Machine Learning and Artificial Intelligence Homework2

Lorenzo Vaiani

15 December 2019

1 Abstract

In this study AlexNet CNN is used to solve an image classification problem. The analayzed dataset is called "Caltech-101". Different combinations of learning rate and step size are used to train and evaluate the model. The experiment is repeated applying transfer learning and data augmentation techniques. Moreover, for transfer learning the final training is repeated three times, each freezing different layers. For data augmentation three different sets of transformation are tested. At the end, another CNN, called VGG11, is used to make a comparison with AlexNet.

2 Convolutional Neural Network

Convolutional Neural Networks (CNN) are a family of neural networks widely used in computer vision. A CNN is a Deep Learning algorithm which can take in an input image, assign importance to various aspects in the image and be able to differentiate one from the other. CNNs follow a layered architecture, typically non-cyclical, which levels can be divided in two main different groups: the feature extraction layers and the classification layers.

The feature extraction layers perform operations to detect image features. Their role is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. The features are not detected looking at the entire images but dividing them in small patches analyzed one at the time. There are several type of operations executed by as many types of layers:

- Convolutional layers, which are the most important layers and from which the CNNs take their name. The operation of these layers is as follows: given an input image, representable as a three-dimensional matrix (two for height and width and one for RGB color values, called depth), a second three-dimensional matrix, called filter, is slid on the image one in order to perform the convolution and calculate the result. Height and width

of the filter are smaller than the original image ones, while the depth is the same. At each step of the convolution the filter slide a distance called stride. Before executing convolution it is possible to insert one or more additional zero-value pixel layers to the edge of the image: it is useful to prevent the image shrinking, since a convolutional step returns a single value for multiple input values analyzed, and to make adequate filter dimensions and stride to the input image. A single convolution layer can apply even more filters to the image. Height and width of the result depend on filter size, stride and padding, while its depth depends on the number of filters applied by the convolutional layer. Given these information it is possible to calculate width W_r and height H_r of the output:

$$W_r = \frac{(W_i - W_f)}{S} + 1 \qquad H_r = \frac{(H_i - H_f)}{S} + 1 \tag{1}$$

where W_i and H_i are width and height of the input image, W_f and H_f are width and height of the filter and S is the stride.

In general, looking at the network architectures, the first convolution layers extract low level features, such as directions, edges and colors. Continuing in the structure these layers will identify increasingly specific features in relation to the type of images they are analyzing.

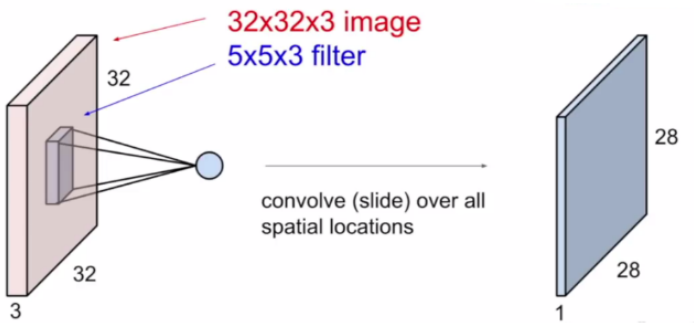

Figure 1: convolution of a 5x5x3 filter over a 32x32x3 image, with stride equals to one and no padding.

- Pooling layers, which are responsible for reducing the spatial size of the extracted features. This is done to decrease the computational power required to process the data through dimensionality reduction. Furthermore, they are useful for extracting dominant features which are rotational and positional invariant. It is common to find one of them after a convolutional layer. These layers work by separately considering various parts of the input matrix and each time selecting a specific value among those currently analyzed (or a combination of them). The pooling layers in the network used in this study perform a specific type of pooling called Max-Pooling: each element of the resulting matrix is the maximum value of

the starting sub-matrix. Width and height of the result can be calculated with the same formulas as the convolutional layers, while the depth is the same as the input. This kind of pooling is able to reduce the output size keeping the relevant information and without adding any new parameter.

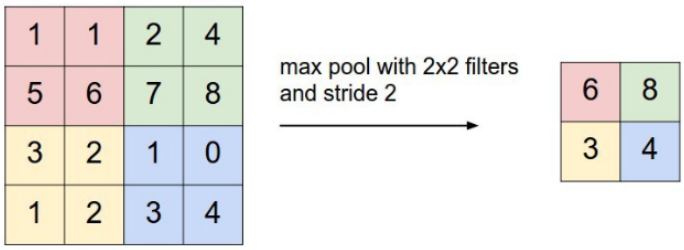

Figure 2: max pool operation example.

- Activation Function layers, which introduce non-linear properties to the network and can also be found between classification layers as for common neural networks. They apply an activation function, that should be differentiable, to transform the output of the previous layer and then send it to the next layer as input. The Rectified Linear Unit (ReLU) function is the most used one. It is very computational efficient because it does not activate all neurons at the same time: it converts all negative inputs to zero and the neuron does not get activated. The fact that this is just one simple operation explains why it is efficient. ReLU does not saturate at positive region, this mean that the gradient will be different from zero for that region, while instead it will be for the negative one.

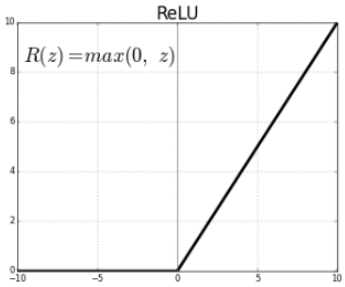

Figure 3: Rectified Linear Unit function representation.

Fully connected layers are used once the input image has been converted into a suitable form for a feed-forward neural network. The high level features extracted by previous layers are flattened into a column vector and used to feed the fully connected part of the network with the purpose of being able to correctly classify the initial image using a Softmax classification technique.

In general, the images in the input dataset are grouped in batches and the CNN is trained for many epochs during each of which the following operations are done, once for each batch:

- Forward pass operation, where the image are processed and classified by the network.
- Loss calculation, where the chosen loss function is calculated using the predicted labels obtained in the previous step.
- Backward pass operation, where the gradient of the loss function is computed. This is done with respect all neurons in the network, starting from the output ones up to the input ones exploiting the chain rule. During this process, the weights of neurons are updated in a certain way depending on the optimization algorithm used.

The CNN used in this study is called AlexNet. As input it requires RGB images of size 224x224. It is composed by five convolution layers each, one followed by a ReLU activation function, three max pooling layers and three fully connected layers. The last fully connected layers is the output layer of the network and it is composed by 1000 neurons that represent the 1000 output classes of ImageNet, the dataset for which AlexNet was designed. The structure of these layers is showed in figure 4.

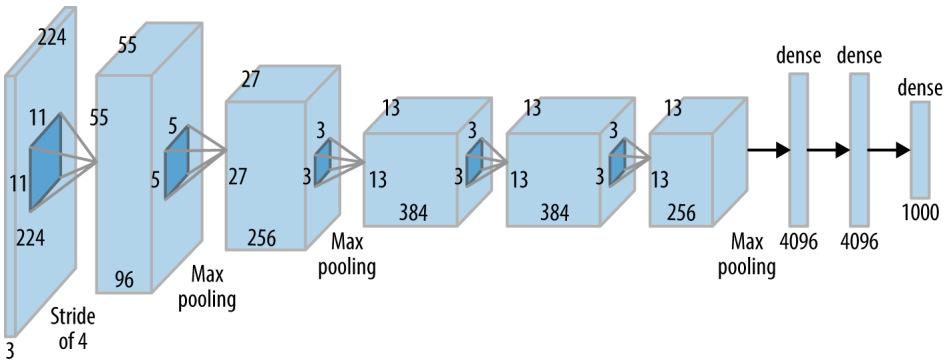

Figure 4: AlexNet layers structure.

3 Data Preparation

In this homework the dataset of images called Caltech-101 is used: it contains almost 10000 images that belong to 102 different classes. One of these classes is a background class (called "BACKGROUND_Google") that is not considered in this study and for this reason it is removed from the dataset, thus obtaining a total of 8677 images. To do this an ad-hoc class called "Caltech" is implemented, starting from the provided template, and it is used instead of the class

"ImageFolder" already available in the pytorch library. This ad-hoc class Caltech follows the "DatasetFolder" pytorch class features whose objects contains information such as the list of image names and the list of corresponding class labels. The only difference is that those list are checked and the images related to the background class are filtered out.

The creation of a Caltech object, in addition to requiring the dataset from which to upload the images, also requires the transformations to be applied. To begin with, the only transformations used are those that are useful for making images suitable as inputs for an AlexNet CNN. This means that the images are resized to a dimension slightly larger than necessary, cropped in the center to the exact size and transformed in to tensors whose values are finally normalized between -1 and 1. This normalization operation is done subtracting from the input values of a channel the mean value of that channel and then dividing by the standard deviation of that channel (mean and standard deviation are both input parameters of the transformation).

The function "make_datasets()" create four different Caltech objects, taking in input the transformations to be associated with each of them. The four objects represent train set, validation test, union of train and validation sets and test set. The function operates in the following way: at first the provided split files are used to divide the entire dataset in train set and test set, respectively two third and one third of the total. Then the train set is again separated into train set and validation set, one half of the original train set each, in a stratified way, that is the percentage of elements belonging to a particular class is the same of the original train set. Since the elements are ordered by class, this is done by selecting the elements of the validation set alternately from the original train set.

Each Caltech object is necessary to create the corresponding "DataLoader" pytorch object, that provide an iterable over the dataset. In these objects the images are loaded, shuffled and grouped in batches. The droplast flag is also activated for dataloaders associated with the train set and the union of train and validation sets, this mean that if the number of images is not divisible by the batch size the last incomplete batch is dropped and its images will be used to train the network in the next epoch. Now the images are ready to feed and evaluate the CNN.

4 Training AlexNet

The function "training_phase()" deals with the training of the network to looking for the best hyperparameters and return them. It is possible to specify the dataloaders to train and evaluate the model, the combinations of hyperparameters to evaluate, the maximum number of training epochs and if it is necessary to use a pretrained model or to freeze some layers of the network. Pytorch provides an already existing implementation of AlexNet but in this step the untrained version is used. AlexNet output layer must be adapted to the 101 class labels of Caltech-101 dataset: to do this the amount of neurons of the last

fully connected layer is decreased from 1000 to 101. Then the loss function, the parameters to optimize, the optimizer and the scheduler are selected. In this study the Cross Entropy Loss implemented by Pytorch is chosen:

$$loss(x, class) = -log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right) \tag{2}$$

The function "get_params()" selects the parameters to optimize. In this step the parameters of both convolutional and fully connected layers are considered but it is also possible to specify which parameters to freeze.
The optimizer holds the current state and updates the parameters based on loss value. To do this the SGD optimization algorithm is used specifying the learning rate (η), momentum (α) and weight decay values. This algorithm update the weights with the following formula:

$$w_{t+1} = w_t - \eta\frac{1}{n}\sum_{i=1}^{n}\nabla Q_i(w_t) + \alpha\Delta w_t \tag{3}$$

where n is the number of batches analyzed at each epoch and Q_i is the loss function computed after the i-th batch observation.
The scheduler adjusts the learning rate in a dynamic way based on the numbers of epochs and the Step Scheduler is used specifying step size and gamma values. This scheduler update the learning rate periodically, multiplying it by gamma after a number of epochs equals to the step size. The number of epochs, the batch size and all the above mentioned parameters for the optimizer and the scheduler are considered the hyperparameters of the neural network.

Next, the network is trained at most for the fixed number of epochs. During each epoch, for each batch in the train dataloader the loss is calculated and the weight of the network are updated. Then, at the end of each epochs the model is evaluated on the validation set, calculating the prediction accuracy. If this accuracy is the highest found so far, the current epoch, the initial learning rate and the step size are stored as best values for those parameters. The training can end before having completed the number of maximum epochs if an early stop criterion is satisfied: every ten epochs the accuracy calculated is stored and if the accuracies recorded during the last three checks (the current one included) have decreasing values the training ends.
After that, the function "final_training()" uses the best hyperparameters values found to create a new model but this time exploiting the images of both train and validation set to train it, without any kind of evaluation. An alternative strategy could be to take the model obtained during the initial training phase and continue training it with only the validation set images, in order to make this step faster.
Eventually, the function "evaluating_phase()" use the test dataloader to evaluate the final model generated by the previous function in order to obtain the accuracy that characterizes it.

In this study most of the hyperparameters are set a priori: the maximum number of epochs is set to 100, the batch size is set to 256, the gamma for

the scheduler is equal to 0.1 and the weight decay and the momentum for the optimizer are equal respectively to $5 * 10^{-5}$ and 0.9. Only some combinations of learning rate and step size are tested in order to identify the best values. Moreover the early stop criterion can be useful to discover a more adequate number of epochs to train the final model.

The tested combinations are:

- $Learning\,Rate = 0.1$ and $Step\,Size = 20$: after 15 epochs the loss takes the NaN (Not a Number) value. This happen due to the high value of the learning rate which increases the loss until it diverges to infinity. During those 15 epochs the accuracy calculated on the validation set is quite bad and never exceeds 20%.

- $Learning\,Rate = 0.01$ and $Step\,Size = 30$: the training proceeds until 111th epoch where the early stop criterion is satisfied and the training ends, reaching the highest accuracy of 48.34% at the 91st epoch. Noteworthy the fact that the accuracy exceeds 45% in less than 50 epochs and takes as much time to earn what it lacks to reach the maximum found. This happens because of the learning rate is decreased by the scheduler every 30 epochs and this means that the convergence is getting slower.

- $Learning\,Rate = 0.01$ and $Step\,Size = 45$: the training end at the 131st epoch due to the early stop criterion. The highest accuracy is reached during the 117th epoch and is 53.32%. The initial fast increase in accuracy is similar to the one of the previous combination but it lasts longer because of the higher step size. Subsequently the best accuracy found with the previous combination is exceeded and after some epochs without any improvement the training ends.

- $Learning\,Rate = 0.001$ and $Step\,Size = 60$: the training end due to the early stop criterion after 31 epochs, during which the accuracy value fluctuates between 15% and 9%. This happens probably because the learning rate is to low and the convergence is to slow.

The best hyperparameters values highlighted by the third combinations and the number of epochs spent before finding the best accuracy are used to retrain the network over the images of both training and validation sets. The test set is used to evaluate the obtained model and the result is an accuracy of 65.61%.

5 Transfer Learning

Transfer learning is a machine learning technique that consists in reuse for a task a model previously trained for another task. This model will be the starting point to solve the new task. Using the knowledge learned during the resolution of another problem brings several benefits: the results are better and faster to get and the pretrained model helps the user to solve the problem of finding the high number of images to decently train the CNN. Transfer Learning is

therefore an ideal choice when the amount of avalaible data to solve a task is not sufficient.

The approach of transfer learning used in this study is called "Fine Tuning": a pretrained model is loaded and its layers are not immediately exploited for classification, but they (or only some of them) are retrained together with other possible untrained new layers. For example, it is possible to decide to maintain the weights (freezing operation) in the layers that extract the low level features and retrain the high level features extractor layers.

Caltech-101 is a very small dataset so it is not the best choice for deep learning tasks. In this case a possible solution is to apply the fine tuning transfer learning technique: Pytorch provide a pretrained model of Alexnet that has as starting point the weights learned from training on Imagenet dataset, a dataset bigger than Caltech-101. To do this most of the operations performed in the previous step are repeated but with the pretrained AlexNet model. The transformations applied to the images are slightly different: the pretrained model expects images normalized in the same way of those of ImageNet, so in the normalization transformation it is necessary to use for the three channels mean values equal to [0.485, 0.456, 0.406] and standard deviation values equal to [0.229, 0.224, 0.225]. The tested combinations of hyperparameters are the same as the previous step:

- $Learning\ Rate = 0.1$ and $Step\ Size = 20$: the loss reaches the NaN (Not a Number) value during the first epoch. This happen because the learning rate is definitely too high.

- $LearningRate = 0.01$ and $StepSize = 30$: the training proceeds until 71st epoch then the early stop ends the training. The highest accuracy reached is 86.44% at the 47th epoch. Noteworthy the fact that the accuracy is much higher than the one obtained without transfer learning (exceeds 80% within the first 3 epochs).

- $Learning\ Rate = 0.01$ and $Step\ Size = 45$: the training end at the 141st epoch due to the early stop criterion. The highest accuracy is reached during the 75th epoch and is 87.34%. The initial fast increase in accuracy is similar to the one of the previous combination but it lasts longer because of the higher step size. Subsequently the best accuracy found with the previous combination is exceeded and after some epochs without any improvement the training ends.

- $Learning\ Rate = 0.001$ and $Step\ Size = 60$: with this combination the training ends after 141 epochs and the accuracy is around 84%. It is good but not enough compared with the others already reached, because probably the starting learning rate is too small.

The best hyperparameters values highlighted by the third combinations and the number of epochs spent before finding the best accuracy are used to retrain the network over the images of both training and validation sets. The test set is used to evaluate the obtained model and the result is an accuracy of 86.69%.

This accuracy is much higher than the one obtained with the untrained AlexNet model and this confirms the usefulness of the applied transfer learning technique.

The best hyperparameters obtained from this transfer learning experiment are also used to retrain two more pretrained models but this time freezing some of their layers: the first one is trained freezing the fully connected layers and the second one is trained freezing the convolutional layers. In the first case the accuracy on the test set is 71.31% while in the second one the accuracy is 86.86%. The result obtained freezing the fully connected layers is considerably lower probably because of the substitution of the output layer: it is loaded in its pretrained version, then it is replaced with the one compatible with the 101 class label of Caltech-101 and finally it is frozen without ever being subjected to training and consequently without its weights ever being updated.

6 Data Augmentation

The more the data, the better the machine learning models will be, but data collection process has a cost (money, time, human effort). Therefore, it sometimes could be necessary to augment existing data to increase the data size that we feed to our CNN. Data Augmentation is another technique that allows to exploit datasets that are not sufficiently large. Data augmentation exploits the invariance property of CNNs according to which a network is able to classify an object independently of its size, shade of colors and orientation in the image. One way to do this is to manipulate the image as first step in order to generate new images which will be considered part of the data set (offline data augmentation). Another approach is to apply transformations to the images of a single batch just before feeding the CNN and then repeat this process for all the batches (online data augmentation). However, it is possible to improve the performance of the model by augmenting the data already present in the dataset.

To do online data augmentation on Caltech-101, training, evaluation and test processes are repeated as in the previous steps, but using three different sets of transformations. In addition to the transformations already used, the following ones have been added:

- "RandomResizedCrop()", that crops the image to random size between 80% and 100% of the original size in a random position and finally resize the cropped image to the specified dimensions, that in this case are 256x256 (it substitutes the "CenterCrop()" transformation previously used).
- "RandomRotation()", that rotates the image of a random angle between the range $(-\Theta, +\Theta)$, where Θ is specified as function parameter. In this study Θ is equal to 30 degrees.
- "ColorJitter()", that changes the brightness, contrast and saturation of an image.

- "Grayscale()", that converts an image to grayscale.
- "RandomHorizontalFlip()", that flips the image in horizontal way.
- "RandomVerticalFlip()", that flips the image in vertical way.

The pretrained model is loaded and it is trained without freezing any layer and evaluated in order to choose the best hyperparameters values from those already used. The compositions of transformation used are:

- RandomResizedCrop + ColorJitter + RandomVerticalFlip: the results are similar to the ones without the use of data augmentation. The best accuracy found is 82.26% with a starting learning rate of 0.01. This accuracy is reached with a step size of 30, and the experiment with step size equals to 45 does not make improvements. The learning rate equals to 0.1 is still too high, while using a starting learning rate of 0.001 the accuracies are still good, between 80% and 81%. Training the final model with the best combinations of hyperparameters found, the accuracy reached on the test set is 84.61%, better than the one obtained during the evaluation, such as the previous step.
- RandomRotation + GrayScale + RandomHorizontalFlip: the best accuracy found is 80.42% with a starting learning rate of 0.01 and step size equals to 30. Again the learning rate equals to 0.1 is too high and the one equals to 0.001 is slightly less efficient. The only difference from the previous transformation composition is that the accuracy is a couple of points lower. This happen probably because the transformations used here hinder the training more than the previous ones. The accuracy on the test set is 82.99%.
- RandomResizedCrop + RandomRotation + (ColorJitter or GrayScale) + (RandomHorizontalFlip or RandomVerticalFlip): the best accuracy found is 81.74% with a starting learning rate of 0.001 and step size equals to 60. This time this combination is better then the ones with a starting learning rate of 0.01, which in any case always maintain good accuracies. The accuracy on the test set is 84.92%.

All these results are a little lower than the one found with transfer learning but without data augmentation. In general, this can happen for several reason: training may need to take more time (training epochs) and/or images or alternatively when the test set is very similar to the train set, as in this case, the transformations may introduce incorrect patterns that damage the classification result.

7 VGG11

To conclude this study, a more modern CNN called VGG11 was tested. This network is part of the VGG (Visual Geometry Group) networks family and has

a more complex structure than AlexNet and whose result are notoriously better than those of AlexNet. Another feature of VGG networks is that the all filters used by convolutional layers have 3x3 dimensions and their application never reduces the input size. This reduction is performed only by MaxPooling layers. VGG11 contains seven convolutional layers, each followed by a ReLU activation function, five max polling operations and three fully connected layers. The order of these layers is showed in figure 5.

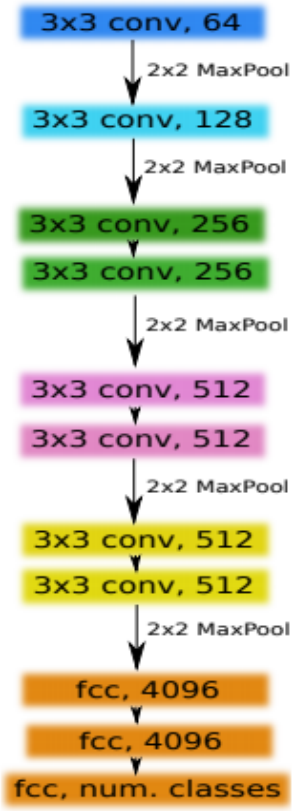

Figure 5: VGG11 layers structure.

In this experiment the VGG11 model is trained with and without the use of transfer learning in order to compare the results with those of AlexNet. The same combinations of hyperparameters are tried and the number of maximum epochs is decreased to 100, because the computation time is very high due to the high number of layers, despite the 3x3 filters simplify the calculations to be performed. Eventually, in order not to saturate the memory of Colab environment given that VGG has a greater number of parameters to optimize with respect to AlexNet, the batch size is reduced to 16.

Training the model without transfer learning the best result is a 52.97% accuracy obtained in 59 epochs with an initial learning rate of 0.01 and an initial step size of 30. Increasing the step size to 45 or decreasing the initial learning rate to 0.001 does not make any improvements. Using the test set to evaluate the final model an accuracy of 62.63% is obtained which is at the same level as that obtained from AlexNet.

Training the model with transfer learning the best result is a 92.63% accuracy obtained in 44 epochs with an initial learning rate of 0.001 and an initial step size of 60. All the other combinations of hyperparameters with an higher learning rate have obtained a NaN (Not a Number) value for the loss during the training. Using the test set to evaluate the final model an accuracy of 91.67% is obtained. This accuracy is pretty much higher than the one reached by AlexNet and this confirms the fact that VGG is a better network. Repeating the final training freezing the convolution layers first and then freezing the fully connected layers the accuracy respectively remains almost the same (92.33%) and loses about 10 percentage points (81.54%), as it happens for AlexNet.