Machine Learning Nanodegree

Capstone Project

Daniel Senna Panizzo, March 2020

I. Definition

Project Overview

The usage of Neural Networks for image diagnosis is quickly growing in the field of medical research and its usage and efficiency are already being tested on clinics, laboratories and hospitals. With a variety of usage possibilities like cancer detection, classification of lesion types or mental illness, the perspectives are enthusiastics. Health professionals are expected to use this technology in their benefit as soon as posible. Recently the Convolutional Neural Networks (CNN) emerged as the most common class of Neural Network for analyzing visual imagery. CNNs are regularized versions of multilayer perceptrons, they take advantage of the hierarquical pattern in data and assemble more complex patterns using smaller and simpler patterns (a.k.a. filtering and pooling the images). This means that a CNN can detect patterns (features) in an image without a pixel losing his correlation with its pixel neighbors.

As I have an intense contact with doctors who work with diagnostic imaging and I frequently discuss the use of new technologies in the area, I intend to use this project as a starting point to delve deeper into the subject and perhaps participate in research projects in the area as Machine Learning engineer.

Problem Statement

This project is based on the "Histopathologic Cancer Detection" Kaggle Competition, a project to identify metastatic tissue in histopathologic scans of lymph node sections. In others words, identify the presence of tumor cells in digital pathology scans. The expected results of this project is to train a Convolutional Neural Network model to make a binary classification of the presence of tumor (true or false) from a given image.

The macro steps included in the development of the CNN model should include:

1. Download the images and files available to train the CNN from Kaggle;
2. Make any preparations in the data if necessary;
3. Make an exploratory data analysis of the images and file;
4. Prepare the training, validation and test datasets;
5. Train a base model CNN and evaluate its accuracy;
6. Train other CNN using transferred learning and evaluate its accuracy

It’s expected at the end of this project to have a useful CNN model capable of predicting if an histopathologic scan of lymph has a tumor or not.

Metrics

We'll evaluate our model on area under the Receiver Operating Characteristic (ROC) curve between the predicted probability and the observed target. ROC curve is a performance measurement for classification problems. It tells how much a model is capable of distinguishing between classes and is plotted with True Positive Rate (TPR) against the False Positive Rate (FPR).

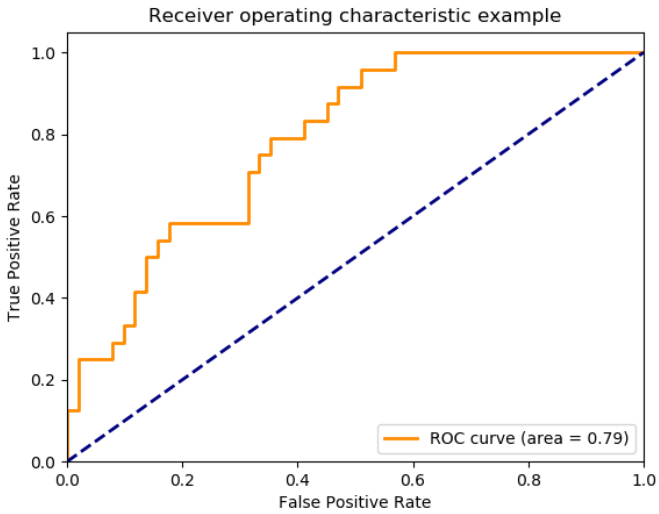

Figure: ROC Curve example from SKLearn

II. Analysis

Data Exploration

The image dataset available for this project is incredibe large with more than 200.000 images labeled as positive or false for tumor. Every image have 96x96px dimensions and the 32x32px center is the area that influences de label (Positive or Negative) for tumor tissue. However the outer area of the image may contain tumor tissue as well.

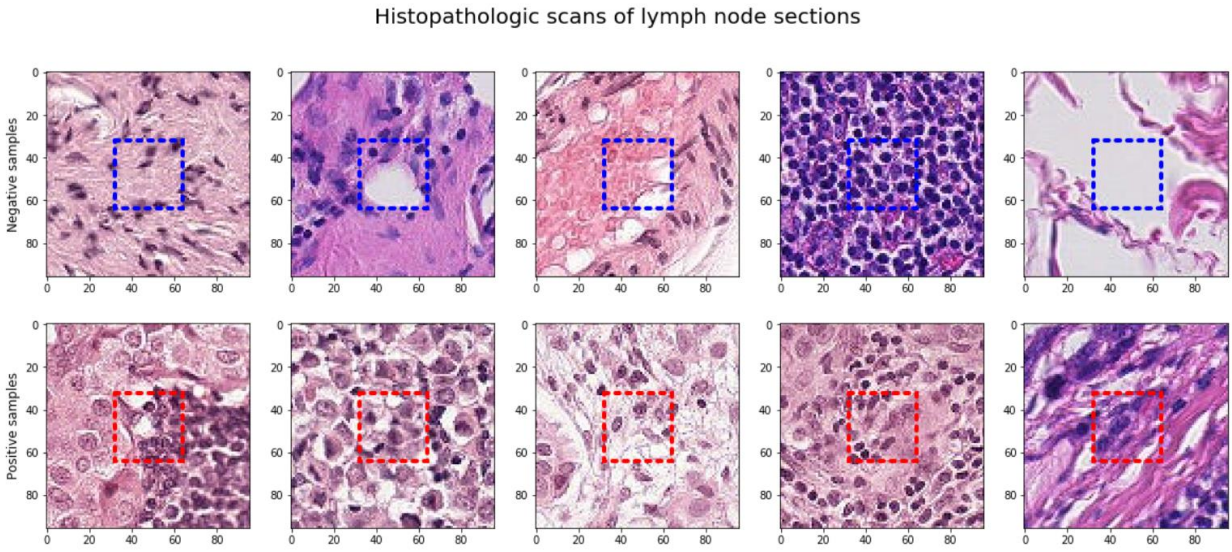

Figure 01: Positive and negative samples of histopathologic scans of lymph node sections.

Besides the images, there's a CSV file containing the name (id) of each image and their respective classification (label) for presence of tumor tissue (0 for negative and 1 for positive).

	id	label
0	1af50a3d4254999b0b486ee9c4fe1310c87f5b07	0
1	1ef450ccba59e14358893b0659b9863ec8b1adf2	0
2	2a88b4c7dc358944bf6791218ddc582082092915	1
3	2e0bc647a770640e952ba3e824b22a18aa5a37b5	0
4	1fb65b2f6a5ab845545d5157a652e4f7a75c53cd	0

Table 01: Sample rows from the CSV file containing the labeled data.

Exploratory Visualization

For this project I limited the data to 10.000 images. The distribution is as expected from the original dataset, around 60% for negative samples and 40% for positive samples.

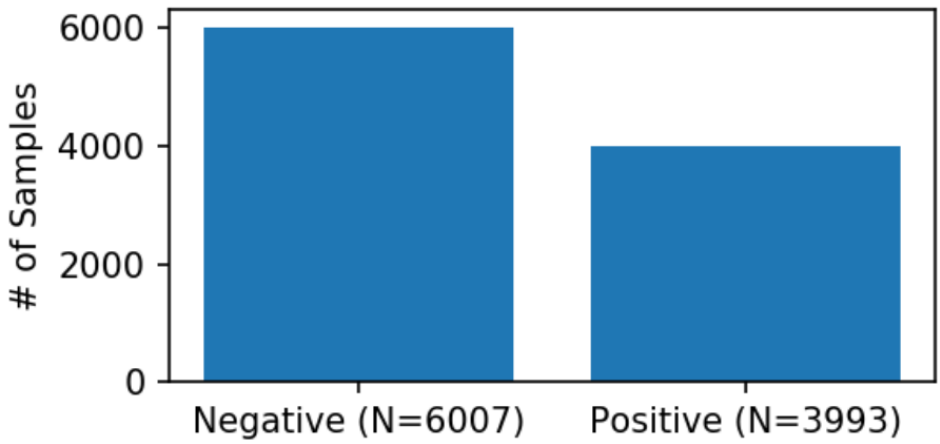

Figure 02: Distribution of images classified for negative and positive tumor tissue.

When analyzing images it's expected to understand every pixel in it. A pixel in an image can be represented as the intensity values of Red, Green and Blue (RGB channel with values from 0 to 255). In other words, we could represent one image as three tables, one for each color, and the cells in those tables holds the value intensity for each pixel. Below we analyze the pixel intensity value distribution on positive and negative image samples in all colors.

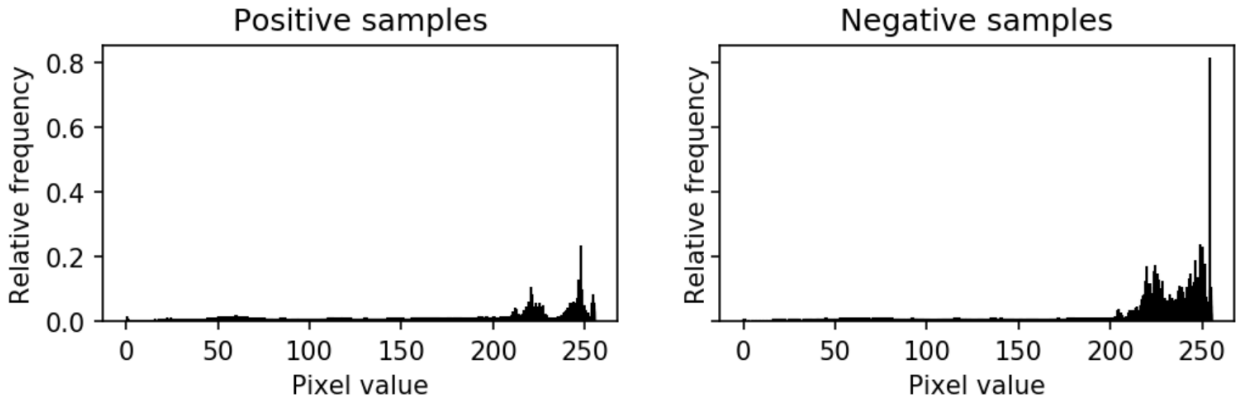

Figure 03: Relative frequency of pixel value for positive and negative tumor tissue.

Here we can spot that the pixel distribution are bimodal, but the difference is that negative samples have higher pixel values frequency than the positive samples for the RGB channel, especially for values above 250.

Algorithms and Techniques

First we'll build a basic CNN model from scratch and evaluate it. This basic model should contain a couple of hidden layers with ReLu activation followed by a fully connected layer finished with Sigmoid activation. After the initial evaluation, we should iterate trough fine tuning the parameters and including some Drop Out layers to avoid overfitting in our model.

Then we'll check if it's possible to improve the results applying pre-trained models (like DenseNet, ResNet and NASNet) available in Keras. Those pre-trained models have complex CNN architectures and were validated over the ImageNet dataset (a large visual database designed for use in visual object recognition software research). The main idea is to connect the output from the pre-trained model to our fully connected layer created in the previous stage and evaluate the results.

Benchmark

As the main benchmark model, we'll use the notebook available from the user "CVxTz" in the Kaggle competition. His model got a score of 0.9709 using CNN with the NasNet pre-trained models. We should also use others competitors notebooks as benchmark for the exploratory data analysis and tests with others pre-trained models.

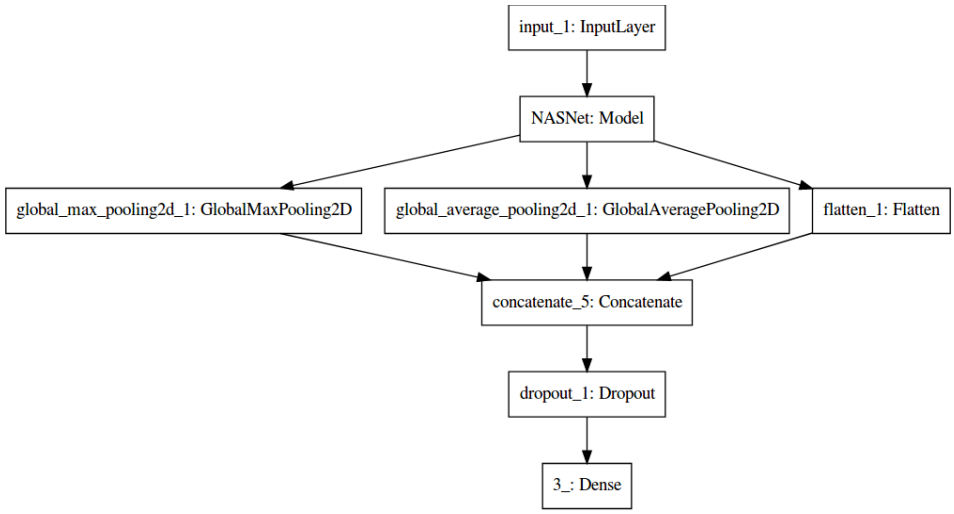

Figure 04: Benchmark CNN architecture macro view - Youness Mansar

III. Methodology

Data Preprocesing

To facilitate the data preprocessing, we'll first prepare our dataframe to store the image ID, its label (1 or 0), the complete path to access the image, a description of the label (0 = False and 1 = True) and a representation of the image as RGB (Red, Green and Blue) pixel values.

	id	label	path	label_desc	rgb
0	1af50a3d4254999b0b486ee9c4fe1310c87f5b07	0	.\train\1af50a3d4254999b0b486ee9c4fe1310c87f5b...	False	[[[161, 126, 146], [203, 169, 186], [229, 192,...
1	1ef450ccba59e14358893b0659b9863ec8b1adf2	0	.\train\1ef450ccba59e14358893b0659b9863ec8b1ad...	False	[[[228, 211, 217], [232, 219, 228], [220, 210,...
2	2a88b4c7dc358944bf6791218ddc582082092915	1	.\train\2a88b4c7dc358944bf6791218ddc5820820929...	True	[[[250, 242, 239], [250, 249, 245], [248, 248,...
3	2e0bc647a770640e952ba3e824b22a18aa5a37b5	0	.\train\2e0bc647a770640e952ba3e824b22a18aa5a37...	False	[[[255, 238, 255], [255, 247, 255], [250, 244,...
4	1fb65b2f6a5ab845545d5157a652e4f7a75c53cd	0	.\train\1fb65b2f6a5ab845545d5157a652e4f7a75c53...	False	[[[38, 17, 94], [14, 0, 74], [148, 119, 209], ...

Table 02: Dataset prepared for usage in training the CNN model.

And before starting training our CNNs, we'll make some common data prepprocessing for images using the Keras Image Preprocessing. For the training and validation data sets, we'll rescale the pixel values to a range between 0 and 1 and randomly flip the images on horizontal and vertical axis. For the test dataset we'll only rescale the images and avoid the data shuffling (so we can correctly compare the results from the prediction).

Implementation

With our data prepared and preprocessed, we can start feeding our CNNs with data for training and validation. All the CNN models used the same parameters when compiling and training the model:

- Adam with a learning rate of 0.001;
- Loss calculated as binary cross entropy;
- Accuracy used as metric for evaluation;
- Images were shuffled when feeding the model for training;
- Training used 10 epochs;

Although the image dataset had much more images for training, I had to reduce the number of steps per epoch because the time to finish the training was prohibitively long on my PC. All the models trained bellow used 100 steps per epoch for training, 30 steps per epoch for validation and 1000 images for test.

Models trained:

- Scratch Model (Base Model);
- Transferred learning with NasNetMobile;
- Transferred Learning with ResNet50V2;
- Transferred Learning with DenseNet201;

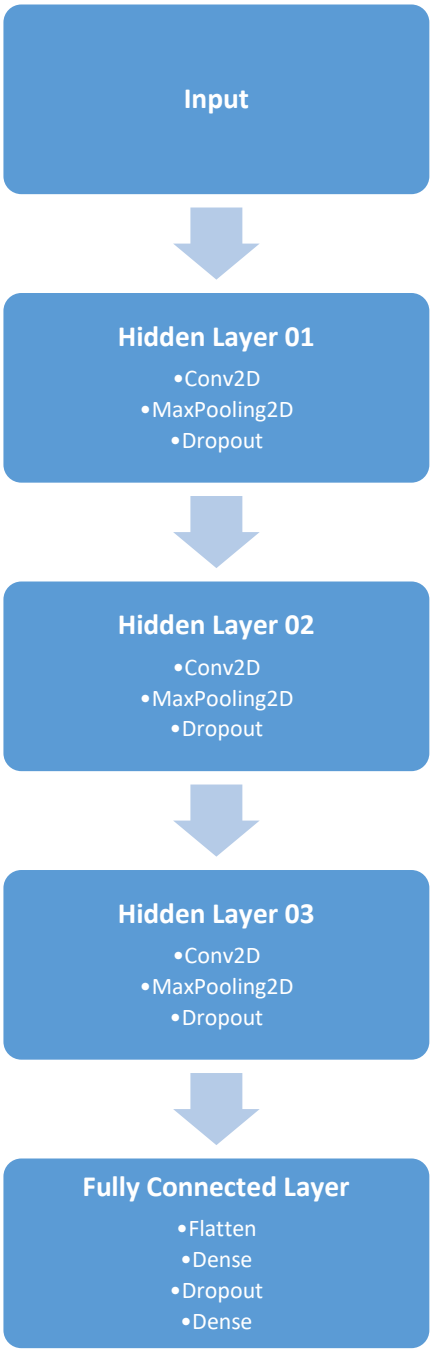

Figure: CNN Base Model (from scratch)

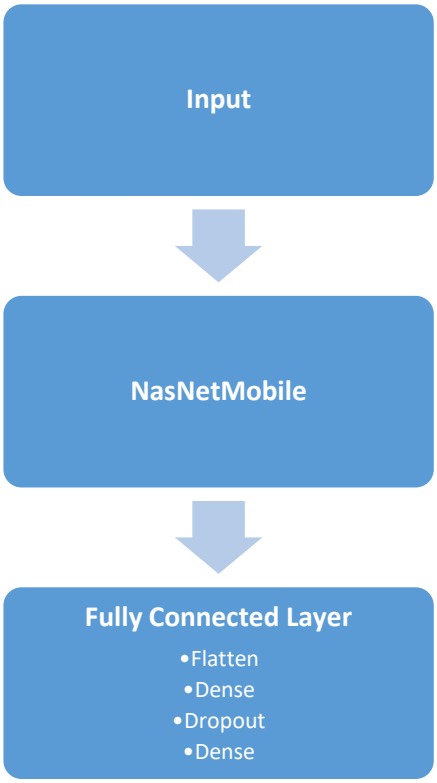

Figure: Transferred Learning with NasNetMobile

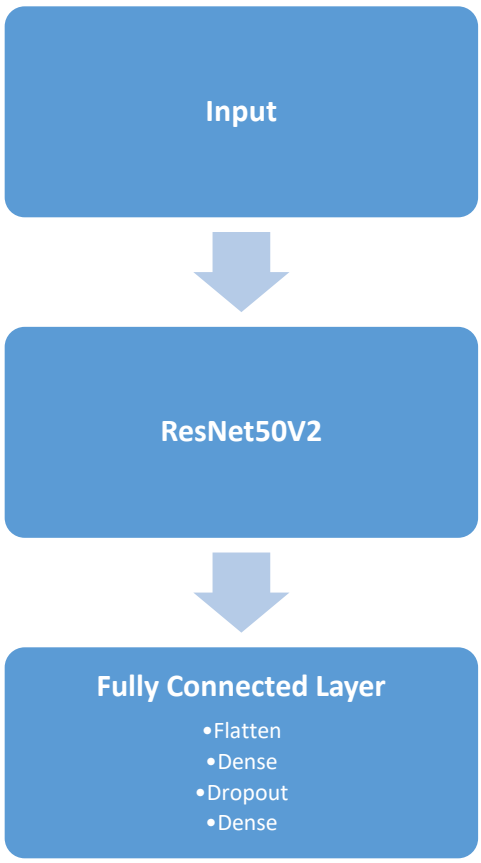

Figure: Transferred Learning with ResNet50V2

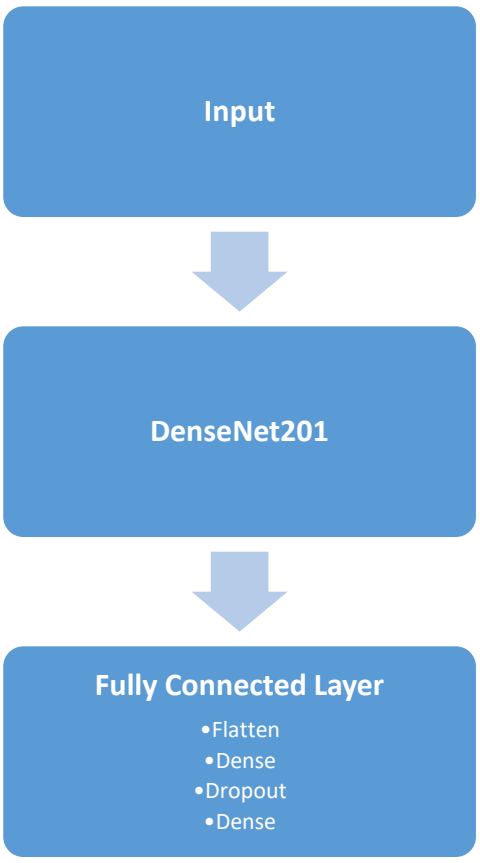

Figure: Transferred Learning with DenseNet201

IV. Results

Model Evaluation and Validation

Individual evaluation of our four models:

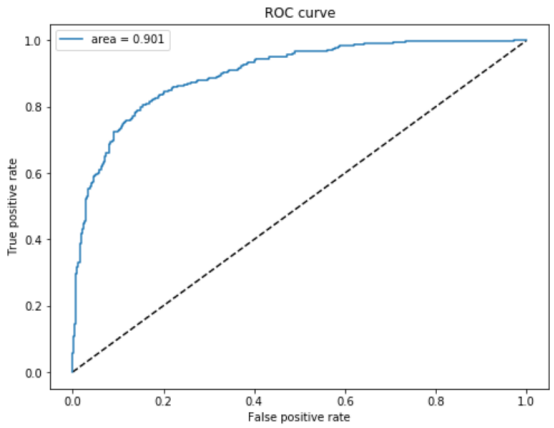

Figure: Scratch AUC-ROC

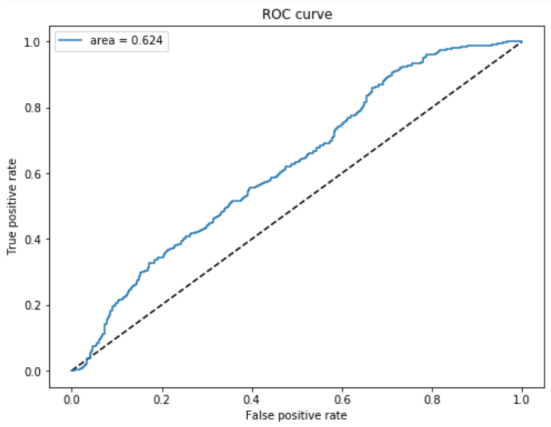

Figure: NasNetMobile AUC-ROC

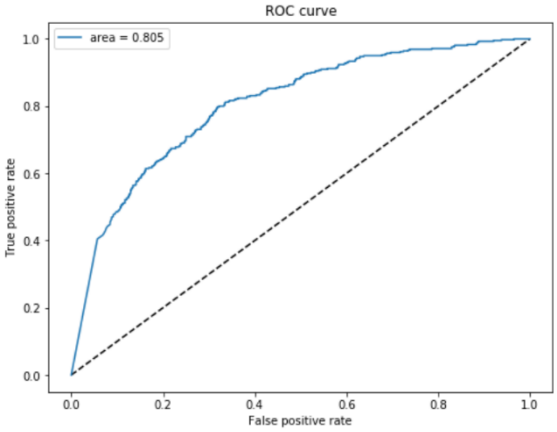

Figure: ResNet50V2 AUC-ROC

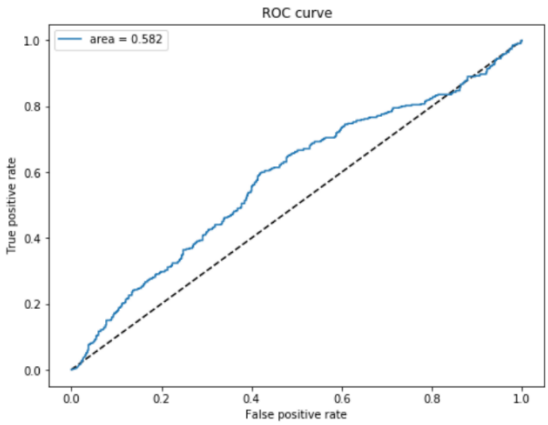

Figure: DenseNet201 NasNetMobile

The final results were:

Model	AUC-ROC
Scratch CNN	0.901
NasNetMobile	0.624
ResNet50V2	0.805
DesneNet201	0.582

Table: Trained models AUC-ROC score.

Justification

It was a little unexpected but the scratch model had the best performance. However, the poor performance over the models using transferred models can be justified. Those models were selected because other competitors on this Kaggle competition got good results wih them. When training this models the weights weren't initiated with the ImageNet validation dataset and the only modification was including the same final layer used on the scratch CNN we made previously. Everything else uses the same parameters and proccess.

This results may be influenced by the limits of my PC process power. We could probably get best results from the pre-trained models if given more time for training with more image samples, steps and epochs.

V. Conclusion

Free-Form Visualization

	id	label	scratch_pred
0	aacd14eef69a8685afb5e87b2c78d45535d0c79f	0	0.616296
1	aa5044a6843c8ccdc6440629c9bfedff5e8df4d3	0	0.330264
2	aa7396a51f9796494654a8e96ba13374cc83b96d	0	0.186724
3	aa7814f917b987ee012314175623389f569c9eea	1	0.943809
4	ab47cc369130b761b1409acccc52c1ff7ed298bd	1	0.794293
...	...	...	...
995	ab3e25c7695fef6c466341188e8fea85704478a0	1	0.419993
996	aa9dbca481b1089bdefc0e1716356f66cd4cea9d	1	0.892185
997	aa081dd9815afecd4deb5132576c29d9255970bc	1	0.953046
998	ab9bc9df8e07cad94feb22e2209b0ed53bfd3df7	0	0.160282
999	ab21fe9202bb97dfff367dcba7809b79dec61657	1	0.315978

Table: Predictions from the CNN Base model (from scratch)

Reflection

In a real world scenario were we use CNN for tumor predictions every improvement in the model is crucial. After all, we would be dealing with human's life and informing the diagnosis of a tumor cell could change lives. So the time needed to train a complex architecture CNN should be the minor of the problems. But it's interesting to know that we can achieve good results from personalized models. For this project scenario with my limitations I'd stay with the Scratch CNN with its simple, fast and good performance.

Improvement

Changing the image size to fit the size nedded to use the weights trained on the ImageNet for the pre-trained models could also improve the results. Others competitors achieved excellent results using more complex final layers at the end of the pre-trained models.