Deliverable 1

Report / Note book

On

Histopathologic Cancer Detection using CNN

Kaggle

Kaggle is a platform for data science competitions where participants compete to create the best models for solving specific problems or analyzing certain data sets. The platform is also used for learning, collaboration, job opportunities, community building, and research in the data science and machine learning fields.

Kaggle is a valuable resource for data scientists and machine learning engineers looking to improve their skills, collaborate with others, and tackle real-world data problems. In this article, you can learn what Kaggle is, how it is used, and what the competitions are like.
Kaggle is a platform for data science competitions, where data scientists and machine learning engineers can compete with each other to create the best models for solving specific problems or analyzing certain data sets. The platform also provides a community where users can collaborate on projects, share code and data sets, and learn from each other's work. Founded in 2010, Google acquired Kaggle in 2017, and the platform is now part of Google Cloud.

What things are included Kernel

- Problem Statement and the Analysis of the Problem Statement
- Data Understanding
- Designing the Model
- Validation And Analysis
    - Metrics
    - Prediction and Activation Visualizations
    - ROC AND AUC

Histopathologic Cancer Detection using CNN

Brief description of the problem and data

Project Topic and Goal

This project is about developing an algorithm to identify metastatic cancer in small image patches extracted from larger digital pathology scans. The dataset used for this competition is a modified version of the PatchCamelyon (PCam) benchmark dataset, which focuses on the detection of metastasis in cancer patients.

The primary problem that this project aims to solve is the automated identification of metastatic cancerin digital pathology images. This will be done by creating deep learning models that can accurately classify whether a given image patch contains evidence of cancer metastasis or not. The dataset provided for this competition simulates this task by presenting binary image classification challenges similar to well-known benchmark datasets like CIFAR-10 and MNIST.

The submissions are evaluated on the area under the ROC curve between the predicted probability andthe observed target.

Data

The dataset consists of many small pathology images for classification. Each image has an image id, and the train_labels.csv file offers the true labels for images in the train folder. The objective is to predict labels for images in the test folder. Positive labels indicate the presence of tumor tissue in the central 32:32px section of a patch (with tumor tissue in the surrounding area not affecting the label). This outer region supports fully-convolutional models that maintain consistency when applied to whole-slide images, without zero-padding.

While the original PCam dataset included duplicate images due to random sampling, the version used inthis competition has no duplicates. The split ratio of train and test data remains consistent with the PCam benchmark.

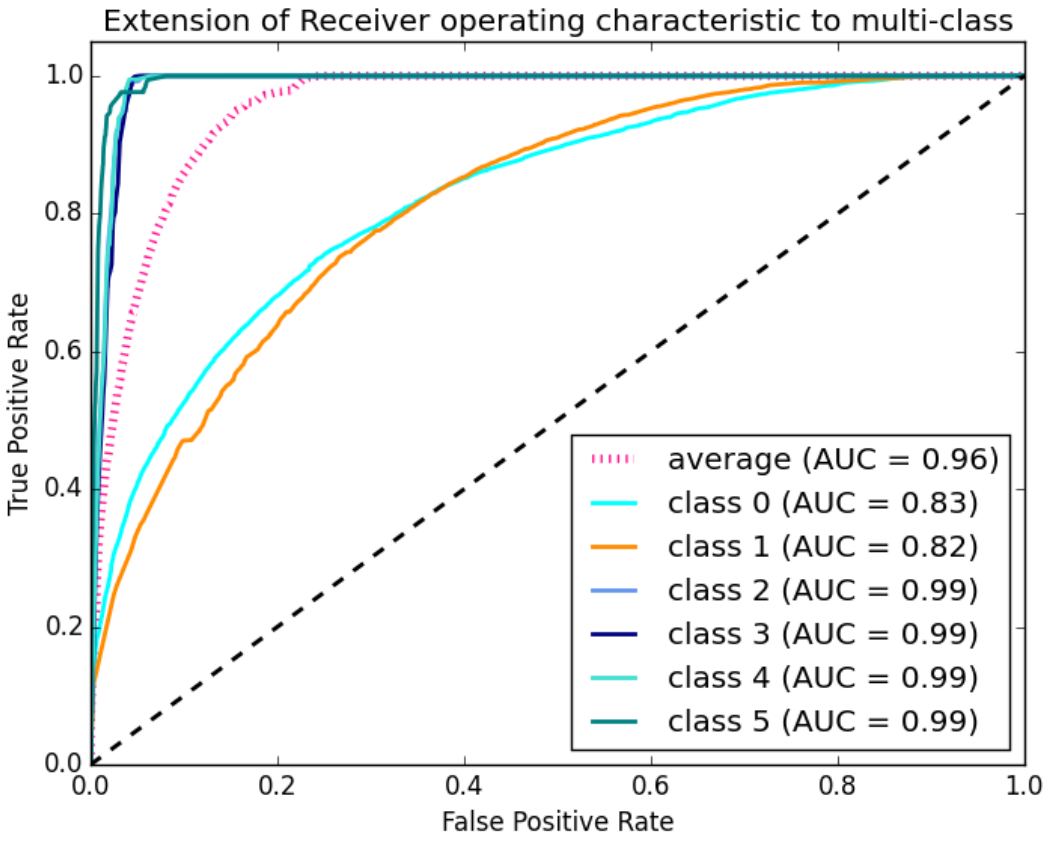
Train and test data combined exhibit a size of rougly 7.76 GB which is quite large. There are 220,000 training train images and 57,000 test images, each represented in a .tif format. TIFF (Tagged Image FileFormat) is a versatile image file format used for various purposes, including photography or medical imaging. Even compressed it maintains high image quality which leads to larger file sizes that can impact the required storage size compared to formats like JPEG or PNG.

For evaluation we are required to create a submission file consisting of a 2-column table. First, there willbe the ID of the test images and second the predicted label (1 = positive, 0 = negative).

Problem Statment

The problem is mainly a BINARY IMAGE CLASSIFICATION PROBLEM. The Problem focuses on identifying the presence of metastases from a 96 * 96 digital histopathology images

Metric Evaluation - Submissions are evaluated on area under the ROC curve between the predicted probability and the observed target.



Analysis of the problem Statment

What Exactly the problem statment conveys to us?

1. The problem deals with the Binary Classification of the Image that has a shape of 96px *96px. It involves identifying the metastases from the 96px* 96px digital histapathology images.

2. One key challenge is that the metastases can be as small as single cells in a large area of tissue.¶

The Histopathological Images:

linkcode

About the Domain:

Obviously, I do not know much about Biology,I made some notes about the following terminologies :

- Histopathology
- Lymphocytes
- Lymph Nodes

So, let us see some of the biological terminologies involved

1. Histopathology - Histopathology is the diagnosis and study of diseases of the tissues, and involves examining tissues and/or cells under a microscope. Histopathologists are responsible for making tissue diagnoses and helping clinicians manage a patient's care.

2. Lymphocytes - Lymphocytes are white blood cells that are also one of the body's main types of immune cells. They are made in the bone marrow and found in the blood and lymph tissue. The immune system is a complex network of cells known as immune cells that include lymphocytes.

Lymph Nodes- Lymph nodes are small lumps of tissue that contain white blood cells, which fight infection. They filter lymph fluid, which is composed of fluid and waste products from your body tissues. Lymph nodes also help activate your immune system if you have an infection.

Exploratory Data Analysis (EDA) — Inspect, Visualize and Clean the Data

Data Understanding

- The dataset contains the histopathological Images, each image is 96px * 96px.
- A positive label indicates that the center 32x32px region of a patch contains at least one pixel of tumor tissue. Tumor tissue in the outer region of the patch does not influence the label. This outer region is provided to enable fully-convolutional models that do not use zero-padding, to ensure consistent behavior when applied to a whole-slide image.

- 'The original PCam dataset contains duplicate images due to its probabilistic sampling,
- however, the version presented on Kaggle does not contain duplicates. We have otherwise maintained the same data and splits as the PCam benchmark.'
- Also, one of the hing is that the problem states that the training Data contains 50/50 Images of both the labels i.e. the training contains equal proportion of both the labels, however on analysis it was found to be nearly equal to 60/40, which we will consider while we design the model

```
# Importing  Libraries
from numpy.random import seed
seed(101)

import pandas as pd
import numpy as np


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.layers import Dense, Dropout, Flatten, Activation
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from tensorflow.keras.optimizers import Adam

import os
import cv2

from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline
tf.random.set_seed(101)
```

```
# Setting Some Pre-Requisites
IMAGE_SIZE=96
IMAGE_CHANNELS=3
SAMPLE_SIZE=80000       # We will be training 80,000 samples from each label
```

```
# So, what are the files which are available?

os.listdir('../input/histopathologic-cancer-detection')
```

```
['sample_submission.csv', 'train_labels.csv', 'test', 'train']
```

```
# So, how many images are there in each of the folder in the training dataset?

print(len(os.listdir('../input/histopathologic-cancer-detection/train')))
print(len(os.listdir('../input/histopathologic-cancer-detection/test')))
```

```
220025
57458
```

```
# Creating a dataframe of all the training images

df_data = pd.read_csv('../input/histopathologic-cancer-detection/train_labels.csv')

# removing this image because it caused a training error previously
df_data[df_data['id'] != 'dd6dfed324f9fcb6f93f46f32fc800f2ec196be2']

# removing this image because it's black
df_data[df_data['id'] != '9369c7278ec8bcc6c880d99194de09fc2bd4efbe']


print(df_data.shape)
```

```
(220025, 2)
```

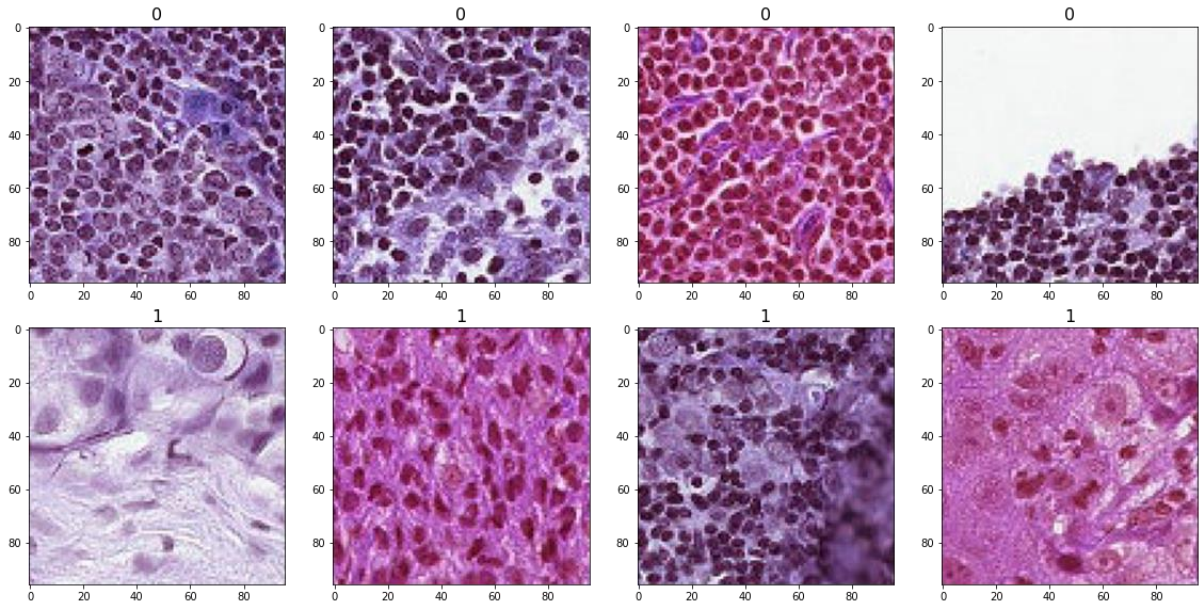
```
df_data['label'].value_counts()
```

```
0    130908
1     89117
Name: label, dtype: int64
```

```
# source: https://www.kaggle.com/gpreda/honey-bee-subspecies-classification

def draw_category_images(col_name,figure_cols, df, IMAGE_PATH):

    """
```

```
    Give a column in a dataframe,
    this function takes a sample of each class and displays that
    sample on one row. The sample size is the same as figure_cols which
    is the number of columns in the figure.
    Because this function takes a random sample, each time the function is run it
    displays different images.
    """


    categories = (df.groupby([col_name])[col_name].nunique()).index
    f, ax = plt.subplots(nrows=len(categories),ncols=figure_cols,
                         figsize=(4*figure_cols,4*len(categories))) # adjust size here
    # draw a number of images for each location
    for i, cat in enumerate(categories):
        sample = df[df[col_name]==cat].sample(figure_cols) # figure_cols is also the sample size
        for j in range(0,figure_cols):
            file=IMAGE_PATH + sample.iloc[j]['id'] + '.tif'
im=cv2.imread(file)
            ax[i, j].imshow(im, resample=True, cmap='gray')
            ax[i, j].set_title(cat, fontsize=16)
    plt.tight_layout()
    plt.show()
```

```
IMAGE_PATH = '../input/histopathologic-cancer-detection/train/'

draw_category_images('label',4, df_data, IMAGE_PATH)
```



DModel Architecture

```
# Create the Train and Validation Sets

df_0=df_data[df_data['label']==0].sample(SAMPLE_SIZE,random_state=101)
df_1=df_data[df_data['label']==1].sample(SAMPLE_SIZE,random_state=101)

# concat the dataframes
df_data = pd.concat([df_0, df_1], axis=0).reset_index(drop=True)
# shuffle
df_data = shuffle(df_data)

df_data['label'].value_counts()
```

```
1    80000
0    80000
Name: label, dtype: int64
```

```
# Now, for the train-test split

# stratify=y creates a balanced validation set.
y = df_data['label']

df_train, df_val = train_test_split(df_data, test_size=0.10, random_state=101, stratify=y)

print(df_train.shape)
print(df_val.shape)
```

```
(144000, 2)
(16000, 2)
```

```
# Create a new directory so that we will be using the ImageDataGenerator
base_dir='base_dir'
os.mkdir(base_dir)

# now we create 2 folders inside 'base_dir':

# train_dir
    # a_no_tumor_tissue
    # b_has_tumor_tissue

# val_dir
    # a_no_tumor_tissue
    # b_has_tumor_tissue



# create a path to 'base_dir' to which we will join the names of the new folders
# train_dir
train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)
```

```
# val_dir
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)



# [CREATE FOLDERS INSIDE THE TRAIN AND VALIDATION FOLDERS]
# Inside each folder we create seperate folders for each class

# create new folders inside train_dir
no_tumor_tissue = os.path.join(train_dir, 'a_no_tumor_tissue')
os.mkdir(no_tumor_tissue)
has_tumor_tissue = os.path.join(train_dir, 'b_has_tumor_tissue')
os.mkdir(has_tumor_tissue)


# create new folders inside val_dir
no_tumor_tissue = os.path.join(val_dir, 'a_no_tumor_tissue')
os.mkdir(no_tumor_tissue)
has_tumor_tissue = os.path.join(val_dir, 'b_has_tumor_tissue')
os.mkdir(has_tumor_tissue)
```

```
# check that the folders have been created
os.listdir('base_dir/train_dir')
```

```
['b_has_tumor_tissue', 'a_no_tumor_tissue']
```

```
# Set the id as the index in df_data
df_data.set_index('id', inplace=True)
```

```
Get a list of train and val images
train_list = list(df_train['id'])
val_list = list(df_val['id'])



# Transfer the train images

for image in train_list:

    # the id in the csv file does not have the .tif extension therefore we add it here
    fname = image + '.tif'
    # get the label for a certain image
    target = df_data.loc[image,'label']

    # these must match the folder names
    if target == 0:
        label = 'a_no_tumor_tissue'
    if target == 1:
        label = 'b_has_tumor_tissue'
# source path to image
    src = os.path.join('../input/histopathologic-cancer-detection/train', fname)
```

```
    # destination path to image
    dst = os.path.join(train_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)


# Transfer the val images

for image in val_list:

    # the id in the csv file does not have the .tif extension therefore we add it here
    fname = image + '.tif'
    # get the label for a certain image
    target = df_data.loc[image,'label']

    # these must match the folder names
    if target == 0:
        label = 'a_no_tumor_tissue'
    if target == 1:
        label = 'b_has_tumor_tissue'


    # source path to image
    src = os.path.join('../input/histopathologic-cancer-detection/train', fname)
    # destination path to image
    dst = os.path.join(val_dir, label, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
```

```
# check how many train images we have in each folder

print(len(os.listdir('base_dir/train_dir/a_no_tumor_tissue')))
print(len(os.listdir('base_dir/train_dir/b_has_tumor_tissue')))
```

```
72000
72000
```

```
# check how many val images we have in each folder

print(len(os.listdir('base_dir/val_dir/a_no_tumor_tissue')))
print(len(os.listdir('base_dir/val_dir/b_has_tumor_tissue')))
```

```
8000
8000
```

```
# Set up the generators
train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'
test_path = '../input/histopathologic-cancer-detection/test'

num_train_samples = len(df_train)
num_val_samples = len(df_val)
train_batch_size = 10
val_batch_size = 10
```

```
train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)
```

```
datagen = ImageDataGenerator(rescale=1.0/255)

train_gen = datagen.flow_from_directory(train_path,
                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                        batch_size=train_batch_size,
                                        class_mode='categorical')

val_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                        batch_size=val_batch_size,
                                        class_mode='categorical')

# Note: shuffle=False causes the test dataset to not be shuffled
test_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                        batch_size=1,
                                        class_mode='categorical',
                                        shuffle=False)
```

```
Found 144000 images belonging to 2 classes.
Found 16000 images belonging to 2 classes.
Found 16000 images belonging to 2 classes.
```

```
kernel_size = (3,3)
pool_size= (2,2)
first_filters = 32
second_filters = 64
third_filters = 128

dropout_conv = 0.3
dropout_dense = 0.3


model = Sequential()
model.add(Conv2D(first_filters, kernel_size, activation = 'relu', input_shape = (96, 96, 3)))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))
```

```
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(dropout_dense))
model.add(Dense(2, activation = "softmax"))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 94, 94, 32)        896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 92, 92, 32)        9248
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 90, 90, 32)        9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 45, 45, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 43, 43, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 41, 41, 64)        36928
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 39, 39, 64)        36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 17, 17, 128)       73856
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 4608)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1179904
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 514
=================================================================
Total params: 1,661,186
Trainable params: 1,661,186
Non-trainable params: 0
_________________________________________________________________
```

In [20]:

```
model.compile(Adam(l
```

```
model.compile(Adam(lr=0.0001), loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# Get the labels that are associated with each index
print(val_gen.class_indices)
```

```
{'a_no_tumor_tissue': 0, 'b_has_tumor_tissue': 1}
```

```
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=2,
                              verbose=1, mode='max', min_lr=0.00001)


callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(train_gen, steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=20, verbose=1,
                   callbacks=callbacks_list
```

```
Epoch 1/20
14400/14400 [==============================] - 242s 17ms/step - loss: 0.4393 - accuracy: 0.7993 - v
al_loss: 0.3816 - val_accuracy: 0.8222
Epoch 2/20
14400/14400 [==============================] - 210s 15ms/step - loss: 0.3384 - accuracy: 0.8533 - v
al_loss: 0.3270 - val_accuracy: 0.8629
Epoch 3/20
14400/14400 [==============================] - 218s 15ms/step - loss: 0.2951 - accuracy: 0.8757 - v
al_loss: 0.3072 - val_accuracy: 0.8646
Epoch 4/20
14400/14400 [==============================] - 215s 15ms/step - loss: 0.2711 - accuracy: 0.8878 - v
al_loss: 0.2442 - val_accuracy: 0.9014
Epoch 5/20
14400/14400 [==============================] - 221s 15ms/step - loss: 0.2524 - accuracy: 0.8971 - v
al_loss: 0.2342 - val_accuracy: 0.9062
Epoch 6/20
14400/14400 [==============================] - 216s 15ms/step - loss: 0.2375 - accuracy: 0.9044 - v
al_loss: 0.2354 - val_accuracy: 0.9105
Epoch 7/20
14400/14400 [==============================] - 215s 15ms/step - loss: 0.2281 - accuracy: 0.9089 - v
al_loss: 0.2237 - val_accuracy: 0.9118
Epoch 8/20
14400/14400 [==============================] - 209s 15ms/step - loss: 0.2184 - accuracy: 0.9126 - v
al_loss: 0.1994 - val_accuracy: 0.9220
Epoch 9/20
14400/14400 [==============================] - 214s 15ms/step - loss: 0.2098 - accuracy: 0.9165 - v
al_loss: 0.2024 - val_accuracy: 0.9214
Epoch 10/20
```

```
14400/14400 [==============================] - 205s 14ms/step - loss: 0.2019 - accuracy: 0.9204 - v
al_loss: 0.1995 - val_accuracy: 0.9219
Epoch 11/20
14400/14400 [==============================] - 217s 15ms/step - loss: 0.1953 - accuracy: 0.9231 - v
al_loss: 0.2069 - val_accuracy: 0.9170
Epoch 12/20
14400/14400 [==============================] - 210s 15ms/step - loss: 0.1905 - accuracy: 0.9256 - v
al_loss: 0.2050 - val_accuracy: 0.9174
Epoch 13/20
14400/14400 [==============================] - 207s 14ms/step - loss: 0.1846 - accuracy: 0.9287 - v
al_loss: 0.1748 - val_accuracy: 0.9341
Epoch 14/20
14400/14400 [==============================] - 209s 15ms/step - loss: 0.1804 - accuracy: 0.9301 - v
al_loss: 0.1934 - val_accuracy: 0.9252
Epoch 15/20
14400/14400 [==============================] - 209s 14ms/step - loss: 0.1738 - accuracy: 0.9335 - v
al_loss: 0.1801 - val_accuracy: 0.9292
Epoch 16/20
14400/14400 [==============================] - 195s 14ms/step - loss: 0.1692 - accuracy: 0.9351 - v
al_loss: 0.1877 - val_accuracy: 0.9262
Epoch 17/20
14400/14400 [==============================] - 211s 15ms/step - loss: 0.1659 - accuracy: 0.9364 - v
al_loss: 0.1799 - val_accuracy: 0.9334
Epoch 18/20
14400/14400 [==============================] - 214s 15ms/step - loss: 0.1614 - accuracy: 0.9381 - v
al_loss: 0.1870 - val_accuracy: 0.9293
Epoch 19/20
14400/14400 [==============================] - 213s 15ms/step - loss: 0.1603 - accuracy: 0.9392 - v
al_loss: 0.1829 - val_accuracy: 0.9321
Epoch 20/20
14400/14400 [==============================] - 200s 14ms/step - loss: 0.1559 - accuracy: 0.9408 - v
al_loss: 0.1655 - val_accuracy: 0.9366
```

In [23]:

 # get the met

```
# get the metric names so we can use evaulate_generator
model.metrics_names
```

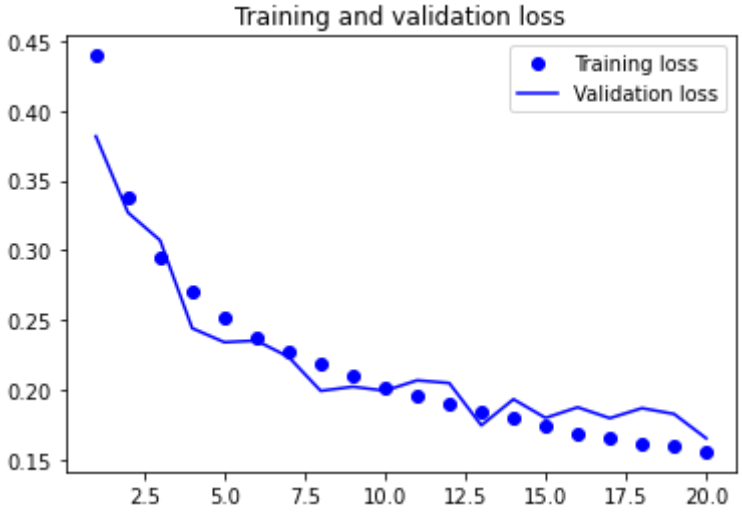
```
['loss', 'accuracy']
```

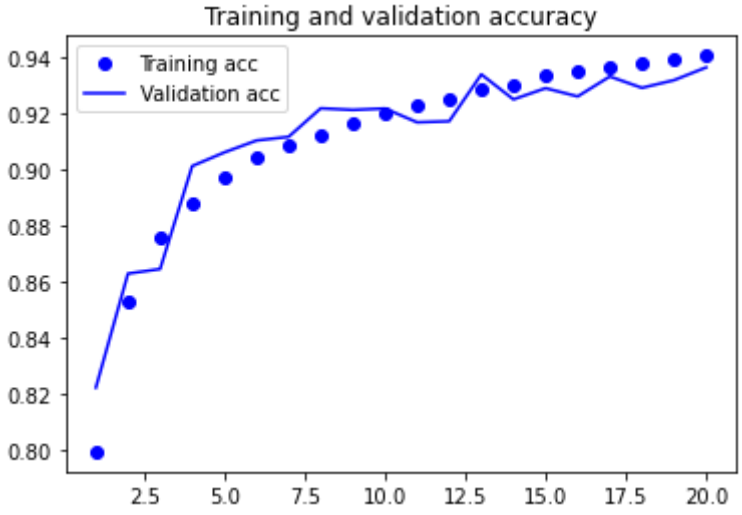
```
# Here the best epoch will be used.



val_loss, val_acc = \
model.evaluate_generator(test_gen,
                         steps=len(df_val))

print('val_loss:', val_loss)
print('val_acc:', val_acc)
```

```
val_loss: 0.16548317670822144
val_acc: 0.9366250038146973
```

```
# display the loss and accuracy curves
import matplotlib.pyplot as plt
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
```





Results and Analysis

Validation and Analysis

```
# make a prediction
predictions = model.predict_generator(test_gen, steps=len(df_val), verbose=1)
```

```
16000/16000 [==============================] - 40s 2ms/step
```

```
predictions.shape
```

```
(16000, 2)
```

```
# This is how to check what index keras has internally assigned to each class.
test_gen.class_indices
```

```
{'a_no_tumor_tissue': 0, 'b_has_tumor_tissue': 1}
```

```
# Put the predictions into a dataframe.
# The columns need to be ordered to match the output of the previous cell

df_preds = pd.DataFrame(predictions, columns=['no_tumor_tissue', 'has_tumor_tissue'])

df_preds.head()
```

	N0_Tumor_Tissue	Has_ Tumor_Tissue
0	0.951828	0.048172
1	0.819612	0.180388
2	0.740446	0.259554
3	0.994519	0.005481
4	0.777217	0.222783

```
# Get the true labels
```

```
y_true = test_gen.classes

# Get the predicted labels as probabilities
y_pred = df_preds['has_tumor_tissue']
```

```
from sklearn.metrics import roc_auc_score

roc_auc_score(y_true, y_pred)
```

```
0.9827305390625
```

```
# Get the labels of the test images.

test_labels = test_gen.classes
test_labels.shape
```

```
(16000,)
```

```
# argmax returns the index of the max value in a row
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))
# Print the label associated with each class
test_gen.class_indices
```

```
{'a_no_tumor_tissue': 0, 'b_has_tumor_tissue': 1}
```

```
from sklearn.metrics import plot_confusion_matrix
```

```
# Delete base_dir and it's sub folders to free up disk space.

shutil.rmtree('base_dir')
#[CREATE A TEST FOLDER DIRECTORY STRUCTURE]

# We will be feeding test images from a folder into predict_generator().
# Keras requires that the path should point to a folder containing images and not
# to the images themselves. That is why we are creating a folder (test_images)
# inside another folder (test_dir).

# test_dir
    # test_images

# create test_dir
test_dir = 'test_dir'
os.mkdir(test_dir)

# create test_images inside test_dir
test_images = os.path.join(test_dir, 'test_images')
os.mkdir(test_images)
# check that the directory we created exists
os.listdir('test_dir')
```

```
['test_images']
```

```
Transfer the test images into image_dir

test_list = os.listdir('../input/histopathologic-cancer-detection/test')

for image in test_list:

    fname = image

    # source path to image
    src = os.path.join('../input/histopathologic-cancer-detection/test', fname)
    # destination path to image
    dst = os.path.join(test_images, fname)
    # copy the image from the source to the destination
    shutil.copyfile(src, dst)
# check that the images are now in the test_images
# Should now be 57458 images in the test_images folder

len(os.listdir('test_dir/test_images'))
```

```
57458
```

```
test_path ='test_dir'


# Here we change the path to point to the test_images folder.

test_gen = datagen.flow_from_directory(test_path,
                                       target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                       batch_size=1,
                                       class_mode='categorical',
                                       shuffle=False)
```

```
Found 57458 images belonging to 1 classes.
```

```
num_test_images = 57458



predictions = model.predict_generator(test_gen, steps=num_test_images, verbose=1)
```

```
57458/57458 [==============================] - 148s 3ms/step
```

```
# Are the number of predictions correct?
# Should be 57458.

len(predictions)
```

```
57458
```

```
# Put the predictions into a dataframe
```

```
df_preds = pd.DataFrame(predictions, columns=['no_tumor_tissue', 'has_tumor_tissue'])

df_preds.head()
```

```
# This outputs the file names in the sequence in which
# the generator processed the test images.
test_filenames = test_gen.filenames

# add the filenames to the dataframe
df_preds['file_names'] = test_filenames

df_preds.head()
```

0	0.000194	0.999806	test_images/00006537328c33e284c973d7b39d340809...
1	0.007043	0.992957	test_images/0000ec92553fda4ce39889f9226ace43ca...
2	0.005235	0.994765	test_images/00024a6dee61f12f7856b0fc6be20bc7a4...
3	0.008798	0.991202	test_images/000253dfaa0be9d0d100283b22284ab2f6...
4	0.991006	0.008994	test_images/000270442cc15af719583a8172c87cd2bd...

```
Create an id column

# A file name now has this format:
# test_images/00006537328c33e284c973d7b39d340809f7271b.tif

# This function will extract the id:
# 00006537328c33e284c973d7b39d340809f7271b


def extract_id(x):

    # split into a list
    a = x.split('/')
    # split into a list
    b = a[1].split('.')
    extracted_id = b[0]

    return extracted_id

df_preds['id'] = df_preds['file_names'].apply(extract_id)
```

0	0.000194	0.999806	test_images/00006537328c33e284c973d7b39d340809...	00006537328c33e284c973d7b39d340809f7271b
1	0.007043	0.992957	test_images/0000ec92553fda4ce39889f9226ace43ca...	0000ec92553fda4ce39889f9226ace43cae3364e
2	0.005235	0.994765	test_images/00024a6dee61f12f7856b0fc6be20bc7a4...	00024a6dee61f12f7856b0fc6be20bc7a48ba3d2
3	0.008798	0.991202	test_images/000253dfaa0be9d0d100283b22284ab2f6...	000253dfaa0be9d0d100283b22284ab2f6b643f6
4	0.991006	0.008994	test_images/000270442cc15af719583a8172c87cd2bd...	000270442cc15af719583a8172c87cd2bd9c7746

```
Get the predicted labels.
# We were asked to predict a probability that the image has tumor tissue
y_pred = df_preds['has_tumor_tissue']

# get the id column
image_id = df_preds['id']
```

Submission

```
submission = pd.DataFrame({'id':image_id,
                           'label':y_pred,
                          }).set_index('id')

submission.to_csv('patch_preds.csv', columns=['label'])
submission.head()
```

00006537328c33e284c973d7b39d340809f7271b	0.999806
0000ec92553fda4ce39889f9226ace43cae3364e	0.992957
00024a6dee61f12f7856b0fc6be20bc7a48ba3d2	0.994765

000253dfaa0be9d0d100283b22284ab2f6b643f6	0.991202
000270442cc15af719583a8172c87cd2bd9c7746	0.008994

```
# Delete the test_dir directory we created to prevent a Kaggle error.
# Kaggle allows a max of 500 files to be saved.

shutil.rmtree('test_dir')
```

Model summary and comparison

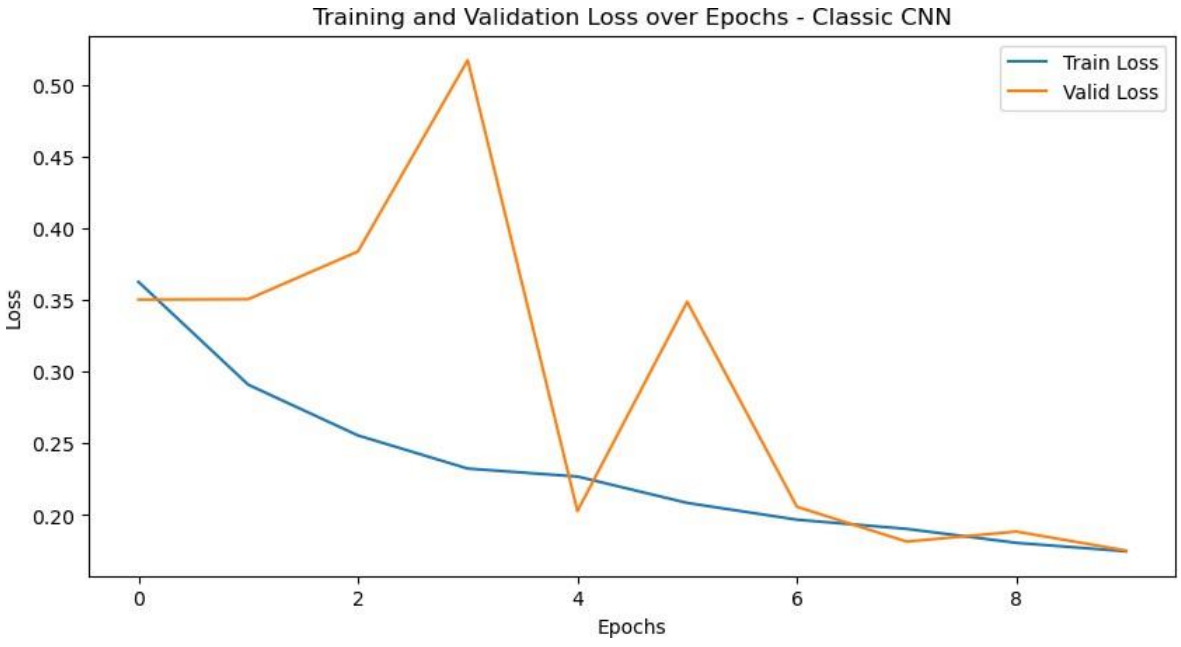
As we have already seen both of our models seem to work quite well on the training data. Now, it's timeto compare them and select the better one for predicting on the test data. First, lets have a closer look at the training and validation losses.
As a reminder, we did a hyperparameter tuning on the learning rate for the first model where we tried out three different values. The best one selected was the following.

```
print("Best learning rate according to hyperparameter search on Classic CNN mode
l: ", learning_rates[best_model_idx])
```

Best learning rate according to hyperparameter search on Classic CNN model: 0.0005

In order to stay within computation time limits we reused this value for the second dense net model. Now, lt's compare the losses of our two models.

```
plot_losses(best_result['train_losses'], best_result['valid_losses'], 'Training
and Validation Loss over Epochs - Classic CNN')
```



```
plot_losses(train_losses_dense, valid_losses_dense, 'Training and Validation Los
s over Epochs - Dense Net')
```

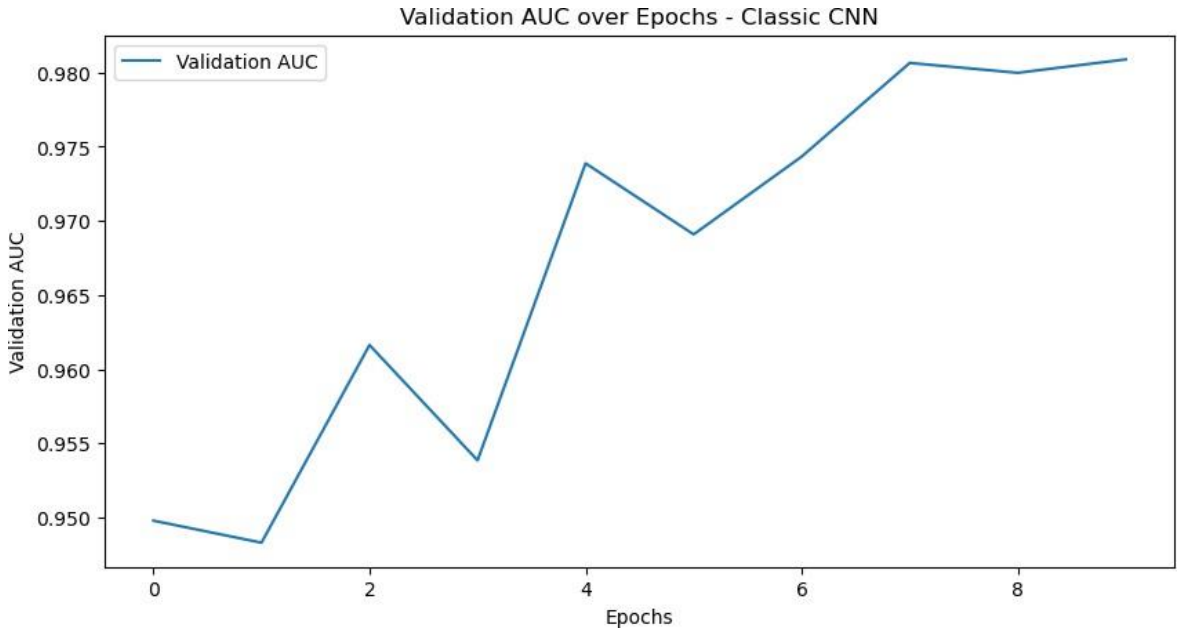
The first plot shows the classic CNN architecture. We observe a decreasing training loss as the epochnumber increases which is what we would expect. The validation loss looks more unsteady but overallalso decreases when reaching the higher epoch numbers. As already mentioned, we could potentially improve here if we would allow for more epochs. But since there is time limit on Kaggle for using the GPU we restrict ourselves a little bit for this project.

The second architecture clearly shows better results. Already after the first epoch there is remarkable difference in training loss, which means the dense network architecture works much better. The same can be said about higher epochs. We constantly get smaller loss values. The reason for the better performance is probably the fact that the connected layer structure of the network can capture more complex patterns in the dataset as compared to the classic CNN structure. Also the fact that the densenetwork is pretrained gives the model a headstart and helps to optimize faster.

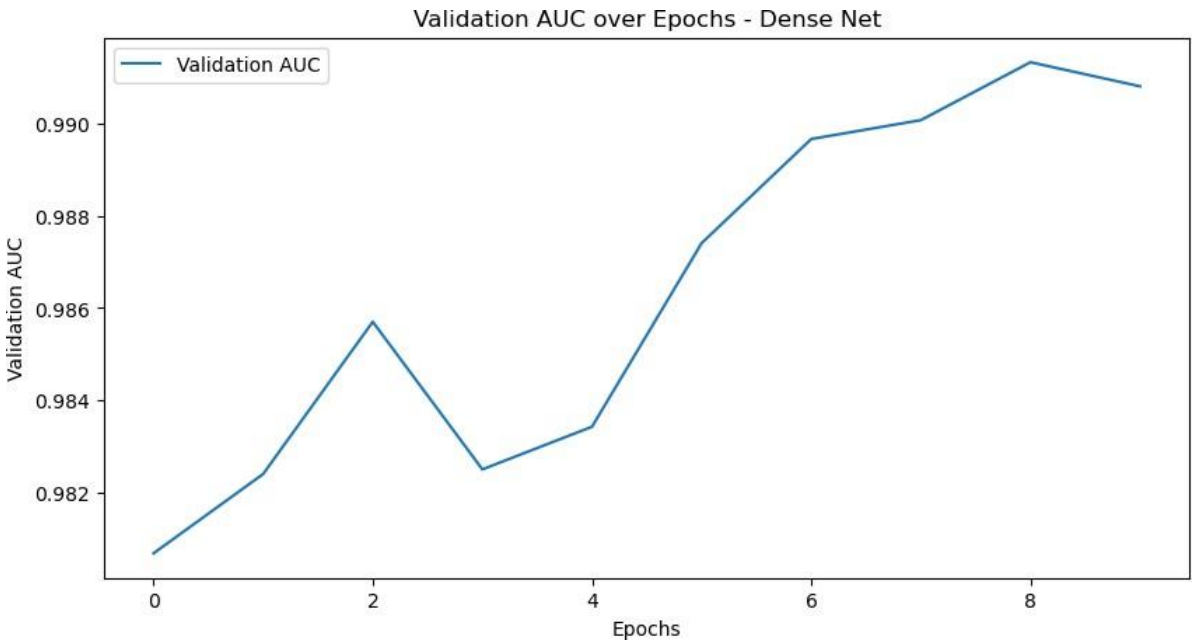
Let's see if our dense model also outperforms when looking at the AUC scores.

In [37]:

```
plot_aucs(best_result['valid_aucs'], 'Validation AUC over Epochs - Classic CNN')
```



```
plot_aucs(valid_aucs_dense, 'Validation AUC over Epochs - Dense Net')
```



Now looking at the AUC scores we can basically conclude the same. The first model shows good resultsand an increasing score over the epochs. Similar to the losses the dense structure starts out at better values than the classic CNN. Even if the increase over epochs is not as significant as with the first model, the absolute values are still slightly higher.

Summarizing the final results everything seems to be pretty easy. But of course, there were some problems during model building and training, e.g. computing time limits, understanding model architectures, and debugging the training process. We will talk about these topics later on in the Learnings and Takeaways section.

Pedicting on test data

We have seen that the dense model performed the best. We will use it now to predict on the test data and create our submission file.

```
clear_memory()
```

In [40]:

```
# turn of gradients
model_dense.eval()
preds = []


# iterate all test images
for batch_i, (data, target) in enumerate(test_loader):
    data, target = data.to(device), target.to(device)
    output = model_dense(data)


    pr = output.detach().cpu().numpy()
    for i in pr:

        preds.append(i)
```

In [41]:

```
# convert probabilities to float
for i in range(len(df_sample_sub)):
    df_sample_sub.label[i] = np.float(df_sample_sub.label[i])
```

In [42]:

```
# create submission file
df_sample_sub.to_csv('submission.csv', index=False)
```

Confusion Matrix

```
import seaborn as sns
import matplotlib.pyplot as plt

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```

Conclusion

Result Summary

As we have already seen in the previous section, the dense network performed better than the CNNnetwork because of its ability to capture more complex structures in our images. However, both modelsdid a good job on the training data. Looking at the performance of the dense network on the test data we saw that we obtain a really good final score.

Learnings and Takeaways

Working with neural networks requires a different style of work compared to simpler supervised learningtechniques like regression analysis. Getting results from model training might last several hours. Therefore, it is really important to work clean and precise so that you avoid having many training iterations. One aspect that might be underestimated but helped me a lot during this project is the usageof print statements in the code. This way you can debug your code better, inspect intermediate results and find errors more easily. Besides that aspect, image data in particalur requires quite different preprocessing steps compared to e.g. textual data. Techniques like image augmentation can really improve your model if used correctly.

Another problem I encountered during my work was the memory limitations of the GPU on Kaggle. On way to solve this issue was to reduce the batch size from 256 down to 128 which leads to less RAM consumptions during training and evaluating. Additionaly, I had to call the garbage collector from time totime in order to free unused memory.

What didn't work

Overall, everything worked quite well. However, it would have been great to work with more epochs inorder to get more precise results. The limited GPU memory really was a problem in this project since image data requires so much disk space. It took quite a few iterations to find suitable values for the number of included images, the batch size or number of epochs while still getting meaningful results.

Possible improvements

One possible improvement would be to include the whole image set instead of just a smaller amount due to memory issues. Also using more epochs would possibly enhance model performance. It might bealso interesting to see how much better the model would perform with a balanced image set compared to the original unbalanced one.

If we had enough computing power, we could also expand our hyperparameter tuning by including moreparameters and a wider value range.