уровня трансляции и меньшей скорости роста сужает сеть и уменьшает параметры, эффективно подавляя переобучение и уменьшая объем вычислений. У DenseNet много преимуществ, и эти преимущества все еще очень очевидны по сравнению с ResNet.

Поэтому в данной работе для сравнения выбраны традиционная модель сверточной нейронной сети и модель DenseNet.

3. Часть кода

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.layers import (
    Dense, Conv2D, MaxPool2D, Dropout, Flatten,
    BatchNormalization, GlobalAveragePooling2D)
from keras.applications.densenet import DenseNet121
from keras import backend as K
from sklearn.metrics import confusion_matrix, classification_report

os.listdir("drive/MyDrive/chestX/chest_xray/chest_xray")
```

```
['.DS_Store', 'test', 'train', 'val']
```

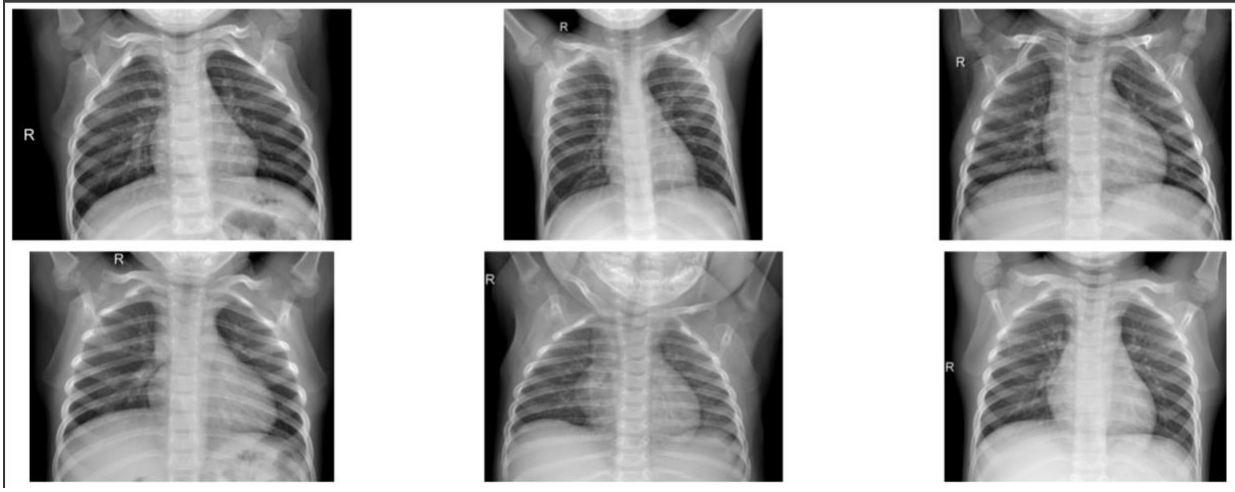
```
len(os.listdir("drive/MyDrive/chestX/chest_xray/chest_xray/train/PNEUMONIA"))
```

```
3876
```

```
train_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/train"
test_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/test"
val_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/val"

print("Train set:\n=====================================")
num_pneumonia = len(os.listdir(os.path.join(train_dir, 'PNEUMONIA')))
num_normal = len(os.listdir(os.path.join(train_dir, 'NORMAL')))
print(f"PNEUMONIA={num_pneumonia}")
print(f"NORMAL={num_normal}")

print("Test set:\n=====================================")
print(f"PNEUMONIA = {len(os.listdir(os.path.join(test_dir, 'PNEUMONIA')))}")
print(f"NORMAL = {len(os.listdir(os.path.join(test_dir, 'NORMAL')))}")

print("Validation set:\n=====================================")
print(f"PNEUMONIA = {len(os.listdir(os.path.join(val_dir, 'PNEUMONIA')))}")
print(f"NORMAL = {len(os.listdir(os.path.join(val_dir, 'NORMAL')))}")

pneumonia = os.listdir("drive/MyDrive/chestX/chest_xray/chest_xray/train/PNEUMONIA")
pneumonia_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/train/PNEUMONIA"

plt.figure(figsize=(20, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(pneumonia_dir, pneumonia[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```

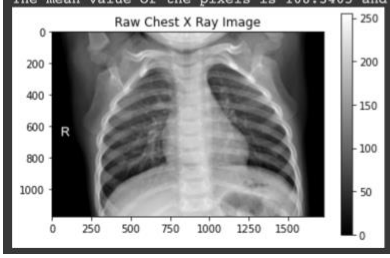
```
Train set:
=====================================
PNEUMONIA=3876
NORMAL=1342
Test set:
=====================================
PNEUMONIA = 390
NORMAL = 234
Validation set:
=====================================
PNEUMONIA = 9
NORMAL = 9
```

```
normal = os.listdir("drive/MyDrive/chestX/chest_xray/chest_xray/train/NORMAL")
normal_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/train/NORMAL"

plt.figure(figsize=(20, 10))

for i in range(9):
    plt.subplot(3, 3, i + 1)
    img = plt.imread(os.path.join(normal_dir, normal[i]))
    plt.imshow(img, cmap='gray')
    plt.axis('off')

plt.tight_layout()
```



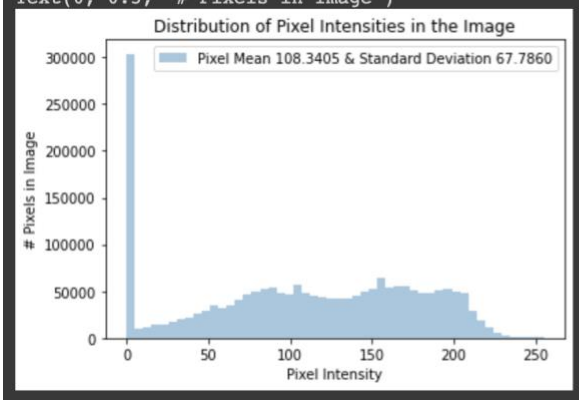
```
normal_img = os.listdir("drive/MyDrive/chestX/chest_xray/chest_xray/train/NORMAL")[0]
normal_dir = "drive/MyDrive/chestX/chest_xray/chest_xray/train/NORMAL"
sample_img = plt.imread(os.path.join(normal_dir, normal_img))
plt.imshow(sample_img, cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {sample_img.shape[0]} pixels width and {sample_img.shape[1]} pixels height, one single color channel.")
print(f"The maximum pixel value is {sample_img.max():.4f} and the minimum is {sample_img.min():.4f}")
print(f"The mean value of the pixels is {sample_img.mean():.4f} and the standard deviation is {sample_img.std():.4f}")
```

```
The dimensions of the image are 1175 pixels width and 1728 pixels height, one single color channel.
The maximum pixel value is 255.0000 and the minimum is 0.0000
The mean value of the pixels is 108.3405 and the standard deviation is 67.7860
```



```
sns.distplot(sample_img.ravel(),
             label=f"Pixel Mean {np.mean(sample_img):.4f} & Standard Deviation {np.std(sample_img):.4f}",
             kde=False)
plt.legend(loc='upper right')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprec
  warnings.warn(msg, FutureWarning)
Text(0, 0.5, '# Pixels in Image')
```



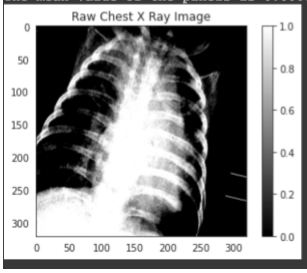

```
image_generator = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    samplewise_center=True,
    samplewise_std_normalization=True
)
```

```
train = image_generator.flow_from_directory(train_dir,
                                            batch_size=8,
                                            shuffle=True,
                                            class_mode='binary',
                                            target_size=(320, 320))

validation = image_generator.flow_from_directory(val_dir,
                                                 batch_size=1,
                                                 shuffle=False,
                                                 class_mode='binary',
                                                 target_size=(320, 320))

test = image_generator.flow_from_directory(test_dir,
                                           batch_size=1,
                                           shuffle=False,
                                           class_mode='binary',
                                           target_size=(320, 320))
```

```
Found 5216 images belonging to 2 classes.
Found 16 images belonging to 2 classes.
Found 624 images belonging to 2 classes.
```

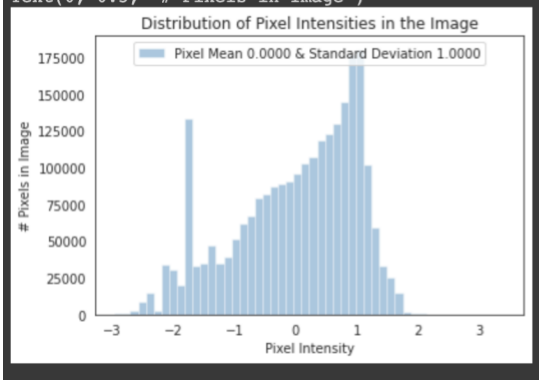
```
sns.set_style('white')
generated_image, label = train.__getitem__(0)
plt.imshow(generated_image[0], cmap='gray')
plt.colorbar()
plt.title('Raw Chest X Ray Image')

print(f"The dimensions of the image are {generated_image.shape[1]} pixels width and {generated_image.shape[2]} pixels height, one single color channel.")
print(f"The maximum pixel value is {generated_image.max():.4f} and the minimum is {generated_image.min():.4f}")
print(f"The mean value of the pixels is {generated_image.mean():.4f} and the standard deviation is {generated_image.std():.4f}")
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
The dimensions of the image are 320 pixels width and 320 pixels height, one single color channel.
The maximum pixel value is 3.3982 and the minimum is -2.9557
The mean value of the pixels is 0.0000 and the standard deviation is 1.0000
```




```
sns.distplot(generated_image.ravel(),
             label=f"Pixel Mean {np.mean(generated_image):.4f} & Standard Deviation {np.std(generated_image):.4f}",
             kde=False)
plt.legend(loc='upper center')
plt.title('Distribution of Pixel Intensities in the Image')
plt.xlabel('Pixel Intensity')
plt.ylabel('# Pixels in Image')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated func
  warnings.warn(msg, FutureWarning)
Text(0, 0.5, '# Pixels in Image')
```



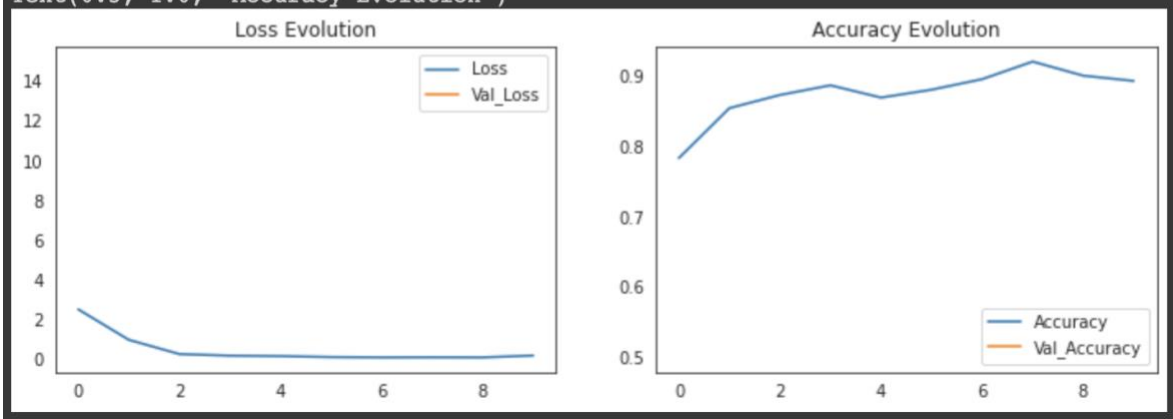
```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(320, 320, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=(320, 320, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPool2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
r = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    class_weight=class_weight,
    steps_per_epoch=100,
    validation_steps=25,
)
```

```
Epoch 1/10
100/100 [==============================] - ETA: 0s - loss: 2.5429 - accuracy: 0.7837WARNING:tensorflow:Your input ran out of data; in
100/100 [==============================] - 224s 2s/step - loss: 2.5429 - accuracy: 0.7837 - val_loss: 15.0002 - val_accuracy: 0.5000
Epoch 2/10
100/100 [==============================] - 171s 2s/step - loss: 1.0102 - accuracy: 0.8550
Epoch 3/10
100/100 [==============================] - 146s 1s/step - loss: 0.2936 - accuracy: 0.8737
Epoch 4/10
100/100 [==============================] - 122s 1s/step - loss: 0.2140 - accuracy: 0.8875
Epoch 5/10
100/100 [==============================] - 106s 1s/step - loss: 0.1923 - accuracy: 0.8700
Epoch 6/10
100/100 [==============================] - 99s 988ms/step - loss: 0.1434 - accuracy: 0.8813
Epoch 7/10
100/100 [==============================] - 97s 963ms/step - loss: 0.1266 - accuracy: 0.8963
Epoch 8/10
100/100 [==============================] - 84s 834ms/step - loss: 0.1317 - accuracy: 0.9212
Epoch 9/10
100/100 [==============================] - 74s 735ms/step - loss: 0.1266 - accuracy: 0.9013
Epoch 10/10
100/100 [==============================] - 73s 730ms/step - loss: 0.2208 - accuracy: 0.8938
```

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')
```

```
Text(0.5, 1.0, 'Accuracy Evolution')
```




```
evaluation = model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

```
624/624 [==============================] - 144s 230ms/step - loss: 1.0176 - accuracy: 0.8077
Test Accuracy: 80.77%
652/652 [==============================] - 393s 603ms/step - loss: 0.2150 - accuracy: 0.9308
Train Accuracy: 93.08%
```

```
base_model = DenseNet121(input_shape=(320, 320, 3), include_top=False, weights='imagenet', pooling='avg')

base_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_t
29089792/29084464 [==============================] - 0s 0us/step
29097984/29084464 [==============================] - 0s 0us/step
Model: "densenet121"
```

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 320, 320, 3)	0	
zero_padding2d (ZeroPadding2D)	(None, 326, 326, 3)	0	input_1[0][0]
conv1/conv (Conv2D)	(None, 160, 160, 64)	9408	zero_padding2d[0][0]
conv1/bn (BatchNormalization)	(None, 160, 160, 64)	256	conv1/conv[0][0]
conv1/relu (Activation)	(None, 160, 160, 64)	0	conv1/bn[0][0]
zero_padding2d_1 (ZeroPadding2D	(None, 162, 162, 64)	0	conv1/relu[0][0]
pool1 (MaxPooling2D)	(None, 80, 80, 64)	0	zero_padding2d_1[0][0]
conv2_block1_0_bn (BatchNormali	(None, 80, 80, 64)	256	pool1[0][0]
conv2_block1_0_relu (Activation	(None, 80, 80, 64)	0	conv2_block1_0_bn[0][0]

```
layers = base_model.layers
print(f"The model has {len(layers)} layers")

print(f"The input shape {base_model.input}")
print(f"The output shape {base_model.output}")
```

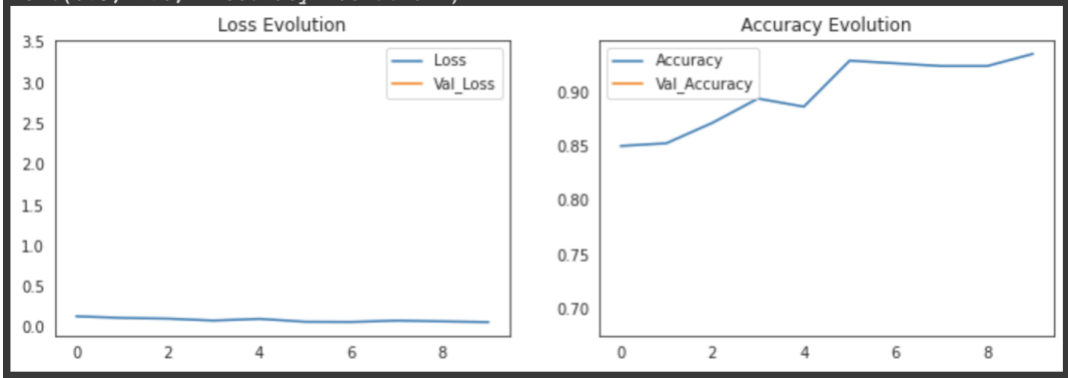
```
The model has 428 layers
The input shape KerasTensor(type_spec=TensorSpec(shape=(None, 320, 320, 3), dtype=tf.float32, name='input_1'), name='input_1', description="created by layer 'input_1'")
The output shape KerasTensor(type_spec=TensorSpec(shape=(None, 1024), dtype=tf.float32, name=None), name='avg_pool/Mean:0', description="created by layer 'avg_pool'")
```

```
base_model = DenseNet121(include_top=False, weights='imagenet')
x = base_model.output

x = GlobalAveragePooling2D()(x)

predictions = Dense(1, activation="sigmoid")(x)

model = Model(inputs=base_model.input, outputs=predictions)
# model.add(base_model)
# model.add(GlobalAveragePooling2D())
# model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

r = model.fit(
    train,
    epochs=10,
    validation_data=validation,
    class_weight=class_weight,
    steps_per_epoch=100,
    validation_steps=25,
)
```

```
Epoch 1/10
100/100 [==============================] - ETA: 0s - loss: 0.1374 - accuracy: 0.8500WARNING:tensorflow:Your input ran ou
100/100 [==============================] - 69s 498ms/step - loss: 0.1374 - accuracy: 0.8500 - val_loss: 3.3577 - val_acc
Epoch 2/10
100/100 [==============================] - 46s 453ms/step - loss: 0.1174 - accuracy: 0.8525
Epoch 3/10
100/100 [==============================] - 46s 457ms/step - loss: 0.1089 - accuracy: 0.8712
Epoch 4/10
100/100 [==============================] - 46s 452ms/step - loss: 0.0850 - accuracy: 0.8938
Epoch 5/10
100/100 [==============================] - 45s 449ms/step - loss: 0.1062 - accuracy: 0.8863
Epoch 6/10
100/100 [==============================] - 45s 446ms/step - loss: 0.0705 - accuracy: 0.9287
Epoch 7/10
100/100 [==============================] - 45s 449ms/step - loss: 0.0682 - accuracy: 0.9262
Epoch 8/10
100/100 [==============================] - 45s 448ms/step - loss: 0.0848 - accuracy: 0.9237
Epoch 9/10
100/100 [==============================] - 46s 455ms/step - loss: 0.0755 - accuracy: 0.9237
Epoch 10/10
100/100 [==============================] - 45s 449ms/step - loss: 0.0662 - accuracy: 0.9350
```

```
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(r.history['loss'], label='Loss')
plt.plot(r.history['val_loss'], label='Val_Loss')
plt.legend()
plt.title('Loss Evolution')

plt.subplot(2, 2, 2)
plt.plot(r.history['accuracy'], label='Accuracy')
plt.plot(r.history['val_accuracy'], label='Val_Accuracy')
plt.legend()
plt.title('Accuracy Evolution')
```

```
Text(0.5, 1.0, 'Accuracy Evolution')
```




```
evaluation = model.evaluate(test)
print(f"Test Accuracy: {evaluation[1] * 100:.2f}%")

evaluation = model.evaluate(train)
print(f"Train Accuracy: {evaluation[1] * 100:.2f}%")
```

```
624/624 [==============================] - 36s 58ms/step - loss: 0.3124 - accuracy: 0.8830
Test Accuracy: 88.30%
652/652 [==============================] - 222s 340ms/step - loss: 0.2731 - accuracy: 0.8836
Train Accuracy: 88.36%
```

4. Вывод

Судя по результатам, показатель точности традиционной модели на обучающем наборе составляет 93,08%, но показатель точности на тестовом наборе составляет всего 80,77%, поэтому модель переоснащена. Однако модель DenseNet, будь то тестовый набор или набор для проверки, имеет точность более 88%.

С точки зрения потери данных, объем потерь модели DenseNet значительно меньше, чем у первой модели, и она хорошо работала на протяжении всего процесса обучения.

Судя по скорости подгонки, поскольку мы внедрили обучение миграции для DenseNet, очевидного процесса подгонки нет, и скорость подгонки данных выше, а эффект лучше.

Таким образом, DenseNet является хорошим выбором модели для анализа медицинских изображений с использованием компьютерного зрения.

5. Список литературы

[1] Гапанюк Ю. Е. Домашнее задание по дисциплине «Методы машинного обучения»[Электронный ресурс] // GitHub. — 2019. — Режим доступа: https://github.com/ugapanyuk/ml_course/wiki/MMO_DZ (дата обращения: 06.05.2019).

[2] You are my Sunshine [Electronic resource] // Space Apps Challenge. — 2017. Access mode: https://2017.spaceappschallenge.org/challenges/earth-and-us/ you-are-my-sunshine/details (online; accessed: 22.02.2019).

[3] dronio. Solar Radiation Prediction [Electronic resource] // Kaggle. — 2017. — Access mode: https://www.kaggle.com/dronio/SolarEnergy (online; accessed: 18.02.2019).

[4] Team The IPython Development. IPython 7.3.0 Documentation [Electronic resource] //Read the Docs. — 2019. — Access mode: https://ipython.readthedocs.io/en/stable/ (online; accessed: 20.02.2019).

[5] Waskom M. seaborn 0.9.0 documentation [Electronic resource] // PyData. — 2018. Access mode: https://seaborn.pydata.org/ (online; accessed: 20.02.2019).

[6] pandas 0.24.1 documentation [Electronic resource] // PyData. — 2019. — Access mode:http://pandas.pydata.org/pandas-docs/stable/ (online; accessed: 20.02.2019).

[7] Chrétien M. Convert datetime.time to seconds [Electronic resource] // Stack Overflow. 2017. — Access mode: https://stackoverflow.com/a/44823381 (online; accessed:20.02.2019).