DTSA 0000 - Introduction to Deep Learning

University of Colorado, Boulder

Master of Science in Data Science

Histopathologic Cancer Detection on Kaggle

1 Introduction

In this competition, we must create an algorithm to identify metastatic cancer in small image patches taken from larger digital pathology scans. The data for this competition is a slightly modified version of the PatchCamelyon (PCam) benchmark dataset (the original PCam dataset contains duplicate images due to its probabilistic sampling, however, the version presented on Kaggle does not contain duplicates).

In this dataset, we are provided with a large number of small pathology images to classify. Files are named with an image id. The train_labels.csv file provides the ground truth for the images in the train folder. We are predicting the labels for the images in the test folder. 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 fullyconvolutional models that do not use zero-padding, to ensure consistent behavior when applied to a whole-slide image.

2 Exploratory Data Analysis(EDA)

2.1 Data Loading and Libraries Import

```
import numpy as np
import pandas as pd
import os
from glob import glob
from random import shuffle
import cv2
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, GlobalMaxPooling2D,
   GlobalAveragePooling2D, Flatten, Concatenate, Dropout, Dense
from keras.models import Model
from keras.applications.nasnet import NASNetMobile,
   preprocess_input
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.losses import binary_crossentropy

# Read the training labels
df_train = pd.read_csv("../input/histopathologic-cancer-
   detection/train_labels.csv")
```

```
id_label_map = {k: v for k, v in zip(df_train.id.values,
   df_train.label.values)}

# Function to extract ID from file path
def get_id_from_file_path(file_path):
    return file_path.split(os.path.sep)[-1].replace('.tif', '')

# Paths to training and test images
labeled_files = glob('../input/histopathologic-cancer-detection
   /train/*.tif')
test_files = glob('../input/histopathologic-cancer-detection/
   test/*.tif')
```

2.2 EDA

Since we are dealing with images, there are different aspects to consider EDA, most of which seem to be unnecessary in our case as we'll see that our base starter model already achieves excellent performance. For the sake of inclusiveness and as a reference for future work, some of the following can be considered during images EDA:

- **Image Dimensions Analysis**: Check the dimensions of the images. Are they all the same size or do they vary? If there's variation, understanding the range and distribution of these dimensions can be important, especially if we plan to resize them for model input.
- **Pixel Intensity Distribution**: Analyze the distribution of pixel intensities across the dataset. This can give insights into the contrast and brightness of our images, which might affect the model's ability to learn from them.
- **Visual Examination of Images**: Manually examine a subset of images from each class (positive and negative). Look for noticeable differences and similarities. This can help in understanding what features might be important for classification.
- **Sample Variability**: Check the variability within each class. Are all positive images similar to each other, or is there a lot of variation? The same goes for negative samples. High intraclass variability can make the classification task more challenging.
- **Check for Class Imbalance**: Beyond the basic distribution, assess the degree of class imbalance. Severe imbalances might require special training techniques like weighted loss functions.
- **Data Augmentation Potential**: Explore how different data augmentation techniques (like rotation, flipping, scaling, etc.) affect the images. This is especially important to know before we augment our data for training.

```
label_counts = df_train['label'].value_counts()
print("Counts of labels:\n", label_counts)
```

We'll get 130908 counts of 0 and 89117 counts of 1. There's a slight imbalance in our training data labels. We ignore them for now.(Later as an improvement suggestion for future work we can resample our dataset in a balanced way so it contains the same counts for our both labels) Our images seem to be of the same shape (96x96). There are some basic augmentations such as flips, rotations, sharpening, and adding blur that can be added in a future work to see if there are improvements in the results.

3 Data Preparation and Helper Functions

This generator function is used for feeding images to the model in batches. It shuffles the files, processes them in chunks (using chunker), and applies preprocessing.

```
# Splitting data into training and validation sets
train, val = train_test_split(labeled_files, test_size=0.1,
   random_state=101010)

# Data generator
def data_gen(list_files, id_label_map, batch_size):
    while True:
        shuffle(list_files)
        for batch in chunker(list_files, batch_size):
            X = [cv2.imread(x) for x in batch]
            Y = [id_label_map[get_id_from_file_path(x)] for x
               in batch]
            X = [preprocess_input(x) for x in X]
            yield np.array(X), np.array(Y)

# Helper function to divide data into chunks
def chunker(seq, size):
    return (seq[pos:pos + size] for pos in range(0, len(seq),
       size))
```

4 Model Selection and Training

There are many ways to experiment with different types of neural networks such as CNNs in the case of image classification. WE can also benefit from the plethora of already_trained models by setting them as base models and add some different layers to add some complexity such as maxpooling layers, and dropout layers(for severing some connections between nuerons). I decided to try the NasNetMobile model that apparently was a successful model being used by other competitors as well[2]. Since we are doing a binary classification,we can use the sigmoid activation function that maps any input value to a range between 0 and 1. (Can be interpreted as the probabilty of the image belonging to the positive class) Note that are different values that can be put as for the input layer for a NasNetMobile, but in our case we can't go smaller than 32x32.[3] Batch_size is a parameter that can be adjusted and lowered if we run into memory allocation issues, but in our case 32 worked fine.

```
# Model creation function
def get_model_classif_nasnet():
    inputs = Input((96, 96, 3))
    base_model = NASNetMobile(include_top=False, input_shape
       =(96, 96, 3))
    x = base_model(inputs)
    out1 = GlobalMaxPooling2D()(x)
    out2 = GlobalAveragePooling2D()(x)
    out = Concatenate(axis=-1)([out1, out2])
    out = Dropout(0.5)(out)
    out = Dense(1, activation="sigmoid")(out)
    model = Model(inputs, out)
    model.compile(optimizer=Adam(0.0001), loss=
       binary_crossentropy, metrics=['accuracy'])
    model.summary()
    return model

# Create and compile the model
model = get_model_classif_nasnet()

# Training parameters
batch_size = 32
h5_path = "model.h5"
checkpoint = ModelCheckpoint(h5_path, monitor='val_accuracy',
   verbose=1, save_best_only=True, mode='max')

# Train the model
history = model.fit(
    data_gen(train, id_label_map, batch_size),
    validation_data=data_gen(val, id_label_map, batch_size),
    epochs=2, verbose=1,
    callbacks=[checkpoint],
    steps_per_epoch=len(train) // batch_size,
    validation_steps=len(val) // batch_size
)
```

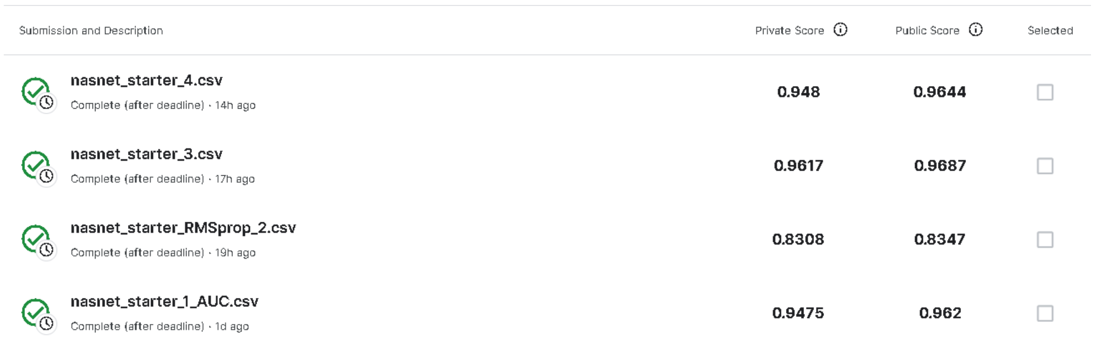
5 Predicting the Labels for Test Images and Submission File

```
# Load best weights and make predictions
model.load_weights(h5_path)

preds = []
ids = []
for batch in chunker(test_files, batch_size):
    X = [preprocess_input(cv2.imread(x)) for x in batch]
    ids_batch = [get_id_from_file_path(x) for x in batch]
    X = np.array(X)
    preds_batch = ((model.predict(X).ravel()*model.predict(X[:,
        ::-1, :, :]).ravel()*model.predict(X[:, ::-1, ::-1, :])
       .ravel()*model.predict(X[:, :, ::-1, :]).ravel())**0.25)
       .tolist()
    preds += preds_batch
    ids += ids_batch
df = pd.DataFrame({'id':ids, 'label':preds})
df.to_csv("nasnet_starter_1.csv", index=False)
df.head()
```

6 Results and Analysis

I submitted 4 slightly different models and the following are the results. Models 1,3 and 4 use Adam optimizer and model 2 uses RMSProp optimizer which we can see may not be ideal in our case. Models 1 and 2 have 50 percent dropout rate and models 3 and 4 use 80 percent dropout rate. Models 1,2 and 3 use a learning rate of 1e-4 and model 4 uses a learning rate of 5e-5.



Submission and Description	Private Score	Public Score	Selected
nasnet_starter_4.csv Complete (after deadline) · 14h ago	0.948	0.9644	☐
nasnet_starter_3.csv Complete (after deadline) · 17h ago	0.9617	0.9687	☐
nasnet_starter_RMSprop_2.csv Complete (after deadline) · 19h ago	0.8308	0.8347	☐
nasnet_starter_1_AUC.csv Complete (after deadline) · 1d ago	0.9475	0.962	☐

7 Conclusion and Future Work

As can be seen from the public and private scores, all models other than 2 (which used RMSProp as its optimizer) performed well at around 0.94-0.96. The main reason for this seems to be related to the nature of our dataset which was nearly balanced and also the NasNetMobile model which without too much modification can be used as a base input layer. A couple of maxpooling layers were also added for dimensional reduction alongside the dropout layer for reducing the complexity and overfitting of the model. I experimented a bit with a few parameters such as the dropout rate, learning rate, and optimizers. Since the scores were already decent with not much complexity, I didn't add basic and sophisticated augmentations in my data generator class. In general, it usually helps a lot using some types of augmentations and resampling (in the case our dataset isn't balanced) for image classification tasks that can be tried in future endeavors.

8 References

[1]https://www.kaggle.com/c/histopathologic-cancer-detection
[2] https://www.kaggle.com/code/CVxTz/cnn-starter-nasnet-mobile-0-9709-lb
[3] https://keras.io/api/applications/nasnet/