# Transfer Learning

An exercise on implementation of Transfer Learning using pretrained Convolutional layers of ResNet50 and CIFAR10 dataset. The dense layer was retrained in the traditional method. Using these large expertly pretrained models gives a huge boost to performance. Specifically, here ResNet50 is 50 layers deep, trained on 1000 different objects, is very apt for the task of image classification in CIFAR10. This can be repurposed for our use through transfer learning where we import pretrained Convolutional layers of ResNet50 and retrain the dense layers on CIFAR10 dataset.

The training time 1h 33s is quite high despite using GPU. Total trainable parameters were 40,583,370 and non-trainable parameters are 315,648. The model has 3 upsampling layers followed by imported resnet50 convolutional layers. Then the tensor is flattened and passed through alternate layers of batch normalized and dense layers for 3 times.

The test accuracy is 92.9% much higher than a 6-layer deep CNN specifically trained on CIFAR10 dataset of 67.6% or any classic ML Models.

Github Link to Notebook:

* CIFAR10 6-layer CNN model <https://github.com/mandalnilabja/soc2022/blob/main/dryrun/CNN%20CIFAR10.ipynb>
* ResNet50 and CIFAR10 model: <https://github.com/mandalnilabja/soc2022/blob/main/Week9Assignment.ipynb>

Reference: <https://medium.com/@andrew.dabydeen/transfer-learning-using-resnet50-and-cifar-10-6242ed4b4245>

# Improvement of CNNpred Implementation

Improvement by hyperparameter tuning and using Bagging ensemble on CNNpred implementation was done.

Following hyperparameters were varied to find the optimal model-

1. Loss Function (mae, binary\_focal\_crossentropy, binary\_crossentropy, hinge)
2. Optimizer (SGD, Adam, Adagrad, Adamax),
3. Epochs (10, 20, 25, 30),
4. Batch Size (128, 64, 32, 16),
5. Dropout Rate (0.05, 0.1, 0.15, 0.2),

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Index | Epochs | Optimizer | Batch Size | Dropout Rate | Loss Function | Accuracy | MAE | F1 |
| 2dCNNpred |  | 20 | Adam | 128 | 0.1 | MAE | 0.48 | 0.52 | 0.55 |
| 2dCNNpred | m1 | 20 | Adam | 64 | 0.1 | MAE | 0.51 | 0.49 | 0.60 |
| 2dCNNpred |  | 20 | Adam | 64 | 0.1 | MAE | 0.54 | 0.46 | 0.66 |
| 2dCNNpred |  | 20 | Adam | 32 | 0.1 | MAE | 0.50 | 0.49 | 0.60 |
| 2dCNNpred |  | 20 | Adam | 16 | 0.1 | MAE | 0.55 | 0.45 | 0.65 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

Thus clearly Optimizer worked best. Also,

Usually Adam (Adaptive Moment Estimation) Optimizer is faster and outperforms others.

So, Adam was chosen. Experimentation on Batch Size was done and optimal batch size of 64 was chosen.

Dropout rate was kept default 0.1.

Github Link to Notebook:

<https://github.com/mandalnilabja/soc2022/blob/main/Week9AssignmentB.ipynb>