POLITECNICO
MILANO 1863

AN2DL - First Homework Report
Autobot

Gabriele Carminati, Gabriele Carrino, Matteo Briscini

carmigab, gabricarr, sbatters

245680, 246970, 259098

November 24, 2024

1 Introduction

This project focuses on *image classification* of 96x96 RGB images of **blood cells** divided between **eight different classes**. The task consist in performing *multiclass classification* on the described dataset using deep learning techniques, in particular using **Convolutional Neural Networks**.

2 Problem Analysis

2.1 Dataset analysis and preprocessing

To better understand the problem the provided dataset was inspected and analyzed, with a focus on **outlier detection**. Two techniques were employed to identify the outliers:

- **PCA**. Principal Component Analysis was used to reduce the dimensionality of the dataset to be able to represent it in a 2 dimensional plot. Then from the obtained plot some outliers were identified.
- **T-SNE**. T-distributed Stochastic Neighbor Embedding is another technique to represent high-dimensionality data in 2 dimension.

The second approach resulted to be slightly better in term of outlier separation, so that was the one used to actually identify and remove the outliers. This process lead to the identification of **two types of outliers** (shown in image) for a total of **1800 images** that were removed from the dataset. In additional to the outlier removal also **8 duplicated images** were removed, reducing the dataset size from 13759 to **11951** images.

2.2 Dataset splitting

To perform the training of the deep learning models the dataset was splitted between training and validation sets with a ratio of **80%, 20%**.

2.3 Class rebalancing

After removing the outliers and splitting the dataset the rate between classes in the training set was measured and the dataset was found to be imbalanced. To overcome this problem the **oversampling** method was applied to the least represented classes in order to reach the same cardinality of the most represented class.

3 Method

After various experiment described in the following paragraph the final model was structured as follow. It is an **ensemble model** made by **5 models** applying the **bagging** technique with **majority voting** to make predictions. The five base models were all model performing **at least 83% accu-**

racy on the test set. Each of them consisted of the **ConvNextBase** model **finetuned** with the given dataset. Before training the dataset was augmented with the **augmetation pipeline** described in the next paragraph, ensuring variability in the training set, such that each of these 5 models was trained on a slightly different dataset thanks to the randomly sampled augmentations. The **final test accuracy** score measured for the ensemble model was **88%**.

4 Experiments

A series of tests and experiment was conducted to reach the final method described in the previous section.

4.1 Model Testing

To determine the best **model** for the task, several were tested on the dataset before class rebalancing. A set of **simple augmentations** was applied, including RandomFlip(), RandomRotation(factor=0.4), RandomContrast(0.2), and RandomTranslation(0.2, 0.2).

The tested models included both custom designs and pre-trained models for transfer learning:

- **Custom CNN**: A model with **4 blocks** of 2-3 convolutional layers, RELU activations, and max pooling, with the final block using global average pooling and a Dense layer.
- **MobilNetV3Small**[13]: A pre-trained model from Keras with **average pooling**, plus an added DropOut and Dense layer for classification.
- **EfficientNetB7**[6]: A Keras pre-trained model with **average pooling** and a **softmax** activation, enhanced by DropOut and Dense layers.
- **ConvNextBase**[4]: Another Keras pre-trained model, featuring **1000 classes**, **average pooling**, and a **softmax** activation, with additional DropOut and Dense layers.

Transfer learning models were fine-tuned, freezing 124, 700, and 200 layers during training. The table summarizes the accuracy results for these models.

Model	Validation	Test
Custom CNN	88.33%	29%
MobileNetV3Small	95.82%	46%
EfficientNetB7	96.24%	58%
ConvNextBase	96.86%	68%

Test accuracy measured by submission on codabench

4.2 Class rebalancing

To improve the measured performance a process of class rebalancing through oversampling was applied for all the following experiments.

Class	Original number of Images
Class 0	1052
Class 1	2381
Class 2	1285
Class 3	2226
Class 4	1049
Class 5	1393
Class 6	2530
Class 7	1843

4.3 Augmentation testing

After identifying ConvNextBase as the most promising model, various **augmentations** were tested to replace the simpler approach previously used. To streamline testing, the custom CNN model was employed due to its faster training time. Each augmentation was applied individually, and its impact on accuracy was measured. The best-performing augmentations were then selected for further experiments.

Augmentation	Accuracy (%)
Empty Model	93.70
ChannelShuffle	94.66
RandomCutout	94.29
JitteredResize	86.70
RandomShear	91.50
GaussianNoise	82.40

4.3.1 Augmentation Pipeline

To improve test performance, a new **augmentation pipeline** was designed to apply a variety of augmentations directly to the dataset before training. This

avoided issues related to casting image and label values or differing input requirements across augmentations. The tested augmentation layers included: CutMix [5], MixUp [12], FourierMix [7], RandAugment [14], JitteredResize [10], GaussianNoise [8], ChannelShuffle [3], and GridMask [9].

Initially, augmentations were applied uniformly to all images. The pipeline then evolved to **expand the dataset** by applying random augmentations multiple times, merging the augmented data with the original dataset to **increase variety and size**. Finally, the pipeline was refined to **apply randomly selected augmentations on a per-image basis**, ensuring even greater variability in the training set.

4.4 Optimizer testing

Different optimizer were also tested. The previous experiment were performed using **Adam**[1]. Other optimizer tested were **Lion**[11] that obtained a **1% average increase** on the classification accuracy, and **AdamW**[2] with a decay rate of $1e-4$ that performed on average **1%** better than Lion. This results lead to the use of AdamW for the next experiments.

4.5 Ensemble model

The last performed experiment was to apply the **bagging** technique to this classifier. The **5 models** trained so far with the highest test classification accuracy were combined in a system that performed predictions using **majority voting**. The idea was to combined different model with good classification accuracy trained on the dataset augmented with different augmentations (randomly selected), in the case that different augmentations made the models able to better classify specific inputs. However the performance increase was not as high as expected, this ensemble model reached **88% accuracy score**, only 1% more than the best single model.

5 Results

The most performing model obtained got a **88% accuracy** score on the test dataset on codabench on unseen data which is a huge improvement with respect to the first model trained. During the process some unexpected outcomes were found, in particular tests showed that an **higher test accuracy** score was reached when performing **augmentation** also on the **validation set**. This can be explained by the fact that probably the unknown test set used to benchmark the models contains also some augmented images and thus augmenting also the validation set during training increases the overall test performance.

6 Discussion

The proposed model and augmentation pipeline were seen to **perform quite well on unseen data**, as shown by the final test accuracy. This is due to the **variety** introduced by the augmentations and the capabilities of the **finetuned model**. However this augmentation sometimes seems too "*aggressive*" leading to a drop in the classification accuracy. As described in the previous section the unexpected behaviour regarding the augmentation on the validation set is strange and will probably need further investigation.

7 Conclusions

To sum up this work the final proposed model reach a **good classification accuracy** on unseen data, thanks to augmentations and finetuning. However some improvements are still possible, further work may include testing **different combination of augmentation** steps.

8 Individual Contributions

Each team member took part in almost every aspect of the project, however this was the main work distribution:

- **Gabriele Carrino**. Initial model testing and ensemble model testing.
- **Matteo Briscini**. Single augmentation testing and optimizers testing.
- **Gabriele Carminati**. Augmentation pipeline and test with different augmentations.

References

[1] Keras Documentation. *Adam Optimizer*. URL: https://keras.io/api/optimizers/adam/.

[2] Keras Documentation. *AdamW Optimizer*. URL: https://keras.io/api/optimizers/adamw/.

[3] Keras Documentation. *ChannelShuffle*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/channelshuffle/.

[4] Keras Documentation. *ConvNeXtBase*. URL: https://keras.io/api/applications/convnext/#convnextbase.

[5] Keras Documentation. *CutMix*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/cutmix/.

[6] Keras Documentation. *EfficientNetB7*. URL: https://keras.io/api/applications/efficientnet/#efficientnetb7.

[7] Keras Documentation. *FourierMix*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/fouriermix/.

[8] Keras Documentation. *GaussianNoise*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/gaussiannoise/.

[9] Keras Documentation. *GridMask*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/gridmask/.

[10] Keras Documentation. *JitteredResize*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/jitteredresize/.

[11] Keras Documentation. *Lion Optimizer*. URL: https://keras.io/api/optimizers/lion/.

[12] Keras Documentation. *MixUp*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/mixup/.

[13] Keras Documentation. *MobileNetV3Small*. URL: https://keras.io/api/applications/mobilenet/#mobilenetv3small.

[14] Keras Documentation. *RandAugment*. URL: https://keras.io/api/layers/preprocessing_layers/image_augmentation/randaugment/.