ELISA GÓMEZ DE LOPE
POSGRADUATE ON DATA SCIENCE AND BIG DATA
SEPTEMBER 2020

CAPSTONE PROJECT

Histopathologic Cancer Detection challenge on Kaggle.

This is a Computer Vision problem where we are tasked with the detection of cancer by identifying metastatic tissue in histopathologic scans of lymph nodes using Deep Learning.

CHALLENGE DESCRIPTION

The goal is to create an algorithm that identifies metastatic cancer in small image patches taken from larger digital pathology scans.

The data was obtained from a (completed) kaggle competition (kaggle link) and 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).

PCam is highly interesting for both its size, simplicity to get started on, and approachability. In the authors' words:

[PCam] packs the clinically-relevant task of metastasis detection into a straight-forward binary image classification task, akin to CIFAR-10 and MNIST. Models can easily be trained on a single GPU in a couple hours, and achieve competitive scores in the Camelyon16 tasks of tumor detection and whole-slide image diagnosis. Furthermore, the balance between task-difficulty and tractability makes it a prime suspect for fundamental machine learning research on topics as active learning, model uncertainty, and explainability.

The official Pcam dataset from GitHub. The data is provided under the CC0 License, following the license of Camelyon16.

DATA DESCRIPTION

The dataset contains a large and balanced number of small pathology images to classify. The images are split into train (220,025 images) and test (57,458) set. Each image file is named with an image id and there is a csv file (*train_labels.csv*) that provides the ground truth for the images in the train folder. The labels for the images in the test folder are to be predicted. 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.

PROJECT STRUCTURE

As a first step, I organized the structure of directories and subdirectories where I would store the input and output data.

The project directories are *data*, *models* and *submission*. The *data* folder contains subfolders: *train* and *test*. In *train* you can find one subfolder per class, in this case there is class *0* (no tumor) and class *1* (has tumor), containing the corresponding images. A piece of code was developed in order to create such subdirectories and move the images to one or other subfolder, since initially all training images were downloaded mixed in the same folder.

Inside *test* directory there is another subdirectory (as if there was only one class: *images*). This structure of subdirectories was necessary for the method utilized to load images.

The images were downloaded and stored as described in data folder. The input data also included 2 csv files that were kept inside data directory: sample_submission.csv and train_labels.csv.

The *models* folder contains the code in Jupyter notebooks and several .h5 files that contain information (mainly the weights) of the different models that were built and trained during the development of this project. They served as checkpoints to which I could always address if I needed a quick implementation without training.

CODE STRUCTURE

The goal is to design a code structure that allows to train and tune models easily, being able to modify the value of hyperparameters and parameters in a straight-forward implementation. In this regard, the code is thought to be modular, breaking the whole project into small pieces of code that are easier to program and modify independently. Following this purpose, I have worked with Jupyter notebooks.

One of the challenges of machine learning projects, and so of this project, is to find a good bias-variance trade off. In this regard, a subset of the training set was separated and used for measuring performance and selecting the best method and parameters. This subset will be called validation set for the sake of clarification and the training set will refer to the remaining of training set once the validation set has been subtracted. The error on training set will be used to diagnose bias if it is high (underfitting, model is too simple) and dev set error will diagnose variance if it is very high compared to the training error (model is overfitting the training set and fails to generalize).

DEEP LEARNING FRAMEWORK: KERAS.

The challenge has been tackled with convolutional neural networks (CNN). CNNs models have been developed within Keras framework, as it is built on top of TensorFlow 2.0 and is easy, intuitive and widely used among deep learning community.

The code developed for this project can be divided into different parts. First part corresponds to packages imports and dataset loading (train and test images together with training labels' and sample submission's csv files). There is a quick exploration of the quantity of images on each set and ratio of positive/negative labels. Some snippets of code are devoted to show some actual images.

The images were loaded into the tensors by using the ImageDataGenerator method *flow_from_directory ()*. Training and validation sets were established on an 85-15 (notebook *1_CNN_customized.ipynb*) and 80-20 (notebook *2_CNN_resnet50.ipynb*) proportions. Additionally, the ImageDataGenerator allows to generate batches of tensor image data with real-time data augmentation (flips, brightness, rotations, zoom, rescale, …). Some transformations were tried in certain implementations (only in train and validation images), but most importantly the rescale 1/255 was always applied (also in test images). Indeed, to my surprise, the application of some of the transformations like brightness and zoom did not necessarily improve the performance.

As mentioned, the type of algorithm selected to model this dataset was Convolutional Neural Networks implemented with Keras. In one of the notebooks (*1_CNN_customized.ipynb*) the model is built by myself based on a build-run-change-repeat cycle. I was also inspired by popular implementations that were used in other Kaggle competitions. Different architectures were explored, from a straight-forward 2 convolutional layers with 'same' padding, relu and 16-32 filters respectively followed by a dense (128) + dense(1) layer and a final sigmoid up to a 16-32-64-128 filters' convolutional layers followed by batchnormalization and relu activations and final dense (128) + dense(1) and sigmoid. In further implementations maxpooling layers and droput were included. Regarding the epochs, I tried 8 and 15 epochs for the different models.

In the second notebook (*2_CNN_resnet50.ipynb*) I downloaded a pre-trained ResNet50 model and added a dense-relu layer followed by a final dense-sigmoid layer (with batchnormalization and droput respectively). The concept I wanted to try was transfer learning. Therefore, I freezed the very first layers of the model and trained the rest so that the transfer learning would be effective and the weights would adapt to this tumor-finding challenge.

For both notebooks, the models included ReduceLROnPlateau() keras callback method, which reduces the learning rate when a metric has stopped improving. In the notebook *2_CNN_resnet50.ipynb* I also included other callback: EarlyStopping(). This method makes the training stop when a monitored metric (validation loss again) has stopped improving.

All models were compiled utilizing Adam optimizer (0.01 initial learning rate), binary cross entropy as a loss function and accuracy as a performance metric.

The assessment of the models was based on the performance on the validation set. I experienced several overfitting problems for the models with higher number of layers and epochs together with lot of data augmentations applied on images. The performance metric measured was the accuracy on validation set, since the dataset was balanced and I thought it would be trustworthy.

Several submissions were uploaded to the Kaggle competition and so externally evaluated on area under the ROC curve between the predicted probability and the observed target.

RESULTS AND CONCLUSIONS

The main motivation of this project was to develop a first approach to deep learning and build models based on neural network architectures. As a beginner and an autonomous learner, I started from almost scratch and after some months working on it, I have the feeling I have learnt a lot and I am ready to deepen further in this field.

Also, I must say my resources were limited, and I think this had a huge impact on the number of things I could try and execute. To run the trainings I evaluated several possibilities such as google colab GPU. The problem was always uploading the +10GB of data to an online or cloud platform. My network resources were not powerful enough to upload to a git repository the data to train, so I could not access it from google colaboratory notebooks. Consequently, the models were trained on my personal computer (Intel i7-3610QM CPU, 2.3GHz, 4 Cores and 8GB RAM). Some trainings took >30h of execution. Therefore, in the end there were multiple parameters and architecture I could try, but I was limited on time and resources. Besides these limitations, I have devoted a lot of effort to this project and gained good hands-on knowledge to kick off in this flourishing field of machine learning. I believe this is the take-home message from this capstone project.

However, there is still a lot of room for improvement. From a more pragmatic and realistic point of view, the results were not very good. I suffered from the typical ML pathologies: the models I developed were most of the time overfitting the training and validation dataset. Those other times, they were underfitting the data and having big bias in the results. As a consequence of the overfitting, the best score by Kaggle was 0.58 despite reaching validation accuracy of 0.91. The 'fortunate' submission corresponds to a customized CNN that I trained for 8 epochs using Adam optimizer (learning rate = 0.01). The model contained the following layers:

```
my_kernel_size = (3,3)
my_pool_size= (2,2)
dropout_conv = 0.3
dropout_dense = 0.3


model = Sequential()
model.add(Conv2D(filters = 16, my_kernel_size, activation = 'relu', input
_shape = (96, 96, 3)))
model.add(Conv2D(filters = 16, my_kernel_size, activation = 'relu'))
model.add(Conv2D(filters = 16, kernel_size = my_kernel_size, padding = 's
ame', use_bias=False))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = my_pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(filters = 32, my_kernel_size, activation ='relu'))
model.add(Conv2D(filters = 32, my_kernel_size, activation ='relu'))
model.add(Conv2D(filters = 32, kernel_size = my_kernel_size, padding = 's
ame', use_bias=False))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = my_pool_size))
model.add(Dropout(dropout_conv))
model.add(Conv2D(filters = 64, my_kernel_size, activation ='relu'))
model.add(Conv2D(filters = 64, my_kernel_size, activation ='relu'))
model.add(Conv2D(filters = 64, kernel_size = my_kernel_size, padding = 's
ame', use_bias=False))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size = my_pool_size))
model.add(Dropout(dropout_conv))


model.add(Flatten())
model.add(Dense(128, use_bias=False))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dropout(dropout_dense))
model.add(Dense(1, activation = 'sigmoid'))
```

OTHER RESOURCES

You can find the 2 notebooks that were mentioned in this document in github together with some of the submissions: https://github.com/elisagdelope/hist_cancer_detection .