Valerio Di Eugenio
S278972

# Assignment 2, Report

The task of this assignment is to evaluate the performances of a net in images classification by applying different strategies in order to achieve a deep understanding of convolutional neural networks. In details, it is used pre-built AlexNet architecture to perform classification on Caltech-101 dataset, composed by more than 9000 images, (split between 101 categories + a Background one to be filtered out) each of about 300x200 pixels. Training and test sets to work with are already provided. Their distribution across the categories is not balanced at all, some of them have around 800 and 435 images, while the majority around 50/60. This unbalancing leads accuracy not to be the best metric to evaluate models, so for this assignment loss will be considere for this purpose.

# DATA PREPARATION

AlexNet expects 224x224 square images as input. In order to fit it, Caltech-101 dataset has been preprocessed through an ad hoc class, where images are firstly resized to 256 pixels and then center-cropped to the desired dimensions. In addition, when images are uploaded in memory, all their channels are normalized to have zero mean and standard deviation of one, in order both to limit the effect of vanishing gradient on the input data and to try to achieve best performances (in fact neural networks are less robust while processing non-centered dataset). Finally, in this preprocessing phase , the 'BACKGROUND_Google' class is filtered out, making the dataset have effectively 101 classes.

After having read the images paths contained in 'train.txt' and 'test.txt'and stored them in appropriate data structures, the initial training set is split into a new training set and a validation set (making them of the same size), paying attention to preserve a balanced percentage of samples of each class between the two sets, in order to make evaluations statistically significant. Therefore, the the situation is the following :
**Training set**: 2892 images.
**Validation set**: 2892 images.
**Test set**: 2893 images.

# TRAINING FROM SCRATCH

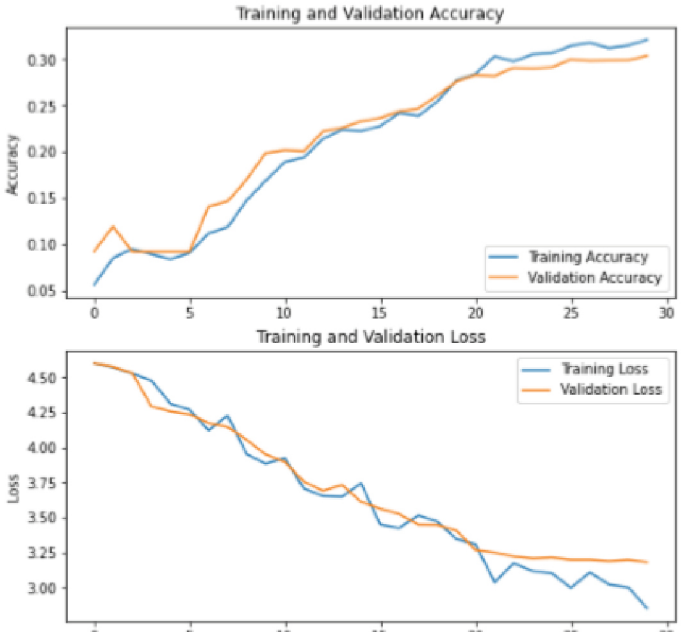
As mentioned before, data are processed through the AlexNet structure: AlexNet consists of 5 Convolutional Layers and 3 Fully Connected Layers. The first two Convolutional layers are followed by the Max Pooling layers, while the third, fourth and fifth convolutional layers are connected directly. The fifth is followed by an Average 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.

Here last classifier is replaced to a new fully connected one with 101 outputs (the number of Caltech-101 classes). ReLU nonlinearity is applied after all the convolution and fully connected layers. Also, Dropout regularization with probability 0.5 is used on the first two FC layers. After each training epoch, the model is evaluated in terms of loss ( Cross Entropy Loss is used in this assignment) and accuracy on the last batch of both validation set (to have an idea about how well or bad our model is generalizing) and training set (to check how well or bad the model is learning), by considering they would present the best performances of each epoch. It is not the most precise value to generalize the result of the entire epoch, but printing every outputs in a chart it is possible to monitor the trend of the resulting curves to analyze the goodness of the model, that matters. The first implementation trains using SGD with momentum for 30 epochs with an initial learning rate of 0.001 and a decaying policy after 20 epochs; the following are the results (the values reported are the mean of three runs, standard deviation here is not reported as pretty small values occurred and they don't affect analysis at all):

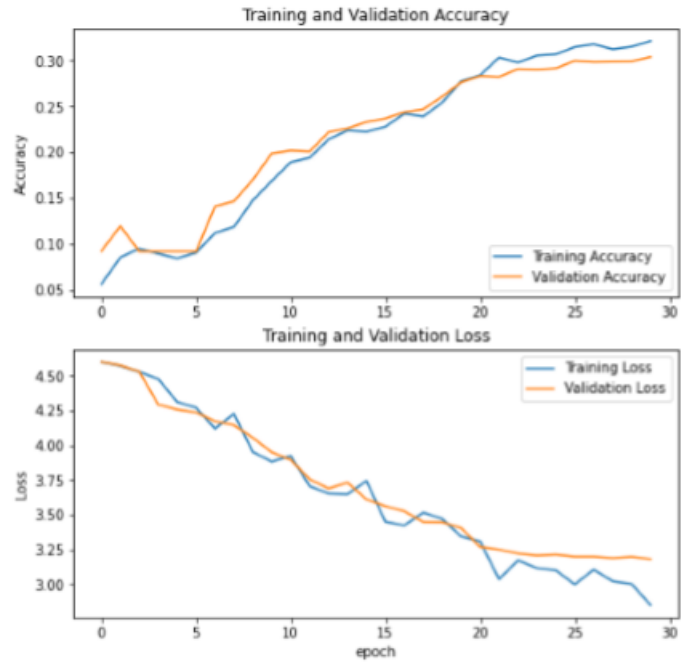

The charts suggest that the model is underfitting the data: the training loss doesn't converge fast, at the end the cost is 4.49 , suggesting that the model is not able to learn the training set at all. The learning rate is too low and the epochs are not enough to allow the model to obtain a sufficiently low error.

In fact, validation and training accuracies are about zero from the first few epochs until the end. A possible solution is to increase the learning rate, the number of epochs or both. Hence, the following hyperparameters are selected to be combined to train the net and check its behavior: -Learning rate=[0.01, 0.05]

- Epochs = [30, 50]. When running with 50 epochs, the policy decay is applied after 30 epochs (also incresing the step size could be a way to improve the performances, as previously the model was even forced to slow down the learning task after 20 epochs)
The charts below show the performances of the nets, starting from those trained on 30 epochs:




- LR: 0.01
- EPOCHS: 30
- STEP SIZE: 20
- TRAININGLOSS: 2.81
- VALIDATION LOSS: 3.20
- TRAINING ACCURACY: 34%
- VALIDATION ACCURACY: 29%

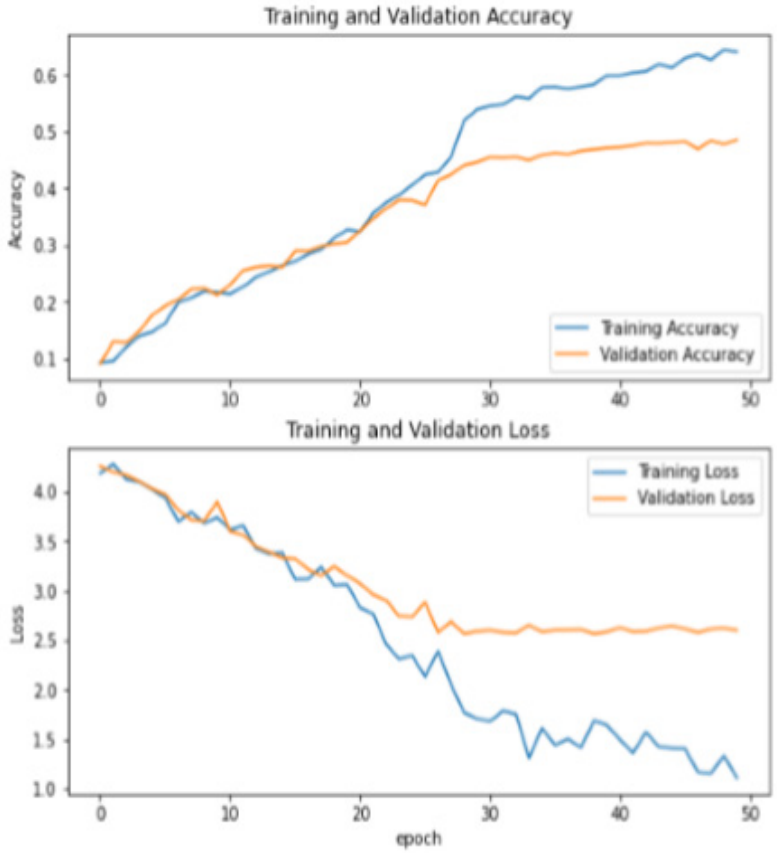



- LR: 0.05
- EPO CHS: 30
- STEP SIZE: 20
- TRAININGLOSS: 2.84
- VALIDATION LOSS: 3.21
- TRAINING ACCURACY: 32% -
VALIDATION ACCURACY: 30%

The charts indicate how performances of our net improve in values when increasing the learning rate. The curves show a similar behavior on both training and validation sets, even though these seem to continue to decrease (in case of loss) and increase (in case of accuracy) at the end of the charts. Moreover, the training loss doesn't converge and presents high values; these facts suggest that perhaps the model is still able to learn more from the data, the training phase could have kept on for more epochs.

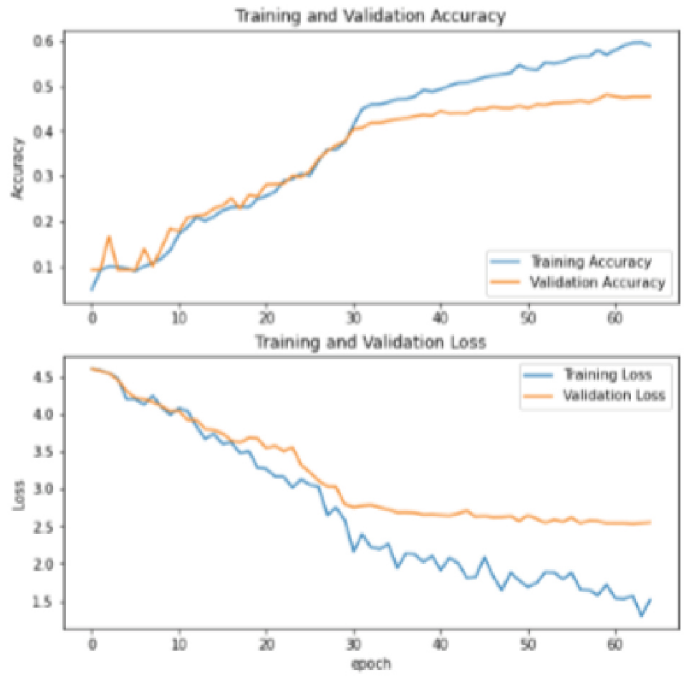

- LEARNING RATE: 0.05
- EPOCHS: 50
- STEP SIZE: 30
- TRAINING LOSS: 0.40
- VALIDATIO N LOSS: 4.36
- TRAINING ACCURACY: 0.91
- VALIDATION ACCURACY: 0.48

By increasing the epochs and maintaining the same learning rate as before, it is possible to notice that our models start overtaining: the loss validation curve starts increasing (in case of loss) from a determined epoch on, while training's keeps on its own decreasing trend. These different behaviors on both train and validation sets indicate that the models have learned the train dataset too well (training loss in fact converges to a low value), but it is not able to generalize to new data, resulting in an increase of generalization error (as mentioned, the validation loss is high in values).




- LR : 0.01
- EPOCHS: 50
- STEP SIZE: 30
- TRAININ G LOSS: 1.69
- VALIDATI ON LOSS: 2.57
- TRAINING ACCURACY: 57%
- VALIDATION ACCURACY: 46%

With learning rate of 0.01 the model doesn't fall into overfitting scenario: the validation loss doesn't start increasing but just flattening (it is possible it's stuck in a point with low gradients). Moreover, the training loss decreasing trend at the end points that the training should be kept on for even more iterations, in order to allow the training loss to converge to a small value (as the trend seems to suggest) and, consequently, to check the behavior of the validation loss. Making the model run for 15 more epochs, the following result is achieved:

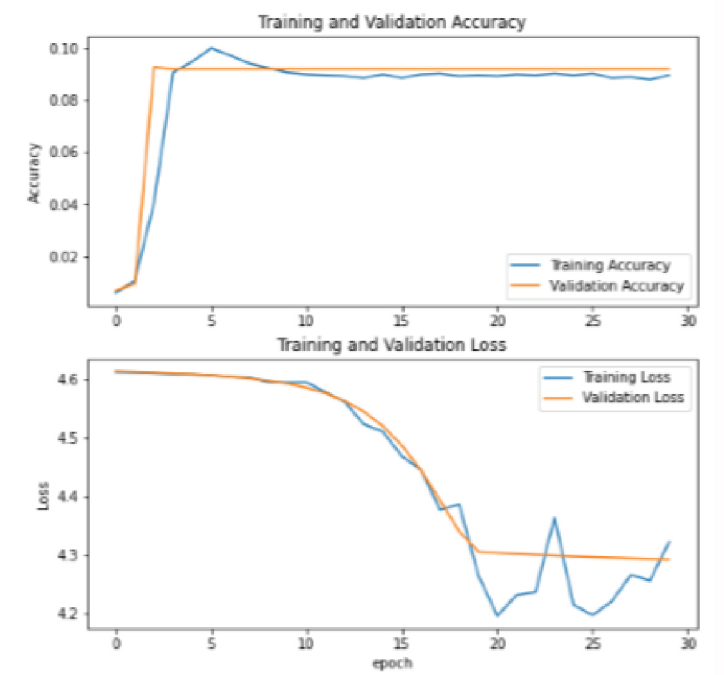



- LR : 0.01
- EPOCHS: 65
- STEP SIZE: 30
- TRAININGLOSS: 1.51
- VALIDATION LOSS: 2.52
- TRAINING ACCURACY: 58%
- VALIDATION ACCURACY: 47%

In 65 epochs the behavior is pretty much the same, even though a slight decrease of the losses is recorded.
By using learning rates greater in magnitude the model diverges: weight updates are too large and the performances of the model oscillate over training epochs  (mainly training set loss).
To try to achieve better results it is followed a different way, to train the net with a different optimization algorithm. Below the charts showing the performances with three different learning rates on 30 epochs, by using Adam optimizer:

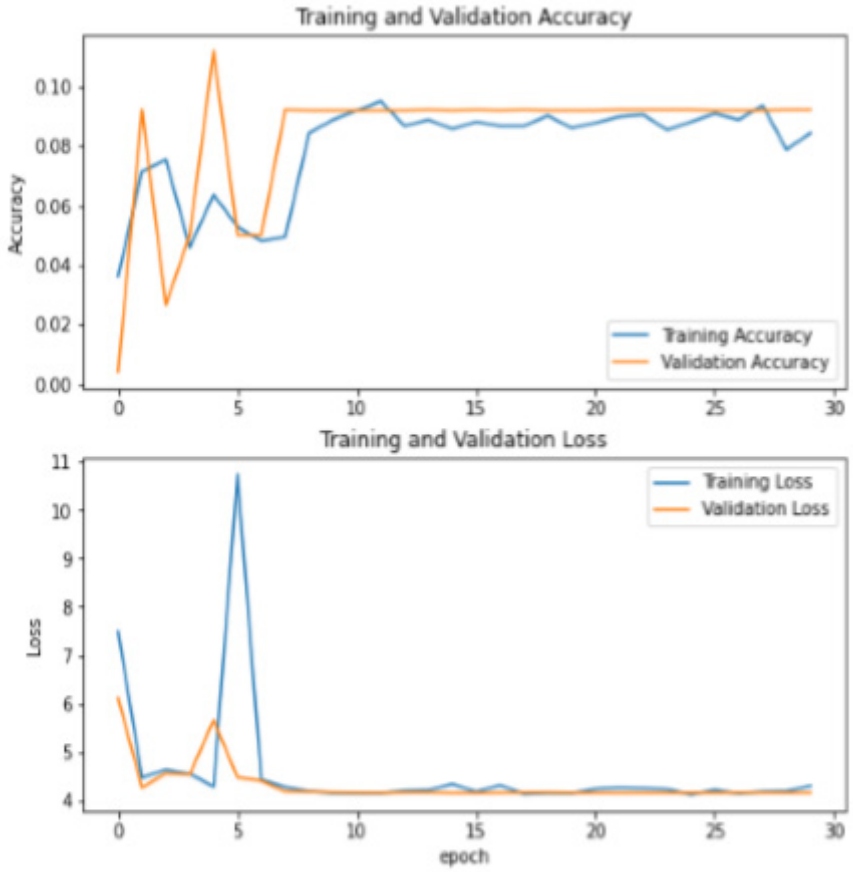

-LEARNING RATE:0.0001
-EPOCHS: 30
- STEP SIZE: 20
-TRAINING ACCURACY: 83%
- VALIDATION ACCURACY:53%
- TRAINING LOSS: 0.50
- VALIDATION LOSS: 2.71

Here the model overfits a bit the data, for the same reasons as before: the training loss curve keeps on with its trend, while the validation starts increasing from araund epoch 15.




- LR: 0.000001
- EPOCHS: 30
- STEP SIZE: 20
- TRAINING ACCURACY: 9%
- VALIDATION ACCURACY: 8%
- TRAINING LOSS: 4.34
- VALIDATION LOSS: 4.25

By decreasing the learning rate the results are not good. Like in the first implementation, the model is not able to learn from the dataset, resulting in flattening low accuracy values and very high loss ones. The training loss doesn't converge to a low value, the error rate is still too high to perform well on unseen data (in fact the validation accuracy is very low, less than 10%). According to this, it was selected a greater learning (0.1) rate with the following results:




- LEARNIN G RATE: 0.1
- EPOCHS: 30
- STEP SIZE:20
- TRAINING ACCURACY: 9%
- VALIDATION ACCURACY: 7%
- TRAINING LOSS: 4.31
- VALIDATION LOSS 4.29

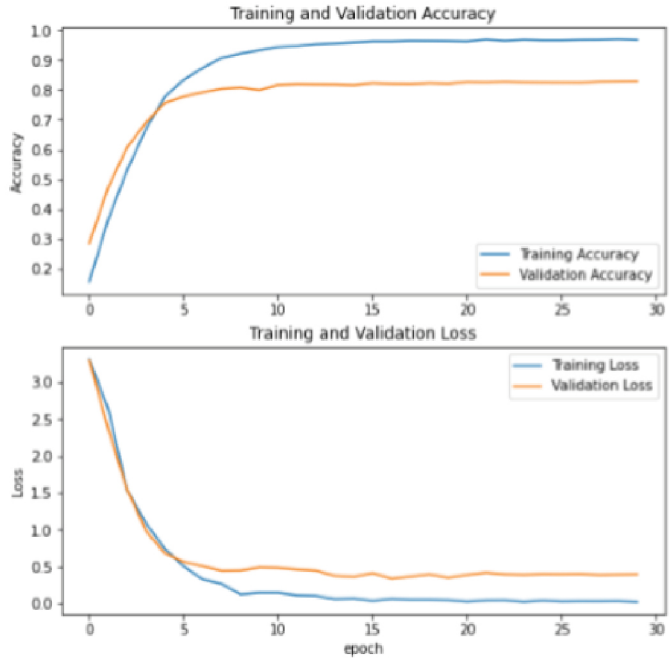
Also this set of hyperparameters doesn't provide good results, but similar issues to the previous one. For a more compact visualization, all the presented results are collected in the following Table:

| LEARNING RATE | EPOCHS | OPTIMIZER | STEP SIZE | TRAINING ACCURACY(%) | VALIDATION ACCURACY(%) | TRAINING LOSS | VALIDATION LOSS |
|---|---|---|---|---|---|---|---|
| 0.001 | 30 | SGD+MOM | 20 | 0.09 | 0.09 | 4.49 | 4.51 |
| 0.01 | 30 | SGD+MOM | 20 | 34 | 29 | 2.81 | 3.20 |
| 0.05 | 30 | SGD+MOM | 20 | 32 | 30 | 2.84 | 3.21 |
| 0.01 | 50 | SGD+MOM | 30 | 57 | 46 | 1.69 | 2.57 |
| 0.05 | 50 | SGD+MOM | 30 | 91 | 48 | 0.40 | 4.36 |
| 0.01 | 65 | SGD+MOM | 30 | 58 | 47 | 1.51 | 2.52 |
| 0.0001 | 30 | ADAM | 20 | 83 | 53 | 0.50 | 2.71 |
| 0.000001 | 30 | ADAM | 20 | 0.09 | 0.08 | 4.34 | 4.34 |

Even keeping testing different sets of hyperparameters the performances don't improve significantly: these charts in general stress how, by training a net from scratch with such few samples, it is not possible to obtain a model which learns well from the training set and at the same time is also capable to well generalize on unseen data (in other words, it's really hard that our model's training loss converges to zero without overfitting on the validation set). It is reasonable to suppose that this fact
is mainly due to the size of the training dataset, which is too small to guarantee good performances for any convolutional neural network. Taking this into account, the model presenting the lowest validation accuracy among those presenting an acceptable tradeoff between overfitting the data and well generalization on new ones is selected for testing. According to this, the best net found has the following hyperparameters: Learning rate: 0.01, Optimizer: SGD + Momentum on 65 epochs with policy decay applied after 30 epochs. The test set is predicted by training the net only with the training set: having already checked the network performances on unseen data (validation ones), similar results are expected, belonging both to the same data distribution. In fact, on the test set an accuracy of 59% is gained. If a new network was trained with training set plus the validation set, even better results would be expected, because of the doubling of the data (validation and training sets are of the same size).

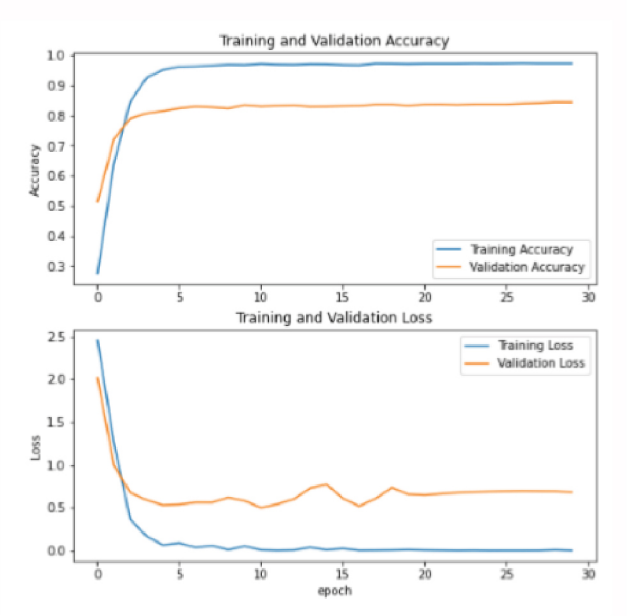
# TRANSFER LEARNING

A way to overcome optimization difficulties is to use transfer learning and finetuning techniques: it is upload AlexNet architecture with its weights (already learned by having trained it on a large dataset quite similar to the current one in terms of pictures subjects, ImageNet) and it is used as starting point for training with Caltech-101 dataset . As it is previously trained to classify 1000 different types of images, the last fully connected layer is reformed to fit the classification classes required by Caltech101. Before feeding the net, the images are preprocessed in the same way described above, by using different parameters to fit the normalization process. By using the same parameters of the first implementation trained the following results, supported by charts, are obtained:




-LEARNING RATE: 0.001
-EPOCHS: 30
-STEP SIZE: 20
-TRAINING ACCURACY: 95%
-VALIDATION ACCURACY: 82%
-TRAINING LOSS: 0.08
-VALIDATION LOSS: 0.71

As expected, performaces significantly improve already from the first run: the model gets almost the same behavior on both training and validation sets the training loss converges to zero and also the validation curve tends to a very low value (it shows no overfitting scenario, as the gap between the two curves is not increasing with epochs). Even using Adam optimizer, the performances remain almost the same:




-LEARNING RATE: 0.0001
- TRAINING LOSS: 0.009
- VALIDATION LOSS: 0.73
-TRAINING ACCURACY: 96%
- VALIDATION ACCURACY: 84%

From the charts and the above , it is possible to note an interesting point: the training loss curve converges to a low value after few epochs and then the more the epochs over the model is running,  the smaller that value becomes, but not significantly (that's the reason why the training phase was performed for no more than 30 epochs, and, at the same time, the training phase could be stopped earlier). Validation loss curves follow a similar trend, even though converging to slightly higher values. In other words, with pretrained net as starting point our resulting model is very often capable to learn features from training set and at the same time to provide good performances in predictions on unseen data. This is why the network already learned from ImageNet generic images patterns which are revealed needful to achieve a good generalization porperty.

Given the previous assumptions, different sets of hyperparameters are tested. The charts of the performances are similar to the previous ones (the only change is due to learning rate, the higher it is, within a certain upper bound value, the faster the model converges) so they are not shown. Only the values of the final results are reported, including the two already shown above :

| LEARNING RATE | OPTIMIZER | EPOCHS | STEP SIZE | TRAINING ACCURACY(%) | VALIDATION ACCURACY(%) | TRAINING LOSS | VALIDATION LOSS |
|---|---|---|---|---|---|---|---|
| 0.001 | SGD+MOM | 30 | 20 | 96 | 83 | 0.04 | 0.71 |
| 0.001 | SGD+MOM | 20 | 15 | 96 | 83 | 0.05 | 0.68 |
| 0.01 | SGD+MOM | 30 | 20 | 97 | 85 | 0.004 | 0.72 |
| 0.01 | SGD+MOM | 20 | 15 | 97 | 85 | 0.0014 | 0.85 |
| 0.0001 | Adam | 30 | 20 | 96 | 84 | 0.09 | 0.73 |
| 0.00001 | Adam | 30 | 20 | 87 | 74 | 0.4 | 0.99 |

The best model is selected according to the lowest validation loss value, among those with the training loss converging to really low values. So, in this case, the best net has this set of hyperparameters:

**-LR : 0.001**

**-OPTIMIZER: SGD + MOMENTUM**

**-EPOCHS : 20**

Which gives a validation loss of 0.68 and a training loss of 0.05. As before, the evaluation on the test set is made by training the net only with the training set, obtaining an accuracy score of 84%.
Now Instead of using pretrained AlexNet as a starting point,  only the fully connected layers first and the convolutional ones then are trained with Caltech dataset. The layers not trained get frozen: it means that their weights are not changed when they are reused in the subsequent task, when backpropagation is done during the net training these  remain untouched.
By training the net with the best set of hyperparameters found so far, the following results are obtained:

**Freezing the convolutional layers:**
**-Training accuracy: 95%**
**-Validation accuracy: 83%**
**-Training loss: 0.26**
**-Validation loss: 0.65**
**-Test accuracy: 84%**

**Freezing the fully connected layers:**

**- Training accuracy: 31%**
**-Validation accuracy: 34%**
**- Training loss: 3.1**
**-Validation loss: 3.5**
**-Test accuracy: 34%**

It is possible to notice the performances gap between the two different methods: the scores obtained by freezing the convolutional layers are significantly higher than those obtained by freezing fully connected layers. This is due to the fact that in the initial layers the features extracted are pretty generic and independent from the particular task. So by freezing the initial stages, the network got is already able to extract meaningful general features and they don't badly affect the classification performances. Instead, by freezing the fully connected our results worsen a lot: the pretrained model had already learnt low level features in the convolutional layers, and the fully connected ones had been set according to those. Training only convolutional layers means to modify the low level features weights and it becomes difficult to adapt them to the next frozen part of the net, that doesn't allow it to well generalize on unseen data.
In conclusion, in this case freezing convolutional layers leads to a similar model to finetuning one, only a really slightly improvement in validation loss is recorded, carried to **0.66**.

# DATA AUGMENTATION

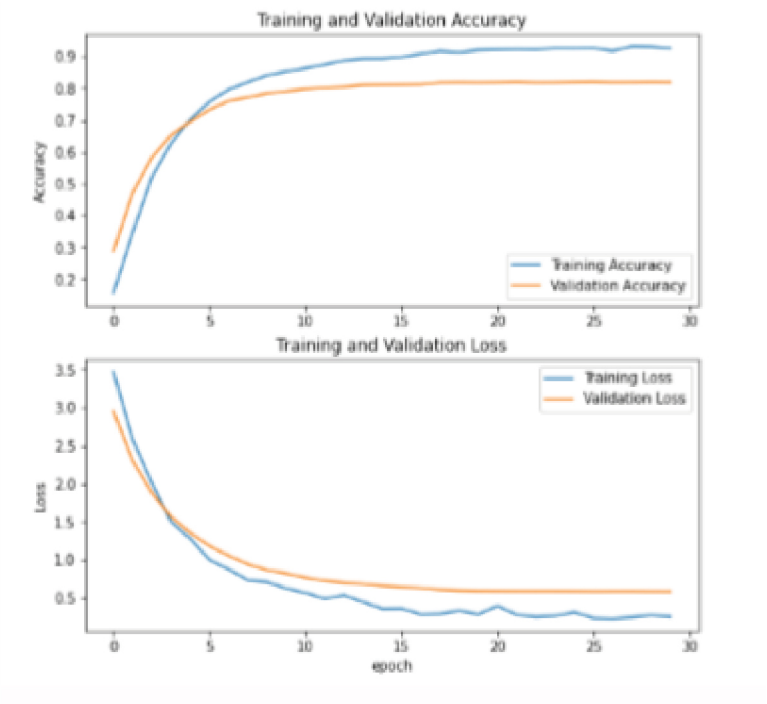
To try, hence, to improve the performances of the net, in particular to reduce overfitting probabilities, the dataset is augmented by using data augmentation technique: during the training phase some alterations are applied to different images already stored in the dataset, and those are considered new distinct ones by the network. [1]
Three different transformations are prepared:

**1)** Vertical flip of the images with probability of 0.5 to be applied on each one combined with a random changing of the brightness, contrast and saturation;
**2)** Horizontal flip of the images with probabiliy of 0.5 to be applied combined with a crop applied at a random location of the images (output size 224x244, with no padding);
**3)** A random rotation of the images of 45 degrees combined with a random changing of the brightness, contrast and saturation; [2]
By applying these transformations to the training set of the net which gave the best performances and evaluating the net on the validation set, the new scores are collected in the table below:

With:
**-AlexNet with transfer learning;**
**-Learning rate : 0.001;**
**-Epochs:20 (policy decay after 15 epochs);**
**- Optimizer: SGD + Momentum;**

| TRANSFORMATION | TRAINING ACCURACY(%) | VALIDATION ACCURACY(%) | TRAINING LOSS | VALIDATION LOSS |
|---|---|---|---|---|
| 1 | 87 | 78 | 0.35 | 0.78 |
| 2 | 95 | 0.82 | 0.06 | 0.75 |
| 3 | 80 | 0.78 | 0.57 | 0.80 |

The scores are about the same obtained with no data augmentation, just slightly worse, but by freezing the convolutional layers and applying the previous transformations, it is possible to notice how the second slightly improves the performances:




The gap between the curves has narrowed, showing a good similar behavior of the net on both the sets, but at the end of the chart they don't seem to have a decreasing trend, so the epochs haven't been increased (even though it would be reasonable, given the percentage of the application of some transformations). The validation loss is 0.64, the lowest so far, while the training is 0.20, slighty higher than before, but even good.

# BEYOND ALEXNET

The classification is performed by using different network architectures with more layers than AlexNet, in particular resnet18 and Vgg16 with transfer learning. Images are preprocessed as usual for this task, and the final layers of the nets are reformed to fit the Caltech-101 classification. With ResNet18, the following performances are obtained (the size of the batches are reduced to 64 to not cause memory issues, given the higher number of layers of the net):

| LEARNING RATE | EPOCHS | STEP SIZE | OPTIMIZER | TRAINING ACCURACY(%) | VALIDATION ACCURACY(%) | TRAINING LOSS | VALIDATION LOSS |
|---|---|---|---|---|---|---|---|
| 0.001 | 30 | 20 | SGD+MOM | 0.99 | 76 | 0.0019 | 0.47 |
| 0.0001 | 30 | 20 | SGD+MOM | 0.97 | 87 | 0.0022 | 0.54 |
| 0.01 | 30 | 20 | SGD+MOM | 0.98 | 0.87 | 0.0021 | 0.54 |
| 0.01 | 45 | 35 | SGD+MOM | 0.99 | 0.84 | 0.0019 | 0.61 |
| 0.01 | 20 | 15 | SGD+MOM | 0.98 | 0.84 | 0.0020 | 0.59 |

As shown, whatever the sets of hyperparameters, performances improve: accuracies are pretty higher and validation losses, in some cases, even lower. Firstly the net was trained by using different learning rates, and then some other hyperparameters tuning was applied: by increasing the epochs the results don't refine on, but it was noticed that the losses quickly converge to zero after a few ones and keep on around that till the end. So, the epochs are reduced to 20, but no significant variation was recorded. Even by using Vgg16 architecture a better model than using Alexnet is obtained:

| LEARNING RATE | EPOCHS | STEP SIZE | OPTIMIZER | TRAINING ACCURACY | VALIDATION ACCURACY | TRAINING LOSS | VALIDATION LOSS |
|---|---|---|---|---|---|---|---|
| 0.001 | 30 | 20 | SGD+MOM | 0.99 | 0.91 | 0.01 | 0.41 |
| 0.0001 | 30 | 20 | SGD+MOM | 0.94 | 0.89 | 0.14 | 0.38 |
| 0.001 | 35 | 25 | SGD+MOM | 0.99 | 0.87 | 0.0002 | 0.42 |
| 0.01 | 10 | 8 | SGD+MOM | 0.97 | 0.88 | 0.0014 | 0.44 |
| 0.01 | 20 | 15 | SGD+MOM | 0.96 | 0.85 | 0.0024 | 0.47 |

In conclusion, it is possible to state how performances strongly depend also on the net architecture used. [3]

References:

[1] https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9

[2] https://discuss.pytorch.org/t/data-augmentation-in-pytorch/7925

[3] https://pytorch.org/docs/stable/torchvision/models.html