Machine Learning and Deep Learning
Politecnico di Torino

Homework 2
Silvia Giammarinaro

Student ID: 269991

June 15, 2020

1 Introduction

In this homework, we analyze the full process of training a Convolutional Neural Network using the Pytorch framework. We consider the model Alexnet and the Caltech101 dataset. AlexNet is a CNN trained on the Imagenet dataset, which contains 1.2 million images with 1000 categories. From 2010 to 2017 it is used in the homonymous challenge, AlexNet is the winning network of the 2012 edition.

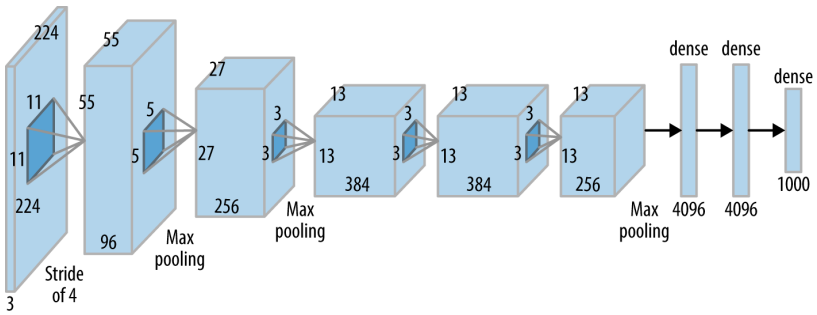

Figure 1: Alexnet's architecture

A Convolutional Neural Network is a sequence of layers in which one volume of data is transformed into another one. Let's start analyzing Alexnet. AlexNet contains convolutional layers, max-pooling layers, ReLU activations, and Softmax. Convolutional layers

compute the output of the neurons connected to small regions of the input, a dot product between their weights and the input is performed. Max-pooling layers are a type of pooling layers, which reduce the volume of the input performing a mathematical operation such as $\max_x(x_i)$ in this case. ReLu layers perform the activation function $max(0, x)$, this type of layer does reduce the input volume. Fully-connected layers compute the class scores and softmax layers are another type of activation layers, these are used in multi-class classification. It returns a discrete probability distribution over all the classes.
We note AlexNet introduces the concept of Local Response Normalization (LRN) and dropout, and it is one of the first networks to use GPUs for the training phase. LRN implements the lateral inhibition concept in neurobiology. This refers to the capacity of an excited neuron to subdue its neighbors. This tends to create a contrast in that area, a local minima, which is what we are looking for. This method is fallen out in favor of other regularization technique such as batch normalization and dropout. Dropout is a regularization technique in which a neuron is active with a probability p, if a neuron is inactive the weights associated with it are not updated with backpropagation algorithm. This technique prevents overfitting.

Label	Count
airplanes	800
Motorbikes	798
Faces_easy	435
Faces	435
watch	239
Leopards	200
bonsai	128
car_side	123
ketch	114
chandelier	107

Figure 2: Top 10 classes of the Caltech101 dataset

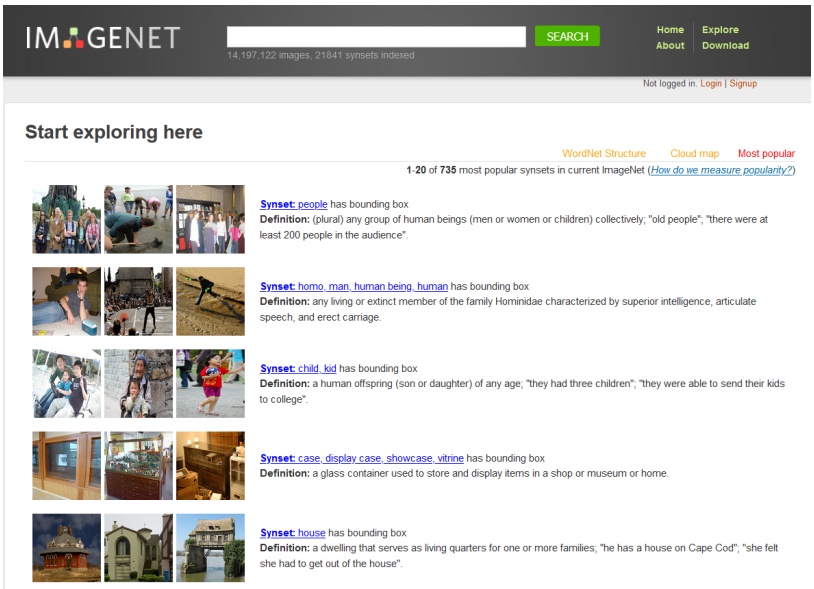

Figure 3: Imagenet's most popular synsets

The Caltech101 dataset contains 9,146 images divided in 102 classes. We choose to remove the background class, so we have 101. We analyze the distribution of the classes and we note that they are highly imbalanced. As an example, this could lead to a more robust classification for the class "airplanes" and a weaker one for another class with less samples.
Although we have to take into account that Alexnet was trained on Imagenet. In the Im-

agenet dataset every picture is associated with a list of synsets, not with just one label as in the Caltech101 dataset. The most popular synsets are reported on its website, the popularity of a synset is defined as follows:
We use the Popularity Percentile index to measure the popularity of a synset. A 90 % Popularity Percentile indicates that the synset is ranked higher in popularity than 90% of all synets. The ranking is derived by combining the number of results returned by Google text search and the word frequency in the British National Corpus.

2 Training from scratch

In this section we devolop different models starting from the Alexnet's architecture, without loading the weights used for the Imagenet competion. We divide our dataset in three different dataloader, train, validation and test and every dataloader is divided into batches, every batch contains 256 samples.
We divide our training process in epochs, for every epoch we iterate over all batches of the train dataloader to train the network and then we compute the accuracy on the validation set. Once the training is finished, we test the best performing model on the test set. We define as the best performing model the one with the lower mean loss per image per epoch.
The method used for update the weight is the stochastic gradient descent algorithm. The SGD algorithm is an optimization algorithm in which we want to find the minimum of a function, in this case the loss function. The loss function determines the gap between our predictions and our target labels, for all the experiments we use the one called Cross Entropy. At each iteration we pick a batch from the set and we compute the gradient of the loss. After using the backpropagation algorithm, we update the network weights using a step size called learning rate. The learning rate is the most important hyperparameter.
A low value can lead to a problem called vanishing gradient, where we are stuck in a local minima of the function. Instead an high value can lead to gradient explosion, in which the values of our weights increase because the step is way to bigger to reach the minimum. The preferred version of the SGD algorithm can be choosen setting an optimizer.
We experiment with SGD, which is the standard one and Adam. If we want to change the learning rate during the epochs we can set a scheduler. We apply the StepLR scheduler, which decreases the learning rate with a factor γ after a number of epochs given by the user. In the end we choose a number of epochs suitable for all configuration just reported.

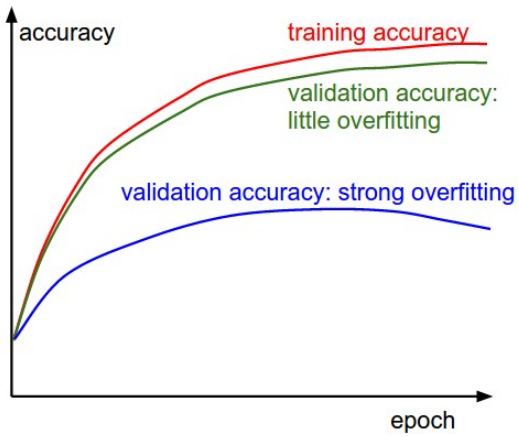

Figure 4: Comparing accuracy's trends

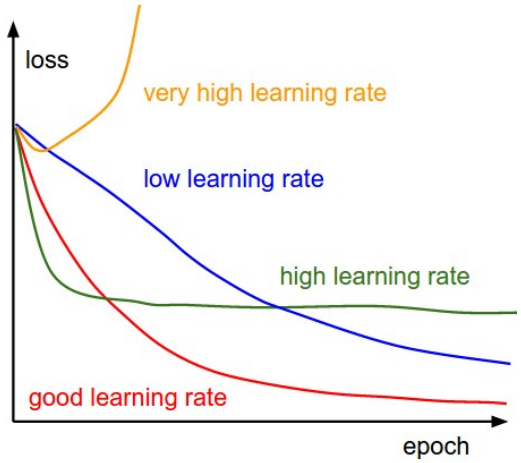

Figure 5: Comparing loss trends

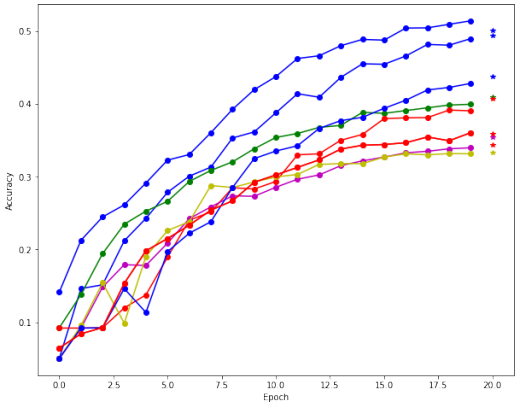

Figure 6: Accuracy per epoch

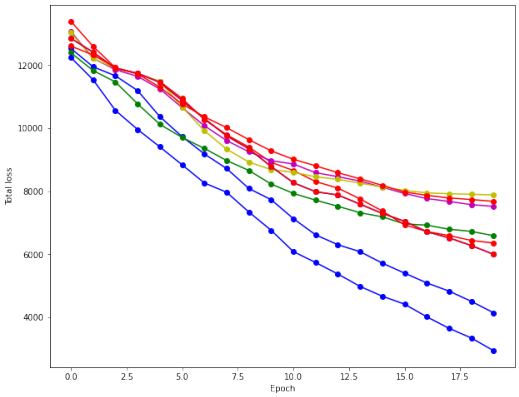

Figure 7: Total Loss per epoch

Batch Size	LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Test Accuracy	Color
256	$2*10^{-4}$	20	10	0.5	Adam	StepLR	**0.469±0.032**	Blue
256	10^{-4}	20	5	0.5	Adam	StepLR	0.377±0.033	Red
256	10^{-3}	25	20	0.1	Adam	StepLR	0.407	Green
256	10^{-3}	20	8	0.25	Adam	StepLR	0.353	Magenta
256	10^{-3}	20	8	0.1	Adam	StepLR	0.333	Yellow

We load the Alexnet model without pretrained weights and we change the last fully connected layer to fit our task.

```
NUM_CLASSES = 101
net = alexnet(pretrained=False)
net.classifier[6] = nn.Linear(4096, NUM_CLASSES)
```

We report five models with different performances, we choose the hyperparamenters with a random search. For two models, we repeat the experiment two more times (additional trials are report with the same color). We note that performances are sightly different per experiment because we shuffle the train dataloader, so at each batch we have different samples which gives us different learning process. For all the trials we create two graphs. The first one is related to the accuracy, for every epoch we compute the accuracy on the validation set and at the end we evaluate the model on the test set (star marker). This convention will be used in the rest of the report. The second one shows the trend of the loss, the sum among all the batches for every epoch. These trends show we are not overfitting, but the learning rate seems quite low.

3 Transfer learning

Transfer learning is a technique used in machine learning in which we use an existing network that accomplishes a similar task to solve our problem. We load the Alexnet model pretrained on Imagenet and we normalize our data using the mean and the standard deviation of Imagenet (0.485, 0.456, 0.406), (0.229, 0.224, 0.225). Starting from a petrained model, we choose a lower learning in order to not perturbe too much the weights and biases computed with Imagenet. We report below our hyperparameters choices and results, we run three experiments for each set of hyperparameters

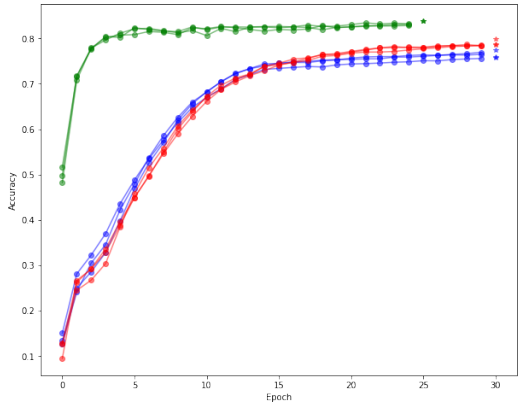

Figure 8: Accuracy per epoch

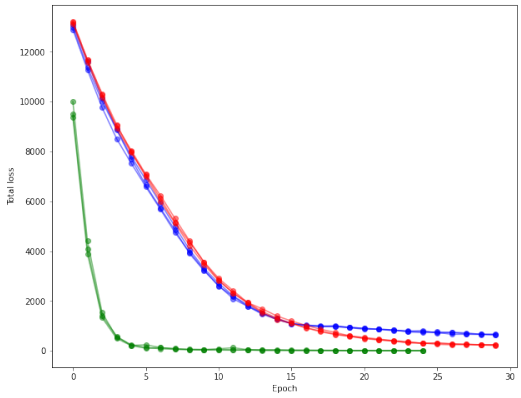

Figure 9: Total loss per epoch

Batch Size	LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Test Accuracy	Color
256	10^{-4}	30	20	0.1	Adam	StepLR	**0.8382±0.0007**	Green
256	10^{-5}	30	25	0.5	Adam	StepLR	0.7930±0.007	Red
256	10^{-5}	30	15	0.25	Adam	StepLR	0.7660±0.009	Blue

We can see our results improve significantly compared to the previous experiment. We can conclude the feautures extracted from Imagenet are comparable to the ones presented

in Caltech101. Now we freeze some layers in order to not modify the starting weights. We freeze all the layers except the convolutional layers, which extract low level features, and the fully connected layers, which extract high level ones. We use the hyperparamenters of the first model, reducing the number of epochs to 20 and removing the scheduler.

```
net_only_conv = alexnet(pretrained=True)
#freeze all the layers
for param in net_only_conv.parameters():
    param.requires_grad = False
# unfreeze only the convolutional layers
for val in net_only_conv.classifier:
  val.requires_grad = True
net_only_conv.classifier[6] = nn.Linear(4096, NUM_CLASSES)
#[...]
net_only_fc = alexnet(pretrained=True)
for param in net_only_fc.parameters():
    param.requires_grad = False
# unfreeze only the fully connected layers
for val in net_only_fc.features:
  val.requires_grad = True
net_only_fc.classifier[6] = nn.Linear(4096, NUM_CLASSES)
```

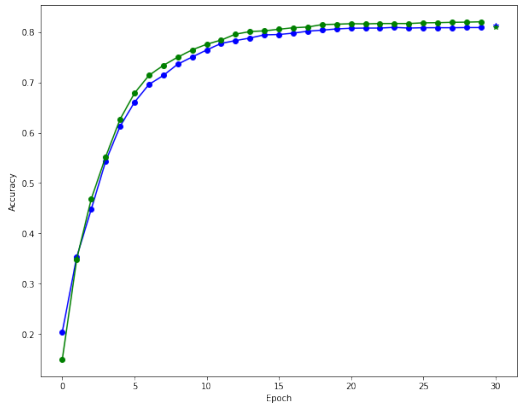

Figure 10: Accuracy per epoch

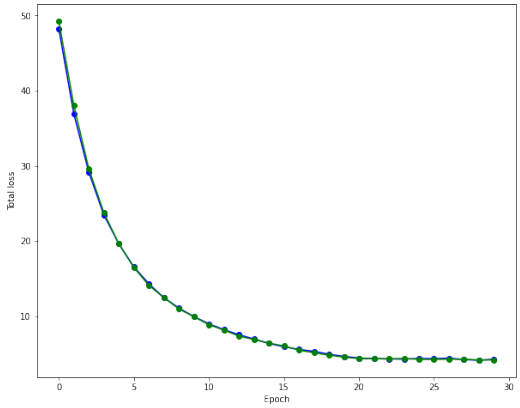

Figure 11: Total loss per epoch

Batch Size	LR	N. Epochs	Optimizer	Active layers	Test Accuracy	Color
256	10^{-4}	20	Adam	Fully Connected	0.817	Blue
256	10^{-4}	20	Adam	Convolutional	0.806	Green

Freezing the features and classifier group of layers, we are not updating other layers such as ReLU, MaxPool2d, which do not have weights updated by the backpropagation algorithm. The graphs show that both networks have very similar performance. As observed

before, we can say that our dataset's features are related to the ones extracted from the Imagenet dataset, freezing the the convolutional layers or the fully connected we obtained almost the same results. This is reasonable because Imagenet is a huge dataset and we suppose all the classes reported in Caltech101 are also presented in the Imagenet.

4 Data augmentation

The data augmentation is a technique used in Deep Learning to create more samples of our dataset. The dataloader apply the transformations to the images at each epoch, so the network is feeded with sightly different samples. Resizing and cropping the images is mandatory because our model requires inputs with size 224x224 pixels. *Train_transform* is applied to both train and validation set and *test_transform* is applied to the test set. We use the best hyperparameters from the previous experiment.
We create three different sets of transformations. The best performing one is composed by the transformations *RandomRotation, RandomHorizontalFlip, ColorJitter* which rotate, flip and change color settings with a probability p. The dataloader apply these transformations at each epoch, so at the epoch the net is "seeing" different inputs. We obtain other two sets modifying the values of the transformation ColorJitter, alterating even more the colors, and adding the GreyScale transformation.

```
train_transform = transforms.Compose([ transforms.Resize(256),
                transforms.RandomRotation(degrees=10),
                transforms.RandomHorizontalFlip(),
                transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize((0.485, 0.456, 0.406),
                    (0.229, 0.224, 0.225) )])

test_transform = transforms.Compose([ transforms.Resize(256),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
                transforms.Normalize((0.485, 0.456, 0.406),
                    (0.229, 0.224, 0.225) )])
```

Set	Test Accuracy	Color
Ours	**0.824**	Blue
Strong ColorJitter	0.821	Green
GreyScale	0.801	Red

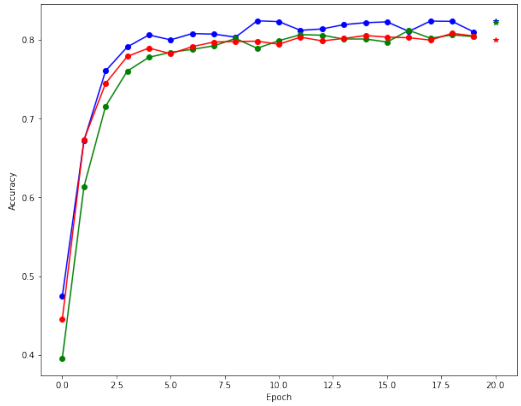

Figure 12: Accuracy per epoch

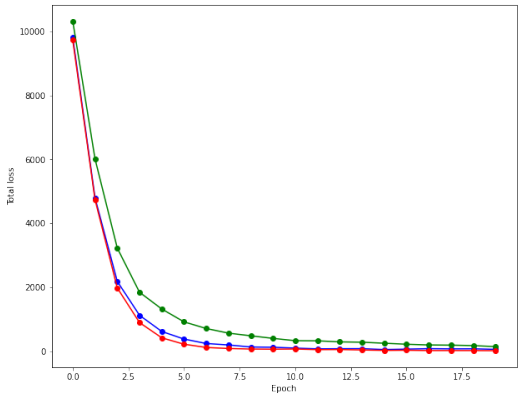

Figure 13: Total loss per epoch

As we can see the accuracy on the test set slightly drop when using GreyScale. This means our network recognise an object also thanks to his colors rather than his shapes and peculiar characteristics (ex: wheels). Moreover, perturbating too much the colors with a strong ColorJitter makes seeing the actual object in the pictures more difficult, even for us. We can conclude the best setting is the first one because we are rotating and flipping the image, which makes the net focus on other aspects of the image, increasing the accuracy. In figure 14 we report the samples obtained with the different transformations.

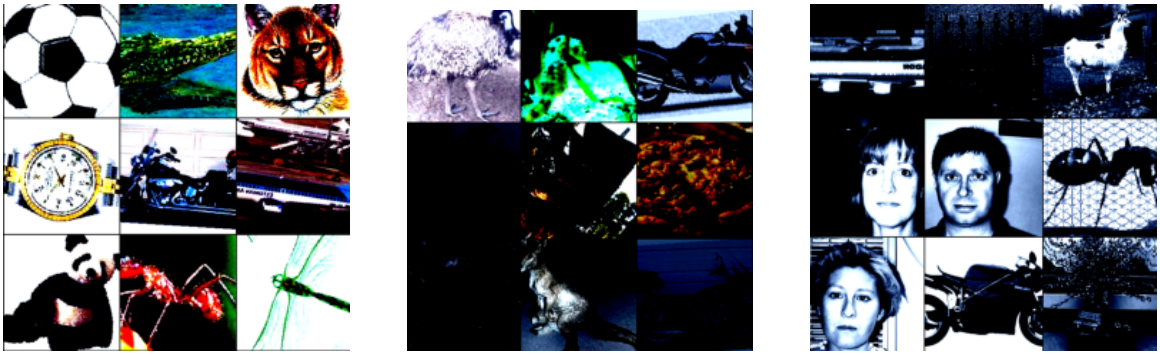

Figure 14: Input images for different set of transformations. From left to right: ours, strong ColorJitter and GreyScale

5 Beyond Alexnet

In the end we run some experiments using two additional models, VGG16 and Resnet18. Both networks have better performances than Alexnet. The VGG model has two variations, VGG16 and VGG19 with 16 and 19 layers. These nets were presented in the 2014 Imagenet competition. In the VGG16 the improvement is given by substituting big kernels (11x11, 5x5) with multiple 3x3 kernels, doing so we are reducing the number of hyperparameters used. The Resnet model was introduced in the 2015 Imagenet challenge. The Resnet architecture introduce the concept of residual block. The residual block implements shortcuts to skip layers in order to avoid the vanishing gradient problem we exposed before. There are multiple versions of ResNet architectures, the number at the end of the model indicates the number of layers.
We start from pretrained models and with a lower batch size due to a GPU limit, these model are bigger than Alexnet. A general rule is to lower the learning rate when the batch size is reduced. We report below the results obtained with the two different configurations. The loss is minimized after 10 epochs, so they're enough to train the networks.

Figure 15: VGG16 architecture

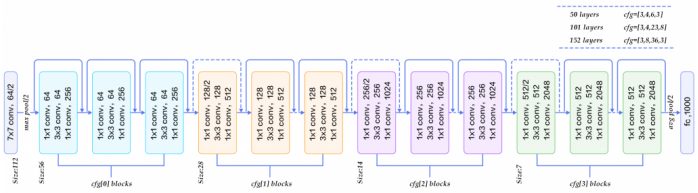

Figure 16: Resnet18 architecture

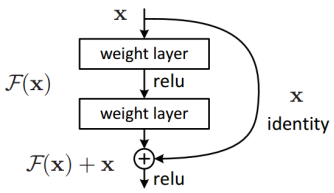

Figure 17: Residual block

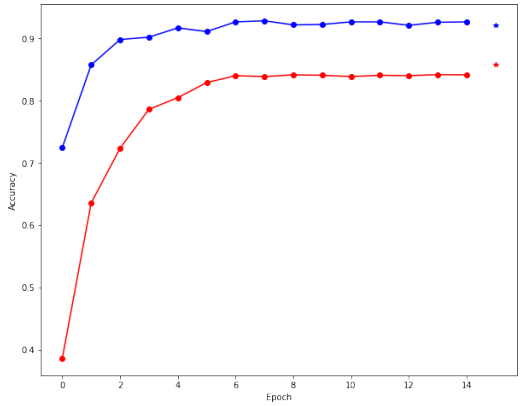

Figure 18: Accuracy per epoch

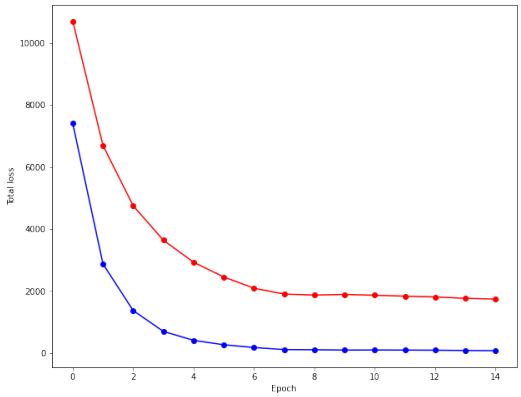

Figure 19: Total loss per epoch

Batch Size	LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Model	Test Acc.	Color
24	10^{-4}	15	7	0,1	Adam	StepLR	Resnet18	0.921	Blue
56	10^{-4}	15	7	0,1	Adam	StepLR	VGG16	0.858	Red

As we expect, the learning process using these model is better than the one analyzed before with AlexNet. These networks are the result of research processes started from AlexNet and other past networks, they are more complex and optimized. They would not win the Imagenet competiton otherwise.

6 References

Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems - Aurélien Géron
Alom, Md. Zahangir Taha, Tarek Yakopcic, Christopher Westberg, Stefan Hasan, Mahmudul Esesn, Brian Awwal, Abdul Asari, Vijayan. The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches. (2018).
Stanford University C231n course's notes
Towards Data science site