Bird Classification on Caltech-UCD dataset using CNNs

Problem Statement : Write code to implement simple **bird image classification** on **customized CNN**.The accuracy has to be **above 95%**. You can use either **Tensorflow** or **pytorch.** You have to use the below dataset.
http://www.vision.caltech.edu/visipedia/CUB-200.html.
Download it, split into train, val, test sets in the ratio (70, 10, 20).

Caltech-USD Birds 200 : The dataset was released by Caltech
It's an image dataset with photos of 200 bird species (mostly North American). For detailed information about the dataset, please see the technical report linked below.

- **Number of categories:** 200
- **Number of images:** 6,033
- **Annotations:** Bounding Box, Rough Segmentation, Attributes

Size of the images folder is 648 MB in .tgz format.
For this classification task we only need images and lists folder.

- Images

The images organized in subdirectories based on species.

- Lists

classes.txt : list of categories (species)
files.txt : list of all image files (including subdirectories)
train.txt : list of all images used for training
test.txt : list of all images used for testing
splits.mat : training/testing splits in MATLAB .mat format

Size after augmentation : 12066 Images (64X64)
Augmentation : Image Flip

- X_train : 16892
- X_test : 4826
- X_valid : 3016

Images were converted into numpy arrays and classes.txt folder was used to label the respective images.

CNN Architecture

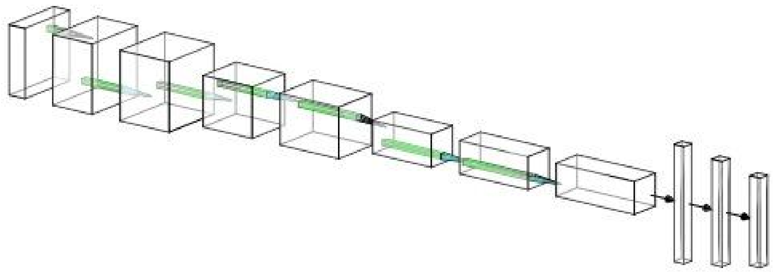
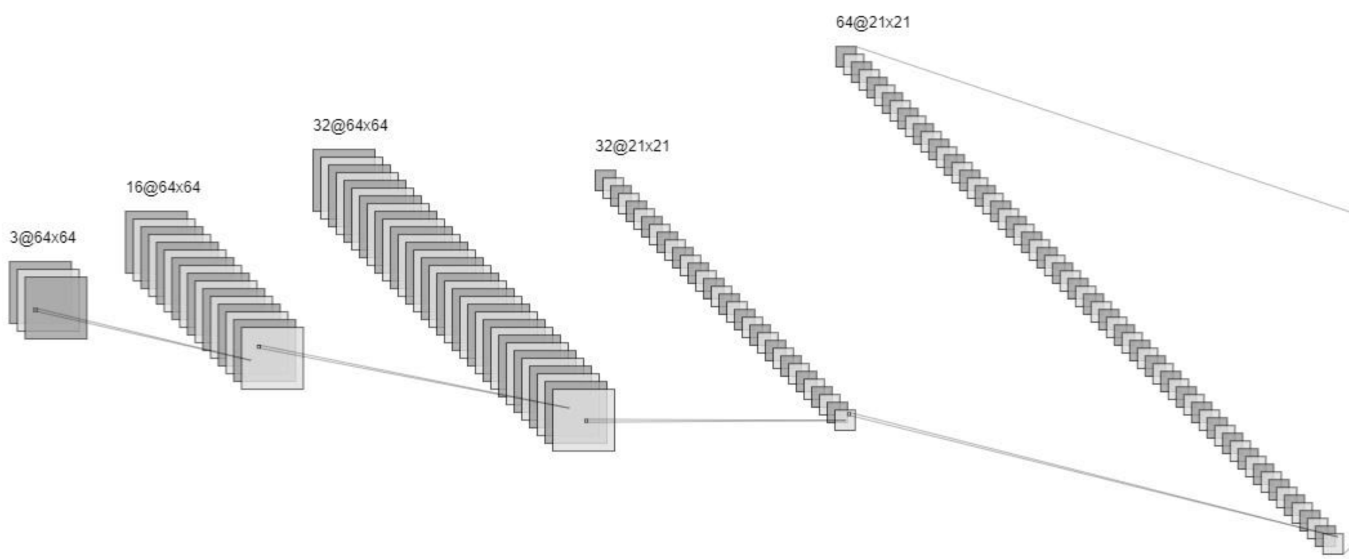
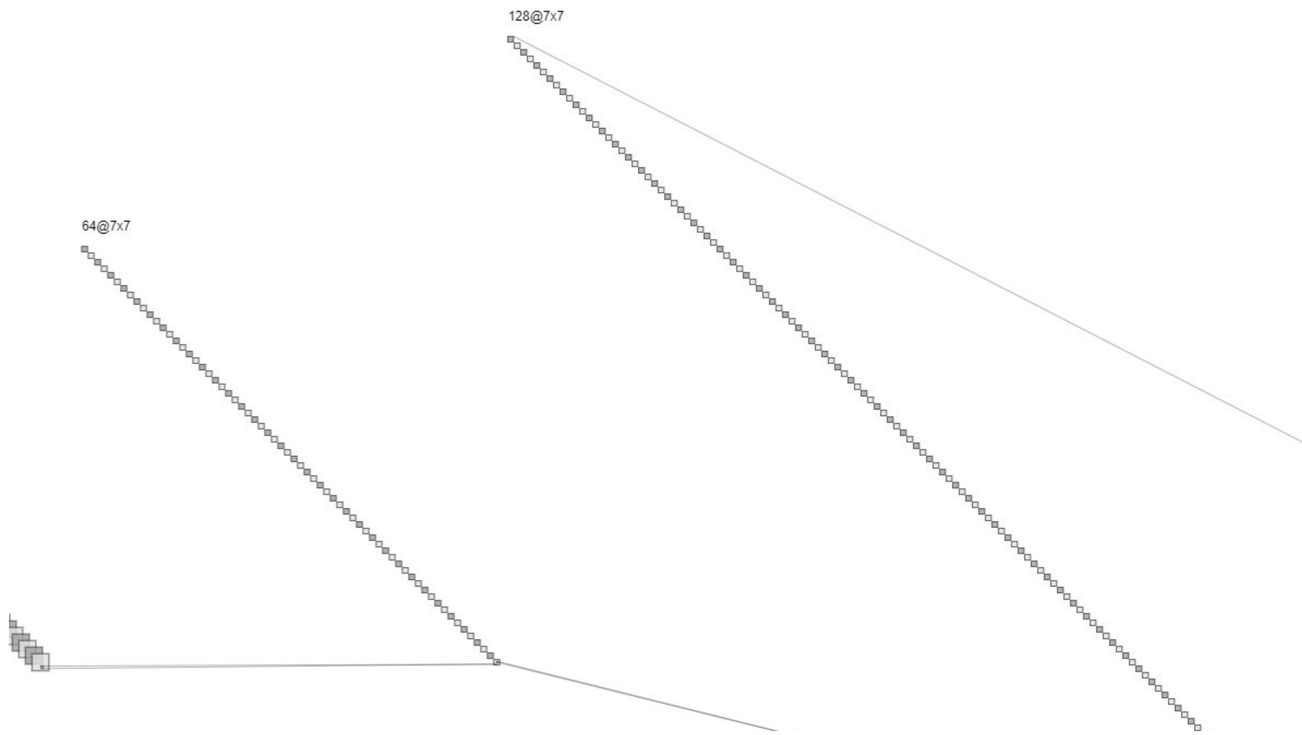


Figure : Alexnet Style Diagram.

Brief Explanation :
We start with 64 X 64 RGB (3 Channels) images and feed it to our 11 layer neural network.
The following diagrams will give an even better idea.



64@21X21 undergoes Max Pooling 3X3 to give 64@7X7 as shown in the next picture. Then followed by a 256@7X7 and another Max Pooling of 3X3 gives 256@2X2 which is flattened out into a 1024 vector.



Which undergoes matrix multiplication to give 512 vectors before undergoing another matrix multiplication to give 201 numbers representing a dummy label and 200 species.To know more about the dummy label go through the notebook.

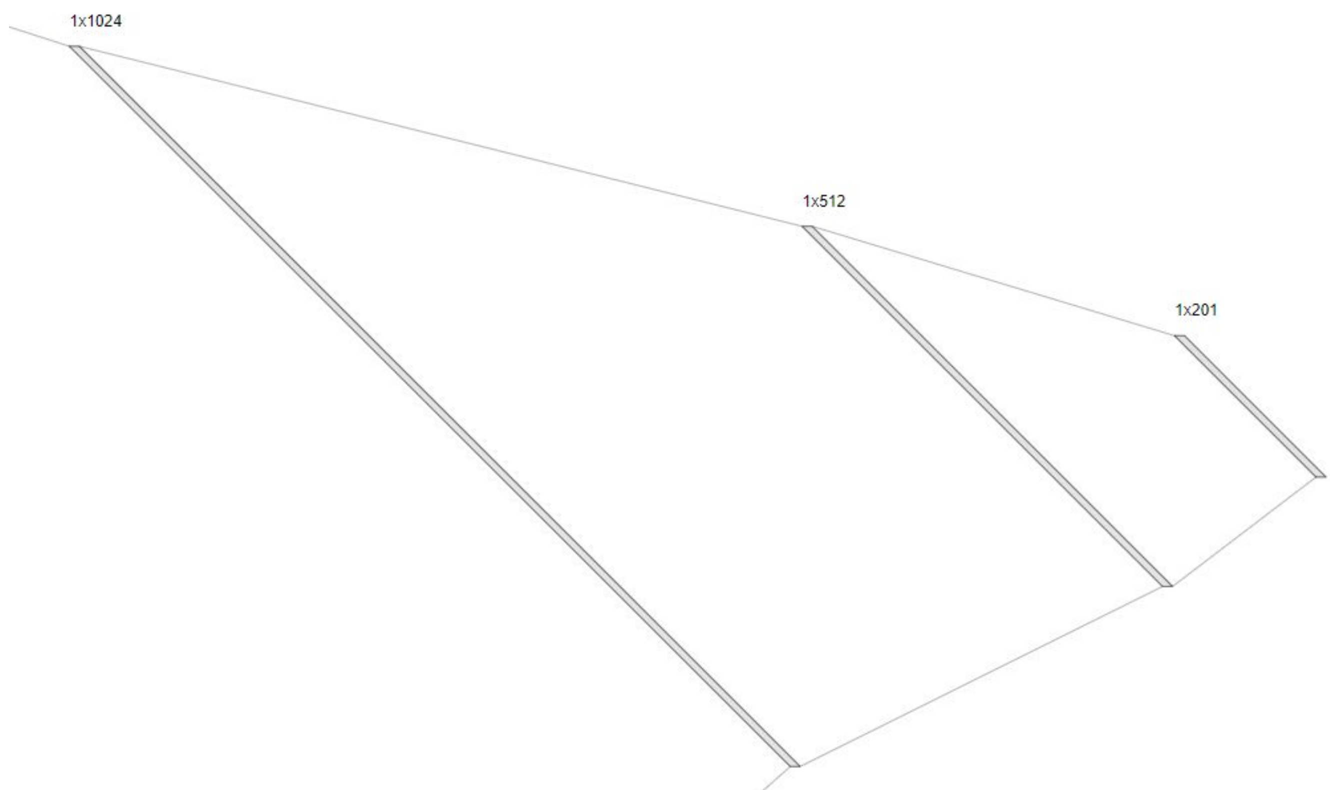


Figure : LeNet Style Diagram

Model Summary

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_97 (Conv2D)           (None, 64, 64, 16)        448
_________________________________________________________________
conv2d_98 (Conv2D)           (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 21, 21, 32)        0
_________________________________________________________________
conv2d_99 (Conv2D)           (None, 21, 21, 32)        9248
_________________________________________________________________
conv2d_100 (Conv2D)          (None, 21, 21, 64)        18496
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_101 (Conv2D)          (None, 7, 7, 128)         73856
_________________________________________________________________
conv2d_102 (Conv2D)          (None, 7, 7, 256)         295168
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 2, 2, 256)         0
_________________________________________________________________
batch_normalization_6 (Batch (None, 2, 2, 256)         1024
_________________________________________________________________
flatten_8 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_25 (Dense)             (None, 512)               524800
_________________________________________________________________
dense_26 (Dense)             (None, 201)               103113
=================================================================
Total params: 1,030,793
Trainable params: 1,030,281
Non-trainable params: 512
_________________________________________________________________
```

Parameters

Library Used: Keras with Tensorflow
Loss Function: Multi-Label CrossEntropy
Optimizer Used : Adam
Learning Rate : Learning rate=0.001, Beta1=0.9, Beta2=0.999
Dropout : 0.5 , that is half of total neurons but only during train time.
Total Number of epochs: 5X25 = 125

Results

Best Accuracy On Training Set : 95.61 %
Best Accuracy On Validation Set : 90.64 %
Best Accuracy On Test Set : 88.56 %

Other Architecture Performances

As the required accuracy was to be more than 95% I tried out a few more existing architectures like Resnet18, Resnet 50, Vgg16 unfortunately I could not get any desired accuracy in fact lower , maybe because the dataset we were dealing with was too small for big architectures.

For these architecture's implementations fast.ai library was used.Data Augmentation was done to avoid overfitting as much as possible using fast.ai Imagelist.

Results are as follows:

Vgg16 with batchnorm : Maximum accuracy of 39% accuracy kept on oscillating back and forth but did not move above 39% in any of the epochs.

37	0.923160	4.221381	0.282753	00:43
38	0.871240	3.810034	0.332504	00:43
39	0.887166	4.233475	0.283582	00:43
40	0.819960	3.717247	0.363184	00:43
41	0.800345	3.731205	0.341625	00:43
42	0.832456	4.130633	0.347430	00:43
43	0.802852	3.734576	0.361526	00:43
44	0.724150	4.140366	0.349917	00:43
45	0.774409	3.527825	0.368159	00:43
46	0.720636	4.611952	0.286899	00:43
47	0.687715	3.976405	0.351575	00:43
48	0.682090	4.057323	0.364013	00:43
49	0.728755	4.377048	0.317579	00:43
50	0.658133	4.329223	0.310116	00:43
51	0.614441	4.138223	0.352405	00:43
52	0.606956	3.973602	0.354063	00:43

68.42% [13/19

16	0.456887	3.726628	0.397181	00:44
17	0.475717	4.158159	0.344113	00:45
18	0.587481	3.702385	0.374793	00:46
19	0.565179	3.575959	0.373964	00:45
20	0.616977	3.777158	0.365672	00:45
21	0.652388	3.701328	0.361526	00:44
22	0.715481	3.758745	0.375622	00:44
23	0.712748	3.859480	0.349088	00:44
24	0.787954	3.813441	0.352405	00:44
25	0.824795	3.777223	0.353234	00:44
26	0.862587	3.675624	0.349917	00:44
27	0.850415	4.277962	0.286899	00:43
28	0.854989	3.567970	0.362355	00:44
29	0.883214	3.784101	0.349088	00:44
30	0.903143	4.082478	0.310116	00:43
31	0.928089	4.575954	0.289386	00:43
32	0.899320	3.648154	0.349917	00:43
33	0.920522	4.033380	0.327529	00:43
34	0.919322	4.303083	0.308458	00:43

Resnet 18 : It also did not perform well, got an mere accuracy of 20% and did not converge any further,image size was changed to as described in paper to fit the model.

```
1 learn = cnn_learner(data, models.resnet18, metrics=
```

```
1 learn.fit_one_cycle(15)
```

epoch	train_loss	valid_loss	accuracy	time
0	6.967589	5.656191	0.008292	00:46
1	6.223145	4.946662	0.063018	00:45
2	5.392600	4.461304	0.098673	00:44
3	4.729103	4.291251	0.115257	00:42
4	4.325078	4.123479	0.133499	00:42
5	4.000518	3.959369	0.155058	00:42
6	3.695888	3.868956	0.169983	00:43
7	3.505283	3.779780	0.178275	00:42
8	3.276067	3.744505	0.187396	00:42
9	3.072647	3.656097	0.198176	00:41
10	2.948643	3.632681	0.207297	00:42
11	2.781055	3.613338	0.192371	00:41
12	2.689140	3.605203	0.204809	00:42
13	2.609393	3.601390	0.203980	00:42
14	2.556580	3.589602	0.203980	00:42

Resnet 50 : Resnet 50: It got an accuracy of about 70% , it did not improve any further.. Since it's a much deeper network different types of augmentations were done like warping, random lightning Unfortunately I do not have the performance picture

Future Work

I think this performance could be increased by using 256X256 pixel inputs with a deeper model than our current architecture but the problem availability of RAM in Google Colab the kernel crashes when I tried to augment 256 X 256 pictures and convert them.I also tried without augmentation the RAM still is not sufficient and kernel crashes. So with a GPU Machine that has a better RAM capacity we can expect better accuracy by carefully redesigning a better architecture and applying good regularization techniques to avoid overfitting and get a better generalization.
I also tried with 128X128 it was not stable, that is it gave different results every time ,and best result it produced was sometimes equal to the result produced by 64X64
Maybe Resnet 18 can also give a good result if we use more data by better augmentation and pre-processing as given in paper.The reason we have got such a less accuracy may be because of incorrect normalization.
Whether Pretrained Network will improve accuracy is debatable because images in this dataset overlap with images in ImageNet. We need to take extreme caution when using networks pre trained as the test set of CUB may overlap with the training set of the original network.