Students: AZIMI Arya, BELOTTI Ottavia, IZZO Riccardo
Student IDs: 10824423, 10657411, 10599996

Homework 1 Report
Artificial Neural Networks and Deep Learning - A.Y. 2022/23

---

1 Definition of the task

The first homework consisted in an image classification task featuring a data set of 96x96 resolution images divided into 8 classes. The public data set made available for the challenge contains 3542 images whose distribution among the classes is more or less uniform except for *Species1* and *Species6* that have significantly less training images than the other classes.

We tackled this task by the use of Convolutional Neural Networks (CNNs).

2 Data Preprocessing and Data Augmentation

We prepared the images for our network with the `ImageDataGenerator` utility provided by Keras. This utility allowed us to split the data set in two: training and validation set with a ratio of 4:1.

Moreover we did data augmentation on the training set to increase the diversity of the overall data set and to increase the robustness of the network. We applied several transformations such as rescale, rotation, zoom, horizontal and vertical flip, horizontal and vertical shift, sheerness variation, brightness variation. As a preprocessing function, as a first attempt we used a custom one to add random Gaussian noise to the images in order to train the net more robustly, but we also tested CutOut and CutMix (see *Sec.* 7).

Finally, with the models using transfer learning, we resized all the images in order to match the input shape they were trained on. For example, in the *Xception* case, we have increased the image size from 96x96 to 299x299, testing as filling methods both `reflect` and `nearest`. This practice greatly improved the overall accuracy of our CNNs.

3 First Model: Custom CNN

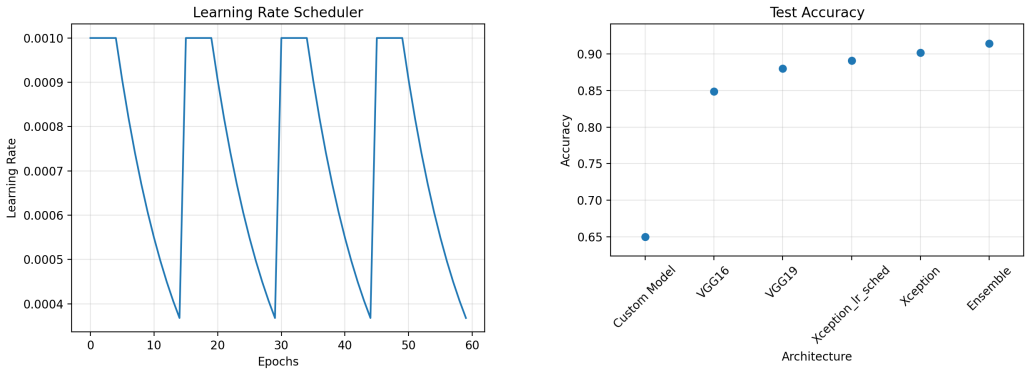
The first attempt was a simple CNN with several convolution and max pooling layers.

We started from the lab code and, after some tuning, we ended up on a final configuration composed of 5 blocks for feature extraction, each one made up of: a Convolution with *ReLU* as activation function, Batch Normalization, Max Pooling, Dropout with 0.2 drop rate (as an optional layer of the block). In fact, three dropout layers have been introduced in order to effectively avoid overfitting along side the Early Stopping mechanism, since we noticed that in its primordial state, the custom net used to score a much lower accuracy on the Codalab's test set than the validation accuracy achieved locally. We chose a low drop rate as 0.2 since convolutions already discard more information than the dense layers, as we didn't wish to lose too much valuable parameters.

Finally, a global average pooling layer prepares the input for the fully connected part of the network, separating it from the feature extraction part.

At last, the classifier is made of: a dense layer with 128 units, batch normalization, dropout with 0.5 drop rate, RandomFourierFeatures layer to implement Quasi-SVM, Dense layer (output) with 8 units and softmax activation function.

Different hyperparameters has been tested with both Keras Tuner and by hand, in particular the ones involved are: Batch size (16, 32, 64, 128), Learning rate $[1e-3, 1e-6]$, Dropout (0.2, 0.5), Number of units in the dense layer (64, 128, 256, 512, 1024), Number of filters in the convolutions (32, 64, 128, 256), Kernel size [(3, 3), (5, 5)] for the convolutions.



(a) Learning Rate scheduling with exponential decay $e^{-0.01}$

(b) Accuracy comparison among the tested architectures

Figure 1: Learning Rate Scheduler and Accuracy

We spent a couple of days on the custom model but as soon as we realized how powerful Transfer Learning and Fine Tuning techniques could be, we decided to switch to the next model.

4 Second Model: Transfer Learning and Fine Tuning

The next step was to try the transfer learning approach and fine tune already well established CNNs. We tried several of them included in the `keras.applications` module and eventually we found *VGG19* and *Xception* to be the most promising. Although, from here on we will refer to *Xception* since it's our best configuration.

After the data augmentation part, the *Xception* net has been imported, we excluded the layers in the top model in order to build our own custom classifier, while the feature extraction part of the original network has been preserved. The final configuration of the fully connected section is: GlobalAveragePooling2D, Dense layer with 256 units and *GELU* activation function, Dropout with 0.3 rate, Dense layer with 128 units and *GELU* activation function, Dropout with 0.3 rate and finally, as output layer, a dense classifier with 8 units (equals to the number of classes). We found the *GELU* activation function slightly more beneficial than the *ReLU* so we sticked to it.

For what concerns the Fine Tuning part, we unfreezed all the layers to possibly let all the model be fully retrained with a very low learning rate (i.e. $1e-5$).

This configuration resulted to be our best stand-alone model so far, with a test accuracy in the Codalab's final phase of $\approx 90\%$ but slightly overfitting, since it suffered from a loss of accuracy of $\approx 2\%$ from the development phase.

5 Third Model: Experiments with CutOut on Augmentation and Learning Rate Scheduler

Having obtained the best results by applying transfer learning on the *Xception* network, we decide to further expand our research with this model. We've noticed a tendency of our network to train for too few epochs, due to the early stopping mechanism stepping in to avoid overfitting. This behavior made us want to try to modulate the learning rate, both in the fitting and in the fine tuning phases, bringing instability to it in order to potentially avoid local minima in the stochastic gradient descent that even the adaptive moment estimation feature of *Adam* algorithm might not avoid. Hence, we introduced a scheduler that forces a periodic spike in the learning rate, keeps it constant[1]for 5 epochs and then lets it decay exponentially for 10 as can be seen in *fig.* (1a). Unfortunately, we haven't had enough time to thoroughly test and fine tune this trick, but even with so little time, it gave satisfying results since it managed to lengthen the training time without introducing much overfitting.

Finally, as a pre-processing of the data augmented images to feed the CNN with, we implemented the CutOut technique, by randomly erasing patches from the training images, to encourage the network to be more resilient and be able to generalize better.

This network actually performed a little bit worse than the previous straight forward Xception one (up to around -2% in accuracy).

6 Final Model: Ensemble of CNNs

Finally, we achieved our best result by combining our best accuracy *Xception* model (*Sec.* 4) with the more experimental version that features the *CutOut* technique for data augmentation and a learning rate scheduler (*Sec.* 5). This scored us an improvement of almost 2% in test accuracy from our best model since the ensembling helped to get rid of some overfitting effect.

The ensemble consists of gathering the predictions for the inputs from each of the two models, averaging the classes probabilities for each image, finally assessing as the final prediction the most likely class from the averaged probabilities (i.e. *argamax*).

7 Further Experiments

Quasi-SVM layer An attempt has been made with a RandomFourierFeatures layer before the output layer, this allows us to "kernelize" the model by applying a non-linear transformation. The result on the custom model was a 5% improvement in the overall accuracy.

Noisy Student weights Tried this specific pre-trained weights for *EfficientNet* series transfer functions instead of widely used `Imagenet` which resulted however in less accuracy.

Different kernel initializers We tried with two simple choices of random and fixed initializer and also using *Glorot* uniform which in some cases gave a 1-2% improvement on the accuracy.

CutMix and CutOut Having used the Cutout method for augmentation, we tried also the mixing as well but it left us with a slightly worse accuracy.

Patience We also tried with different values of patience and we ended up using a patience of 20 where we could end the fitting not too early and also not ending up overfitting.

Bonus: Vision Transformer When we started to dabble in the transfer learning domain, we came across very new approches to image classification that drifted from the CNN one. We decided to try out of curiosity the keras import of *DeiT*, a vision transformer trained to perform image classification. This piqued our interest and immediately gave us a very good result in accuracy, leaving us impressed by the strength of such a tool, but immediately after discarded it since it wasn't the focus of the homework.

8 Conclusions

As was mentioned before, we ended up using an ensemble of two *Xception* transfer models: our best base model (Sec. 4) and the one featuring Cutout and a custom learning rate scheduling (Sec. 5). This homework gave us the opportunity to experiment with our own CNN, but also made us understand how much time and understanding of the problem domain is needed to achieve results comparable to the already popular networks.

[1] The base learning rate in the fitting part is set at $1e-3$ and in the fine tuning part at $1e-5$