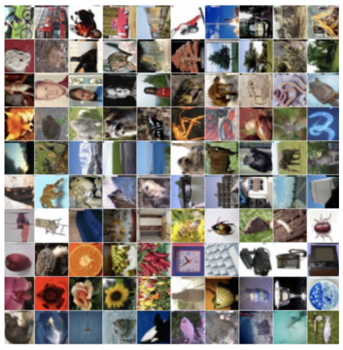
National Technical University
School of Electrical & Computer Engineering
Neural Networks and Intelligent Computer Systems

Deep Learning

Objective

The aim of this study was to familiarize ourselves with the neural network architectures with Deep Networks (Architecture) As part of this work we are also invited to familiarize ourselves with the TensorFlow 2.0 library and optimize models in the CIFAR-100 dataset. For implementation we used jupyter notebook in Google Colaboratory environment, utilizing the built-in GPU. We studied two learning techniques with the implementation of our own models "from scratch" and Transfer Learning. The models described below were trained according to the pronunciation code and the following baseline parameters: Adam, Sparse Categorical Cross-Entropy Loss, 128 Batch Size. The early stopping method was also applied to end the training. More specifically, each model is trained until the validation loss does not improve for 25 consecutive seasons.

Data



The dataset consists of 60000 32x32 color images belonging to 100 classes (600 images per class). The dataset is divided into atrain set of 50000 images (500 per class) and a test set of 10000 images (100 per class).   This is an evolution and clearly more difficult version of the CIFAR-10 dataset. The 100 classes are grouped into 20 superclasses. So each image comes with a fine label for the class and a coarse label for the superclass to which it belongs.

We apply augmentation to our data from the beginning and more specifically here ImageDataGenerator in the introduction of the dataset. There is an increase in training time by 2 to 3 times, which makes sense since our data has multiplied. The augmentation we applied concerns the mirroring of the image in relation to the horizontal axis, zooming, rotation and shift along the length and width. Finally, it should be mentioned that the Generator has the option for scaling (1/255) which essentially normalizes the values of each image to [0,1]. This was avoided, however, because it was done earlier when the dataset was loaded.

By exploring ways to speed up training times, prefetching was implemented so that the data that is expected to be executed soon will be prepared and made immediately available. We also apply caching, using TensorFlow techniques, where the data is stored in cache so that it does not take long to reload it in each training session.

In our case it is possible to directly load all the images and therefore no further processing is required in terms of memory and storage. Then we give the data to our model through a train and validation generator, with limited batch size where he takes care to load the real image for each path provided.

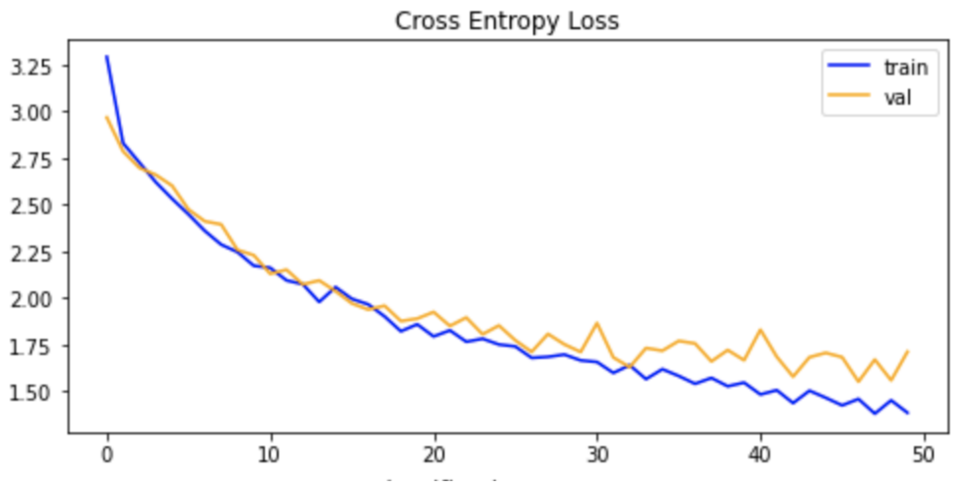
Initial non-optimized architectures

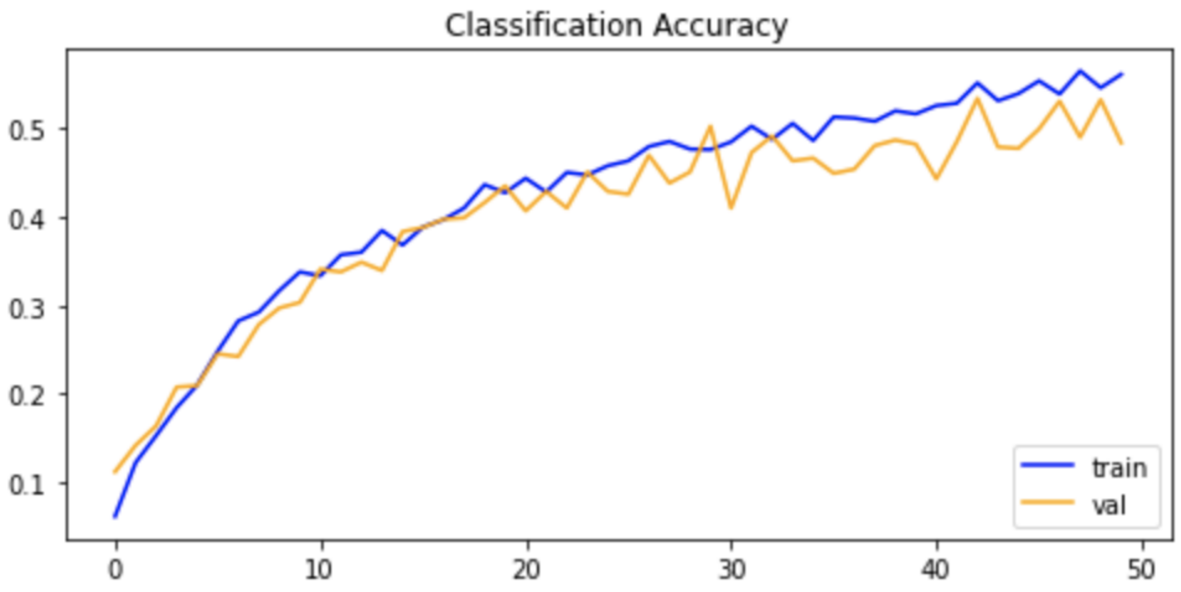
“from scratch” models

The simple model is a simple Sequential network with 3 convergent and 2 dense levels. H αρχιτεκτονική αναλυτικά:

1. 2D Standard Convolution Layer with 32 output filters, ReLU activation and 3X3 Convolution Kernel
2. 2D Max Pooling Layer over a 2X2 Kernel
3. 2D Standard Convolution Layer with 64 output filters, ReLU activation and 3X3 Convolution Kernel
4. 2D Max Pooling Layer over
5. 2D Standard Convolution Layer with 64 output filters, ReLU activation and 3X3 Convolution Kernel
6. Flattening Layer in order to adjust the next Dense Layers
7. Dense Layer with 64 output cells and ReLU activation
8. Dense Layer with 100 output cells (1 for each class) and Softmax activation for probabilistic output

Training the model for 50 seasons we get the following curves:



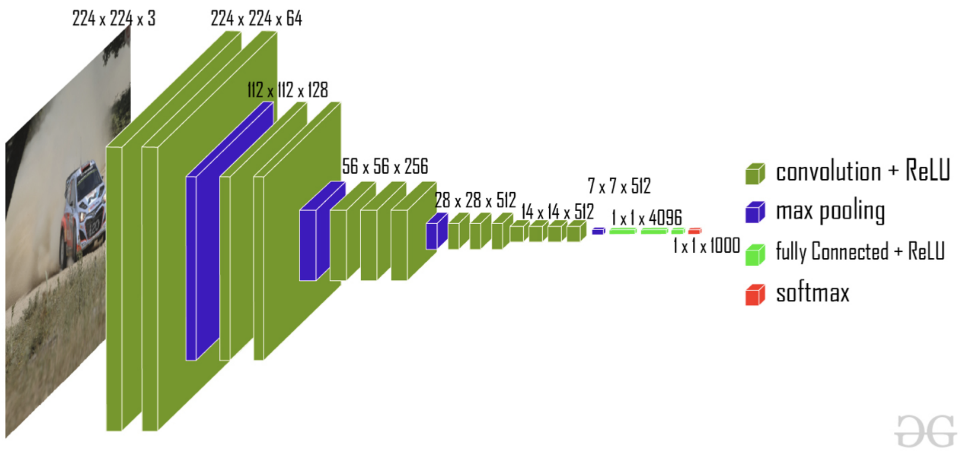


Loss: 1.62
Accuracy: 0.51

We notice that the model can be greatly improved, since it is underfitting, since both train_loss and val_loss are decreasing functions until the end, while the corresponding accuracy is ascending.

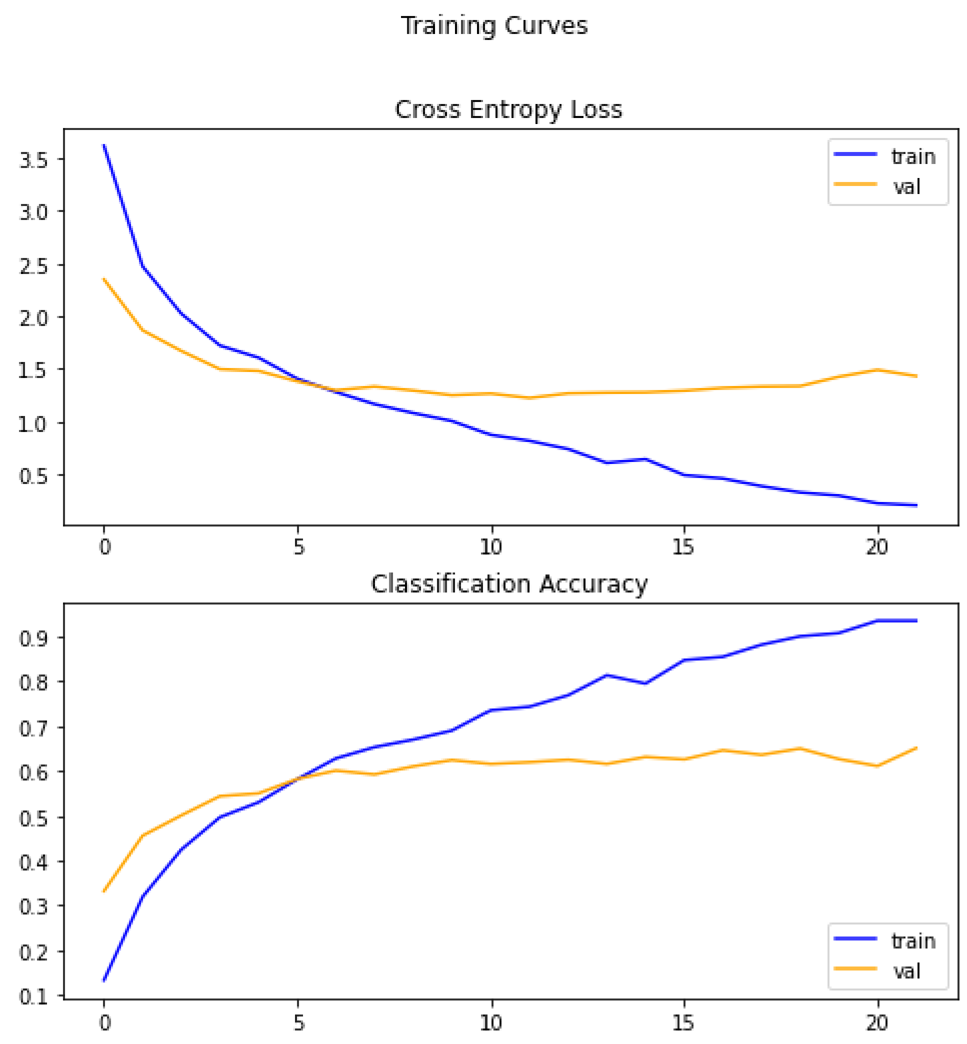
Transfer Learning Models

VGG16



VGG-16 architecture

Having seen the low performance of the above model, we use ransfer learning, in order to see better results this time. More specifically, we use VGG16 which has pre-trained weights and we will add dropout, GlobalAveragePooling2D and a fully-connected. We define the variable trainable = False, in order to train all weights, trained and not. We use Adam optimizer and sparse cross entropy loss function we get the following curves:



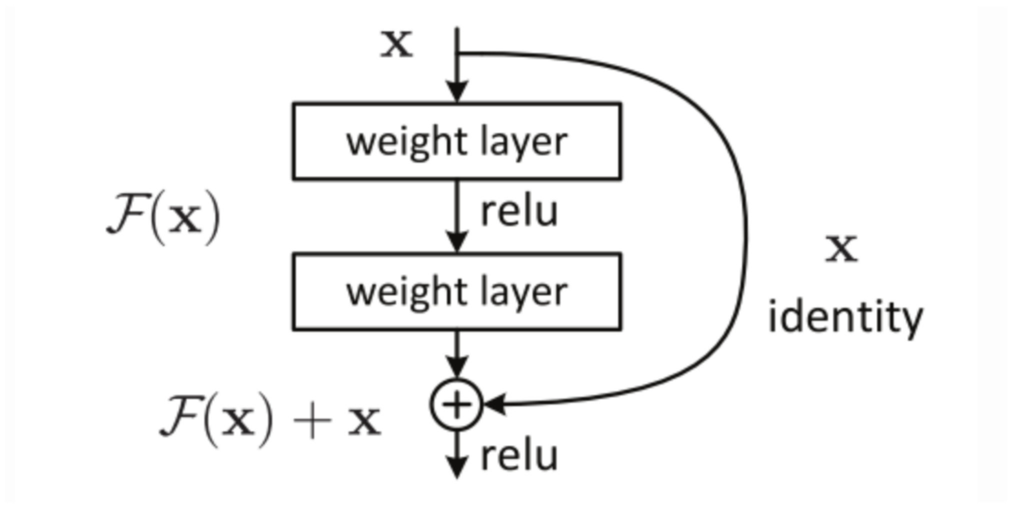
Loss: 1.42
Accuracy: 0.67

Having an accuracy score of 67%, we see that by using transfer learning, we achieve much better performance than in "from scratch".

Optimized Architectures

We try different ones to optimize the techniques we studied above.

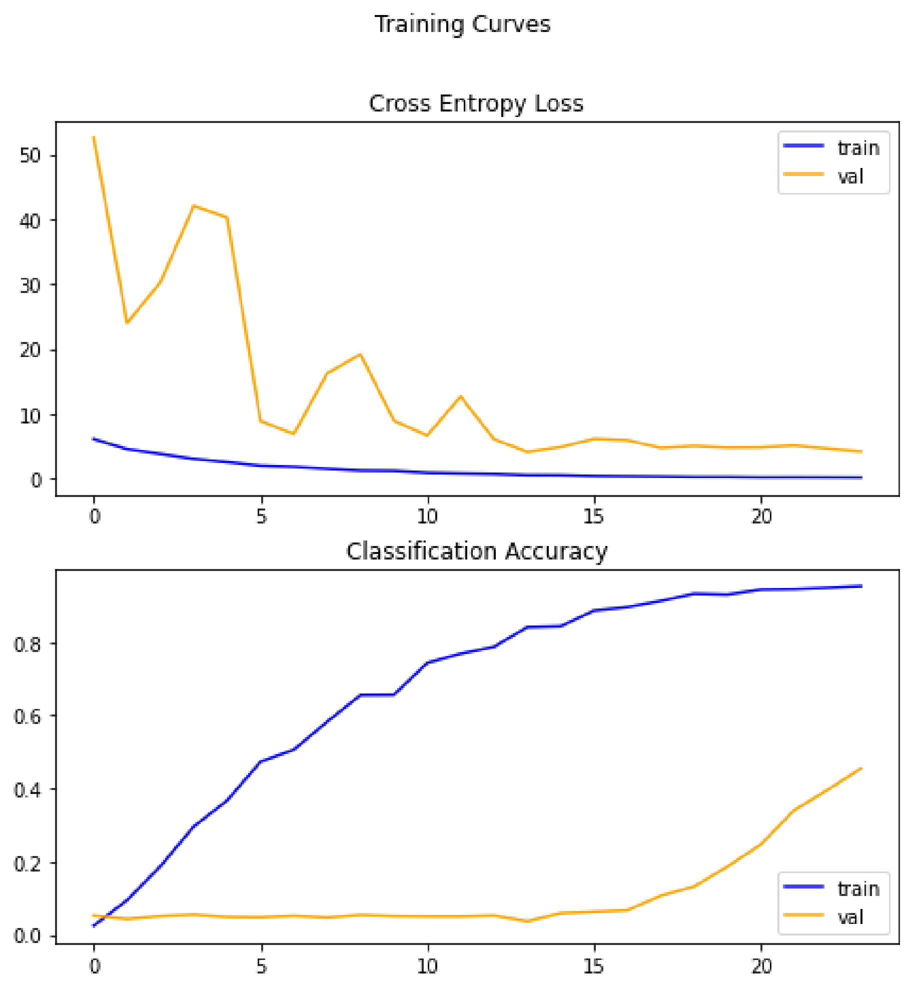
ResNet



x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

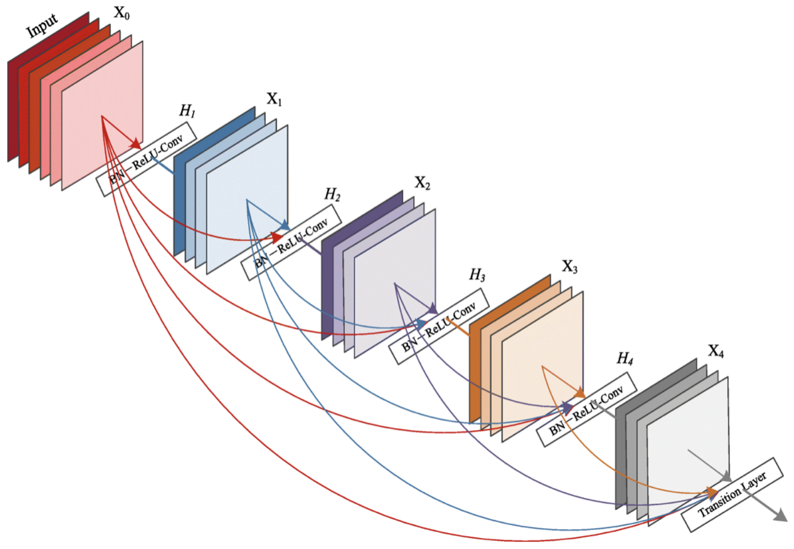
The ResNet model, a powerful backbone model often used in computer vision tasks. The peculiarity of this model is the identity shortcut connection, ie the connection of the result of one layer with a "remote" layer, ignoring the next ones in the series.

Training Curves:



Loss: 4.22
Accuracy: 0.44
The resulting test score was 44%

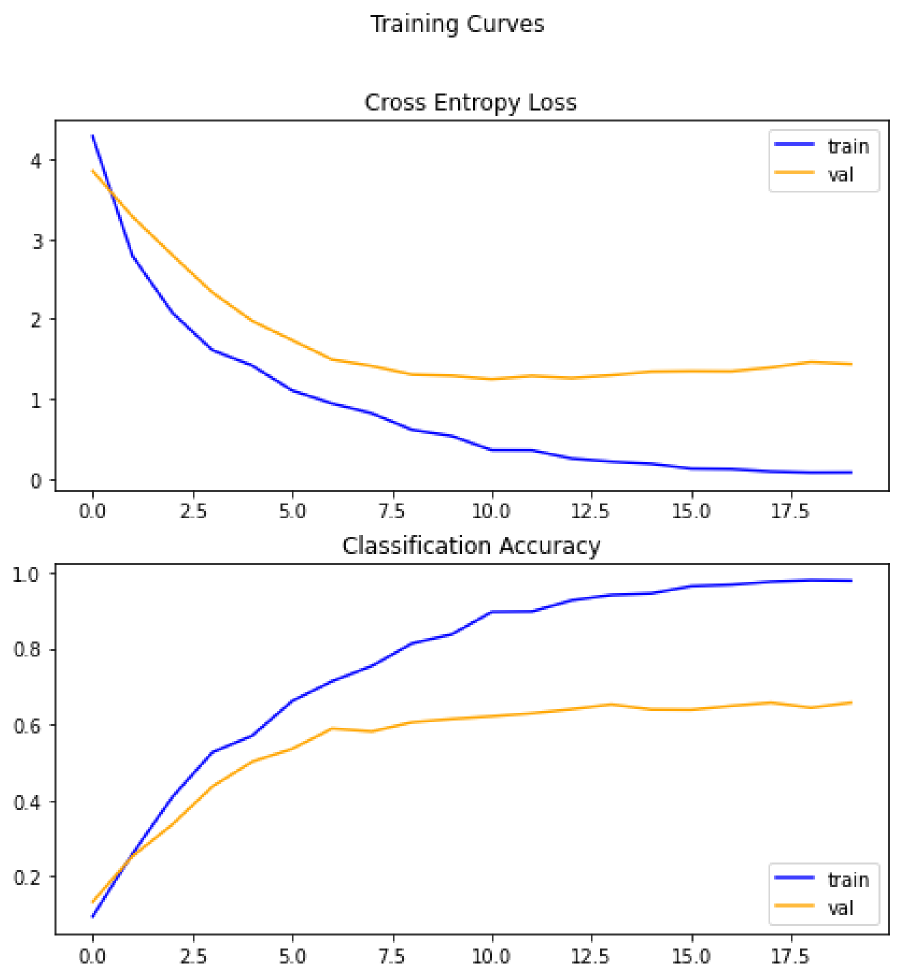
DenseNet



The latest model is DenseNet. Unlike the previous architecture where one level can receive input from any previous one, here it receives multiple inputs from all previous levels, simulating a kind of collective knowledge processing. This allows the network to be simpler in structure and memory requirements and computations. The network architecture includes similar patterns to the previous ones.

The default values of the parameters mentioned above were used for the training of the models with learning transfer. In addition, the weights of the trained networks are kept frozen, ie they do not change during the retraining. The models were uploaded from the TensorFlow libraries and each was supplemented with a Dropout layer to reduce overfitting on ImageNet and a Dense layer for proper retraining.

Training curves:



loss: 1.36
accuracy: 0.67
Test score 67%

Sources

- https://www.geeksforgeeks.org/vgg-16-cnn-model/
- https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the- right-timeusing-early -stopping /
- https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning - model-performance /
- https://brandonmorris.dev/2018/06/30/wide-resnet-pytorch
- https: // machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-timeusing-early-stopping /
- https://datascience.stackexchange.com/questions/29719/how-to-set-batch-size-steps- per-epoch-and-validation-steps
- https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning - model-performance /
- https://towardsdatascience.com/a-comprehensive -guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://scholar.smu.edu/cgi/viewcontent.cgi?article=1091&context=datasciencereview [8]
- https://machinelearningmastery.com/ how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks / I
- https: // benc hmarks.ai/cifar-100
- https://github.com/
- https://www.cs.toronto.edu/~kriz/cifar.html
- Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15, 1 (January 2014), 1929–1958
- K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and
- Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 77
- G. Huang, Z. Liu, L. vd Maaten and KQ Weinberger, "Densely Connected Convolutional Networks," 2017 IEEE Conference on
- Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, 2017, pp. 2261-2269.