Assignment2_s263138

December 13, 2019

0.0.1 Ivan Murabito-s263138

1 Homework 2 DeepLearning and experiments on AlexNet

2 Training from scratch

Install requirements

Import libraries

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

Set Arguments

Block of given arguments (LR,STEP_SIZE,EPOCHS etc)

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)

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

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

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
```

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
```

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)

Prepare Network

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

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

2.1 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.

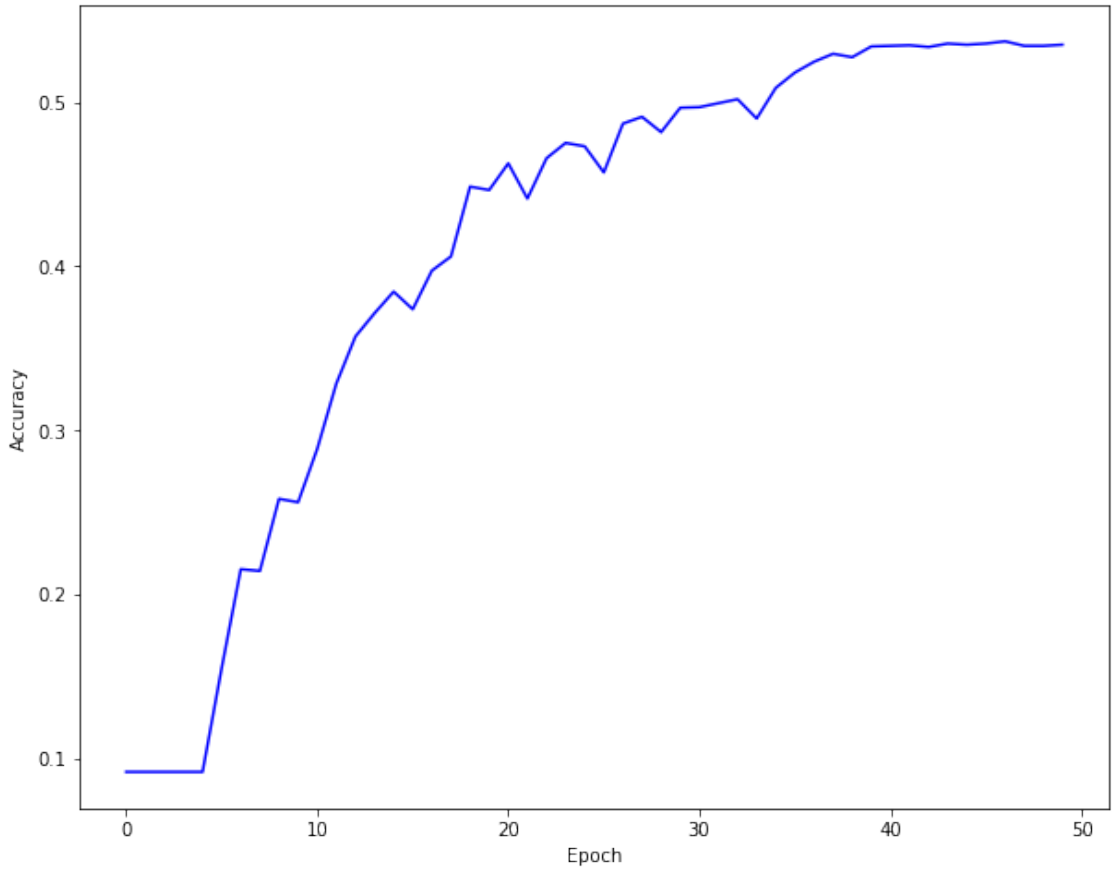
the printed output below show the epoch,loss,val_score at the end two graphs showing the trend of these variables over the epochs.

the printed output below show (..one of) the best model that i found over experiments: - **LR:0.05 - 50 epochs - step size:35**

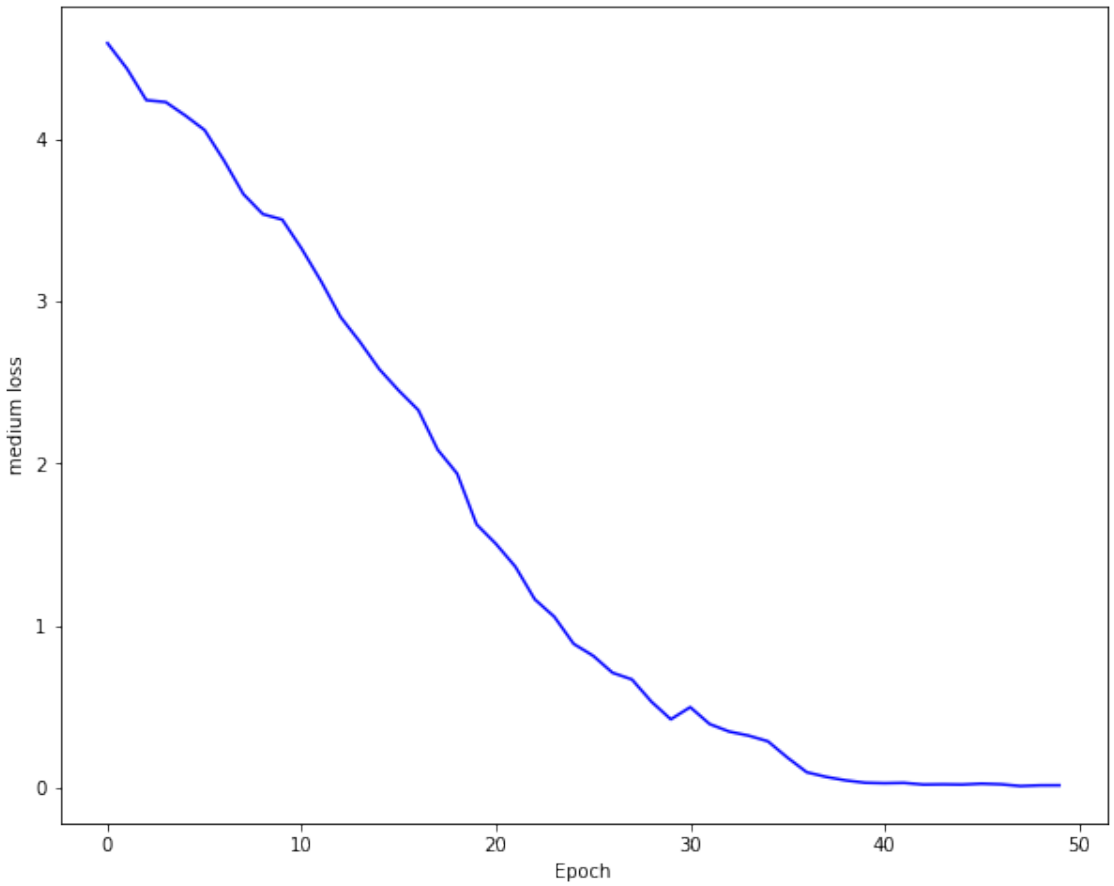
```
epoch 1/50, LR = [0.05]  loss : 4.593657623637807 validation score:
0.09197786998616875  BEST MODEL found!
epoch 2/50, LR = [0.05]  loss : 4.4370271075855605 validation score:
0.09197786998616875
epoch 3/50, LR = [0.05]  loss : 4.2419892224398525 validation score:
0.09197786998616875
epoch 4/50, LR = [0.05]  loss : 4.2301602797074755 validation score:
0.09197786998616875
epoch 5/50, LR = [0.05]  loss : 4.147969852794301 validation score:
0.09197786998616875
epoch 6/50, LR = [0.05]  loss : 4.057804692875255 validation score:
0.15525587828492393  BEST MODEL found!
epoch 7/50, LR = [0.05]  loss : 3.869585644115101 validation score:
0.21542185338865838  BEST MODEL found!
epoch 8/50, LR = [0.05]  loss : 3.662203983827071 validation score:
0.2143845089903181
epoch 9/50, LR = [0.05]  loss : 3.5391142151572486 validation score:
0.258298755186722  BEST MODEL found!
epoch 10/50, LR = [0.05]  loss : 3.505119735544378 validation score:
0.25622406639004147
epoch 11/50, LR = [0.05]  loss : 3.3253775509920986 validation score:
0.2887275242047026  BEST MODEL found!
epoch 12/50, LR = [0.05]  loss : 3.1246937838467685 validation score:
0.3284923928077455  BEST MODEL found!
epoch 13/50, LR = [0.05]  loss : 2.904759233648127 validation score:
0.35719225449515907  BEST MODEL found!
epoch 14/50, LR = [0.05]  loss : 2.750786781311035 validation score:
0.37136929460580914  BEST MODEL found!
epoch 15/50, LR = [0.05]  loss : 2.58216853575273 validation score:
0.38450899031811897  BEST MODEL found!
epoch 16/50, LR = [0.05]  loss : 2.450655048543757 validation score:
0.37378976486860305
epoch 17/50, LR = [0.05]  loss : 2.331090060147372 validation score:
0.39730290456431533  BEST MODEL found!
epoch 18/50, LR = [0.05]  loss : 2.086157787929882 validation score:
0.40594744121715076  BEST MODEL found!
epoch 19/50, LR = [0.05]  loss : 1.939303083853288 validation score:
0.448478561549101  BEST MODEL found!
epoch 20/50, LR = [0.05]  loss : 1.6259516694329001 validation score:
0.4464038727524205
epoch 21/50, LR = [0.05]  loss : 1.5055184797807173 validation score:
0.46265560165975106  BEST MODEL found!
epoch 22/50, LR = [0.05]  loss : 1.364645611156117 validation score:
0.4412171507607192
epoch 23/50, LR = [0.05]  loss : 1.1630618897351352 validation score:
0.4657676348547718  BEST MODEL found!
epoch 24/50, LR = [0.05]  loss : 1.0565672516822815 validation score:
```

```
0.475103734439834  BEST MODEL found!
epoch 25/50, LR = [0.05]  loss : 0.8889344822276722 validation score:
0.4730290456431535
epoch 26/50, LR = [0.05]  loss : 0.8157401518388228 validation score:
0.45712309820193636
epoch 27/50, LR = [0.05]  loss : 0.710897380655462 validation score:
0.48686030428769017  BEST MODEL found!
epoch 28/50, LR = [0.05]  loss : 0.669519001787359 validation score:
0.49100968188105115  BEST MODEL found!
epoch 29/50, LR = [0.05]  loss : 0.532810628414154 validation score:
0.48167358229598894
epoch 30/50, LR = [0.05]  loss : 0.4236672656102614 validation score:
0.49654218533886585  BEST MODEL found!
epoch 31/50, LR = [0.05]  loss : 0.49858367984945123 validation score:
0.49688796680497926  BEST MODEL found!
epoch 32/50, LR = [0.05]  loss : 0.3943786052140323 validation score:
0.49930843706777317  BEST MODEL found!
epoch 33/50, LR = [0.05]  loss : 0.34824153916402295 validation score:
0.5017289073305671  BEST MODEL found!
epoch 34/50, LR = [0.05]  loss : 0.3231798234311017 validation score:
0.4899723374827109
epoch 35/50, LR = [0.05]  loss : 0.2879230698401278 validation score:
0.5086445366528354  BEST MODEL found!
epoch 36/50, LR = [0.005000000000000001]  loss : 0.1879965290427208 validation
score: 0.5179806362378977  BEST MODEL found!
epoch 37/50, LR = [0.005000000000000001]  loss : 0.09711063043637709 validation
score: 0.5245504840940526  BEST MODEL found!
epoch 38/50, LR = [0.005000000000000001]  loss : 0.06843117522922429 validation
score: 0.5293914246196404  BEST MODEL found!
epoch 39/50, LR = [0.005000000000000001]  loss : 0.047134770757772705 validation
score: 0.5273167358229599
epoch 40/50, LR = [0.005000000000000001]  loss : 0.03254999609833414 validation
score: 0.5338865836791148  BEST MODEL found!
epoch 41/50, LR = [0.005000000000000001]  loss : 0.029788913374597378 validation
score: 0.5342323651452282  BEST MODEL found!
epoch 42/50, LR = [0.005000000000000001]  loss : 0.03175303238359364 validation
score: 0.5345781466113416  BEST MODEL found!
epoch 43/50, LR = [0.005000000000000001]  loss : 0.021273088895461777 validation
score: 0.5335408022130014
epoch 44/50, LR = [0.005000000000000001]  loss : 0.022669729861346157 validation
score: 0.5356154910096819  BEST MODEL found!
epoch 45/50, LR = [0.005000000000000001]  loss : 0.0212044683708386 validation
score: 0.534923928077455
epoch 46/50, LR = [0.005000000000000001]  loss : 0.02689671397886493 validation
score: 0.5356154910096819
epoch 47/50, LR = [0.005000000000000001]  loss : 0.022916724566708912 validation
score: 0.5369986168741355  BEST MODEL found!
epoch 48/50, LR = [0.005000000000000001]  loss : 0.012008548629554834 validation
```

```
score: 0.5342323651452282
epoch 49/50, LR = [0.005000000000000001]  loss : 0.016575152731754562 validation
score: 0.5342323651452282
epoch 50/50, LR = [0.005000000000000001]  loss : 0.016787321052768013 validation
```




```
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
```

##**Test** evaluate the best model on test set

```
test set  score: 0.5368129968890425
```

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	0.016	0.54	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.

3 Transfer Learning

in this section I do the same experiments as before but this time on the pretrained net.

for 'pretrained' i mean:

```
Use the weights learned by training on a large related dataset as a starting point for
training on the small dataset
```

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!

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
```

load alexnet pretrained

the procedure is the same as before ,except for:

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

Alexnet structure:

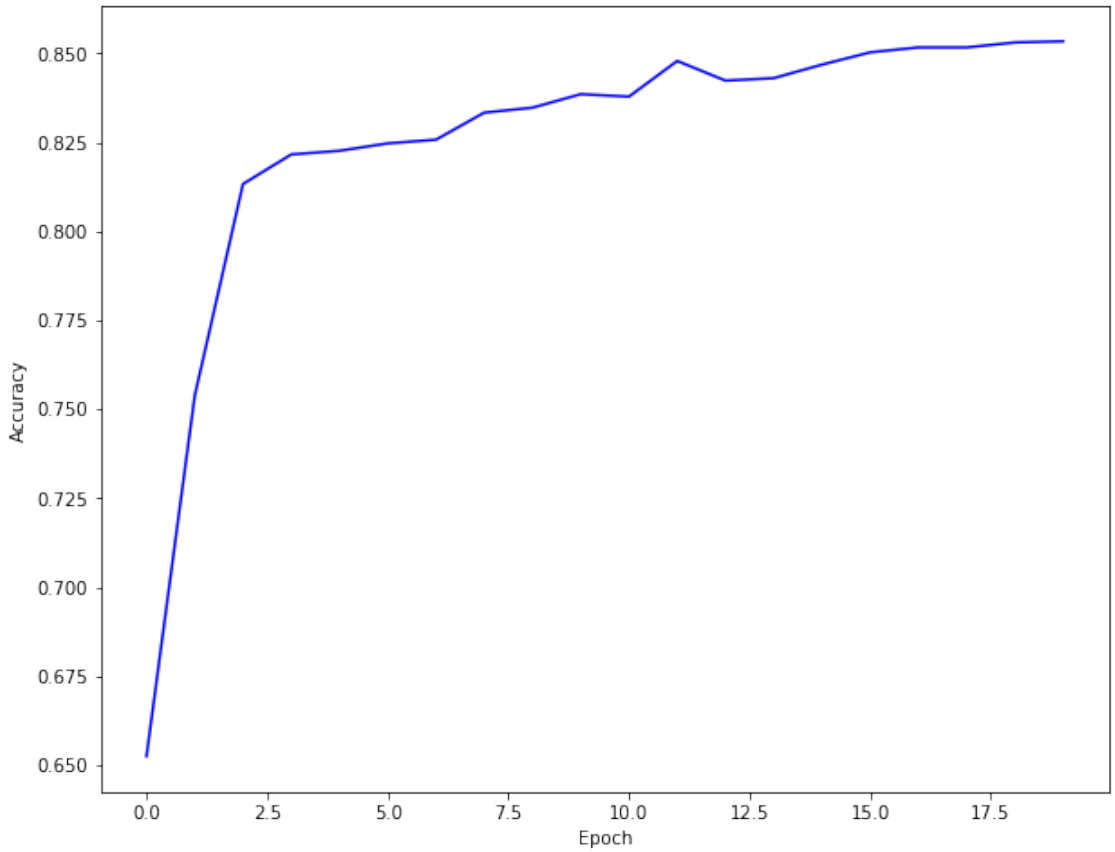
```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))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

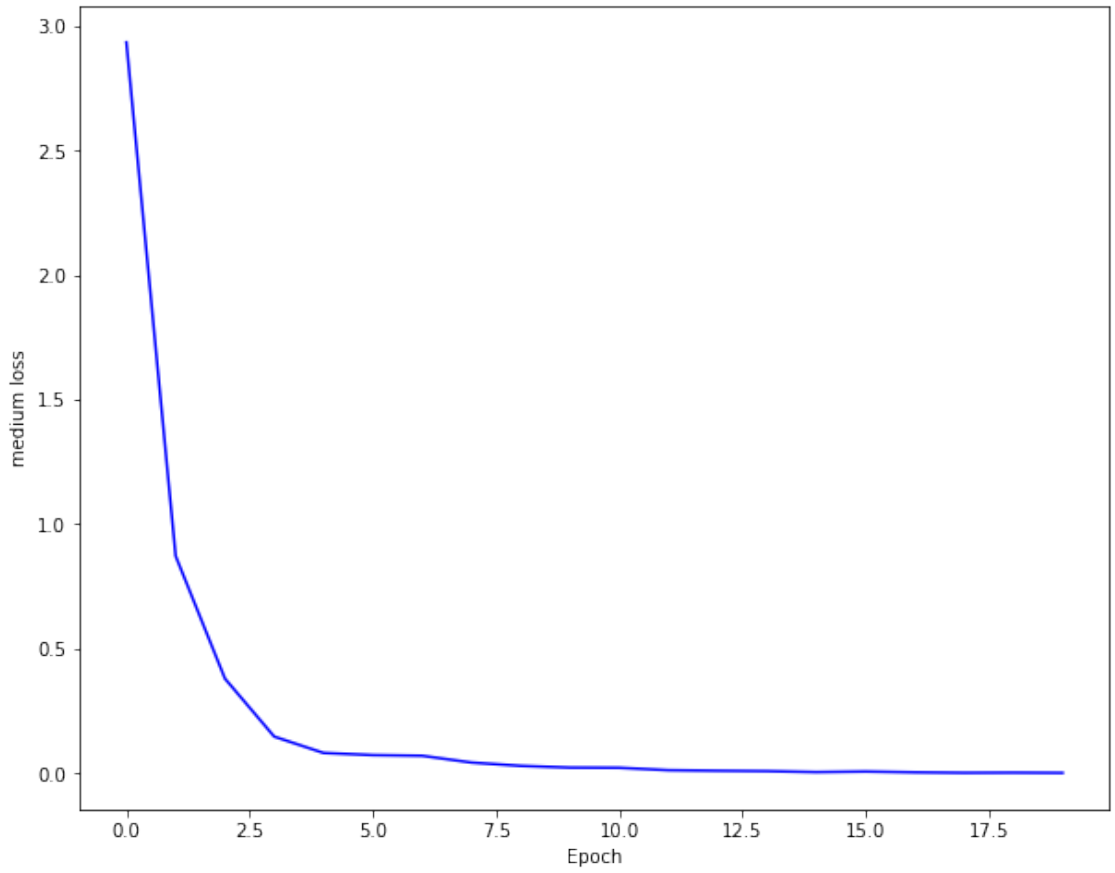
```
ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 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)
  )
)
```

Train the model pretrained and found best hyper-params

```
epoch 1/20, LR = [0.01]  loss : 2.9351478815078735 validation score:
0.6524896265560166  BEST MODEL found!
epoch 2/20, LR = [0.01]  loss : 0.8723358620296825 validation score:
0.7538035961272476  BEST MODEL found!
epoch 3/20, LR = [0.01]  loss : 0.38032312420281494 validation score:
0.8132780082987552  BEST MODEL found!
epoch 4/20, LR = [0.01]  loss : 0.14736244624311273 validation score:
0.8215767634854771  BEST MODEL found!
epoch 5/20, LR = [0.01]  loss : 0.0816753598099405 validation score:
0.8226141078838174  BEST MODEL found!
epoch 6/20, LR = [0.01]  loss : 0.0730857253074646 validation score:
0.8246887966804979  BEST MODEL found!
epoch 7/20, LR = [0.01]  loss : 0.07041364942084659 validation score:
0.8257261410788381  BEST MODEL found!
epoch 8/20, LR = [0.01]  loss : 0.04311365938999436 validation score:
0.8333333333333334  BEST MODEL found!
epoch 9/20, LR = [0.01]  loss : 0.029842208393595436 validation score:
0.834716459197787  BEST MODEL found!
epoch 10/20, LR = [0.01]  loss : 0.022857032038948753 validation score:
0.8385200553250346  BEST MODEL found!
epoch 11/20, LR = [0.01]  loss : 0.022098914804783733 validation score:
0.8378284923928078
epoch 12/20, LR = [0.01]  loss : 0.012373168190771883 validation score:
```

```
0.8478561549100968  BEST MODEL found!
epoch 13/20, LR = [0.01]  loss : 0.01010569151152264 validation score:
0.8423236514522822
epoch 14/20, LR = [0.01]  loss : 0.008650402792475441 validation score:
0.843015214384509
epoch 15/20, LR = [0.01]  loss : 0.004677711562676864 validation score:
0.8468188105117566
epoch 16/20, LR = [0.001]  loss : 0.007059651003642516 validation score:
0.8502766251728907  BEST MODEL found!
epoch 17/20, LR = [0.001]  loss : 0.0037108432840217242 validation score:
0.8516597510373444  BEST MODEL found!
epoch 18/20, LR = [0.001]  loss : 0.0018900687044317071 validation score:
0.8516597510373444
epoch 19/20, LR = [0.001]  loss : 0.0024731972000815654 validation score:
0.853042876901798  BEST MODEL found!
epoch 20/20, LR = [0.001]  loss : 0.0016257847574624147 validation score:
0.8533886583679114  BEST MODEL found!
```





lr	epochs	step	loss	score (validation)
0.05	20	15	div!	div!
0.01	20	15	0.001	0.853
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

```
test score: 0.85309367438645
```

3.1 Freezing layer

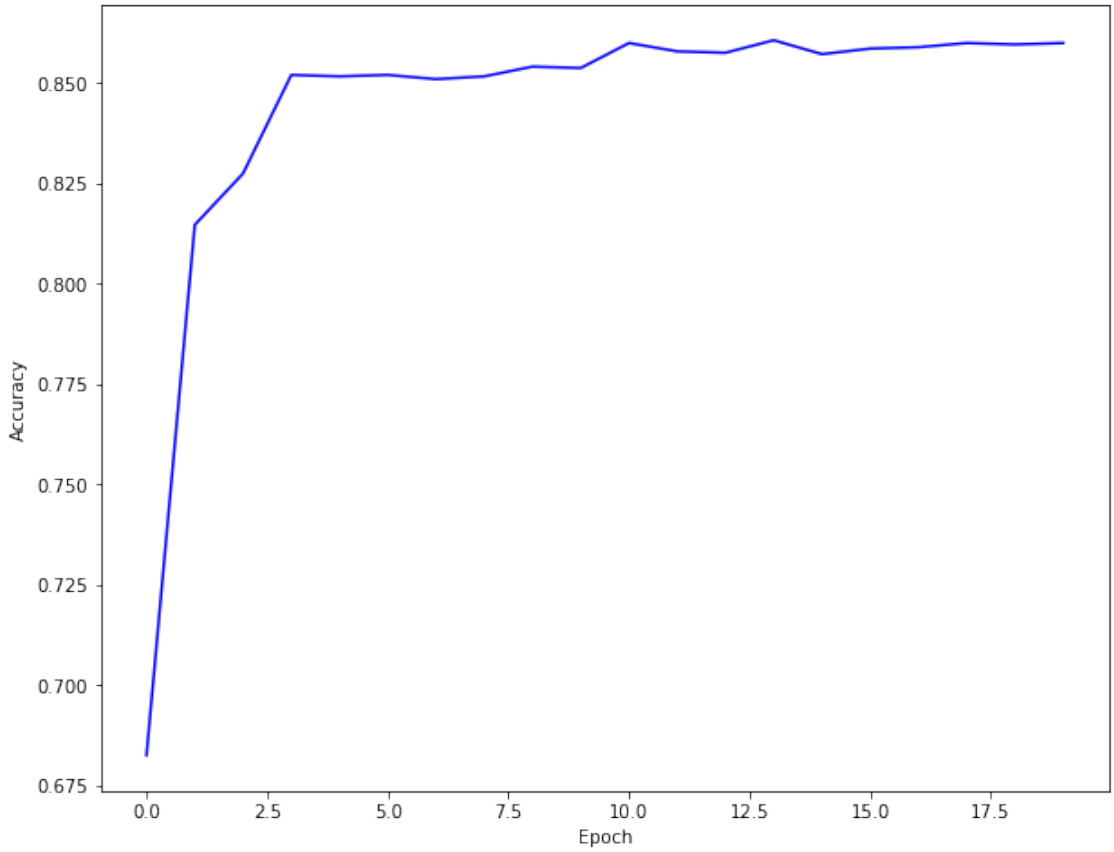
- LR= 0.01
- 20 epochs
- step size = 15

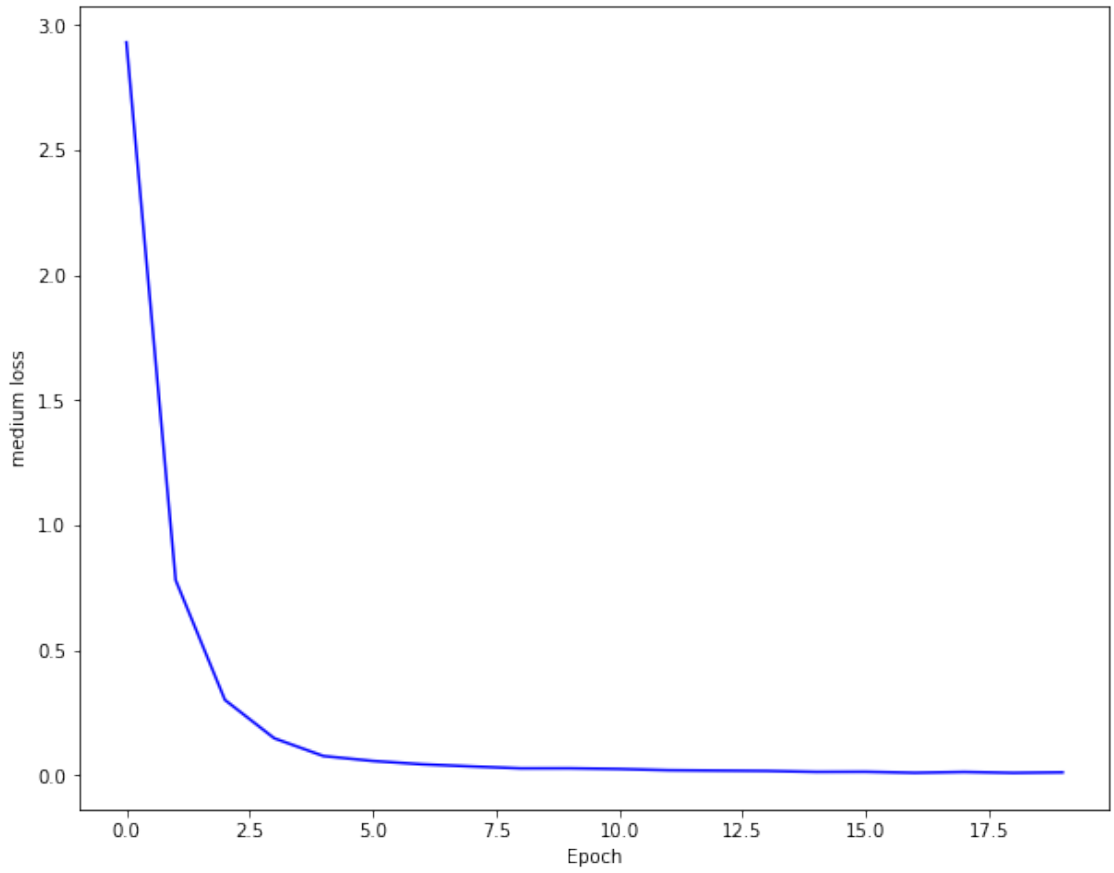
in this section I try the hyper-parameters found previously: - training only the fully connected layers - training only the convolutional layers

training and test only the fully connected layers

```
epoch 1/20, LR = [0.01]  loss : 2.9304029074582187 validation score:
0.6825726141078838  BEST MODEL found!
epoch 2/20, LR = [0.01]  loss : 0.7810212861407887 validation score:
0.8146611341632088  BEST MODEL found!
epoch 3/20, LR = [0.01]  loss : 0.3022911792451685 validation score:
0.8274550484094052  BEST MODEL found!
epoch 4/20, LR = [0.01]  loss : 0.14918840337883343 validation score:
0.8520055325034578  BEST MODEL found!
epoch 5/20, LR = [0.01]  loss : 0.0780270675366575 validation score:
0.8516597510373444
epoch 6/20, LR = [0.01]  loss : 0.05839121781966903 validation score:
0.8520055325034578
epoch 7/20, LR = [0.01]  loss : 0.045082019472664055 validation score:
0.8509681881051175
epoch 8/20, LR = [0.01]  loss : 0.03657011366025968 validation score:
0.8516597510373444
epoch 9/20, LR = [0.01]  loss : 0.028887460177594967 validation score:
0.8540802213001383  BEST MODEL found!
epoch 10/20, LR = [0.01]  loss : 0.029052125120704823 validation score:
0.8537344398340249
epoch 11/20, LR = [0.01]  loss : 0.02634473703801632 validation score:
0.8599585062240664  BEST MODEL found!
epoch 12/20, LR = [0.01]  loss : 0.02134986154057763 validation score:
0.8578838174273858
epoch 13/20, LR = [0.01]  loss : 0.01982255703346296 validation score:
0.8575380359612724
epoch 14/20, LR = [0.01]  loss : 0.01852575960484418 validation score:
0.8606500691562933  BEST MODEL found!
epoch 15/20, LR = [0.01]  loss : 0.014661048623648558 validation score:
0.857192254495159
epoch 16/20, LR = [0.001]  loss : 0.015154805373061787 validation score:
0.8585753803596127
epoch 17/20, LR = [0.001]  loss : 0.01108616827563806 validation score:
0.8589211618257261
epoch 18/20, LR = [0.001]  loss : 0.014354618266224861 validation score:
0.8599585062240664
epoch 19/20, LR = [0.001]  loss : 0.010773626918142492 validation score:
0.859612724757953
```

```
epoch 20/20, LR = [0.001]  loss : 0.012936360456726768 validation score:
0.8599585062240664
```



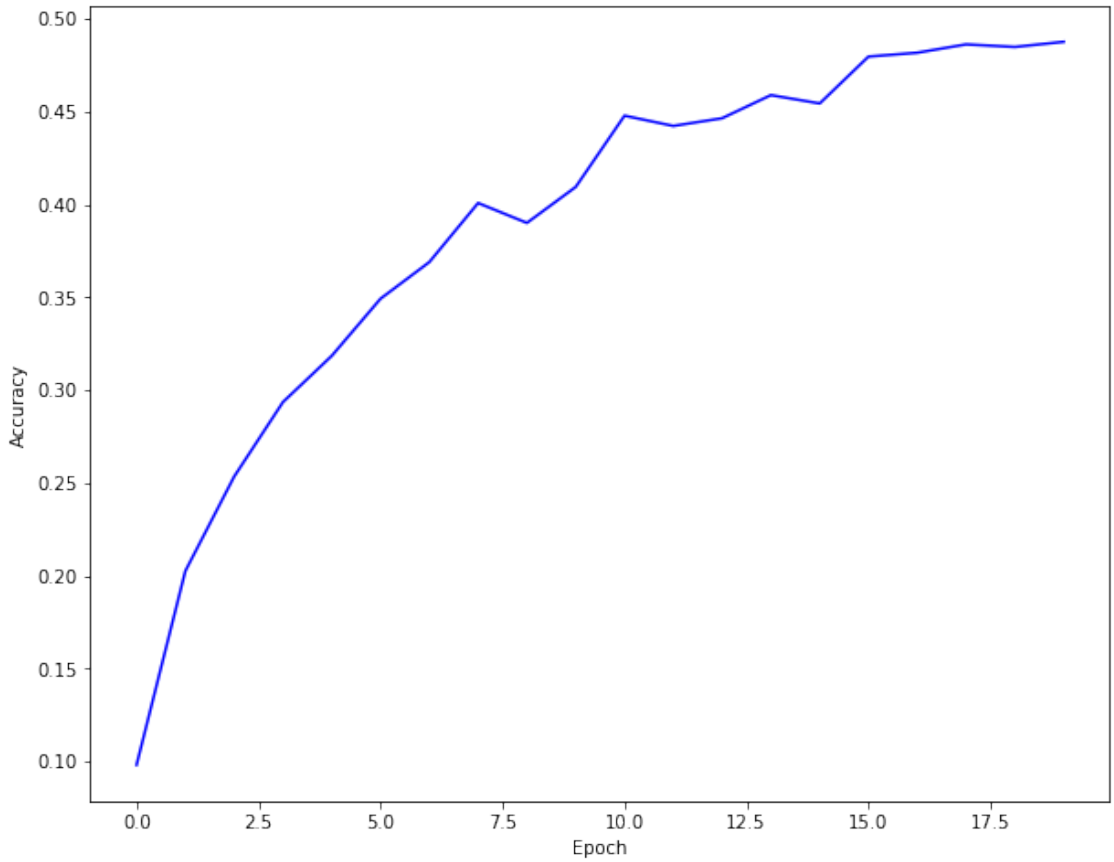


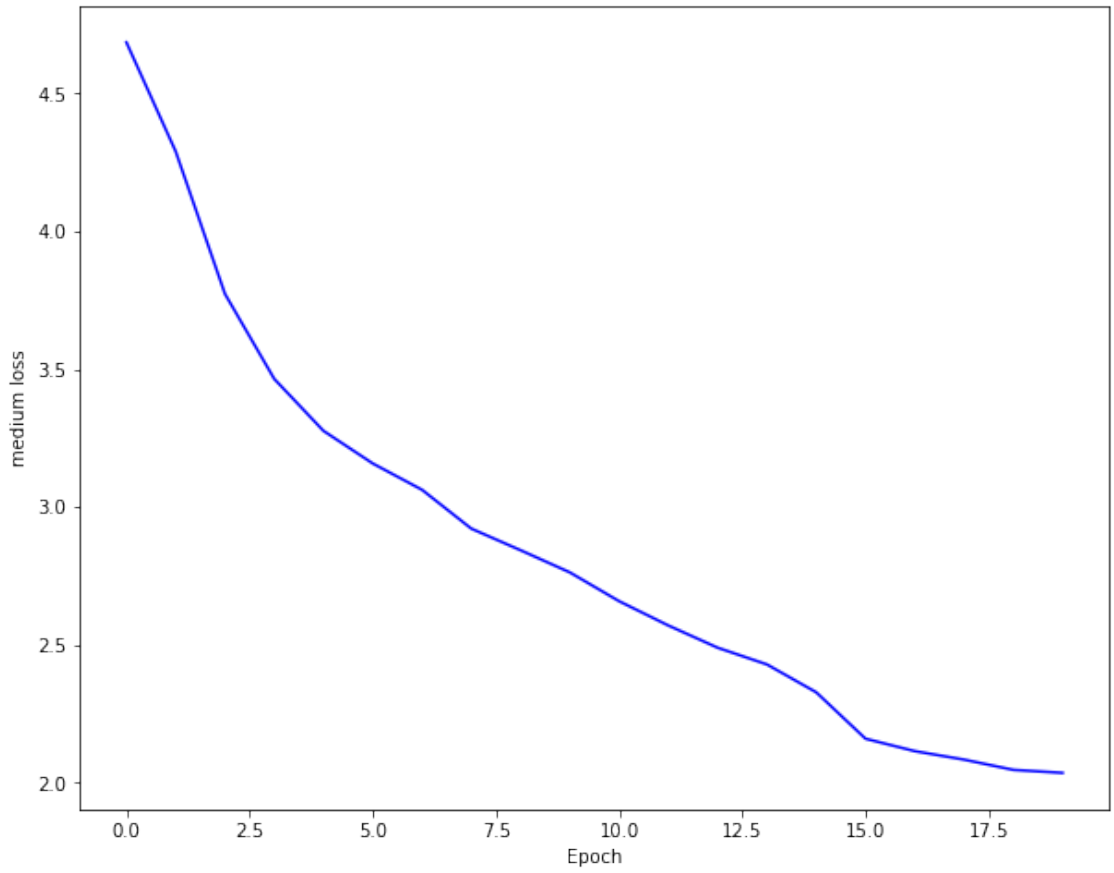

```
(full conn layers) test score: 0.8603525751814726
```

training and test only the conv layers

```
epoch 1/20, LR = [0.01]  loss : 4.68530425158414 validation score:
0.09820193637621023  BEST MODEL found!
epoch 2/20, LR = [0.01]  loss : 4.289907932281494 validation score:
0.20262793914246197  BEST MODEL found!
epoch 3/20, LR = [0.01]  loss : 3.773331642150879 validation score:
0.25345781466113415  BEST MODEL found!
epoch 4/20, LR = [0.01]  loss : 3.46509296243841 validation score:
0.29356846473029047  BEST MODEL found!
epoch 5/20, LR = [0.01]  loss : 3.2764456272125244 validation score:
0.3184647302904564  BEST MODEL found!
epoch 6/20, LR = [0.01]  loss : 3.1582499850880015 validation score:
0.34923928077455046  BEST MODEL found!
epoch 7/20, LR = [0.01]  loss : 3.062823317267678 validation score:
0.36894882434301524  BEST MODEL found!
epoch 8/20, LR = [0.01]  loss : 2.922023144635287 validation score:
```

```
0.40076071922544954  BEST MODEL found!
epoch 9/20, LR = [0.01]  loss : 2.8436369462446733 validation score:
0.3900414937759336
epoch 10/20, LR = [0.01]  loss : 2.7632934830405493 validation score:
0.4094052558782849  BEST MODEL found!
epoch 11/20, LR = [0.01]  loss : 2.6588441241871226 validation score:
0.44778699861687415  BEST MODEL found!
epoch 12/20, LR = [0.01]  loss : 2.570653720335527 validation score:
0.44225449515905946
epoch 13/20, LR = [0.01]  loss : 2.490176200866699 validation score:
0.4464038727524205
epoch 14/20, LR = [0.01]  loss : 2.429528691551902 validation score:
0.45885200553250344  BEST MODEL found!
epoch 15/20, LR = [0.01]  loss : 2.328540086746216 validation score:
0.45435684647302904
epoch 16/20, LR = [0.001]  loss : 2.160326675935225 validation score:
0.47959889349930845  BEST MODEL found!
epoch 17/20, LR = [0.001]  loss : 2.1151971600272437 validation score:
0.48167358229598894  BEST MODEL found!
epoch 18/20, LR = [0.001]  loss : 2.084452347321944 validation score:
0.48616874135546334  BEST MODEL found!
epoch 19/20, LR = [0.001]  loss : 2.0472929911179976 validation score:
0.4847856154910097
epoch 20/20, LR = [0.001]  loss : 2.0364019762385976 validation score:
0.487551867219917  BEST MODEL found!
```


0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch



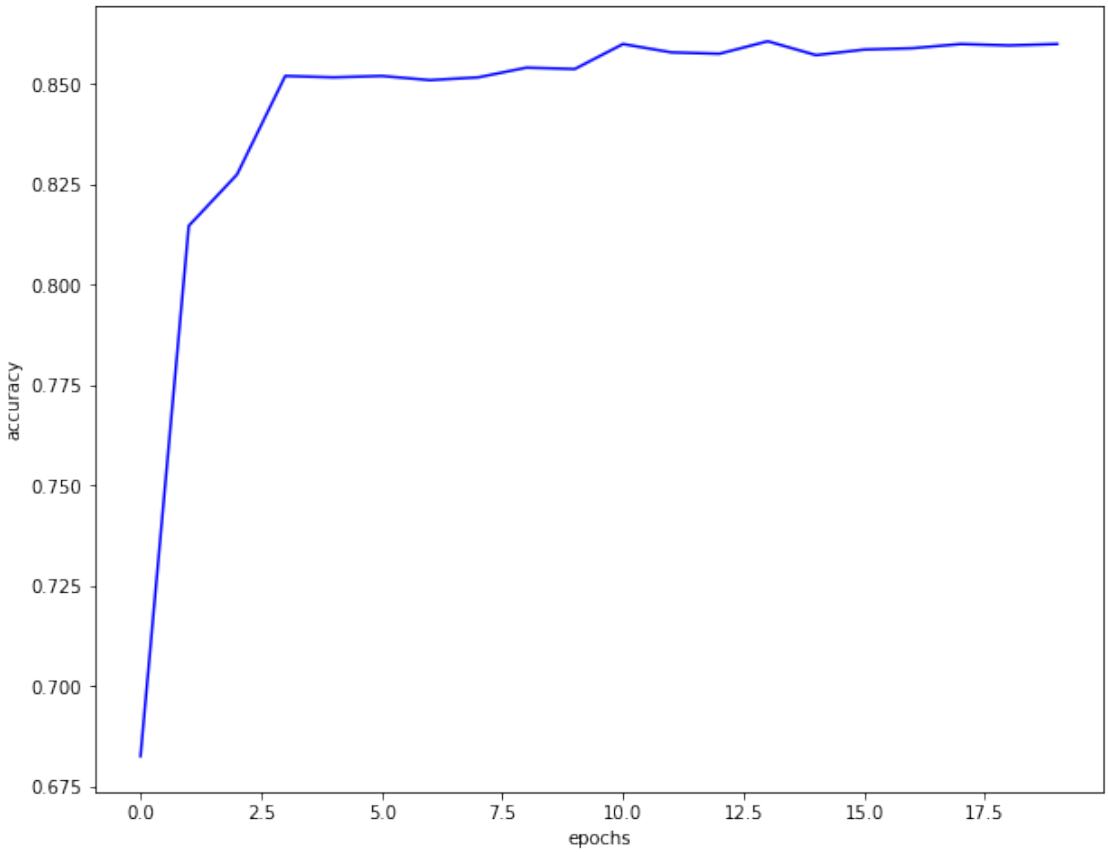

```
(conv layers) test score: 0.4873833390943657
```

Compare results

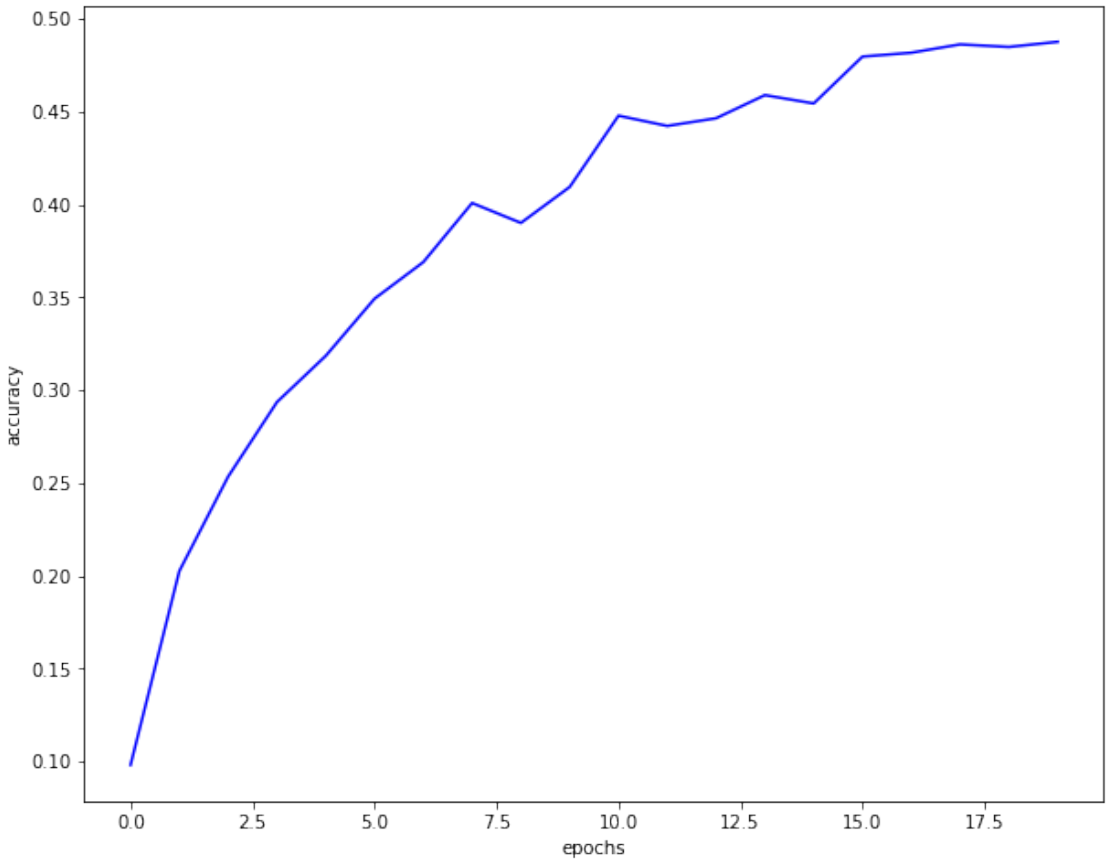
as it was easy to imagine,training only the FC, the model converge much much faster then training only the CONV,

in this case even the results (training only FC) are a little bit better than training the entire network, I didn't expect this behavior, probably because it manages to converge earlier than train the entire network.

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



ACCURACY OBTAINED WHEN TRAINING CONVOLUTIONAL LAYER



compare three different type of training:

type of training (layers)	loss	score(val)	score(test)
all	0.001	0.85	0.85
only Fully connected	0.01	0.86	0.86
only convolutional	2	0.48	0.48

4 Data augmentation

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

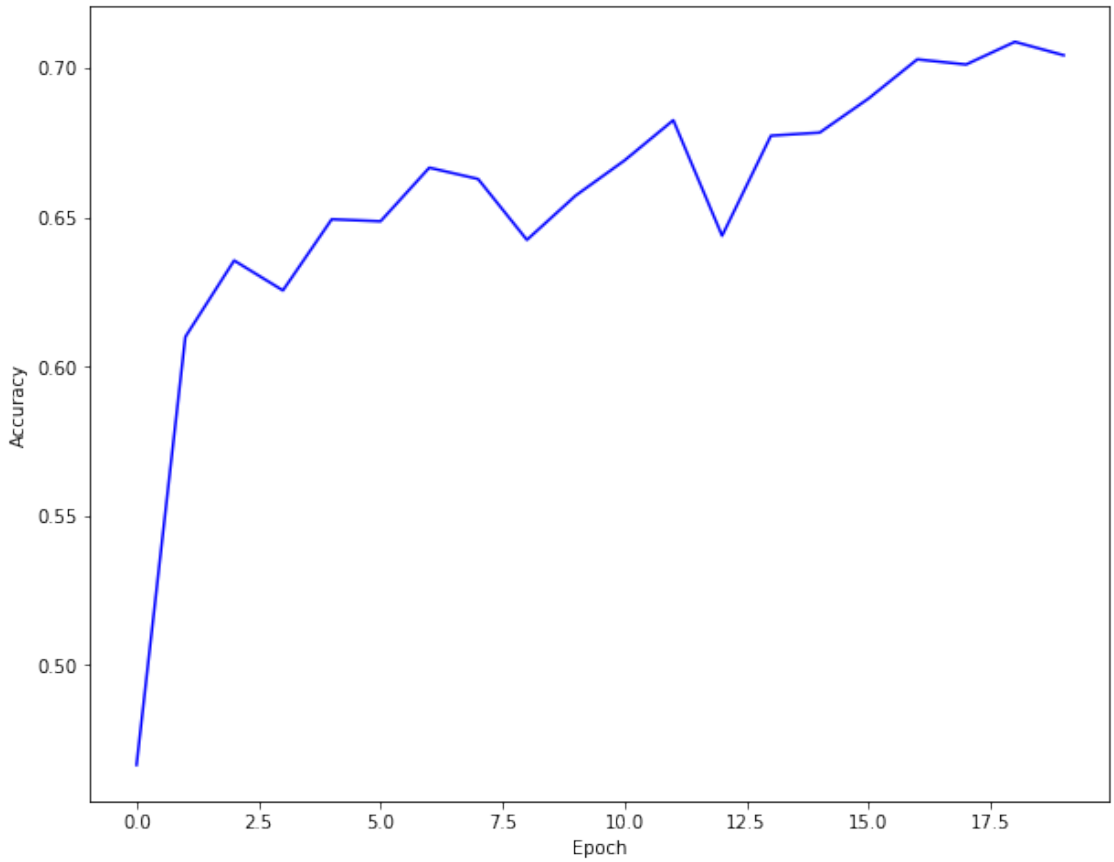
4.1 Prepare datasets with new transforms

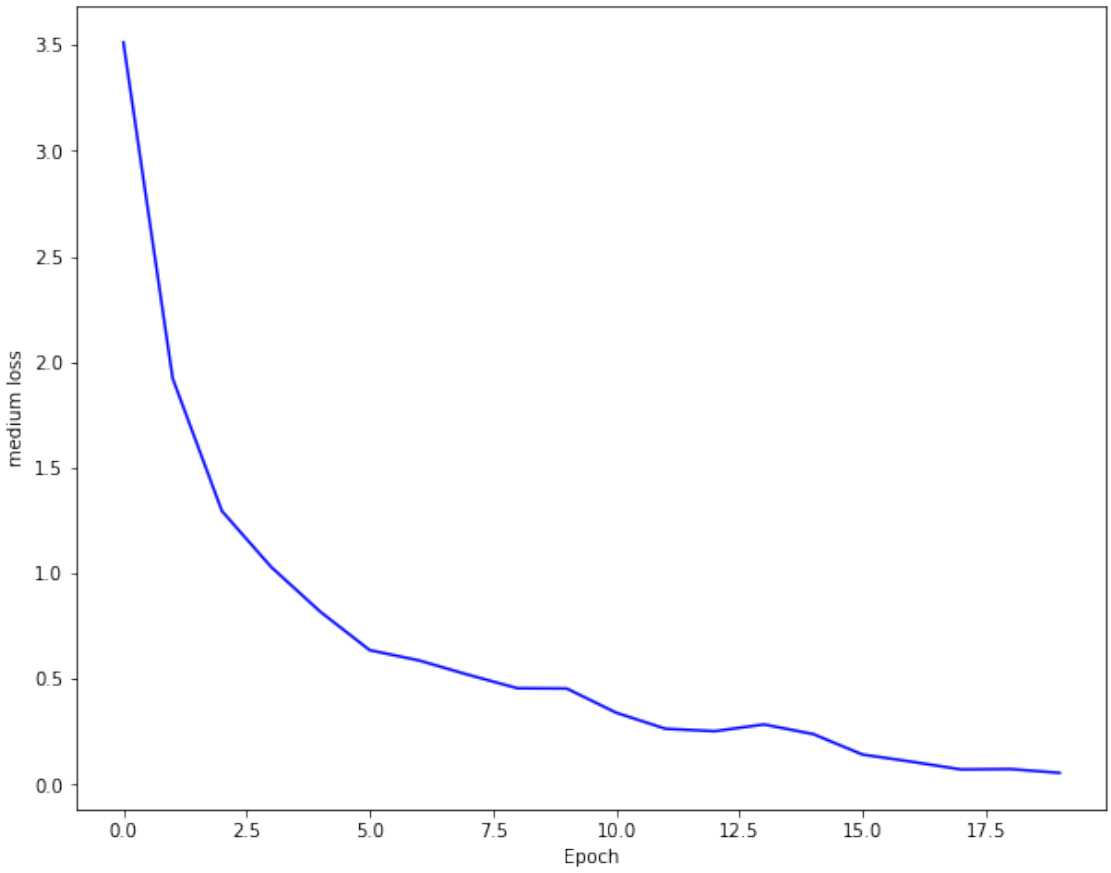
```
[stock transforms]
 Compose(
    Resize(size=256, interpolation=PIL.Image.BILINEAR)
    CenterCrop(size=(224, 224))
    ToTensor()
```

```
    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


epoch 1/20, LR = [0.01]  loss : 3.51328706741333 validation score:
0.4664591977869986  BEST MODEL found!
epoch 2/20, LR = [0.01]  loss : 1.9228394681757146 validation score:
0.6099585062240664  BEST MODEL found!
epoch 3/20, LR = [0.01]  loss : 1.2947211265563965 validation score:
0.6355463347164592  BEST MODEL found!
epoch 4/20, LR = [0.01]  loss : 1.029720203443007 validation score:
0.6255186721991701
epoch 5/20, LR = [0.01]  loss : 0.8175604180856184 validation score:
0.6493775933609959  BEST MODEL found!
epoch 6/20, LR = [0.01]  loss : 0.6362503062595021 validation score:
0.648686030428769
epoch 7/20, LR = [0.01]  loss : 0.5876784324645996 validation score:
```

```
0.6666666666666666  BEST MODEL found!
epoch 8/20, LR = [0.01]  loss : 0.519824204119769 validation score:
0.6628630705394191
epoch 9/20, LR = [0.01]  loss : 0.4559565945105119 validation score:
0.6424619640387276
epoch 10/20, LR = [0.01]  loss : 0.4546502015807412 validation score:
0.6573305670816044
epoch 11/20, LR = [0.01]  loss : 0.3403643559325825 validation score:
0.6690871369294605  BEST MODEL found!
epoch 12/20, LR = [0.01]  loss : 0.26439601995728235 validation score:
0.6825726141078838  BEST MODEL found!
epoch 13/20, LR = [0.01]  loss : 0.2528823668306524 validation score:
0.6438450899031812
epoch 14/20, LR = [0.01]  loss : 0.2846778224815022 validation score:
0.6773858921161826
epoch 15/20, LR = [0.01]  loss : 0.23916759274222635 validation score:
0.6784232365145229
epoch 16/20, LR = [0.001]  loss : 0.14226132766766983 validation score:
0.6898340248962656  BEST MODEL found!
epoch 17/20, LR = [0.001]  loss : 0.10827260396697304 validation score:
0.7029737206085753  BEST MODEL found!
epoch 18/20, LR = [0.001]  loss : 0.07181944393298843 validation score:
0.7012448132780082
epoch 19/20, LR = [0.001]  loss : 0.07363670827312903 validation score:
0.7088520055325035  BEST MODEL found!
epoch 20/20, LR = [0.001]  loss : 0.055547263473272324 validation score:
0.7043568464730291
```


0.70
0.65
0.60
0.55
0.50
Accuracy
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch




```
test score: 0.7034220532319392
```

	transforms	loss	score (test)
stock	center-crop	0.001	0.85
1	random crop	0.018	0.833
2	center-crop + random horizontal flip	0.008	0.84
3	center-crop + random rotation (180°)	0.05	0.7

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)

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

5 Beyond AlexNet

5.1 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*

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)
  )
)
```

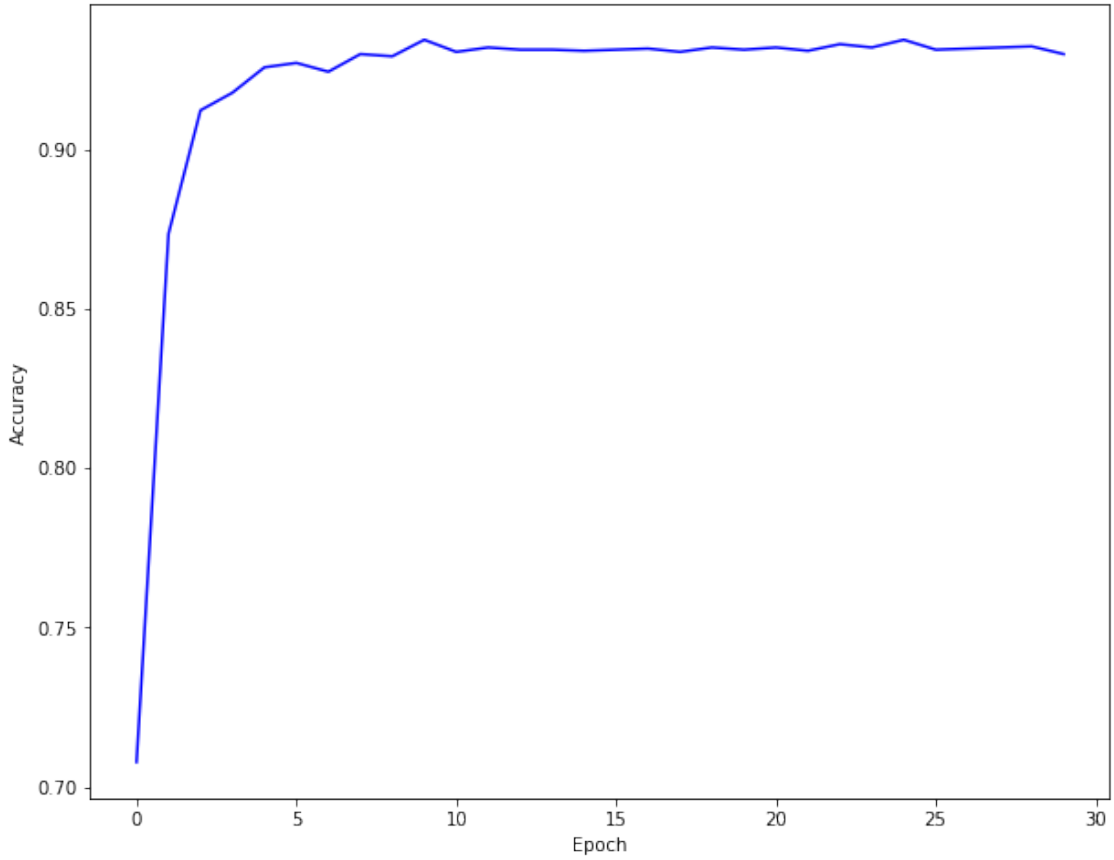
Test with resnet

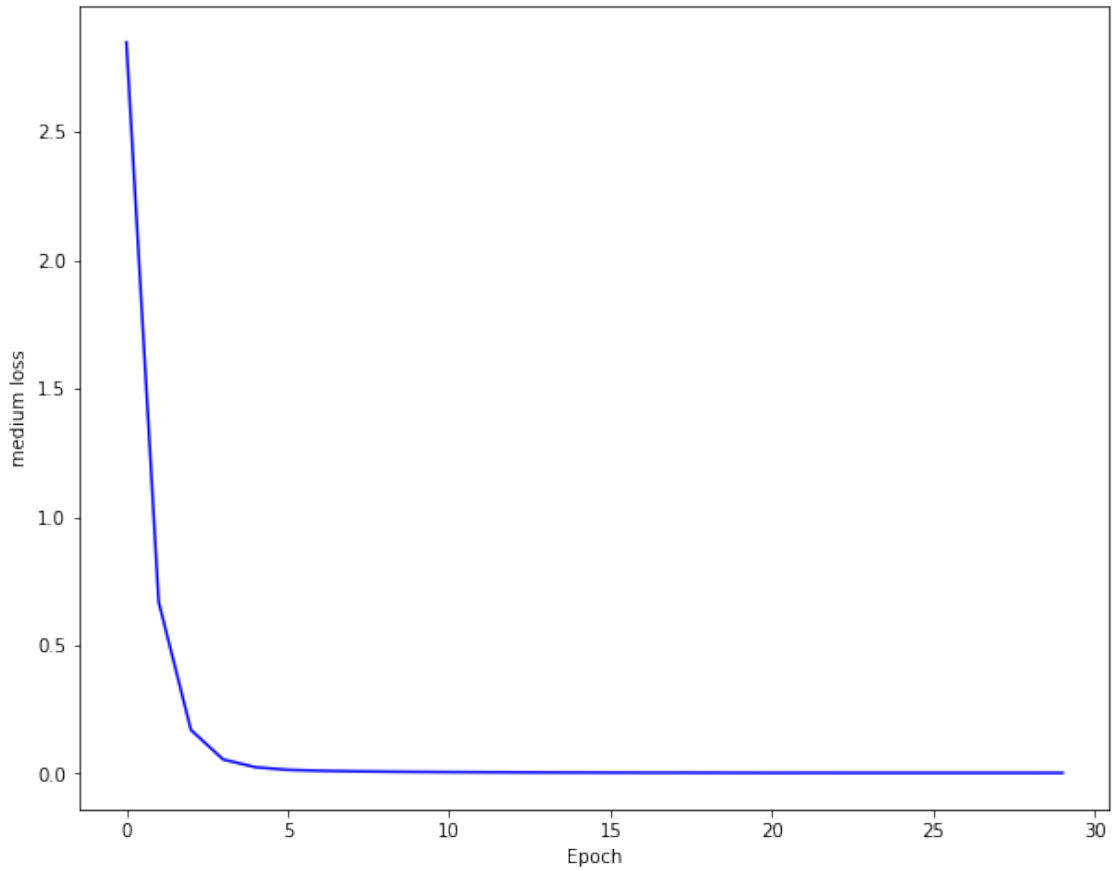
```
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 5784 Images and label
FOUND 101 CLASSES [accordion,airplanes..]
Loaded 2893 Images and label
Train Dataset: 5784
Test Dataset: 2893
Training split: 2892
Validation split: 2892


epoch 1/30, LR = [0.02]  loss : 2.8487802147865295 validation score:
0.7078146611341632  BEST MODEL found!
epoch 2/30, LR = [0.02]  loss : 0.6694851856340062 validation score:
0.8734439834024896  BEST MODEL found!
epoch 3/30, LR = [0.02]  loss : 0.17020560157569972 validation score:
0.9121715076071922  BEST MODEL found!
```

```
epoch 4/30, LR = [0.02]  loss : 0.05504009423946792 validation score:
0.9177040110650069  BEST MODEL found!
epoch 5/30, LR = [0.02]  loss : 0.025027294177562 validation score:
0.9256569847856155  BEST MODEL found!
epoch 6/30, LR = [0.02]  loss : 0.014805962289260193 validation score:
0.9270401106500692  BEST MODEL found!
epoch 7/30, LR = [0.02]  loss : 0.011133433192629705 validation score:
0.9242738589211619
epoch 8/30, LR = [0.02]  loss : 0.00903661233711649 validation score:
0.9298063623789765  BEST MODEL found!
epoch 9/30, LR = [0.02]  loss : 0.0077403674579479475 validation score:
0.9291147994467497
epoch 10/30, LR = [0.02]  loss : 0.006755170052532445 validation score:
0.9343015214384509  BEST MODEL found!
epoch 11/30, LR = [0.02]  loss : 0.005994507788934491 validation score:
0.9304979253112033
epoch 12/30, LR = [0.02]  loss : 0.005518850997429003 validation score:
0.931881051175657
epoch 13/30, LR = [0.02]  loss : 0.0049747168827293945 validation score:
0.9311894882434302
epoch 14/30, LR = [0.02]  loss : 0.004568059431304308 validation score:
0.9311894882434302
epoch 15/30, LR = [0.02]  loss : 0.004292370402254164 validation score:
0.9308437067773168
epoch 16/30, LR = [0.02]  loss : 0.003849303762597794 validation score:
0.9311894882434302
epoch 17/30, LR = [0.02]  loss : 0.003668967443941669 validation score:
0.9315352697095436
epoch 18/30, LR = [0.02]  loss : 0.0036597805817357516 validation score:
0.9304979253112033
epoch 19/30, LR = [0.02]  loss : 0.0033935179447077894 validation score:
0.931881051175657
epoch 20/30, LR = [0.02]  loss : 0.003229987225495279 validation score:
0.9311894882434302
epoch 21/30, LR = [0.002]  loss : 0.0029694586886431684 validation score:
0.931881051175657
epoch 22/30, LR = [0.002]  loss : 0.003075212621214715 validation score:
0.9308437067773168
epoch 23/30, LR = [0.002]  loss : 0.0030360738111829214 validation score:
0.9329183955739973
epoch 24/30, LR = [0.002]  loss : 0.002968169568868523 validation score:
0.931881051175657
epoch 25/30, LR = [0.002]  loss : 0.0029787851963192225 validation score:
0.9343015214384509
epoch 26/30, LR = [0.002]  loss : 0.0030267180091786113 validation score:
0.9311894882434302
epoch 27/30, LR = [0.002]  loss : 0.0029612925718538463 validation score:
0.9315352697095436
```

```
epoch 28/30, LR = [0.002]  loss : 0.002985047081231394 validation score:
0.931881051175657
epoch 29/30, LR = [0.002]  loss : 0.0029466069536283612 validation score:
0.9322268326417704
epoch 30/30, LR = [0.002]  loss : 0.0029693162597885184 validation score:
0.9298063623789765
```






```
test score: 0.9339785689595576
```