s263138_AIML_homework2

December 16, 2019

1 Ivan Murabito

2 Homework 2 DeepLearning and experiments on AlexNet

3 Introduction

The purpose of this homework is to analyze and test the **AlexNet** convolutional network through various experiments;

- plain training and test
- transfer learning
- data augmentation
- **AlexNet**: created in 2012 by Alex Krizhevsky, it was the first convolutional network to use the RELU as an activation function, to add non-linearity and improve its speed, and one of the first **deep** cnn to have important results in 2012 imageNet challenge. follow the classical convolutional network scheme:

  some conv layer followed by a pooling, and in the end of the network the FC layers.

  In particular AlexNet has 8 layers — 5 convolutional and 3 fully-connected, and about 62 Million parameters.
- **Dataset**: in this homework we use the caltech-101 dataset which, as the name implies, has 102 classes and about 9k images including different objects.

4 Training from scratch

4.0.1 Install requirements

4.0.2 Import libraries

In this section i've installed and imported all stuffs needed for this homework (numpy,matplot,pil,torchvision etc etc)

4.0.3 Set Arguments

Block of given arguments (LEARNING RATE,STEP_SIZE,EPOCHS etc)

4.0.4 Define Data Preprocessing

I defined the transformations to apply to images in the dataset (both training and evaluation), at first:

- resize (256)
- center-crop (224)
- to Tensor
- normalize (mean=.5 .5 .5) (std=.5 .5 .5)

4.0.5 My custom dataset (Caltech) class

I created my custom dataset class, starting from the one released. that load the images respectively from given train.txt and test.txt files. I created two new functions that help the creation of dataset

- load_images(..)
- find_classes(..)

(easy to imagine their purpose :))
the dataset class are:

- init(..) : initial processes, when data are loaded (from txt in this case)
- len(..) : return the size of data
- getitem(..) : return data and label at orbitary index

4.0.6 Useful function

I wrote several usefull functions that I will use throughout the homework in particular

- evaluate(..): test a model on given dataset
- plot_graph(..)
- train_and_validate(..): very useful function; given a model, optimizer, n_epoch and datasets and dataloaders

train the model and evaluate it on each epoch on validation set.
at the end (after last epoch) plot two graphs : loss / epochs accuracy / epochs and returns the best model found

4.0.7 Prepare Dataset

in this section i loaded two dataset from .txts; train and test (5784 images for train and 2893 for test) nb: obviously the images are balanced for each class.

```
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 5784 Images and label
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 2893 Images and label
Train Dataset: 5784
Test Dataset: 2893
```

4.0.8 Split train into train and validation

I've splitted train in train and validation (each 2892 images), to maintain the balance I used odd indexes for validation and even for training. (nb: this trick work only because the images are ordered) for all the homework these datasets will be used.

```
Training split: 2892
Validation split: 2892
```

4.0.9 Prepare Dataloaders

from pytorch docs:

`Data loader. Combines a dataset and a sampler, and provides an iterable over the given dataset.`

Dataloaders help to put in ram a batch of data from dataset, help with parallelization and provide some usefull tools like shuffling and more. there is a dataloader for each split (train,test,val)

4.0.10 Prepare Network

- Load the Alexnet (pytorch implementation)
- change fc layer from 1000 output to 101 output (caltech classes)

4.0.11 Prepare Training

Define:

- criterion: cross entropy
- param to oprimize: at firsts step of homework i will optimize all params of network (net.parameters())
- optimizer: the optimizer update the weights based on the loss,in this case is used SGD with momentum
- scheduler : dynamically change the LR

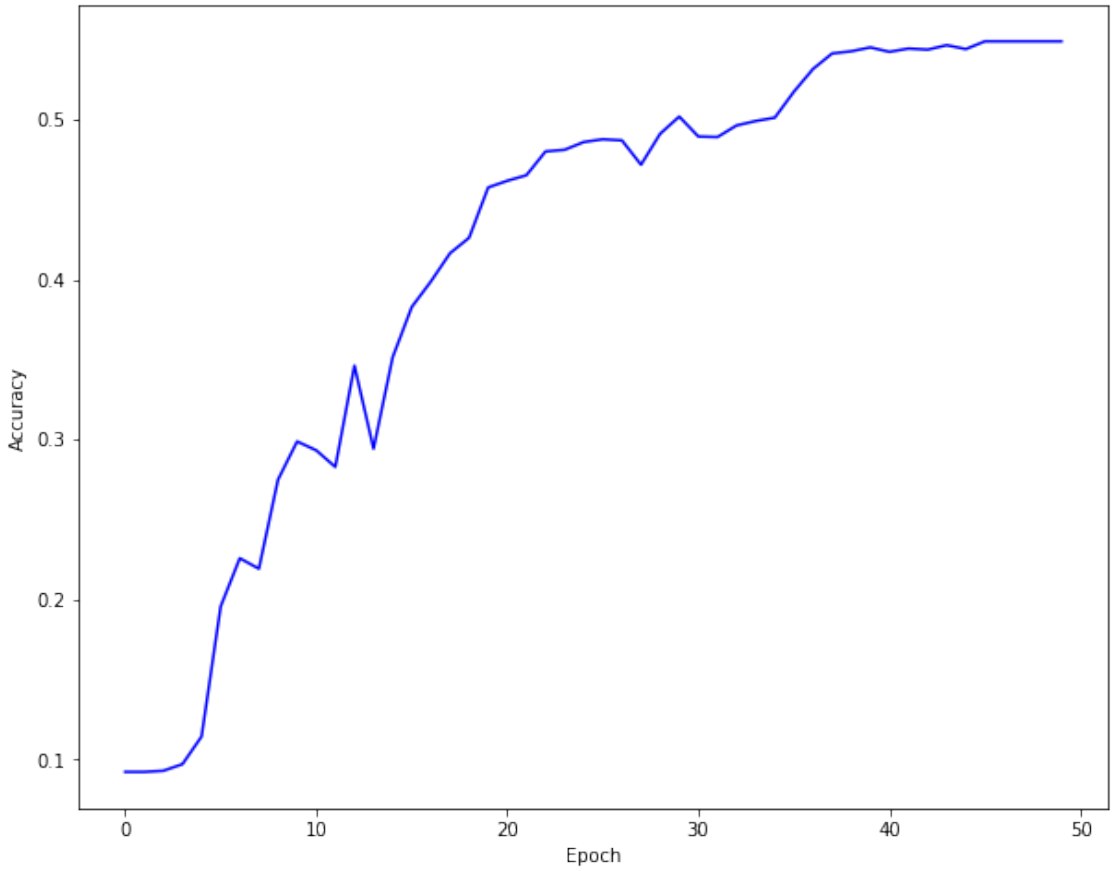
4.1 Train

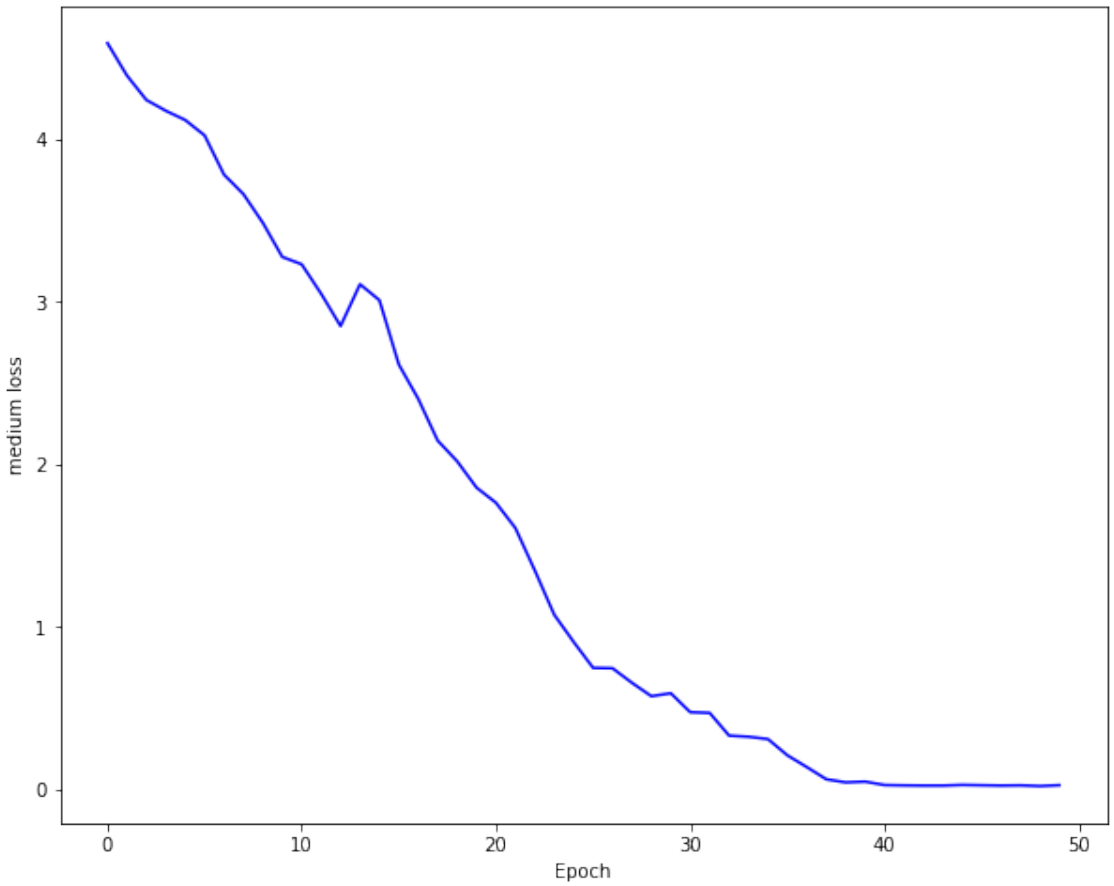
To find a good model I didn't put all the parameters to optimize in a double/triple for (like a gridSearch) but I did several experiments analyzing the progress of each model, first modifying the learning rate (the one given to me was too small, especially after splitting the train in train + validation) and then the step size and the number of epochs. for each epoch the model is evaluated on validation set. the printed output below show the epoch,loss,val_score at the end two graphs showing the trend of these variables over the epochs. the printed output below show (..one of) the best model that i found over experiments:

- **LR:0.05**
- **50 epochs**
- **step size:35**

```
epoch 1/50, LR = [0.05]  loss : 4.5911 score: 0.0923  BEST!
epoch 2/50, LR = [0.05]  loss : 4.3924 score: 0.0923
epoch 3/50, LR = [0.05]  loss : 4.2419 score: 0.0930  BEST!
epoch 4/50, LR = [0.05]  loss : 4.1754 score: 0.0972  BEST!
epoch 5/50, LR = [0.05]  loss : 4.1184 score: 0.1145  BEST!
epoch 6/50, LR = [0.05]  loss : 4.0238 score: 0.1957  BEST!
epoch 7/50, LR = [0.05]  loss : 3.7824 score: 0.2258  BEST!
epoch 8/50, LR = [0.05]  loss : 3.6626 score: 0.2192
epoch 9/50, LR = [0.05]  loss : 3.4860 score: 0.2749  BEST!
epoch 10/50, LR = [0.05]  loss : 3.2764 score: 0.2988  BEST!
epoch 11/50, LR = [0.05]  loss : 3.2311 score: 0.2932
epoch 12/50, LR = [0.05]  loss : 3.0501 score: 0.2828
epoch 13/50, LR = [0.05]  loss : 2.8514 score: 0.3461  BEST!
epoch 14/50, LR = [0.05]  loss : 3.1087 score: 0.2943
epoch 15/50, LR = [0.05]  loss : 3.0103 score: 0.3513  BEST!
epoch 16/50, LR = [0.05]  loss : 2.6146 score: 0.3828  BEST!
epoch 17/50, LR = [0.05]  loss : 2.4033 score: 0.3987  BEST!
epoch 18/50, LR = [0.05]  loss : 2.1470 score: 0.4163  BEST!
epoch 19/50, LR = [0.05]  loss : 2.0203 score: 0.4260  BEST!
epoch 20/50, LR = [0.05]  loss : 1.8580 score: 0.4575  BEST!
epoch 21/50, LR = [0.05]  loss : 1.7648 score: 0.4616  BEST!
epoch 22/50, LR = [0.05]  loss : 1.6093 score: 0.4651  BEST!
epoch 23/50, LR = [0.05]  loss : 1.3483 score: 0.4799  BEST!
epoch 24/50, LR = [0.05]  loss : 1.0761 score: 0.4810  BEST!
epoch 25/50, LR = [0.05]  loss : 0.9089 score: 0.4858  BEST!
epoch 26/50, LR = [0.05]  loss : 0.7496 score: 0.4876  BEST!
epoch 27/50, LR = [0.05]  loss : 0.7473 score: 0.4869
epoch 28/50, LR = [0.05]  loss : 0.6576 score: 0.4716
epoch 29/50, LR = [0.05]  loss : 0.5754 score: 0.4910  BEST!
epoch 30/50, LR = [0.05]  loss : 0.5936 score: 0.5017  BEST!
epoch 31/50, LR = [0.05]  loss : 0.4770 score: 0.4893
epoch 32/50, LR = [0.05]  loss : 0.4728 score: 0.4889
epoch 33/50, LR = [0.05]  loss : 0.3331 score: 0.4962
epoch 34/50, LR = [0.05]  loss : 0.3262 score: 0.4990
epoch 35/50, LR = [0.05]  loss : 0.3117 score: 0.5010
epoch 36/50, LR = [0.005000000000000001]  loss : 0.2119 score: 0.5173  BEST!
epoch 37/50, LR = [0.005000000000000001]  loss : 0.1397 score: 0.5315  BEST!
epoch 38/50, LR = [0.005000000000000001]  loss : 0.0649 score: 0.5411  BEST!
epoch 39/50, LR = [0.005000000000000001]  loss : 0.0455 score: 0.5425  BEST!
epoch 40/50, LR = [0.005000000000000001]  loss : 0.0503 score: 0.5450  BEST!
epoch 41/50, LR = [0.005000000000000001]  loss : 0.0291 score: 0.5422
epoch 42/50, LR = [0.005000000000000001]  loss : 0.0271 score: 0.5443
epoch 43/50, LR = [0.005000000000000001]  loss : 0.0259 score: 0.5436
epoch 44/50, LR = [0.005000000000000001]  loss : 0.0259 score: 0.5463  BEST!
epoch 45/50, LR = [0.005000000000000001]  loss : 0.0300 score: 0.5439
epoch 46/50, LR = [0.005000000000000001]  loss : 0.0279 score: 0.5488  BEST!
epoch 47/50, LR = [0.005000000000000001]  loss : 0.0259 score: 0.5488
```

```
epoch 48/50, LR = [0.005000000000000001]  loss : 0.0273 score: 0.5488
epoch 49/50, LR = [0.005000000000000001]  loss : 0.0229 score: 0.5488
epoch 50/50, LR = [0.005000000000000001]  loss : 0.0283 score: 0.5488
```





4.2 Test

evaluate the best model on test set

```
score: 0.5465
```

LR	EPOCH	STEP	LOSS	MAX SCORE ON VALIDATION SET	SCORE ON TEST
0.1	30	20	diverge		
0.05	60	20	0.34	0.47	
0.05	40	10	3.9	0.12	
0.05	50	35	0.02	0.55	0.54
0.01	40	25	0.37	0.34	
0.001 (stock)	30 (stock)	20 (stock)	2.5	0.16	

as you can see from the table there is a lot of difference in the results of the various tests with different hyperparameters. it can be noted, however, that the starting learning rate 0.001 is too

small to converge in acceptable times/epochs. nb: i evaluated on test set only the best model that i found, all others models are evaluated (for each epochs) only on validation set, and the validation score refers only the best model in all iterations. instead the loss is the loss at the last epochs.

5 Transfer Learning

in this section I do the same experiments as before but this time on the pretrained net (on imagenet). it can be seen that the pretrained network converges much much faster, so I preferred to use fewer epochs also to reduce the execution time!

5.0.1 Prepare dataset with new transform (mean and std of imageNet)

the procedure is the same as before but this time in the normalize function (into transforms) i use mean and std of imagenet:

- mean: (0.485, 0.456, 0.406),
- std: (0.229, 0.224, 0.225)

```
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 5784 Images and label
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 2893 Images and label
Train Dataset: 5784
Test Dataset: 2893
Training split: 2892
Validation split: 2892
```

5.0.2 Load alexnet pretrained

the procedure is the same as before ,except for:

```
net = alexnet(pretrained=True)
```

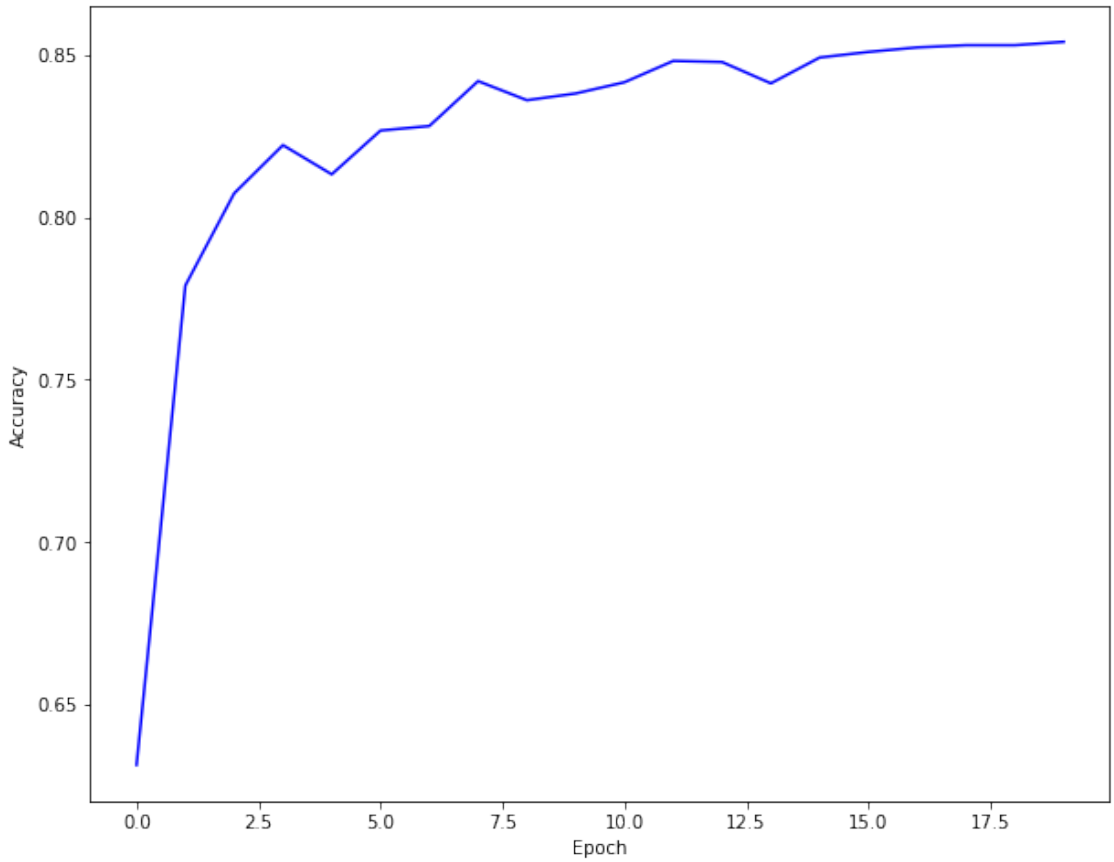
5.0.3 Alexnet structure:

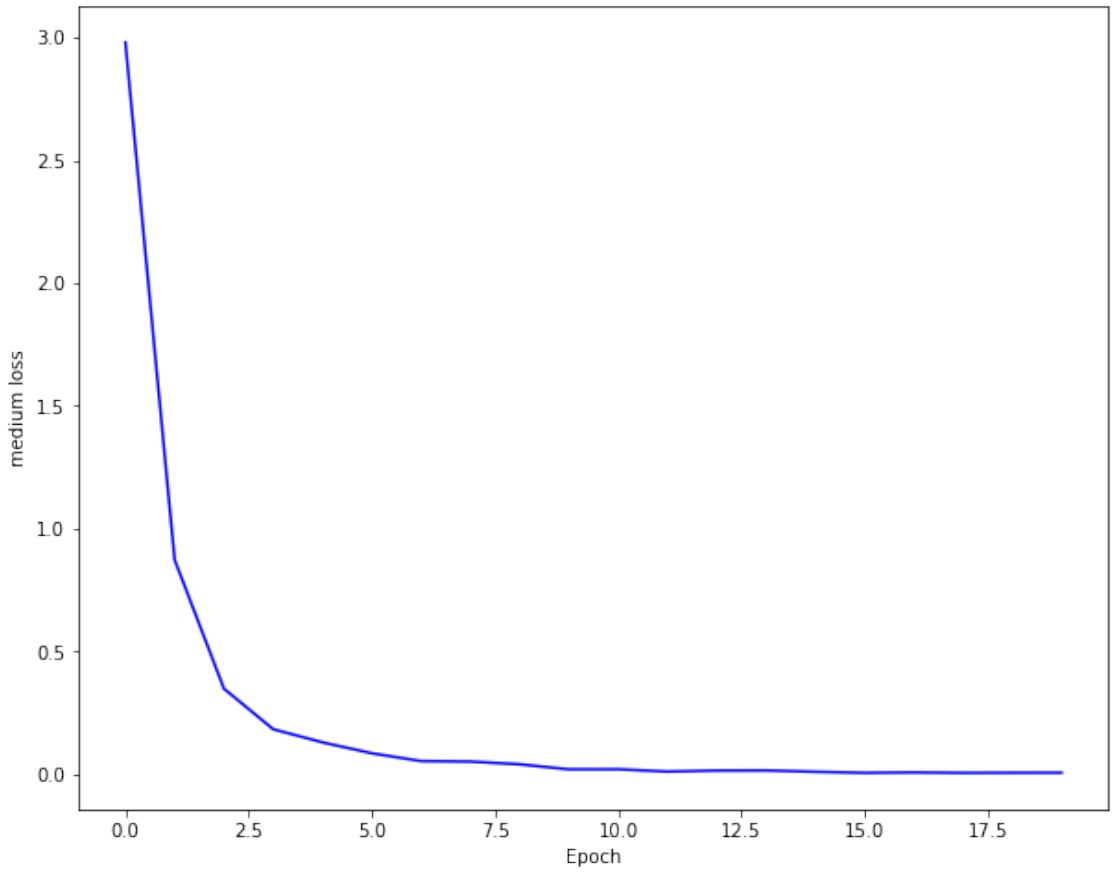
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=101, bias=True)
  )
)
```

5.0.4 Train the model pretrained and found best hyper-params

```
epoch 1/20, LR = [0.01]  loss : 2.9812 score: 0.6314  BEST!
epoch 2/20, LR = [0.01]  loss : 0.8728 score: 0.7790  BEST!
epoch 3/20, LR = [0.01]  loss : 0.3481 score: 0.8074  BEST!
epoch 4/20, LR = [0.01]  loss : 0.1828 score: 0.8223  BEST!
epoch 5/20, LR = [0.01]  loss : 0.1294 score: 0.8133
epoch 6/20, LR = [0.01]  loss : 0.0848 score: 0.8268  BEST!
epoch 7/20, LR = [0.01]  loss : 0.0531 score: 0.8281  BEST!
epoch 8/20, LR = [0.01]  loss : 0.0510 score: 0.8420  BEST!
epoch 9/20, LR = [0.01]  loss : 0.0400 score: 0.8361
epoch 10/20, LR = [0.01]  loss : 0.0195 score: 0.8382
epoch 11/20, LR = [0.01]  loss : 0.0198 score: 0.8416
epoch 12/20, LR = [0.01]  loss : 0.0103 score: 0.8482  BEST!
epoch 13/20, LR = [0.01]  loss : 0.0145 score: 0.8479
epoch 14/20, LR = [0.01]  loss : 0.0150 score: 0.8413
epoch 15/20, LR = [0.01]  loss : 0.0095 score: 0.8492  BEST!
epoch 16/20, LR = [0.001]  loss : 0.0052 score: 0.8510  BEST!
epoch 17/20, LR = [0.001]  loss : 0.0068 score: 0.8524  BEST!
epoch 18/20, LR = [0.001]  loss : 0.0050 score: 0.8530  BEST!
epoch 19/20, LR = [0.001]  loss : 0.0055 score: 0.8530
epoch 20/20, LR = [0.001]  loss : 0.0057 score: 0.8541  BEST!
```


0.85
0.80
0.75
0.70
0.65
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch



lr	epochs	step	loss	score (validation)
0.05	20	15	div!	div!
0.01	20	15	0.001	0.854
0.01	30	15	0.0012	0.852
0.008	20	15	0.0024	0.85
0.005	15	10	0.007	0.84
0.005	30	25	0.002	0.853
0.001	30	25	0.0015	0.83

as you can see the results are much better than the previous model. even at the second epoch we reach a 75% accuracy on the validation set, which for me is a very good result; until the accuracy reach, after 20 epoch, just over 85%, impressive!

but i tuned again different hyperparameters because the best ones in the "not pretrained" model didn't perform very well.

i could even continue to decrease the number of epochs and by making a more precise tuning, obtain even better results

```
score: 0.8496
```

5.1 Freezing layer

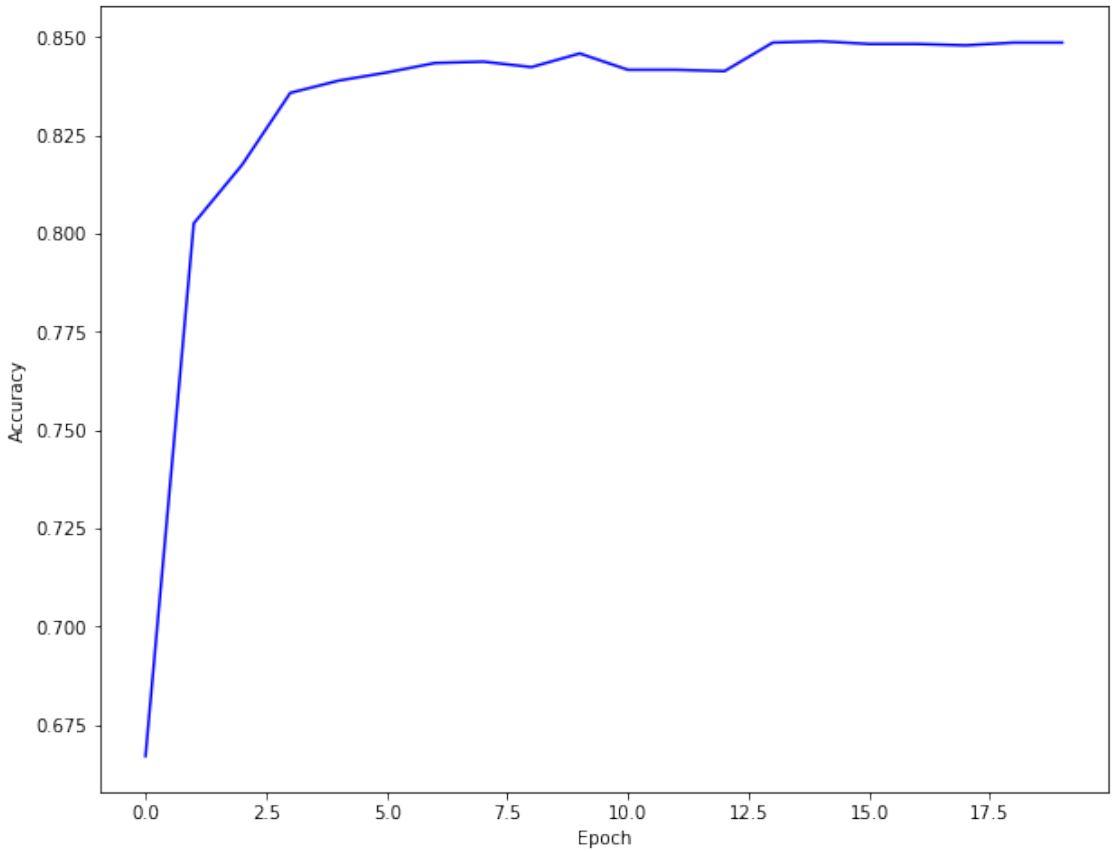
- LR= 0.01
- 20 epochs
- step size = 15

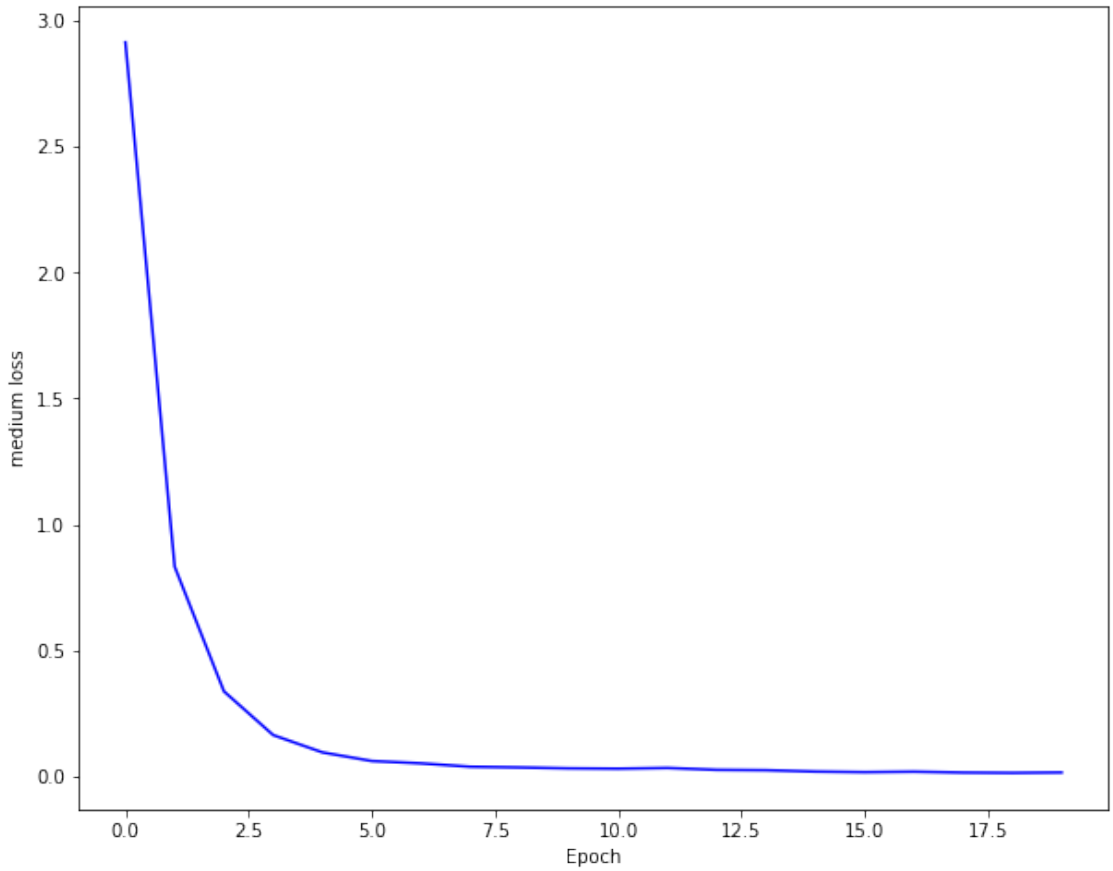
in this section I try the hyper-parameters found previously:

- freezing the convolutional layers
- freezing the fully connected layers

5.1.1 Training and test freezing convolutional layers

```
epoch 1/20, LR = [0.01]  loss : 2.9148 score: 0.6670  BEST!
epoch 2/20, LR = [0.01]  loss : 0.8320 score: 0.8026  BEST!
epoch 3/20, LR = [0.01]  loss : 0.3365 score: 0.8174  BEST!
epoch 4/20, LR = [0.01]  loss : 0.1628 score: 0.8358  BEST!
epoch 5/20, LR = [0.01]  loss : 0.0940 score: 0.8389  BEST!
epoch 6/20, LR = [0.01]  loss : 0.0598 score: 0.8409  BEST!
epoch 7/20, LR = [0.01]  loss : 0.0505 score: 0.8434  BEST!
epoch 8/20, LR = [0.01]  loss : 0.0371 score: 0.8437  BEST!
epoch 9/20, LR = [0.01]  loss : 0.0342 score: 0.8423
epoch 10/20, LR = [0.01]  loss : 0.0306 score: 0.8458  BEST!
epoch 11/20, LR = [0.01]  loss : 0.0293 score: 0.8416
epoch 12/20, LR = [0.01]  loss : 0.0322 score: 0.8416
epoch 13/20, LR = [0.01]  loss : 0.0247 score: 0.8413
epoch 14/20, LR = [0.01]  loss : 0.0233 score: 0.8485  BEST!
epoch 15/20, LR = [0.01]  loss : 0.0179 score: 0.8489  BEST!
epoch 16/20, LR = [0.001]  loss : 0.0155 score: 0.8482
epoch 17/20, LR = [0.001]  loss : 0.0174 score: 0.8482
epoch 18/20, LR = [0.001]  loss : 0.0138 score: 0.8479
epoch 19/20, LR = [0.001]  loss : 0.0127 score: 0.8485
epoch 20/20, LR = [0.001]  loss : 0.0142 score: 0.8485
```


0.850
0.825
0.800
0.775
0.750
0.725
0.700
0.675
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch

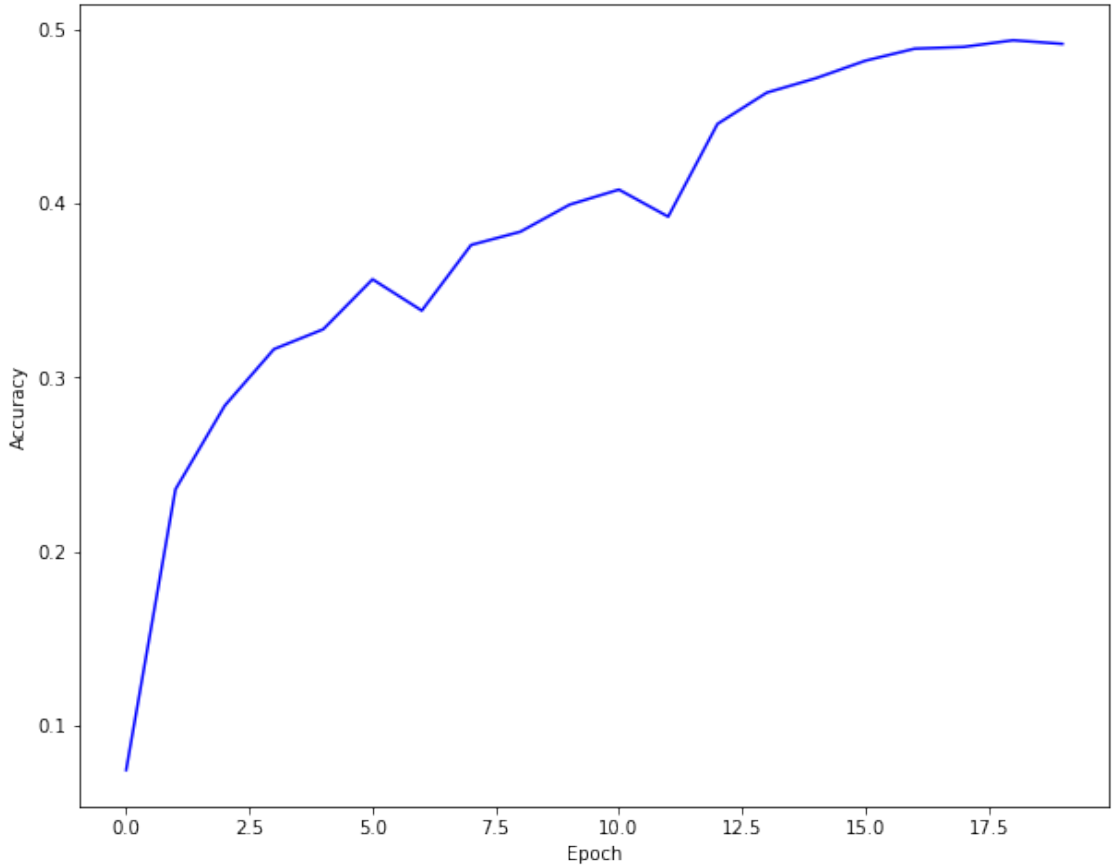


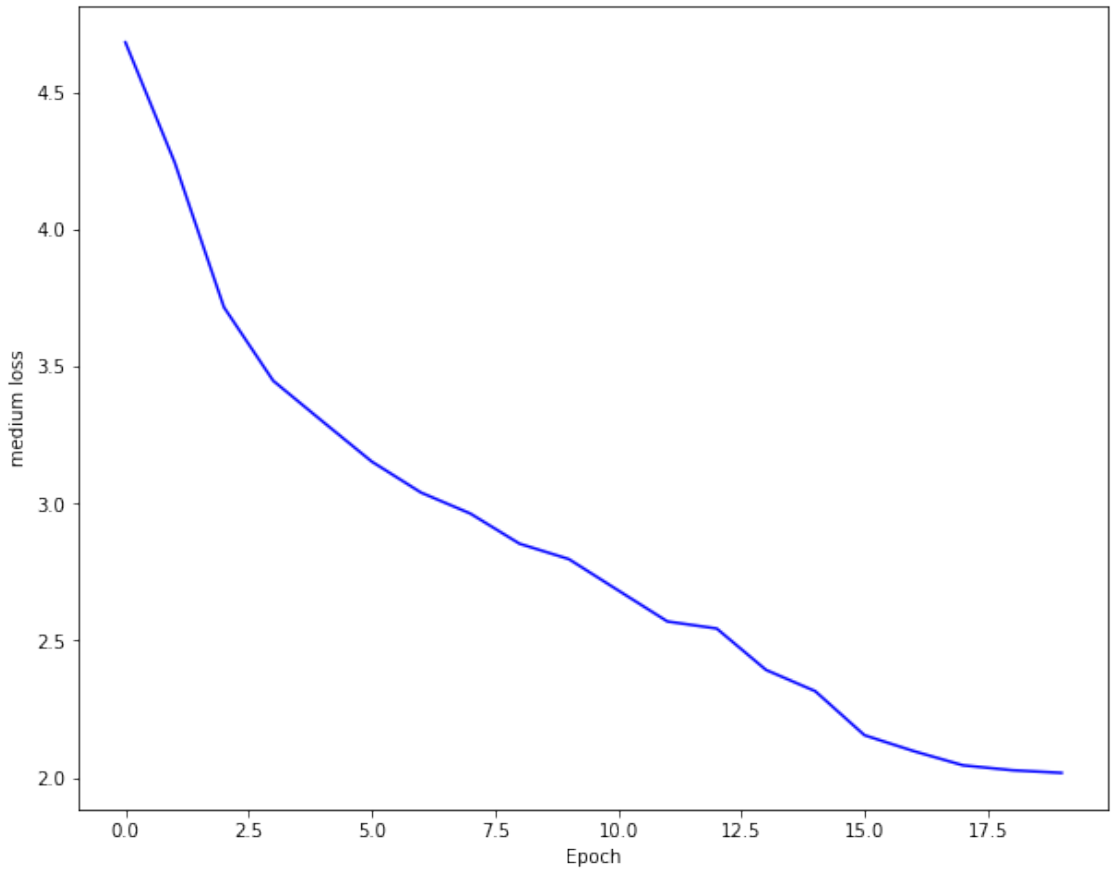

```
score: 0.8441
```

5.1.2 Training and freezing the fully connected layers

```
epoch 1/20, LR = [0.01]  loss : 4.6834 score: 0.0747  BEST!
epoch 2/20, LR = [0.01]  loss : 4.2454 score: 0.2358  BEST!
epoch 3/20, LR = [0.01]  loss : 3.7166 score: 0.2839  BEST!
epoch 4/20, LR = [0.01]  loss : 3.4482 score: 0.3164  BEST!
epoch 5/20, LR = [0.01]  loss : 3.3001 score: 0.3278  BEST!
epoch 6/20, LR = [0.01]  loss : 3.1533 score: 0.3565  BEST!
epoch 7/20, LR = [0.01]  loss : 3.0402 score: 0.3385
epoch 8/20, LR = [0.01]  loss : 2.9637 score: 0.3762  BEST!
epoch 9/20, LR = [0.01]  loss : 2.8535 score: 0.3838  BEST!
epoch 10/20, LR = [0.01]  loss : 2.7974 score: 0.3994  BEST!
epoch 11/20, LR = [0.01]  loss : 2.6833 score: 0.4080  BEST!
epoch 12/20, LR = [0.01]  loss : 2.5698 score: 0.3925
epoch 13/20, LR = [0.01]  loss : 2.5440 score: 0.4457  BEST!
epoch 14/20, LR = [0.01]  loss : 2.3931 score: 0.4637  BEST!
epoch 15/20, LR = [0.01]  loss : 2.3154 score: 0.4720  BEST!
```

```
epoch 16/20, LR = [0.001]  loss : 2.1546 score: 0.4820  BEST!
epoch 17/20, LR = [0.001]  loss : 2.0965 score: 0.4889  BEST!
epoch 18/20, LR = [0.001]  loss : 2.0448 score: 0.4900  BEST!
epoch 19/20, LR = [0.001]  loss : 2.0268 score: 0.4938  BEST!
epoch 20/20, LR = [0.001]  loss : 2.0172 score: 0.4917
```






```
score: 0.4860
```

Compare results

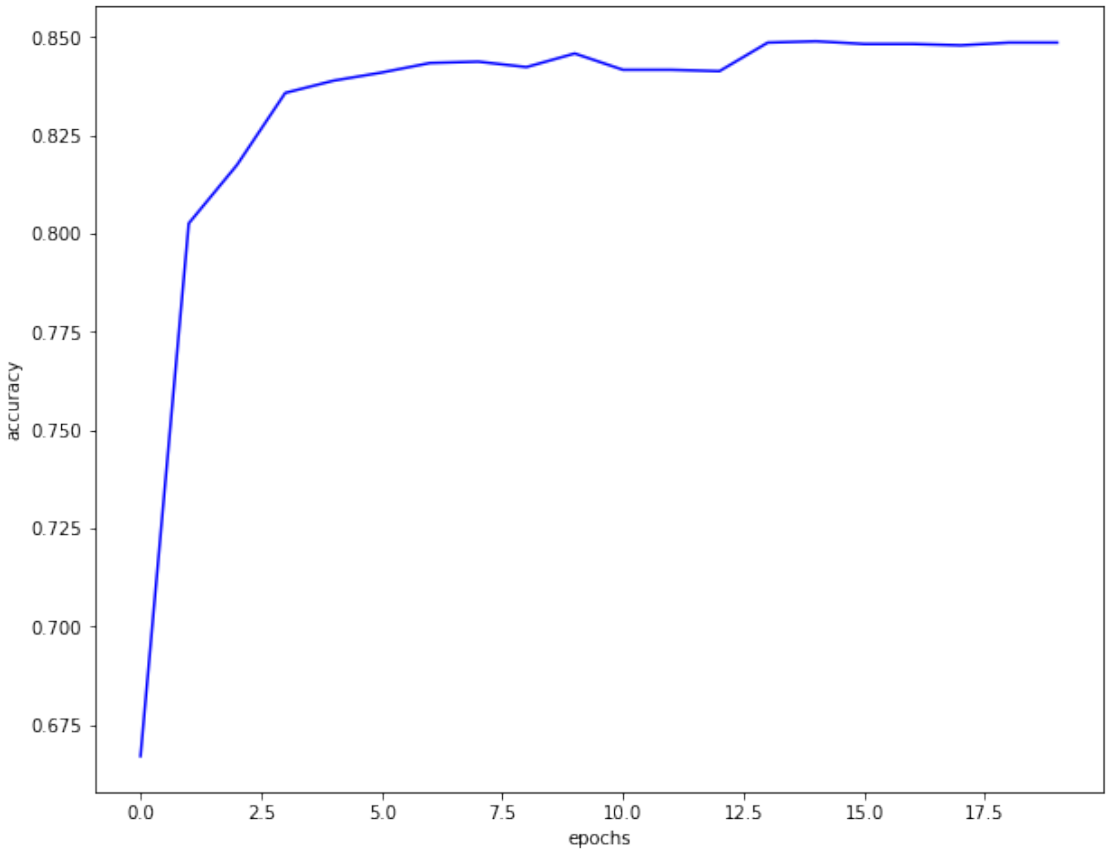
as it was easy to imagine,training only the FC, the model converge much much faster then training only the CONV,

in this case even the results (training only FC) are a very very similiar than training the entire network!

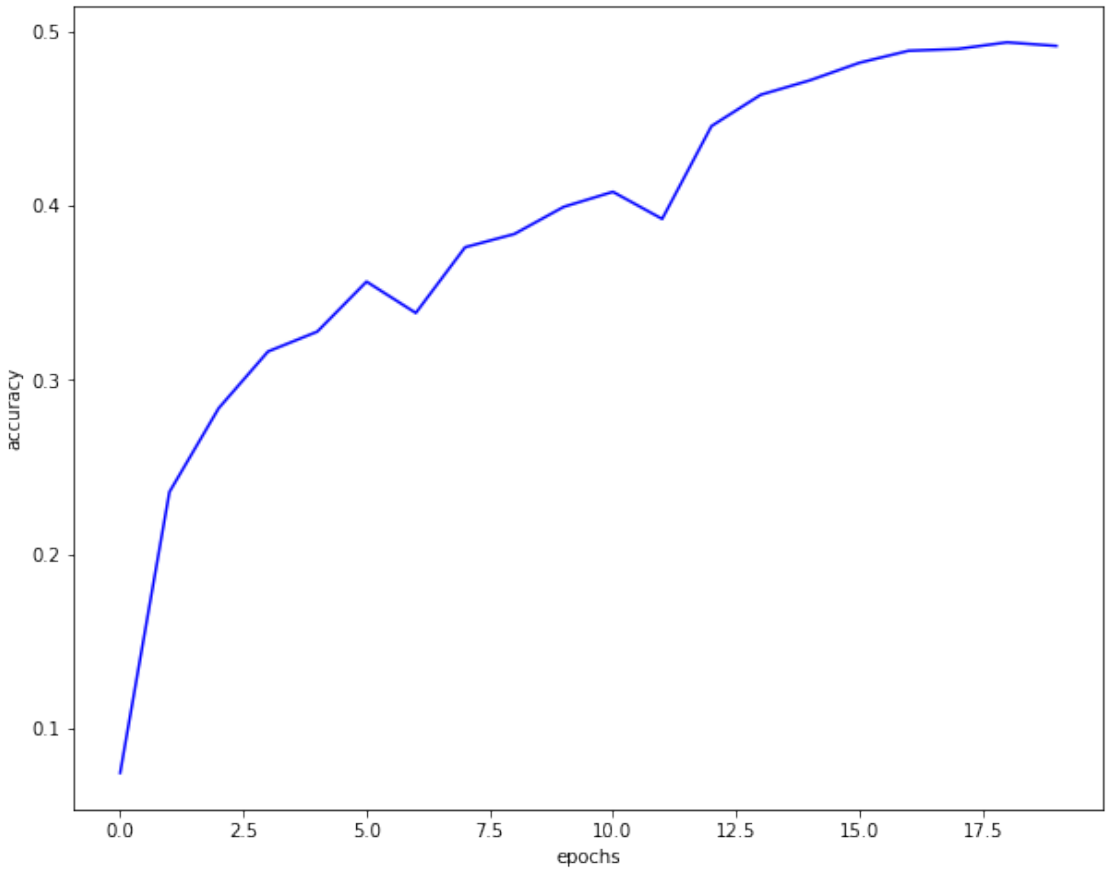
this is because normally when we do transfer learning it is good practice to do fine-tuning on the highest levels of the network,And not on all levels. this because going down deep on the net, the first layers are those that identify more generic features, such as borders, shadows etc, while the layers at the upper (generally the FC layers) are those that identify more specific features. we do not need to re-train also the convolutionary layers because their weights are already adequate.

type of training (layers)	loss	score(val)	score(test)
all	0.001	0.85	0.85
only Fully connected	0.01	0.84	0.84
only convolutional	2	0.49	0.48

```
ACCURACY OBTAINED WHEN TRAINING FULLY CONNECTED LAYER
```



ACCURACY OBTAINED WHEN TRAINING CONVOLUTIONAL LAYER



6 Data augmentation

in this section I experiment with different types of image transformations: - (stock) centerCrop - randomCrop - centerCrop + randomHorizontalFlip - centerCrop + randomRotation(180°)

6.1 Prepare datasets with new transforms

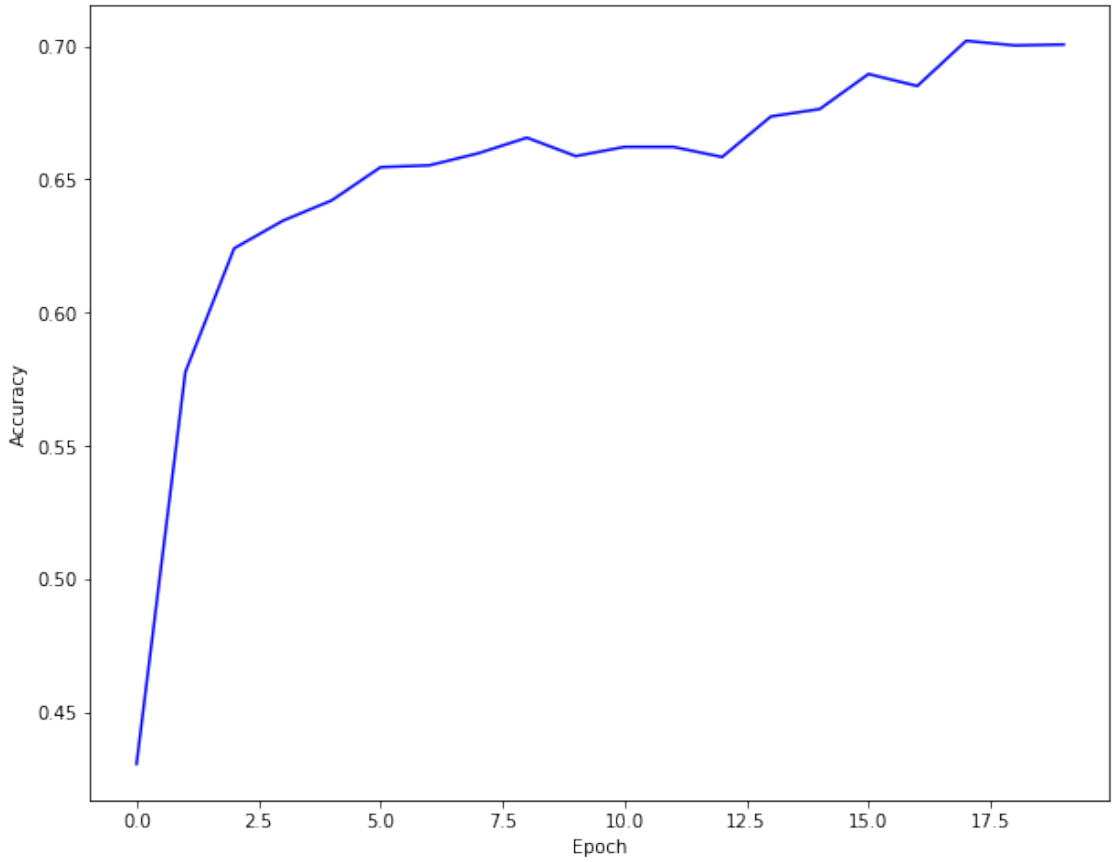
```
[stock transforms]
 Compose(
    Resize(size=256, interpolation=PIL.Image.BILINEAR)
    CenterCrop(size=(224, 224))
    ToTensor()
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
)
[transforms #1]
 Compose(
    Resize(size=256, interpolation=PIL.Image.BILINEAR)
    RandomCrop(size=(224, 224), padding=None)
    ToTensor()
```

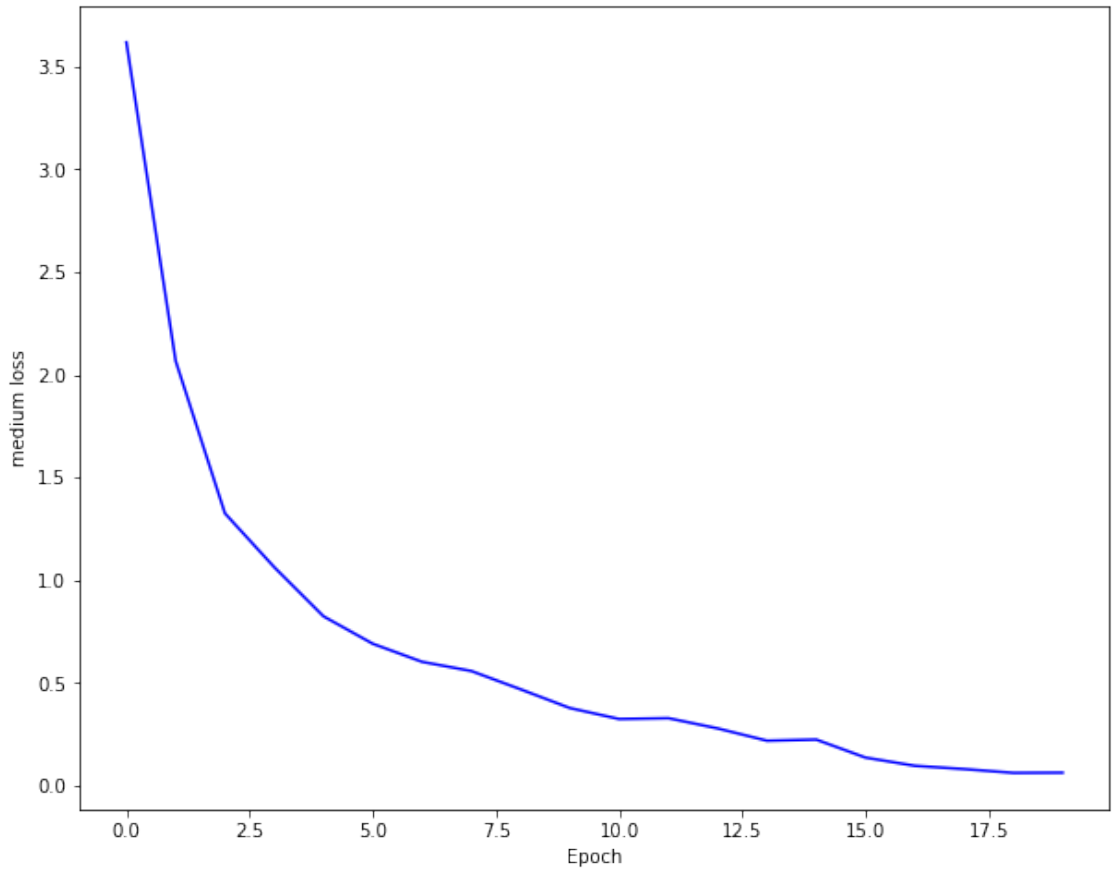
```
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
)
[transform #2]
 Compose(
    Resize(size=256, interpolation=PIL.Image.BILINEAR)
    CenterCrop(size=(224, 224))
    RandomHorizontalFlip(p=0.5)
    ToTensor()
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
)
[transform #3]
 Compose(
    Resize(size=256, interpolation=PIL.Image.BILINEAR)
    CenterCrop(size=(224, 224))
    RandomRotation(degrees=(-180, 180), resample=False, expand=False)
    ToTensor()
    Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
)
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 5784 Images and label
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 2893 Images and label
Train Dataset: 5784
Test Dataset: 2893
Training split: 2892
Validation split: 2892
```

6.1.1 Test the new transformations

```
epoch 1/20, LR = [0.01]  loss : 3.6166 score: 0.4308  BEST!
epoch 2/20, LR = [0.01]  loss : 2.0666 score: 0.5778  BEST!
epoch 3/20, LR = [0.01]  loss : 1.3252 score: 0.6241  BEST!
epoch 4/20, LR = [0.01]  loss : 1.0635 score: 0.6345  BEST!
epoch 5/20, LR = [0.01]  loss : 0.8243 score: 0.6421  BEST!
epoch 6/20, LR = [0.01]  loss : 0.6909 score: 0.6546  BEST!
epoch 7/20, LR = [0.01]  loss : 0.6029 score: 0.6553  BEST!
epoch 8/20, LR = [0.01]  loss : 0.5582 score: 0.6598  BEST!
epoch 9/20, LR = [0.01]  loss : 0.4692 score: 0.6656  BEST!
epoch 10/20, LR = [0.01]  loss : 0.3781 score: 0.6587
epoch 11/20, LR = [0.01]  loss : 0.3247 score: 0.6622
epoch 12/20, LR = [0.01]  loss : 0.3288 score: 0.6622
epoch 13/20, LR = [0.01]  loss : 0.2789 score: 0.6584
epoch 14/20, LR = [0.01]  loss : 0.2192 score: 0.6736  BEST!
epoch 15/20, LR = [0.01]  loss : 0.2246 score: 0.6763  BEST!
epoch 16/20, LR = [0.001]  loss : 0.1370 score: 0.6895  BEST!
epoch 17/20, LR = [0.001]  loss : 0.0971 score: 0.6850
epoch 18/20, LR = [0.001]  loss : 0.0811 score: 0.7019  BEST!
```

```
epoch 19/20, LR = [0.001]  loss : 0.0627 score: 0.7002
epoch 20/20, LR = [0.001]  loss : 0.0636 score: 0.7006
```






```
score: 0.6371
```

	transforms	loss	score (test)
stock	center-crop	0.001	0.85
1	random crop	0.018	0.833
2	center-crop + random horizontal flip	0.01	0.78
3	center-crop + random rotation (180°)	0.07	0.64

this time I couldn't get better results than the previous ones (but quite similar). however I think that using different types of transformations greatly helps to reduce overfitting (wee can see this on the loss wich is medium higher than the previous) and improve accuracy

Another way can be increase the cardinality of data by concatenate different sets with different transformations

7 Beyond AlexNet

7.1 Resnet

Very deep network using residual connection.

Resnet is very different model and much deeper than alexnet;
it starts from the hypothesis that deep models are more difficult to optimize;
solution: use layers to fit residual F(x) = H(x) + x instead of H(x) directly
resnet is composed by "residual blocks", every block has two 3x3 conv layers
since the network is much deeper, it is necessary to lower (even a lot!) the batch size to avoid saturating the gpu ram.

*I've also tried the **vgg16** but i couldn't get it to work at its best, and the training was really really slow. and sometimes (even with a very lower batch size) bring Colab to crash for too much Vram request*

7.1.1 Resnet vs alexnet vs vgg16

we can see the huge structural differences between these models.

```
-------ALEXNET-----
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

```
-------RESNET-----
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
```

```
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
-------VGG16-----
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
```

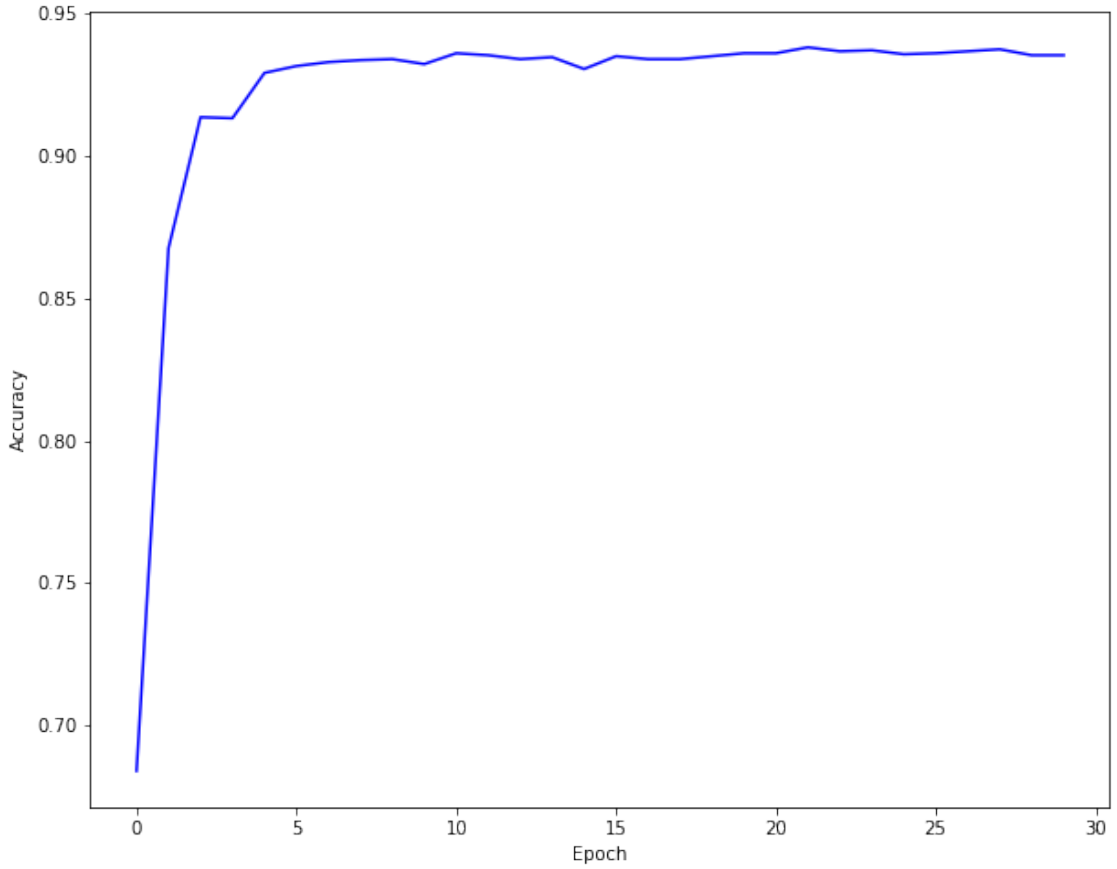
```
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

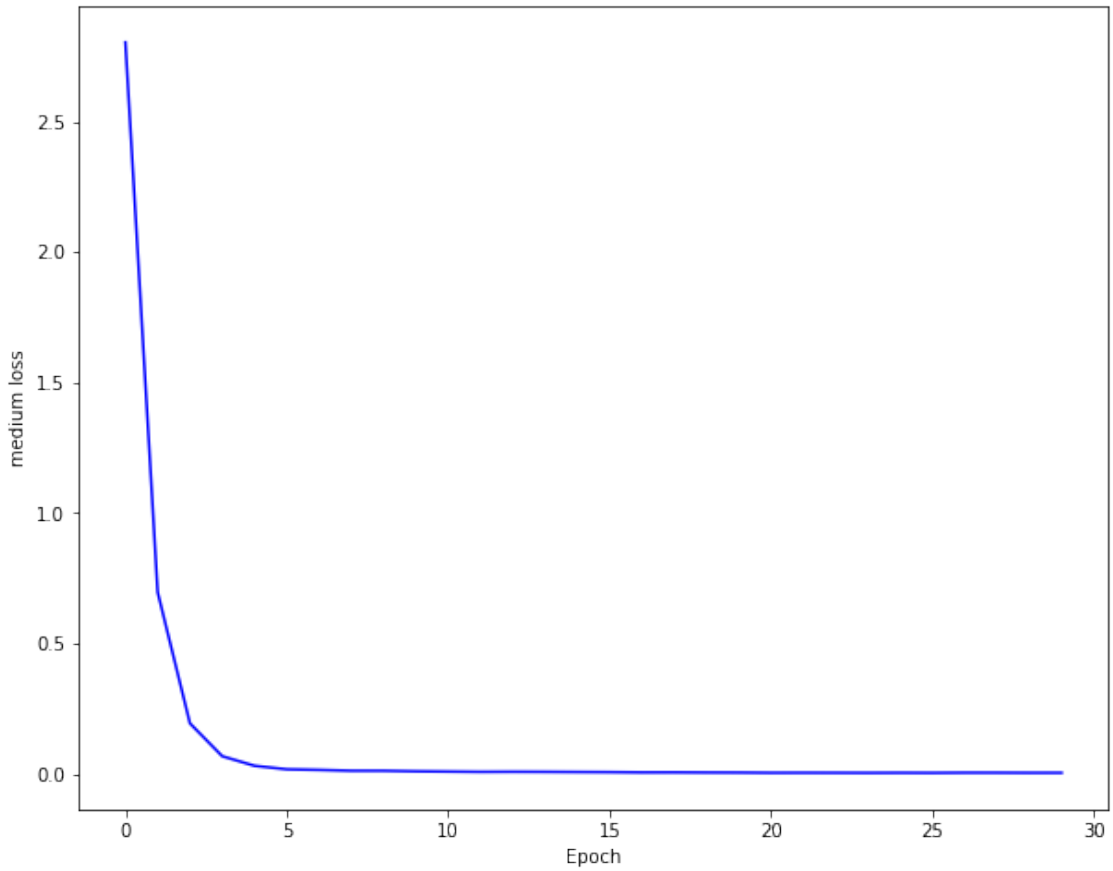
7.1.2 Test with resnet

```
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 5784 Images and label
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 2893 Images and label
Train Dataset: 5784
Test Dataset: 2893
Training split: 2892
Validation split: 2892


epoch 1/30, LR = [0.02]  loss : 2.8040 score: 0.6840  BEST!
epoch 2/30, LR = [0.02]  loss : 0.6968 score: 0.8676  BEST!
epoch 3/30, LR = [0.02]  loss : 0.1940 score: 0.9136  BEST!
epoch 4/30, LR = [0.02]  loss : 0.0682 score: 0.9132
epoch 5/30, LR = [0.02]  loss : 0.0314 score: 0.9291  BEST!
```

```
epoch 6/30, LR = [0.02]  loss : 0.0180 score: 0.9315  BEST!
epoch 7/30, LR = [0.02]  loss : 0.0157 score: 0.9329  BEST!
epoch 8/30, LR = [0.02]  loss : 0.0124 score: 0.9336  BEST!
epoch 9/30, LR = [0.02]  loss : 0.0123 score: 0.9340  BEST!
epoch 10/30, LR = [0.02]  loss : 0.0103 score: 0.9322
epoch 11/30, LR = [0.02]  loss : 0.0093 score: 0.9360  BEST!
epoch 12/30, LR = [0.02]  loss : 0.0082 score: 0.9353
epoch 13/30, LR = [0.02]  loss : 0.0085 score: 0.9340
epoch 14/30, LR = [0.02]  loss : 0.0082 score: 0.9346
epoch 15/30, LR = [0.02]  loss : 0.0076 score: 0.9305
epoch 16/30, LR = [0.02]  loss : 0.0071 score: 0.9350
epoch 17/30, LR = [0.02]  loss : 0.0061 score: 0.9340
epoch 18/30, LR = [0.02]  loss : 0.0063 score: 0.9340
epoch 19/30, LR = [0.02]  loss : 0.0057 score: 0.9350
epoch 20/30, LR = [0.02]  loss : 0.0054 score: 0.9360
epoch 21/30, LR = [0.002]  loss : 0.0046 score: 0.9360
epoch 22/30, LR = [0.002]  loss : 0.0047 score: 0.9381  BEST!
epoch 23/30, LR = [0.002]  loss : 0.0046 score: 0.9367
epoch 24/30, LR = [0.002]  loss : 0.0043 score: 0.9371
epoch 25/30, LR = [0.002]  loss : 0.0046 score: 0.9357
epoch 26/30, LR = [0.002]  loss : 0.0043 score: 0.9360
epoch 27/30, LR = [0.002]  loss : 0.0050 score: 0.9367
epoch 28/30, LR = [0.002]  loss : 0.0049 score: 0.9374
epoch 29/30, LR = [0.002]  loss : 0.0045 score: 0.9353
epoch 30/30, LR = [0.002]  loss : 0.0046 score: 0.9353
```


0.95
0.90
0.85
0.80
0.75
0.70
Accuracy
0
5
10
15
20
25
30
Epoch




```
score: 0.9378
```