

Medical Image Multi Label Classification

Student: Lucian Istrati - 511 - Data Science

lucian.istrati@s.unibuc.ro

lucian.istrati@my.fmi.unibuc.ro

Introduction

The task at hand is to be able to perform three classification for any scan image. Each image has 3 binary labels associated with it.

Dataset Analysis

The images are split across the datasets in the following manner:

- 12.000 images in the training set;
- 3.000 images in the validation set;
- 5.000 images in the testing set.

In order to load the data we call:

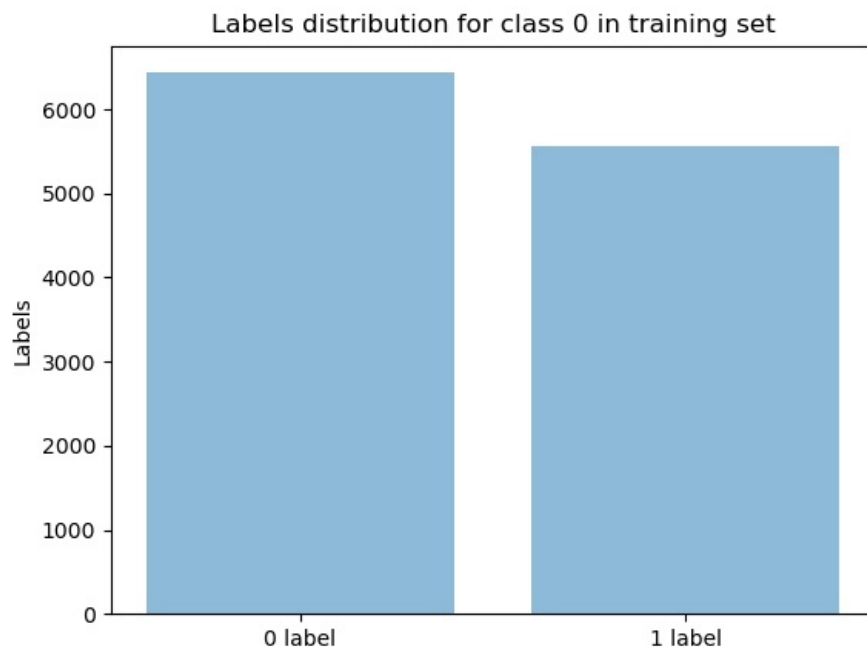
- `load_data()` from `main.py` in order to load the images;
- `load_labels()` from `main.py` to load the labels for the images from training and validations sets;

Some observations about the data:

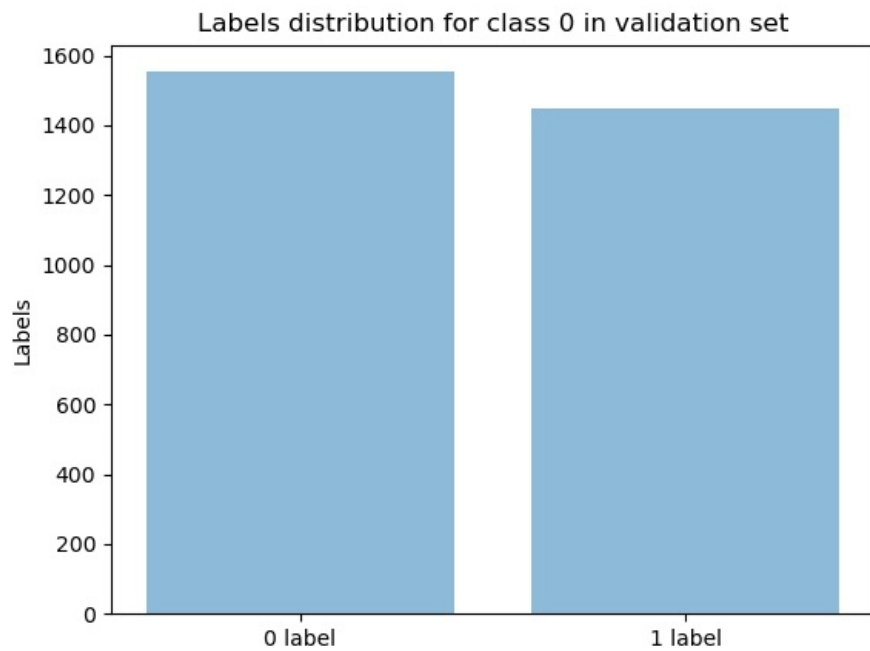
- images are 64 by 64 by 3, but since they are grayscale we can use a 64 by 64 by 3 or 64 by 64 by 1 images for models training.
- at a first close look through the data many images that were totally black were observed, because of this I decided to investigate how many of these black images were there in total. In total, 5000 black images were found. Since there were this many black ones I also searched for other possible duplicates in the dataset between normal images, but none were found. Trainings were done both without most of these 5000 black images, except for one with a 0 label on each task and with a 1 label on each task. Also, this quarter of the dataset was labelled with both zeroes and ones.
- one quarter of the images are black (4985 out of 15.000). A removal of all black images except for one image was performed.

Labels distribution

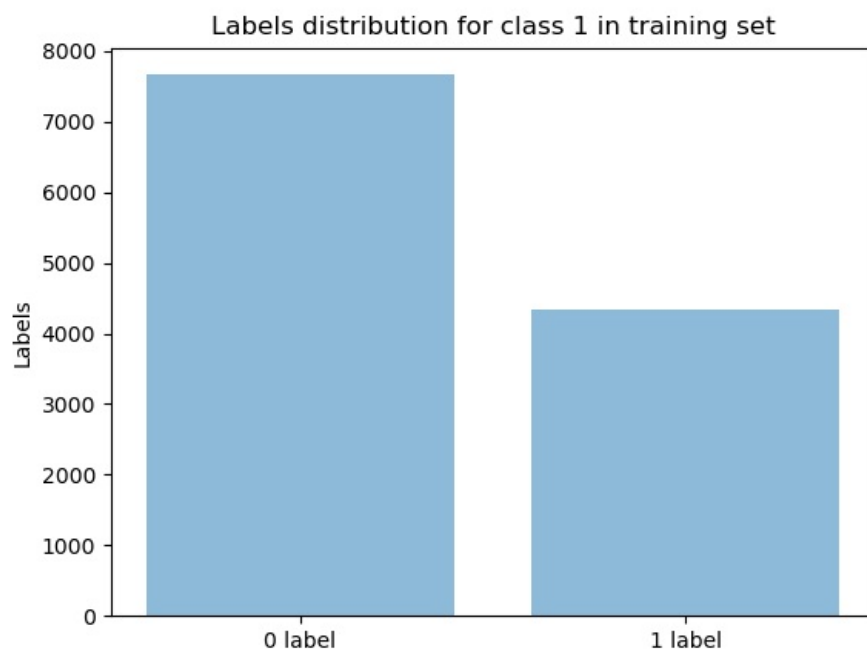
- Training set first task:



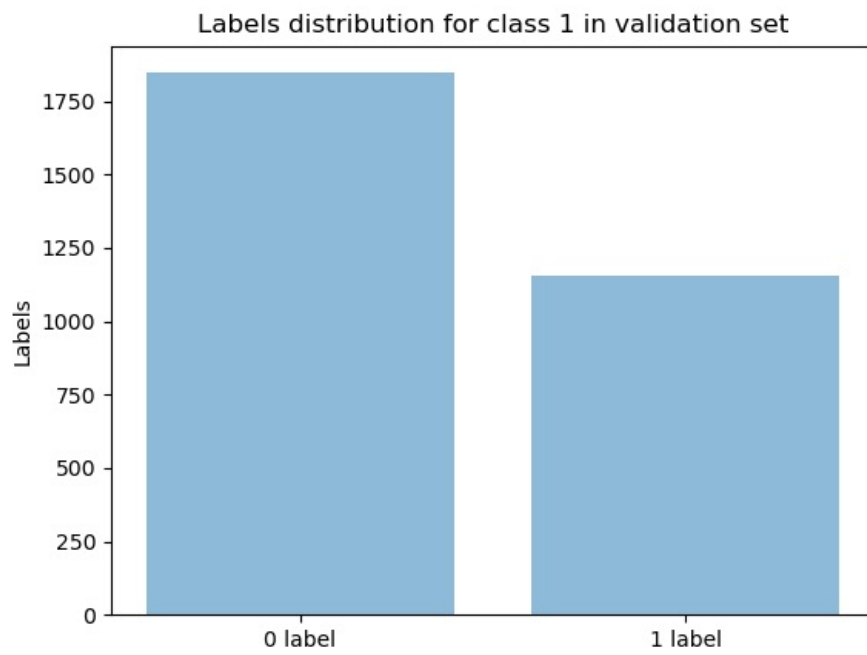
-



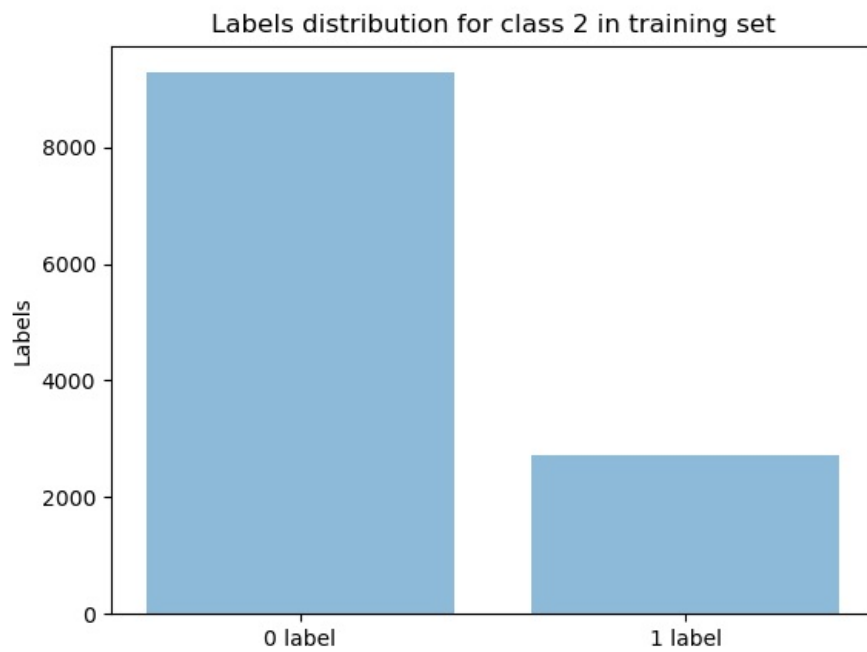
- Validation set first task:
- Training set second task:



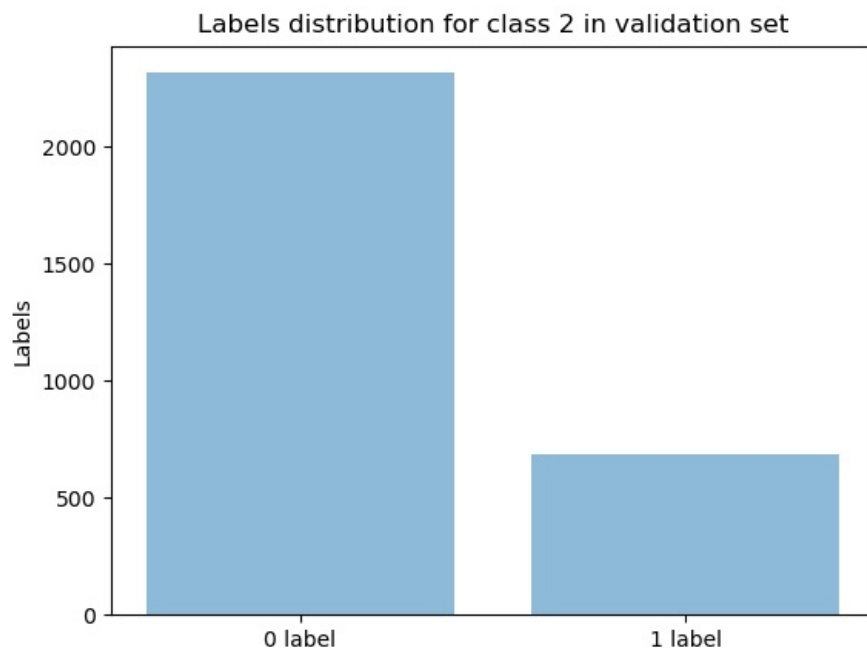
-
- Validation set second task:



-
- Training set third task:



-
- Validation set third task:



-

1. Convolutional neural network model. Five architectures were tried out for this type of model as this one proved to yield the best results from the start out of the 3 types of models:

conv2d_input	input:	[(None, 64, 64, 1)]
InputLayer	output:	[(None, 64, 64, 1)]



conv2d	input:	(None, 64, 64, 1)
Conv2D	output:	(None, 62, 62, 256)



max_pooling2d	input:	(None, 62, 62, 256)
MaxPooling2D	output:	(None, 20, 20, 256)



flatten_1	input:	(None, 20, 20, 256)
Flatten	output:	(None, 102400)



dense_1	input:	(None, 102400)
Dense	output:	(None, 256)



dense_2	input:	(None, 256)
Dense	output:	(None, 2)

• 1.a

input_1	input:	[(None, 64, 64, 3)]
InputLayer	output:	[(None, 64, 64, 3)]



conv2d_1	input:	(None, 64, 64, 3)
Conv2D	output:	(None, 64, 64, 25)



max_pooling2d_1	input:	(None, 64, 64, 25)
MaxPooling2D	output:	(None, 32, 32, 25)



conv2d_2	input:	(None, 32, 32, 25)
Conv2D	output:	(None, 16, 16, 50)



max_pooling2d_2	input:	(None, 16, 16, 50)
MaxPooling2D	output:	(None, 8, 8, 50)



batch_normalization	input:	(None, 8, 8, 50)
BatchNormalization	output:	(None, 8, 8, 50)



conv2d_3	input:	(None, 8, 8, 50)
Conv2D	output:	(None, 4, 4, 70)



output	output:	(None, 4, 4, 70)
--------	---------	------------------

max_pooling2d_3	input:	(None, 4, 4, 70)
MaxPooling2D	output:	(None, 2, 2, 70)



batch_normalization_1	input:	(None, 2, 2, 70)
BatchNormalization	output:	(None, 2, 2, 70)



flatten_2	input:	(None, 2, 2, 70)
Flatten	output:	(None, 280)



dense_3	input:	(None, 280)
Dense	output:	(None, 100)



dense_4	input:	(None, 100)
Dense	output:	(None, 100)



dropout	input:	(None, 100)
Dropout	output:	(None, 100)



dense_5	input:	(None, 100)
Dense	output:	(None, 2)

• 1.b

conv2d_4_input	input:	[(None, 64, 64, 3)]
InputLayer	output:	[(None, 64, 64, 3)]



conv2d_4	input:	(None, 64, 64, 3)
Conv2D	output:	(None, 64, 64, 50)



conv2d_5	input:	(None, 64, 64, 50)
Conv2D	output:	(None, 64, 64, 75)



max_pooling2d_4	input:	(None, 64, 64, 75)
MaxPooling2D	output:	(None, 32, 32, 75)



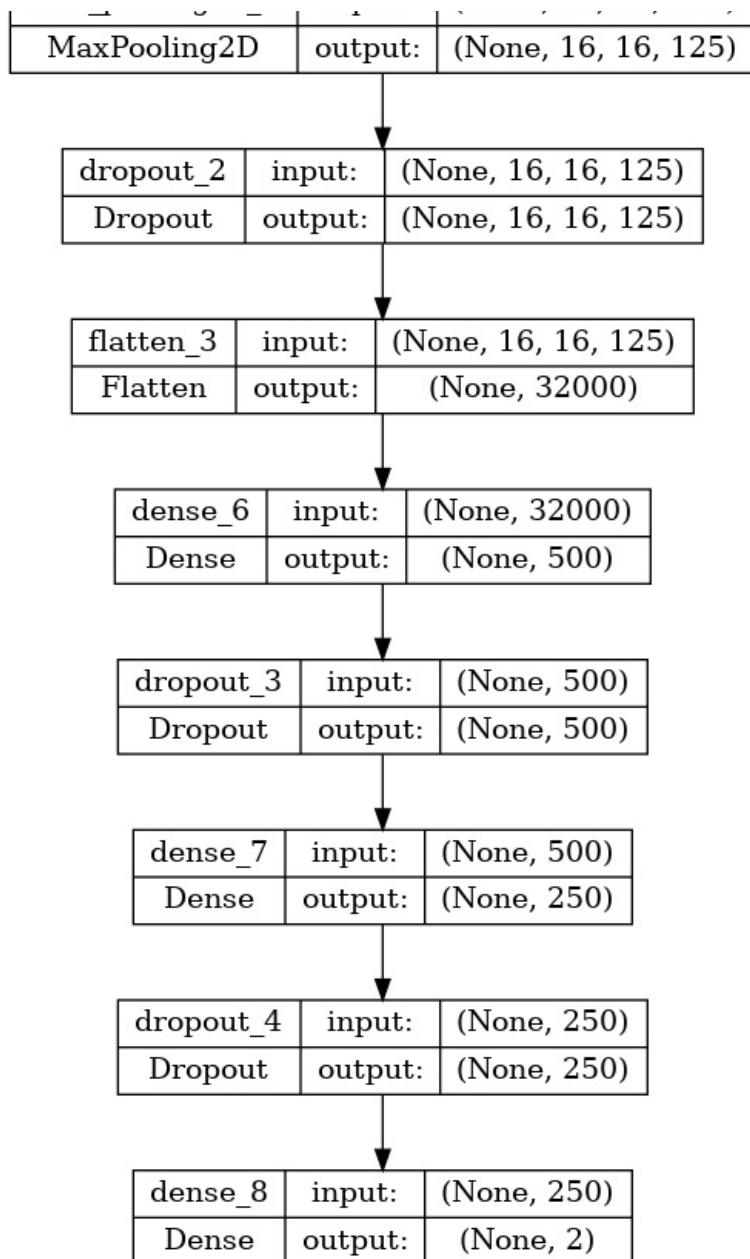
dropout_1	input:	(None, 32, 32, 75)
Dropout	output:	(None, 32, 32, 75)



conv2d_6	input:	(None, 32, 32, 75)
Conv2D	output:	(None, 32, 32, 125)



max_pooling2d_5	input:	(None, 32, 32, 125)
-----------------	--------	---------------------



- 1.c

conv2d_7_input	input:	[(None, 64, 64, 3)]
InputLayer	output:	[(None, 64, 64, 3)]



conv2d_7	input:	(None, 64, 64, 3)
Conv2D	output:	(None, 60, 60, 32)



max_pooling2d_6	input:	(None, 60, 60, 32)
MaxPooling2D	output:	(None, 12, 12, 32)



conv2d_8	input:	(None, 12, 12, 32)
Conv2D	output:	(None, 8, 8, 64)



max_pooling2d_7	input:	(None, 8, 8, 64)
MaxPooling2D	output:	(None, 1, 1, 64)



flatten_4	input:	(None, 1, 1, 64)
Flatten	output:	(None, 64)



dense_9	input:	(None, 64)
Dense	output:	(None, 1024)



dropout_5	input:	(None, 1024)
Dropout	output:	(None, 1024)



dense_10	input:	(None, 1024)
Dense	output:	(None, 2)

• 1.d

conv2d_9_input	input:	[(None, 64, 64, 3)]
InputLayer	output:	[(None, 64, 64, 3)]



conv2d_9	input:	(None, 64, 64, 3)
Conv2D	output:	(None, 64, 64, 32)



conv2d_10	input:	(None, 64, 64, 32)
Conv2D	output:	(None, 62, 62, 32)



max_pooling2d_8	input:	(None, 62, 62, 32)
MaxPooling2D	output:	(None, 31, 31, 32)



conv2d_11 input: (None, 31, 31, 32) Conv2D output: (None, 31, 31, 32)

dropout_6	input:	(None, 31, 31, 32)
Dropout	output:	(None, 31, 31, 32)



conv2d_11	input:	(None, 31, 31, 32)
Conv2D	output:	(None, 31, 31, 64)



conv2d_12	input:	(None, 31, 31, 64)
Conv2D	output:	(None, 29, 29, 64)



max_pooling2d_9	input:	(None, 29, 29, 64)
MaxPooling2D	output:	(None, 14, 14, 64)



dropout_7	input:	(None, 14, 14, 64)
Dropout	output:	(None, 14, 14, 64)



conv2d_13	input:	(None, 14, 14, 64)
Conv2D	output:	(None, 14, 14, 128)



conv2d_14	input:	(None, 14, 14, 128)
Conv2D	output:	(None, 12, 12, 128)



activation	input:	(None, 12, 12, 128)
Activation	output:	(None, 12, 12, 128)



max_pooling2d_10	input:	(None, 12, 12, 128)
MaxPooling2D	output:	(None, 6, 6, 128)



dropout_8	input:	(None, 6, 6, 128)
Dropout	output:	(None, 6, 6, 128)



conv2d_15	input:	(None, 6, 6, 128)
Conv2D	output:	(None, 6, 6, 512)

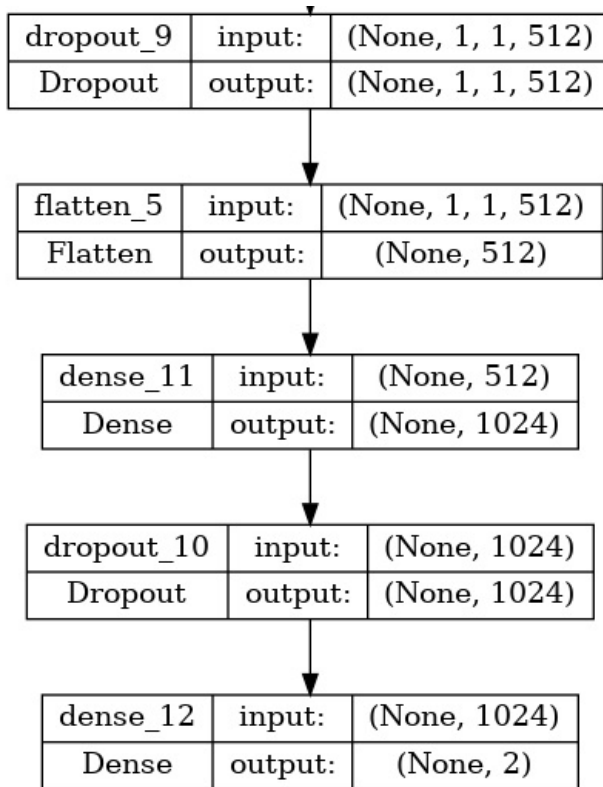


conv2d_16	input:	(None, 6, 6, 512)
Conv2D	output:	(None, 2, 2, 512)



max_pooling2d_11	input:	(None, 2, 2, 512)
MaxPooling2D	output:	(None, 1, 1, 512)





- 1.e
- Convolution neural networks are different from vanilla fully connecte neural networks because of the convolutional and pooling layers. Convolutions work by sliding a filter (which must be smaller than the original image, otherwise we are performing a deconvolution instead of convolution) and when sliding the filter we are essentially multiply term by term the filter with a subimage of the original image and then we add it up (so essntially a scalar product that leads to downsizing the image in order to increase the amount of information contained per pixel in the image). These convolutions are then followed by max pooling operations which lead to choosing the maximal value of pixels from continuous subimages slided from the original image.

2. Recurrent neural network model. Only one architecture was tried out for this type of model.

lstm_input	input:	[(None, 64, 3)]
InputLayer	output:	[(None, 64, 3)]



lstm	input:	(None, 64, 3)
LSTM	output:	(None, 64, 128)



lstm_1	input:	(None, 64, 128)
LSTM	output:	(None, 64, 128)



flatten_6	input:	(None, 64, 128)
Flatten	output:	(None, 8192)



dense_13	input:	(None, 8192)
Dense	output:	(None, 64)



dense_14	input:	(None, 64)
Dense	output:	(None, 2)

- Recurrent neural networks were developed as a need to model sequential data sampled as points in time, so generally speaking is less suitable for image and more appropriate for time series data.

3. Fully connected neural network model. Two architectures were tried out for this type of model:

dense_15_input	input:	[(None, 64, 3)]
InputLayer	output:	[(None, 64, 3)]



dense_15	input:	(None, 64, 3)
Dense	output:	(None, 64, 256)



dense_16	input:	(None, 64, 256)
Dense	output:	(None, 64, 256)



dense_17	input:	(None, 64, 256)
Dense	output:	(None, 64, 2)

- 3.a

dense_18_input	input:	[(None, 64, 3)]
InputLayer	output:	[(None, 64, 3)]



dense_18	input:	(None, 64, 3)
Dense	output:	(None, 64, 256)



dense_19	input:	(None, 64, 256)
Dense	output:	(None, 64, 100)



dropout_11	input:	(None, 64, 100)
Dropout	output:	(None, 64, 100)



batch_normalization_2	input:	(None, 64, 100)
BatchNormalization	output:	(None, 64, 100)



dense_20	input:	(None, 64, 100)
Dense	output:	(None, 64, 2)

- 3.b
- Fully connected neural networks are models comprised of multiple layers of perceptrons aligned in deep depths in order to captures as many non-linearities as possible. There are many tweaks that can be done to this tpe of neural networks such as: dropout layers which drop neurons in order to increase the robustness of the network or batch-normalization to perform the normalization at batch level at an intermediary layer.

cnn 2 is also 89% smth, as rnn lstm is 89% smth as well

Tried out some augmentations, such as the following:

- Rotate - rotate an image up to a certain degree
- Flip - flip an image horizontally or vertically
- Shear - shear an image
- Scale - downsize an image or upsize it
- Pad - add padding to the borders of a image
- Blur - blur an image and make it more fuzzy
- Crop - crop parts from an image
- CutOut - crop and extracts the parts from the image (like a masking of inside pixels)

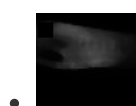
This is an example of how these augmentations turned out to look like:

Original image:


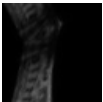

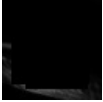

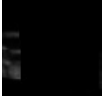

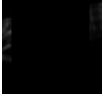



•

Augmented images:



•

- 
- 
- 
- 
- 
- 
- 
- 
- 

There were no substantial differences or improvement after performing augmentations of the dataset.

Number of epochs choices:

- 5 - around 5 or +/-3 epochs it was underfitting;
- 10 - best results (above this, it went into overfitting);

Learning rate choices:

- 1e-3 - best results
- 1e-2 - worse results
- 1e-1 - worse results

Optimizer choices:

- Adam - better results
- SGD (Stochastic Gradient Descent) - worse results

Loss function choices:

- Binary Crossentropy - insignificant differences
- Categorical Crossentropy - insignificant differences

Metrics function choices:

- F1Score - final choice as the kaggle metric was also this one
- Accuracy - no significant difference, but went for F1 since it was also used on kaggle

Data preprocessing techniques:

- resizing from (64, 64, 3) to (64, 64, 1)

- normalization by dividing to 255
- downsizing to (32, 32)
- converting the images to float32 precision

average precision per class on validation set with various models (tables/figures for these)

Submitting

In order to then create a submission file we call:

- `create_sample_submission()` from `main.py`

Methodology

Experiments

The experiments were mainly about experimenting with the three types of architectures: fully connected, recurrent and convolutional by varying the learning rate, optimizer, loss function, number of epochs, batch size and depth of the network.

Future works

A possible future work could be to try out some other more advanced transformers based architectures or deeper architectures such as Efficient Net, VGG or AlexNet. Maybe some other augmentations could be tried out as well. Of course new sources of data could also improve the performance as the dataset at hand is quite small for a task where high performance can be obtained mostly with large convoluted neural networks.

Conclusion

In the `src` folder there are five `py` files:

- `data_analysis.py` - where some analysis is performed on the dataset
- `main.py` - the main on which the other files rely on
- `network_visualizer.py` - functions necessary for creating the plots of the NNs architectures
- `solt_augmentation.py` - image augmentation is located
- `train_neural_net.py` - util functions for training various neural networks

To conclude, convolutional neural networks proved to yield the best results when compared to recurrent neural networks with long-short-term memory or other more simple fully connected neural networks.

Also, out of the 5 cnn tried out architectures the deepest one proved to yield the best results overall managing to achieve a public score of 0.915 and a private score of 0.907, so a difference of about 0.8% between the public and the private scores.

References

<https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/> <https://www.antoniofurioso.com/artificial-intelligence/convolutional-neural-network-what-it-is-and-how-it-works/> <https://towardsdatascience.com/inside-convolutional-neural-network-e1c4c1d44fa2> <https://medium.com/analytics-vidhya/an-overview-of-convolutional-neural-network-cnn-a6a3d67ce543> <https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/> <https://medium.com/analytics-vidhya/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90> <https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-networkcnn/>