After we tried transfer learning obtaining bad performances we decided to try to build a simple CNN model from scratch. We began with the construction of a base model that we have then tried to improve adding more convolutional blocks. This is a common pattern in CNN designing. It consists in stop the adding of convolutional blocks when the add of the last convolution block doesn’t increase the performances of the latest model built. The base model architecture is composed of a convolutional block and a simple classification part. The convolutional block is composed of two convolutional layers with 32 activation maps each and a 3x3 filter size. This size because of the small local features in skin cancer lesions. These layers are followed by a max pool layer to reduce the number of parameter to be learnt. The classification part is composed of a flatten layer to flat the output of the convolutional block, a fully connected layer and finally the softmax layer to compute the probability distribution over the seven classes of the dataset.

These are the performances of the five model we have built following this design pattern. F1 scores for all classes, macro average F1 score and test accuracy are shown. Why we chose the proposed model?

Because it has better performances in terms of macro average and even F1 scores in all the classes compared to the other models. It has less performances in Melanoma class but we prefer to be accurate in all the classes and not only one.

Under the table is possible to see the architecture of all the model we built.

This is the architecture of the proposed model. It is different from the base model for the number of convolutional blocks on it. Following the design pattern, with the adding of a new convolutional blocks we doubled the number of activation maps for the convolutional layers. This helps the network to identify more complex features in skin lesions with the increase of the depth of the network. The dropout layers help to mitigate the overfitting problem. We increased the percentage of frozen units with the increase of the depth of the network because more parameters have to be learnt and more the overfitting would occur. Why we think this architecture is the best? Because we think this is the correct depth for this specific task. The network is depth enough to distinguish well skin cancer lesions, in fact the adding of a new convolutional block has decreased all the metrics presented in the previous table.

We have chosen the Adam optimizer because it was the unique that made the network capable to learn something. In fact with the other optimizers we tested, such as RMSprop and the simple SGD we had the same issue of transfer learning. The network wasn’t able to learn from our data. We tried three different learning rate values. We started with 0.001 learning rate that’s the common choice in computer vision. This caused the network to overfit very soon. So we tried a smaller learning rate, 0.00001 learning rate. The convergency was stable but it took too many epochs to reach a good accuracy. We then selected the 0.0001 learning rate as a good tradeoff between the previous learning rates. In fact, you can see that the overfitting is limited compare to 0.001 and the convergency is faster than 0.00001.

We then selected 32 batch size that was the default in Keras. We tried to train our models with this spcific batch size and we obtained good results so we kept it to its default value.

We selected 200 epochs to allow our models to learn as much as possible from training data.

We used these three regularization techniques to try to deal with the problem of imbalanced data. Dropout layer is a common technique that consists in freezing a percentange of the units of the network at each epoch. We decided to decay the learning rate if after 3 epochs the validation accuracy doesn’t increase. We finally used early stopping. If after 10 epochs of training the validation accuracy doesn’t improve, the training will be stopped and the weights corresponding to the best validation accuracy will be restored.

These are the performances of our proposed model. As you can see there are difficulties in predicting well dermatofibromas and melanomas. In fact there are difficulties in distinguish between dermatofibromas and basal cell carcinomas but also between melanomas and melanocytic nevis. 46% of melanomas are classified lika a nv and the 33% of dermatofibromas are classified like a bcc. Finally predictions are biased toward the majority class that is nv.

To improve the performances of our proposed model we tried to deal with the problem of imbalanced data using two different techniques.

Class weighing allows to make the dataset balanced for the model without changing the dataset.

Oversampling is more invasive, in fact it consists in increasing the minority classes to make the dataset balanced.

In particular, the class weighting technique consists in a loss function penalization that gives more emphasis to minority classes and less emphasis to majority classes, in such a way that makes the dataset balanced for the model. In this way is possible for example to count each melanoma example like 10 melanocytic nevis examples. As you can see from the confusion matrix there are less difficulties in predicting dermatofibromas and melanomas and less bias toward the majority class, so this is a better model in terms of predicting power. This shows that with a balanced dataset our model could achieve better performances.

So, we tried the oversampling techniques. After training, test, validation split, the majority classes counted 5000 examples, so we used data augmentation to oversample minority classes. From the confusion matrix is possible to see that with this technique the model reintroduces difficulties in distinguish between classes and we think this is due to overfitting issue. This is caused by the aggressive data augmentation we performed, in fact the minority class was augmented from only 80 examples to 5000 examples. In the figure on the right is possible to see the overfitting problem of this last model built.

In conclusion, we obtained our best performances with the class weighting technique and this demonstrate that with more data CNN can achieve good performances, probably better than expert dermatologists. This is an important milestone for oncology.

Our model performances are bad if compared with the performances of other researchers, but they used transfer learning across all the layers. Instead we tried to build a model from scratch and to use other approaches.

Finally, we didn’t have enough time to complete our experiments, so these are possible future works:

* oversampling HAM10000 with higher quality images bring from Hospital datasets;
* try to apply one vs all approach to HAM10000. This consists in building 7 classifier, one for each specific class. This could improve the performances on specific classes;
* try to apply transfer learning across all the layers of the pre-trained architecture. We didn’t have time and resources to try this strategy;
* test our proposed model on other open access datasets to see the generalization power of our model.

random weights inizialization -> deve essere fatto per permettere alla rete di apprendere infatti se i pesi fossero inizializzati ugualmente, a causa della simmetria della rete neurale feed-forward ogni aggiornamento cambierebbe i pesi nella stessa maniera. Ciò significa che un intero strato farebbe la stessa cosa che potrebbe fare un singolo neurone. È necessario quindi rompere queste simmetrie con la random weights inizialization. I pesi devono essere poi inizializzati a bassi valori per aiutare SGD, infatti per evitare fin da subito una lenta convergenza a causa del vanishing del gradiente è necessario tenere i pesi il più bassi possibile in valore assoluto, per fare in modo che l'input delle activation function sia il più basso possibile e quindi la derivata massima.

momentum -> permette di stabilizzare la convergenza e renderla più veloce, ad ogni aggiornamento dei pesi si tiene conto del gradiente appena calcolato e di una parte del gradiente precedentemente calcolato. Questa informazione è contenuta nel vettore velocity. Il parametro alpha permette di settare il tradeoff tra i gradienti da utilizzare -> questa tecnica fa si che si evitano gli zig-zag (vettori in direzioni opposte causati da randomizzazione di sgd) del normale sgd e vincola i vettori ad andare verso la soluzione. Inoltre, in caso di plateu questo aiuta perché essendo il gradiente molto piccolo, dopo ogni aggiornamento questo viene sommato ai gradienti precedenti, aumentando di volta in volta la size dello spostamento nella funzione errore.

activation function types -> le funzioni di attivazione permettono di rendere gli output delle unità all'interno di un certo range. Infatti l'input della funzione di attivazione è imprevedibile, in quanto dato dalla somma pesata di tutti gli input dell'unità. Inoltre le funzioni di attivazioni permettono di capire quali sono le unità che ad ogni epoca contribuiscono o meno all'apprendimento. Queste sono dette unità attivate. Esistono varie funzioni di attivazione, le più comuni erano sigmoid e tanh ma avevano il problema del vanishing del gradiente. Per cui si preferisce utilizzare la funzione ReLu, dove la derivata sulla parte positiva della funzione è sempre costante e di conseguenza anche il gradiente, che non sarà mai piccolo, a meno che l'output della ReLu non sia 0 perché l'input è compreso tra -inf e 0. In tal caso, se molte unità dovessero avere questo problema è necessario utilizzare una funzione diversa, ad esempio la leaky rely, che invece di spegnere totalmente alcune unità durante il training, penalizza semplicemente gli input piccoli, evitando il mancato contributo di alcune unità durante il training.

batch normalization -> per evitare vanishing del gradiente si normalizzano gli input delle unità. L'input è sempre bounded all'interno di un range in questo modo. Cosi si evitano valori troppo grandi o troppo piccoli dell'input che comportebbe un output della funzione di attivazione su una delle sue code, dove il gradiente è molto piccolo. Questo problema poi rischia di propagarsi lungo tutta la rete.

nesterov momentum -> è differente dal momentum solo per come viene calcolato il gradiente.

adam optimizer -> combina RMSprop con momentum.

activation maps -> più sono le activation maps e più filtri vengono applicati all'immagine di input con scale differenti, questo permette di trovare dei key points che sono più robusti alle trasformazioni dell'input, come ad esempio la risuluzione dell'immagine.

gaussian filter -> i parametri mu e sigma permetteono di gestire quanto si deve tener conto del vicinato dei pixel ad ogni applicazione del filtro, e quindi determinano la poteza dello smoothing effect applicato sull'immagine. A seconda di come questi parametri cambiano si avranno bluring effect differenti. È possibile ottenere il medesimo output con combinazioni diverse di filter size e parametri della gaussiana. Inoltre è possibile ottenere la stessa immagine con diversa risoluzione applicando il gaussian filter.