Final Project Experimental Report

- **Dataset:** The Dataset selected for the Final Project is SUN397 which belongs to Task-2. The dataset contains different classes belonging to environmental scenes, places and the objects within. The database contains 108,753 images of 397 categories, used in the Scene Understanding (SUN) benchmark. The number of images varies across categories, but there are at least 100 images per category.

- **Training/Testing Partition:** For the results of this experiment we have chosen at random one of the available ten training and testing partitions provided by the authors of the dataset. The randomly chosen partition is the fifth partition which has two files namely: **Training_05.txt** and **Testing_05.txt**. Each of these files contains 19850(50x397) image paths where each class has fifty images belonging to it.

- **Image Augmentation:** Each image undergoes different augmentation methods. The methods used are listed as follows:
  1. **Rotation:** Degree range for random rotations(-10, +10).
  2. **Symmetric Warp:** Change the angle at which image is viewed.(-0.2, +0.2)
  3. **Zoom range:** Range for random zoom(1, 1.1).
  4. **Left Right flip:** Randomly flip inputs left or right.
  5. **Brightness:** Changes brightness of Images(0.4, 0.6).
  6. **Contrast:** Changes contrast of Images(0.8, 1.25).
  7. **Crop and Pad:** Randomly crops and pads the remaining part of image.

- **Feature Fusion Technique:** We have used adaptive2D pooling method for feature fusion. This method combines max pooled and average pooled features and then passes it to fully connected layers.

- **Classifier Model:** The classification method used in this experiment is D-CNN. The model used in the experiment is ResNet50. We have trained this network structure by using the Place365 weights as the initial weights. The default input size for this model is (3, 224,224).

- **Architecture:** The architecture used in this experiment contains the Resnet50 structure where the dense layers are replaced with an adaptive2D pooling layer followed by fully connected layers. The model architecture for the top layers is as shown in figure-1:

```
      BatchNorm2d-160           [-1, 2048, 7, 7]           4,096
             ReLU-161           [-1, 2048, 7, 7]               0
       Bottleneck-162           [-1, 2048, 7, 7]               0
            Conv2d-163            [-1, 512, 7, 7]       1,048,576
      BatchNorm2d-164            [-1, 512, 7, 7]           1,024
             ReLU-165            [-1, 512, 7, 7]               0
            Conv2d-166            [-1, 512, 7, 7]       2,359,296
      BatchNorm2d-167            [-1, 512, 7, 7]           1,024
             ReLU-168            [-1, 512, 7, 7]               0
            Conv2d-169           [-1, 2048, 7, 7]       1,048,576
      BatchNorm2d-170           [-1, 2048, 7, 7]           4,096
             ReLU-171           [-1, 2048, 7, 7]               0
       Bottleneck-172           [-1, 2048, 7, 7]               0
AdaptiveMaxPool2d-173            [-1, 2048, 1, 1]               0
AdaptiveAvgPool2d-174            [-1, 2048, 1, 1]               0
AdaptiveConcatPool2d-175               [-1, 4096, 1, 1]               0
          Flatten-176                    [-1, 4096]               0
      BatchNorm1d-177                    [-1, 4096]           8,192
          Dropout-178                    [-1, 4096]               0
           Linear-179                     [-1, 512]       2,097,664
             ReLU-180                     [-1, 512]               0
      BatchNorm1d-181                     [-1, 512]           1,024
          Dropout-182                     [-1, 512]               0
           Linear-183                     [-1, 397]         203,661
================================================================
Total params: 25,818,573
Trainable params: 25,818,573
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.57
Forward/backward pass size (MB): 286.71
Params size (MB): 98.49
Estimated Total Size (MB): 385.77
----------------------------------------------------------------
```

Figure-1

- **Training Process:** We have executed the training process in two steps.

  Step-1: The training occurs for only the fully connected layers for the initial 5 epochs. This step finetunes our classifier for the given dataset.

  Step-2: The convolution layers are unfreezed and all the layers are trained for 2 epochs. This step finetunes our model for the given dataset.

  Parameters for model:

  1. **Optimizer:** Adam
  2. **Loss_Function:** Categorical_Crossentropy

  Discriminative Learning Rate: Differential Learning Rates (LR) are used for faster, more efficient transfer learning. The general idea is to use a lower Learning Rate for the earlier layers, and gradually increase it in the latter layers.

  One-cycle Policy: Varying the learning rate between reasonable bounds can actually increase the accuracy of the model in fewer steps. Here, we start with a small learning rate (like 1e-4,

1e-3) and increase the lr after each mini-batch till the loss starts exploding. Once loss starts exploding stop the range test run. Then plot the learning rate vs loss and choose the learning rate at-least one order lower than the learning rate where the loss is minimum.

- **Results:** The results obtained after both the training steps for three runs is as shown in igure-2. The average testing accuracy obtained after three runs is **69.313**%.

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	3.045159	1.589482	0.564685	0.435315	08:26
1	2.031103	1.512841	0.590529	0.409471	06:58
2	1.580198	1.282151	0.641209	0.358791	07:02
3	1.176071	1.126957	0.685441	0.314559	07:01
4	0.915919	1.084229	0.693199	0.306801	06:58
0	0.864151	1.077442	0.695113	0.304887	06:57
1	0.859532	1.073482	0.694559	0.305441	06:56
2	0.825874	1.072657	0.695365	0.304635	06:56

(a) Run-1

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	3.053918	1.602363	0.572796	0.427204	07:32
1	2.055854	1.556024	0.587355	0.412645	07:34
2	1.587337	1.293446	0.640353	0.359647	07:34
3	1.196448	1.118837	0.680453	0.319547	07:39
4	0.943023	1.081918	0.692040	0.307960	07:35
0	0.865293	1.080561	0.691839	0.308161	07:41
1	0.841334	1.074188	0.693401	0.306599	07:40
2	0.835416	1.072479	0.693098	0.306902	07:39

(b) Run-2

epoch	train_loss	valid_loss	accuracy	error_rate	time
0	2.999628	1.613311	0.563678	0.436322	08:56
1	2.015862	1.564137	0.582368	0.417632	07:33
2	1.614134	1.292570	0.642065	0.357935	07:31
3	1.196015	1.127366	0.679043	0.320957	07:33
4	0.927991	1.086176	0.689773	0.310227	07:35
0	0.862587	1.079984	0.690781	0.309219	07:40
1	0.843513	1.077093	0.692091	0.307909	07:42
2	0.839007	1.076651	0.691033	0.308967	07:39

(c) Run-3

Figure-2

- **Training time:** Average training time for each run is **57 min 15 s.**
- **Log Files:** The log files for all the runs are available in the project directory.
- **Model Weights:** The trained model weights are available in the models folder.

NOTE: To replicate the experiment follow the instructions mentioned in the readme.txt file.