AN2DL - First Homework Report
OverFritti

Angelo Baturi, Daniele Bergamaschi, Lorenzo Bottelli, Gianluca Carta

1 Introduction

The goal of the project was to implement a CNN to solve a multi-class classification problem. We were provided with a dataset containing 96x96 RGB images of blood cells categorized into eight classes, each representing a particular cell state.

2 Data analysis and balancing

The original dataset was composed of 13759 images, and contained some outliers (Shrek and Rick Astley). After (sadly) removing them we analyzed the class distribution, noticing a not negligible imbalance (the minority class [4] had 849 samples and majority class [6] had 2330 samples). To tackle this problem we firstly tried to apply basic geometric image augmentation techniques (such as RandomRotation, RandomFlip and RandomTranslation) considering class weights in order to reduce imbalance while avoiding to introduce too much noise to the original dataset. While looking for alternative solutions we tried to use SMOTE, an oversampling technique to generate synthetic data, to up-sample the minority classes while avoiding to create duplicates that could lead to overfitting. We were satisfied with the outcomes and decided to stick with it.

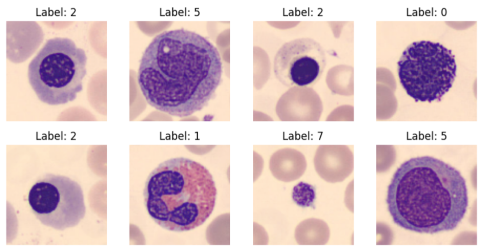


Figure 1: examples of synthetic images generated using SMOTE

It's important to note that data balancing was applied exclusively on the training set in order to maintain the original class distribution on the validation and test sets.

To perform our experiments we decided to keep 10% of samples to test our models locally and opted for a 80/20 split for training and validation.

3 Data Augmentation

In this section we delve deeper into one of the most crucial aspect of the challenge that rack our brains. Our initial approach was to apply basic augmentations that preserved the structure of each image, maintaining as much as possible the similarity to the original dataset. However we obtained poor results, but we were unsure if the main reason behind was about the augmentations adopted or the complexity of the model. So we started investigating more advanced augmentation techniques while using the same model (initially our custom CNN).

We started to use advanced augmentations techniques from Keras_CV that let images almost completely unrecognizable for a human eye and we realized how differently a neural network learns with respect to us. Even if we were skeptical, results also convinced us how the fundamental features remain detectable and allow the network to learn. We found out that MixUp, CutMix and GridMask worked pretty good for us, and we applied them on the training set.

Another strategy we tried was to implement our own augmentation pipeline that consisted in randomly picking one transformation for each of the three categories that we defined (geometric, color and distortion transformations) and applying them sequentially on the given image. In the end we replaced this technique with the straightforward application of RandAugment and AugMix, that guaranteed better results.

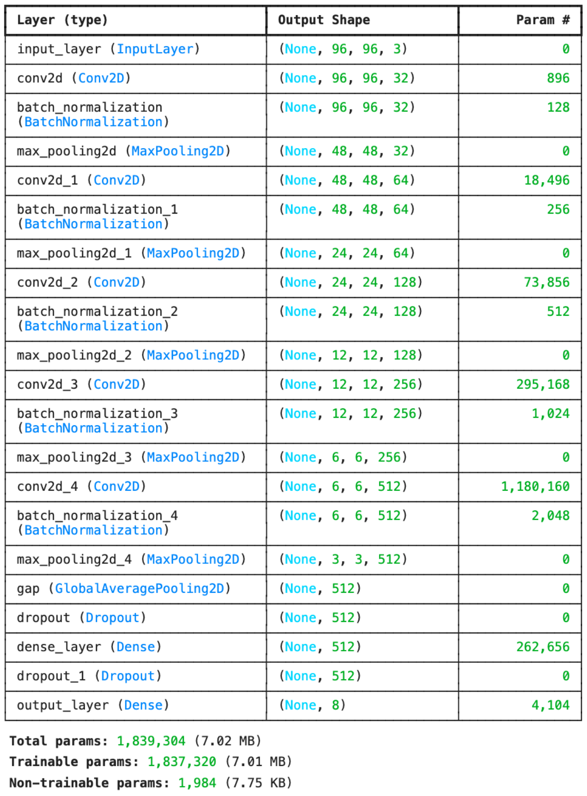
Later, we also tried to apply these augmentations to the validation and test sets separately in order not to add biases and avoid data leakage. We noticed that the results obtained by our models in this way were similar to the ones obtained on the submission test set. Therefore, to cope with the malfunctioning of the Codabench platform and to test our models on a more heterogeneous set we decided to adopt it as our local evaluation technique.

4 CNN from scratch

Our first approach was to build a custom CNN from scratch. In all our experiments we used Softmax as activation function in the output layer and Categorical Crossentropy as loss function. After an initial experimental phase we decided to use ReLU in the hidden layers, Adam as optimizer and HeUniform as weight initializer. The input images were normalized and the input labels were one-hot encoded.

We started with a basic structure consisting of 4 convolutional layers followed by a flatten layer and only one dense layer acting as output. As expected, the model performed poorly on the submission test set. Our intent was then to increase the complexity of the model while adopting techniques to reduce overfitting. We replaced the flatten layer with a Global Average Pooling layer to reduce the number of parameters and added dropout and regularization to dense layers. Moreover, we decided to add Batch Normalization layers to stabilize training. This model structure convinced us, so we tested it using different augmentation pipelines and techniques as discussed in section 3.

Our last model is represented in the figure below and obtained a score of 0,65 on our local evaluation on the augmented test set.



```
Layer (type)                            Output Shape                 Param #
input_layer (InputLayer)                (None, 96, 96, 3)                  0
conv2d (Conv2D)                         (None, 96, 96, 32)               896
batch_normalization                     (None, 96, 96, 32)               128
(BatchNormalization)
max_pooling2d (MaxPooling2D)            (None, 48, 48, 32)                 0
conv2d_1 (Conv2D)                       (None, 48, 48, 64)            18,496
batch_normalization_1                   (None, 48, 48, 64)               256
(BatchNormalization)
max_pooling2d_1 (MaxPooling2D)          (None, 24, 24, 64)                 0
conv2d_2 (Conv2D)                       (None, 24, 24, 128)           73,856
batch_normalization_2                   (None, 24, 24, 128)              512
(BatchNormalization)
max_pooling2d_2 (MaxPooling2D)          (None, 12, 12, 128)                0
conv2d_3 (Conv2D)                       (None, 12, 12, 256)          295,168
batch_normalization_3                   (None, 12, 12, 256)            1,024
(BatchNormalization)
max_pooling2d_3 (MaxPooling2D)          (None, 6, 6, 256)                  0
conv2d_4 (Conv2D)                       (None, 6, 6, 512)          1,180,160
batch_normalization_4                   (None, 6, 6, 512)              2,048
(BatchNormalization)
max_pooling2d_4 (MaxPooling2D)          (None, 3, 3, 512)                  0
gap (GlobalAveragePooling2D)            (None, 512)                        0
dropout (Dropout)                       (None, 512)                        0
dense_layer (Dense)                     (None, 512)                  262,656
dropout_1 (Dropout)                     (None, 512)                        0
output_layer (Dense)                    (None, 8)                      4,104

Total params: 1,839,304 (7.02 MB)
Trainable params: 1,837,320 (7.01 MB)
Non-trainable params: 1,984 (7.75 KB)
```

Figure 2: summary of the model of our CNN

5 Transfer learning + Fine tuning

In order to achieve better results, we decided to move to pre-trained models from Keras Applications . Our strategy was to initially apply Transfer Learning and Fine Tuning techniques to the smallest models, to be able to make a reasonable comparison while considering time and computation constraints. We started from the pre-trained weights of ImageNet and we added dense layers to be trained. Then we fine-tuned the models, trying with a varying number of frozen layers.

These are the best results we obtained on local test for each model:

Model	Accuracy (local)
DenseNet121	84,87%
ResNet50	78,68%
MobileNetV2	77,46%
EfficientNetV2B0	**86,20%**
InceptionV3	84,39%

Being EfficientNetV2B0 the best model among the small ones, we decided to experiment with bigger models from the EfficientNet family. As expected, the best result on Codabench test was obtained with the largest one, with a simple Transfer Learning + Fine Tuning freezing 80 layers.

Model	Accuracy (submission)
EfficientNetV2L	**89%**

We also wanted to experiment with the Gradual Unfreezing technique. For time and resources constraints we applied it on the EfficientNetV2B3 model. We repeatedly fine-tuned the model with a decreasing number of frozen layers and we found these results on our local test:

Phase 1	:	N=75	85,87%
Phase 2	:	N=45	87,46%
Phase 3	:	N=15	87,79%

This was enough to convince us of the power of Gradual Unfreezing, that probably would have brought even better results with EfficientNetV2L.

As last experiment, we tried to re-train EfficientNetV2L also on the data previously used for validation and testing in order to maximize the utilization of data at our disposal. We obviously couldn't evaluate our model locally having no more set of unseen data, so we submitted the model on Codabench and we were surprised to notice a slight drop of 1%. This could be due to the fact that we didn't perform class balancing on the new training data and that we ran the new training for just 30 epochs.

6 Further improvements

As mentioned before, we also thought about some improvements that could be applied to increase the accuracy of our models and here are some examples:

- Apply Gradual Unfreezing to larger models
- Use layer-specific learning rate
- In the last training step done on the previous validation and test data, apply class balancing

7 Conclusions

We think that our approach of starting from simple models and trying to understand how to effectively apply the techniques seen in class gave us a good understanding of which factors can have the biggest influence on the model's learning. For example we were surprised about the huge impact of image augmentations, and a big part of our focus throughout the challenge has been on experimenting different techniques and pipelines. Also, by training both simple and larger models we acquired awareness of the fact that every decision is a trade-off and not always going for the biggest model is the final solution, especially because of time and computation requirements.

8 Contributions

Angelo Baturi	Data augmentation, custom CNN, TL+FT on Inception
Daniele Bergamaschi	Data balancing, TL+FT on DenseNet and MobileNet
Lorenzo Bottelli	Data augmentation, custom CNN, TL+FT on EfficientNet
Gianluca Carta	Data analysis, TL+FT on ResNet, Gradual unfreezing

9 References

- Enhancing Classification Accuracy for Imbalanced Image Data Using SMOTE, link
- RandAugment: Practical automated data augmentation with a reduced search space, link
- AugMix: A simple data processing method to improve robustness and uncertainty, link
- EfficientNet: Rethinking model scaling for Convolutional Neural Networks, link