Histopathological cancer detection

Detecting cancer in a patch of a histological slide, using a convolutional neural network.

# Milestone 1: Baseline Model

## 1. The analysis of data and construction of features

Our data consists of unique patches from larger histological slides, from the PatchCamelyon (PCam) dataset. This is 220.025 labeled images in total, with a 50/50 split of positive and negative samples.

These patches were mined from the Camelyon16 dataset. This dataset contains 400 hematoxylin (which binds to DNA, coloring it) and eosin (which colors cytoplasm) (H&E) stained Whole Slide Images (WSI) of sentinel lymph node tissue. Slides were collected from two different centers and digitised at x40 objective. Data was oversampled at 10x. To prevent selecting background patches, slides are converted to Hue Saturation Value color channels (HSV), blurred, and patches filtered out if maximum pixel saturation was below 0.07 (which was validated to not throw out tumor data in the training set). The patch-based dataset is sampled by iteratively choosing a WSI and selecting a positive or negative patch with probability *p*. Patches are rejected following a stochastic hard-negative mining scheme with a small CNN, and *p* is adjusted to retain a balance close to 50/50.\*

The original PCam dataset consists of 96x96px images. 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. Our data are these center patches. Training data was split into 70% training data and 30% validation data.

Resized image data was synchronously stored in one list and labels in the other, resulting in two arrays for each list.

Convolutional Neural Networks (CNN) generally provide the best results for image prediction, this is why this type of model is chosen. A disadvantage of this model is it’s lack of interpretability. The features the model learns are hard to define and can only be selected manually, by specifying the weights for a filter by hand.

\* citation: <https://github.com/basveeling/pcam>

## 2. The inputs and structure of the model

Because the input data are images, a convolutional neural network was used as a model. Our baseline model

The input data is presented as tiff images of 32 x 32 pixels with 3 color channels. The collection of tiff images was converted to a four-dimensional Numpy array.

The dimensions of this Numpy array is as follows:

Dimension 0: The different images

Dimension 1: The rows of the images

Dimension 2: The columns of the images

Dimension 3: The three colour channels

The network architecture of the baseline model is as follows:

* Convolutional layer 1
* Maxpool layer 1
* Convolutional layer 2
* Maxpool layer 2
* Flatten layer
* Hidden layer 1
* Output layer

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input\_shape=(32, 32, 3)))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))

model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())

model.add(layers.Dense(64, activation='relu'))

model.add(layers.Dense(2, activation='softmax'))

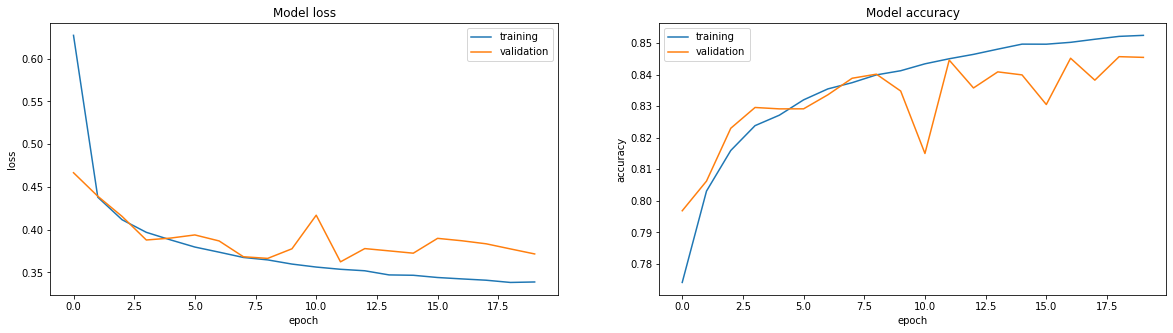
train\_and\_evaluate(model, train\_images, train\_labels, val\_images, val\_labels)

## 3. The training methodology and results

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

For the baseline model, we used categorical cross entropy as loss function and Adam as optimizer for gradient descent. The learning rate for the adam optimizer was set to the default value of 0.001. We trained the CNN for 20 epochs.

## 4. Comparison to previous models and analysis of results



*Graph 1: Model loss and accuracy for the baseline model.*

Our baseline model performed surprisingly well. The final validation accuracy was 84.5%, which is far above chance (50%).

For a future model, we expect that we can increase the rate by which the accuracy increases per epoch by normalizing the input data and/or by increasing the learning rate.

Notes for 1st feedback session:

* Notebook crashes and empties RAM every time the model finishes training. To make the model run again after this, the entire notebook needs to be restarted and run from the start. Why does this happen and is there a way to fix this?
* Validation accuracy seems to fluctuate a lot between epochs. What could be the source of these fluctuations and how can we limit them?

Possible tweaks:

* Changing learning rate
* Increasing number of epochs
* Normalizing input
* Adding dropout layers
* Deepening the model
* Adding batch normalization layers
* Data augmentations