Machine Learning and Deep Learning

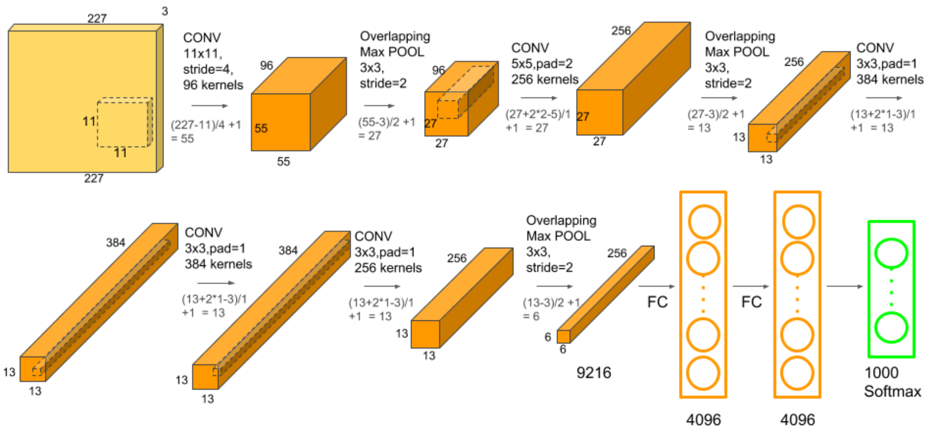
Politecnico di Torino

Homework 2 Report

Student ID: s275090

Convolutional Neural Network for image classification

In this homework we are going to train a CNN for image classification. In this case we use AlexNet. AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers that extract interesting features in an image.



The first two Convolutional layers are followed by the Overlapping Max Pooling layers, used to downsample the width and height of the tensors. The third, fourth and fifth convolutional layers are connected directly. The fifth convolutional layer is followed by an Overlapping Max Pooling layer, the output of which goes into a series of two fully connected layers. The second fully connected layer feeds into a softmax classifier with 1000 class labels.

ReLU nonlinearity is applied after all the convolution and fully connected layers.

The Dataset: Caltech-101

Caltech 101 is a data set of digital images. It is intended to facilitate Computer Vision research and techniques and is most applicable to techniques involving image recognition classification and categorization. Caltech 101 contains a total of 9,146 images, split between 101 distinct object categories (faces, watches, ants, pianos, etc.) and a background category. The size of each image is roughly *300 x 200 pixels.*

The code

The code is divided into 4 steps:

1. *Preprocessing* - In this block we define the transforms for training and evaluation phases. For each of them we resize the images to 256 input and crop a central square patch of the image of 224. Then we turn PIL image to torch.Tensor and finally we normalize tensor with mean and standard deviation.

2. *Datasets* – Caltech 101 is divided into train and test set by folders. We divide the train in half to obtain a valuation set where to evaluate our model after each epoch. We use the StratifiedShuffleSplit to not filter out entire classes from the sets. After that, the samples for train, valid and test are *2892, 2892, 289*3. For this work we filter out the BACKGROUND class.

3. *Training* – We load AlexNet model. It has 1000 output neurons, corresponding to the 1000 Imagenet's classes but we need 101 outputs for Caltech-101. We just changed the last layer with a FC layer with 101 outputs.

   For classification, we use cross entropy.

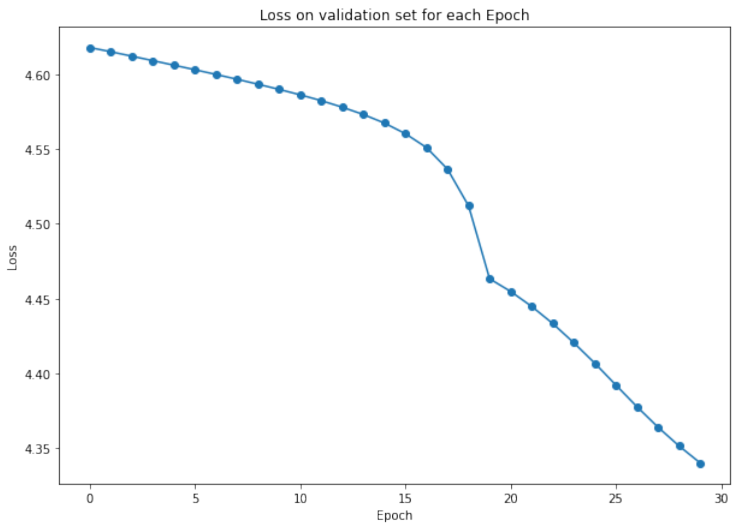
   For optimizer, we use SGD with momentum, and we optimize over all the parameters of AlexNet.

   Finally, for scheduler that dynamically changes learning rate, we use the step(-down), which multiplies LR by gamma every N-epochs.

   After defining the training, we start iterating over the epochs where after each of it we evaluate the model on the validation set.

4. *Validation/Testing* - In this block we use only the best performing model on the validation set for testing.

The current implementation trains using SGD with momentum for 30 epochs with an initial LR of 0.001 and a decaying policy after 20 epochs.



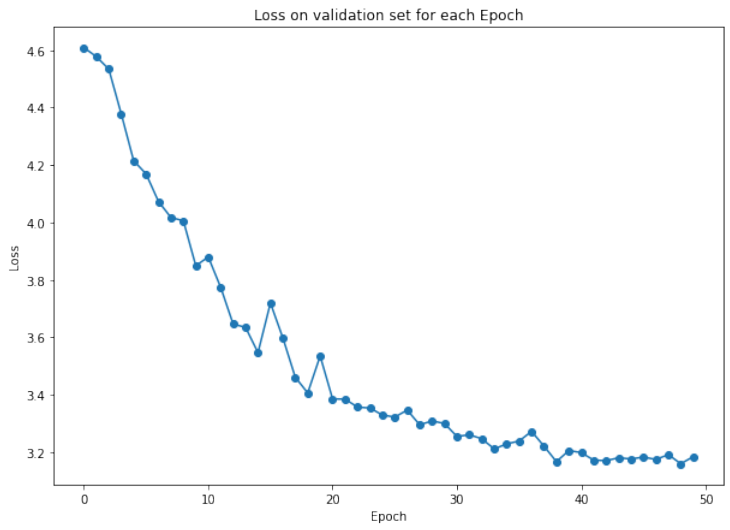
From the graph we can see that the performance of our model is not very good. The loss after 30 epochs decrease but still remains high. Maybe we have a problem of vanishing gradient. In fact, with the best performing model, we get an accuracy of *0.091977* on validation set and *0.0919460* on test set. So, we need to change some parameters.

Experiment

Now we experiment with two different sets of hyperparameters.

1. LR = 0.01 - DECAYING POLICY = 20 - NUM_EPOCH = 50

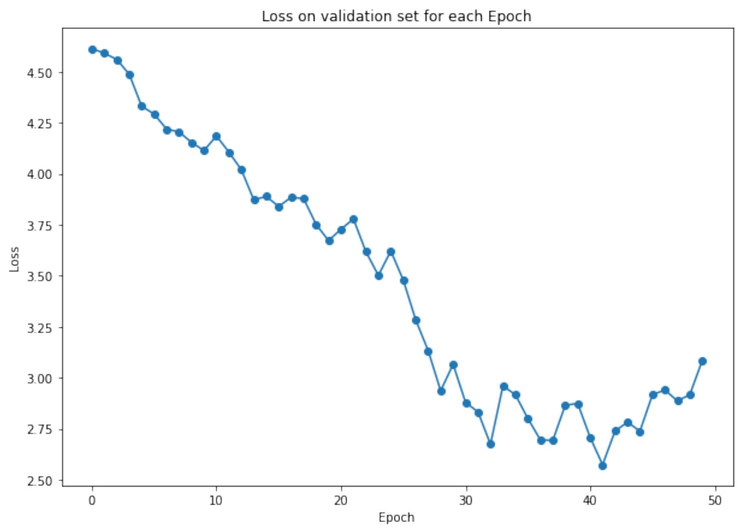
   Our model decreases too slowly, so we need to increase the learning rate to help the model to learn faster and we increase the number of epochs.



From the graph we can see that now the model gets a loss better than before, but after 40th epoch seems to have stuck in a sub-optimal. On the valuation set, with the best performing model, we get an accuracy of *0.32295* and on the test set *0.3294.* Much better than before.

2. LR = 0.01 – DECAYING POLICY = 40 - NUM_EPOCH = 50

   In the first experiment we had a sub-optimal after 40 epochs. Now to escape from this point we increase to 40 the decaying policy.



We can see that the curve goes down to 40 epoch and it starts growing again later. Then, the loss suddenly rises as the model fails to converge. On the valuation set, with the best performing model, we get an accuracy of *0.48443* and on the test set *0.49982*.

Transfer Learning

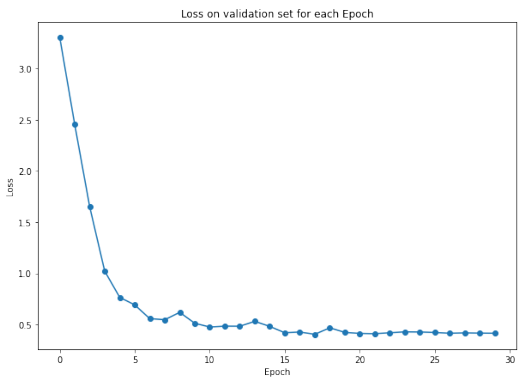
Caltech 101 is a small dataset to train our network and get a good result. Transfer learning can help us. It is a research problem in ML that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem.

We use a pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task.

We load AlexNet with weights trained on the ImageNet dataset (pretrained=True), a large visual database designed for use in visual object recognition software research, and we change the Normalize function of Data Preprocessing to Normalize using ImageNet's mean and standard deviation *[(0.485, 0.456, 0.406), (0.229, 0.224, 0.225)].*

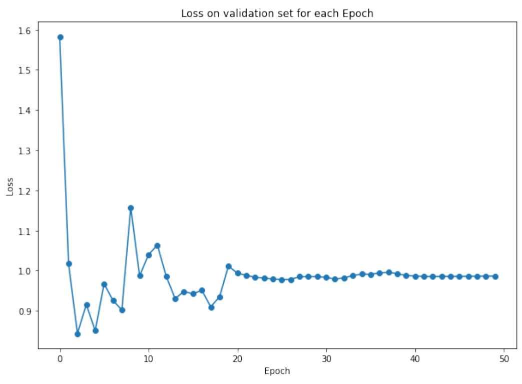
Now, we run experiments with three different sets of hyperparameters:

1 - LR = 0.001 - DEC POLICY = 20 - EPOCH = 30



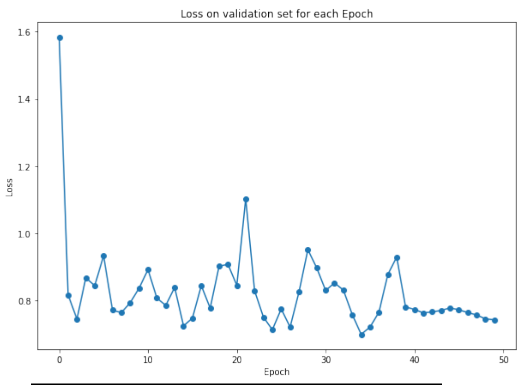
	Accuracy
Validation	*0.83264*
Test	*0.83166*

2 - LR = 0.01 - DEC POLICY = 20 - EPOCH = 50



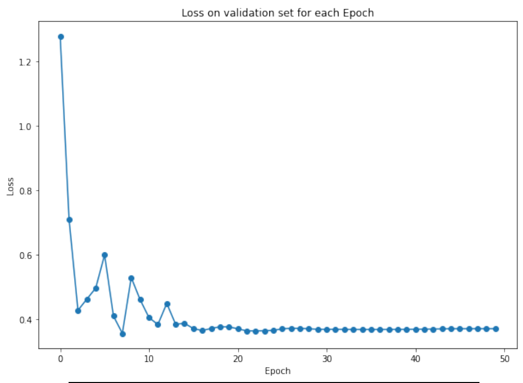
	Accuracy
Validation	*0.8136*
Test	0.8161

3 - LR = 0.01 - DEC POLICY = 40 - EPOCH = 50



	Accuracy
Validation	0.83990
Test	0.84410

4 - LR = 0.01 - DEC POLICY = 15 - EPOCH = 50

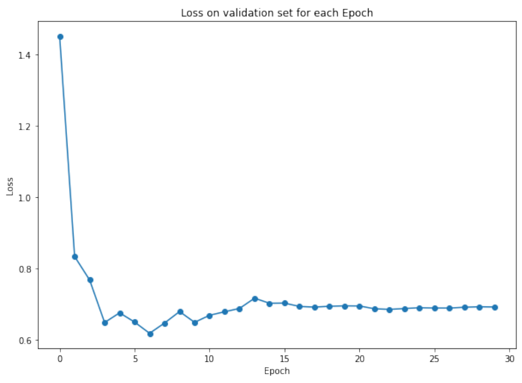


	Accuracy
Validation	0.84094
Test	0.84445

With transfer learning we get much better results, especially in the first experiment where without pre-training we got a bad result. The third experiment finds an optimal accuracy, but we have an oscillating performance, where the weights diverge. Finally, in the last experiment we found the best accuracy and a good curve trend.

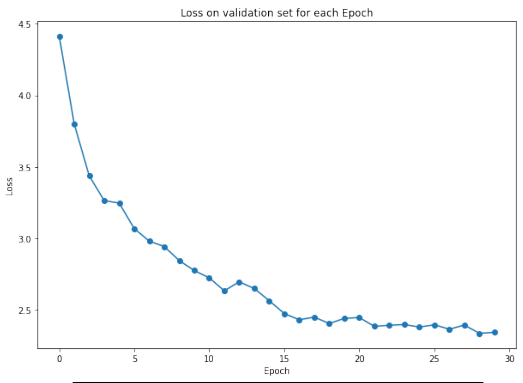
To improve our model, we can also “freeze” part of the network and only train certain layers to prevent overfitting in some cases. So, with the best hyperparameters obtained in the previous step, we experiment only the fully connect layers and then only the convolutional layers.

Optimizing over FC layer



	Accuracy
Validation	*0.*84336
Test	*0.*84618

Optimizing over Convolutional layers



	Accuracy
Validation	*0.*5131
Test	*0.*5274

Freezing only the conv layers, we don’t get an improvement on test and validation test. Initially we have a sharp drop in the loss and the oscillating problem is less evident, but we have a vanishing gradient. On the contrary, freezing only the fc layer, the result worsens, and the descent is slower.

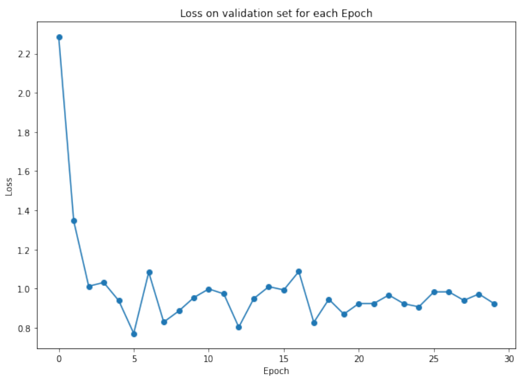
Data Augmentation

There are a limited number of images in some categories, we can use image augmentation to artificially increase the number of images seen by the network. For example, we randomly resize and crop the images and also flip them horizontally.

All of the data is also converted to Torch Tensors before normalization. The normalization values are standardized for ImageNet.

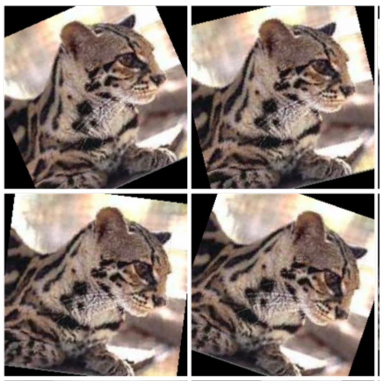
We apply three different sets of preprocessing for training images (from these tests we reduce epochs to 30):

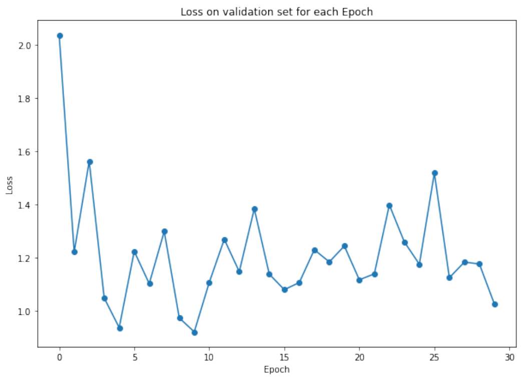
1 – We randomly flip the image horizontally and convert it to grayscale



	Accuracy
Validation	*0.*8385
Test	*0.*7573

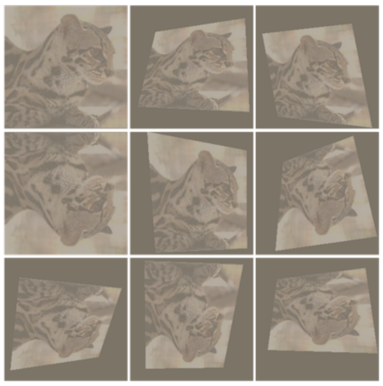
2 – We randomly rotate the image by angle and change the brightness, contrast and saturation

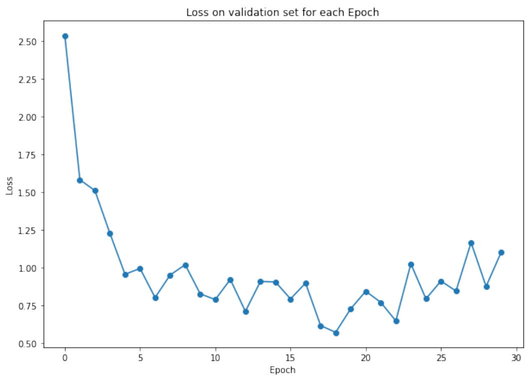




	Accuracy
Validation	*0.*8115
Test	*0.*8285

3 – We randomly flip the image vertically and perform perspective transformation





	Accuracy
Validation	*0.*7655
Test	*0.*6187

None of the sets of preprocessing improved our result and often the weights diverge. Perhaps because the data of test and validation are very similar to training data.

Beyond AlexNet

With AlexNet we got good results, but we had problems of oscillating performance and vanishing gradient. To further improve our result, we will try other networks: ResNet18 and VGG16.

For these networks, we will use the best hyperparameters found (LR = 0.01 - DEC POLICY = 15 - EPOCH = 30).

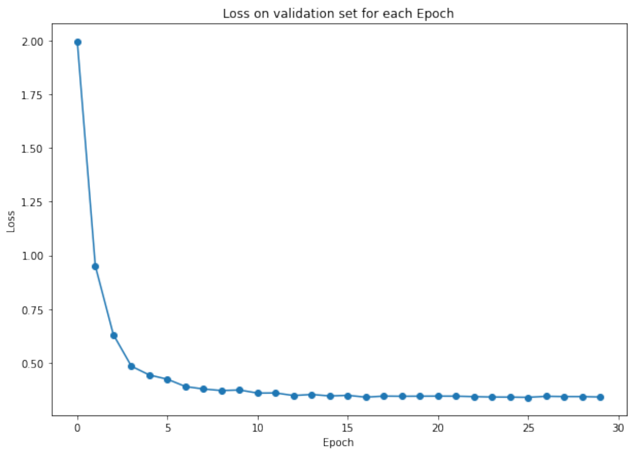
ResNet18

ResNet18 is a convolutional neural network that is 18 layers deep. This network utilizes skip connections to jump over some layers and to avoid the problem of vanishing gradients.

To implement ResNet we have to adapt our code. From this network we obtain an output size of 1000, so we have to change the FC layer to have an output of n classes.

net.fc = nn.Linear(512, NUM_CLASSES)

Also, Resnet consumes GPU memory, so we decrease the batch size to 128.

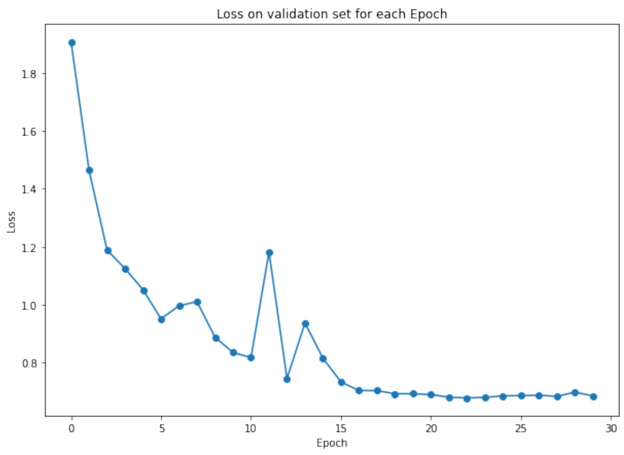


	Accuracy
Validation	*0.*92634
Test	*0.*92810

With this network we get the best result. Moreover, with skip connections we no longer have problems with oscillating performance.

VGG16

VGG16 is makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another. However, there are two drawbacks, it is slow to train, and the network architecture weights themselves are quite large, so we reduce the batch size to 32.



	Accuracy
Validation	*0.86825*
Test	*0.*86795

We get a slightly better result but don't keep the same problems we had with AlexNet.