CS6220
Big Data System and Analytics
Assignment 4

Sathvik Karatattu Padmanabha

October 2024

Student Session: cs6220-A
GT ID: xx4032361

Topic: Programming

Problem 2:

The Power of ensemble learning

Contents

1 Introduction and Readme **3**
1.1 Problem . . . 3
1.2 Execution Environment . . . 3

2 Notebook Details **4**

3 Dataset Details **5**

4 Base Model Details **10**

5 Experiments **11**
5.1 Baseline Training and Testing . . . 11
5.2 Focal Model Negative Examples Testing . . . 14
5.3 Ensemble Testing . . . 18

6 Optional Part: Using a second Dataset - MNIST **20**
6.1 Experiment Details . . . 20
6.2 Experiments . . . 20

7 Conclusion **22**

8 Workflow diagram **23**

9 Screenshots of Execution **24**

10 References **26**

11 Appendix **27**
11.1 Confusion Matrices for MNIST Experiments . . . 27
11.2 MNIST Data EDA . . . 29

1 Introduction and Readme

1.1 Problem

I have chosen the ensemble learning problem. I have chosen image classification. I have chosen the CIFAR-10 dataset to perform experiments. There is an optional task to use another dataset, for this, I have used MNIST dataset.

I chose 4 pre-trained models [5][11] to be the base models. I chose the following models

- denseNet
- vgg19
- resNet
- mobileNetV2

Then, I am performing all possible ensemble combinations to determine the best team. There is a section in the problem statement that asks to choose a focal model. I am choosing vgg19 as the focal model.

1.2 Execution Environment

I have chosen pytorch [5] as the ML library. This is because the datasets and the models are readily available in pytorch. I have used the PACE environment with NVIDIA A100-PCIE-40GB to execute the experiments and used the jupyter interactive app in the PACE dashboard for convenience. I used anaconda for library management.

The .ipynb book submitted in the zip file contains the complete code. The majority of the code is mostly inspired from a single kaggle notebook [1]. However, I have added changes as required by the problem statement and other references are mostly from pytorch open conversation posts [2][3][4][8] and other notebooks [6][7].

The github repo containing the code is provided below. However, you need to request access from me as it is private. I have anyways provided the .ipynb book along with the submission.

https://github.com/SathvikKP/Ensemble_Learning_CIFAR10.git

To setup the environment, I have provided the anaconda .yml file. But here is a list of commands that I used to setup the environment.

```
module load anaconda3
conda create --prefix ./bda_4 python=3.9
conda activate ./bda_4
conda install matplotlib numpy
conda install -c conda-forge ipykernel
python -m ipykernel install --user --name myenv --display-name "bda_4"
conda install pytorch-gpu torchvision torchaudio pytorch-cuda=11.8 -c pytorch -
c nvidia
conda install scikit-learn
conda install seaborn
conda env export > environment.yml
```

2 Notebook Details

I have neatly added the markup cells describing the actions taking place in the code cells. I have also modularized the functions in original code to make it easy for the reader to analyze.

- Step 1: Import Libraries
- Step 2: Check for GPU and declare global variables
- Step 3: Define the validation, train and test functions
- Step 4: Write Auxillary Functions for Data preprocessing, model loading and Exploratory Data Analysis
- Step 5: Classes from original code for DenseNet, VGG19, ResNet and MobileNetV2.
- Step 6: Function to perform tasks as required in section 5: Compare other models against focal model wrongly predicted examples
- Step 7: Functions to Display the results of Comparison
- Step 8: Functions for testing out the ensemble permutations
- Step 9: Data Loading and Performing EDA
- Step 10: Create the base models and train them
- Step 11: Peform Testing of base models
- Step 12: Compare performance of other models by keeping VGG19 as focal model
- Step 13: Run the ensemble predictions over all permutations
- Step 14: OPTIONAL : DATASET 2 - MNIST

The optional part for using another dataset (MNIST) yielded me very high accuracies. Hence, I performed data preprocessing such as mislabelling, noise addition etc

3 Dataset Details

Pytorch has the CIFAR dataset readily available. torchvision.datasets.CIFAR10 is the the inbuilt function to load the data. Note that the pytorch provides only train and test split. We need to split the train dataset into train and valid dataset. The ratio of Train:Valid:Test that I have chosen is 4:1:1. I have performed exploratory data analysis and have provided the details below. Figures 1, 2, 3, 4, 5 and 6 show the exploratory data analysis part. Note that the images shown can be flipped as during data loading using pytorch, I applied some transformations, random flip and rotate before Exploratory Data Analysis. I could not reverse the flips while printing. Table 1 shows the CIFAR-10 standard dataset size. However, we are resizing the dataset for our models, hence, the statistics change which is reported in Table 2.

Metric	**Train Set (CIFAR-10)**	**Test Set (CIFAR-10)**
Total Number of Images	50,000	10,000
Number of Images per Class	5,000 per class (for 10 classes)	1,000 per class (for 10 classes)
Total Image Size (in pixels)	153,600,000	30,720,000
Average Image Size (in pixels)	3,072.00	3,072.00
Total Size of Images (in bytes)	Approx. 153.00 MB	Approx. 30.72 MB
Image Resolution	3 x 32 x 32	3 x 32 x 32

Table 1: Standard CIFAR-10 Dataset Metrics

Metric	**Train Set**	**Validation Set**	**Test Set**
Total Number of Images	40000	10000	10000
Number of Images per Class	0: 3939, 1: 4016, 2: 4034, 3: 4008, 4: 4005, 5: 3975, 6: 4029, 7: 4022, 8: 3999, 9: 3973	0: 1061, 1: 984, 2: 966, 3: 992, 4: 995, 5: 1025, 6: 971, 7: 978, 8: 1001, 9: 1027	0: 1000, 1: 1000, 2: 1000, 3: 1000, 4: 1000, 5: 1000, 6: 1000, 7: 1000, 8: 1000, 9: 1000
Total Image Size (in pixels)	1966080000	491520000	491520000
Average Image Size (in pixels)	49152.00	49152.00	49152.00
Total Size of Images (in bytes)	7500.00 MB	1875.00 MB	1875.00 MB
Image Resolution	3 x 128 x 128	3 x 128 x 128	3 x 128 x 128

Table 2: Summary of Train, Validation, and Test Set Metrics

Figure 1: Train Dataset Distribution

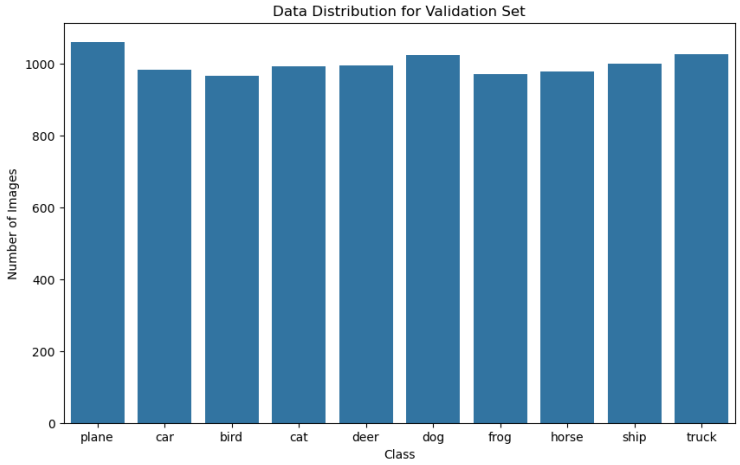


Figure 2: Validation Dataset Distribution

Figure 3: Test Dataset Distribution

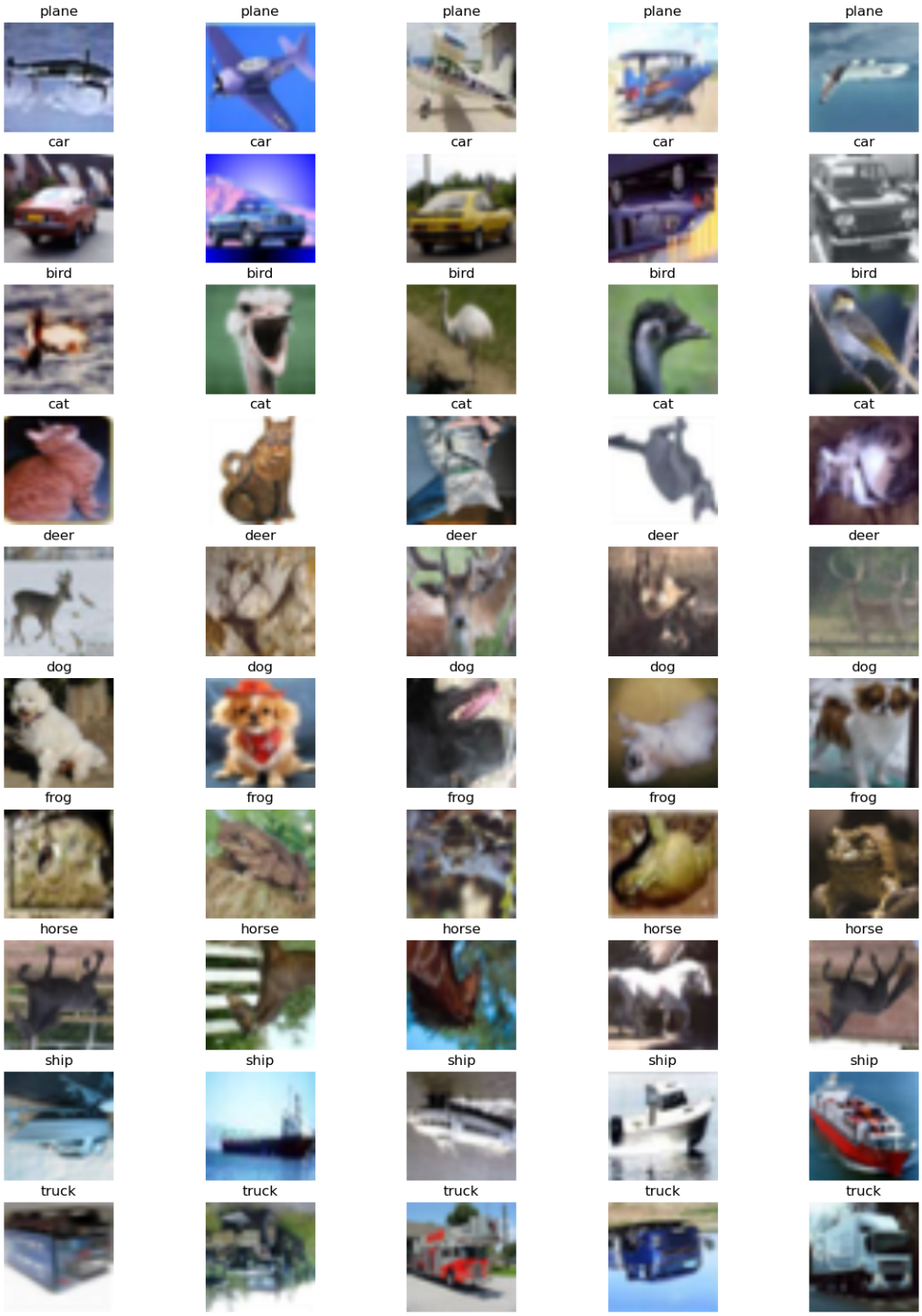


Figure 4: Sample images for each class in train dataset

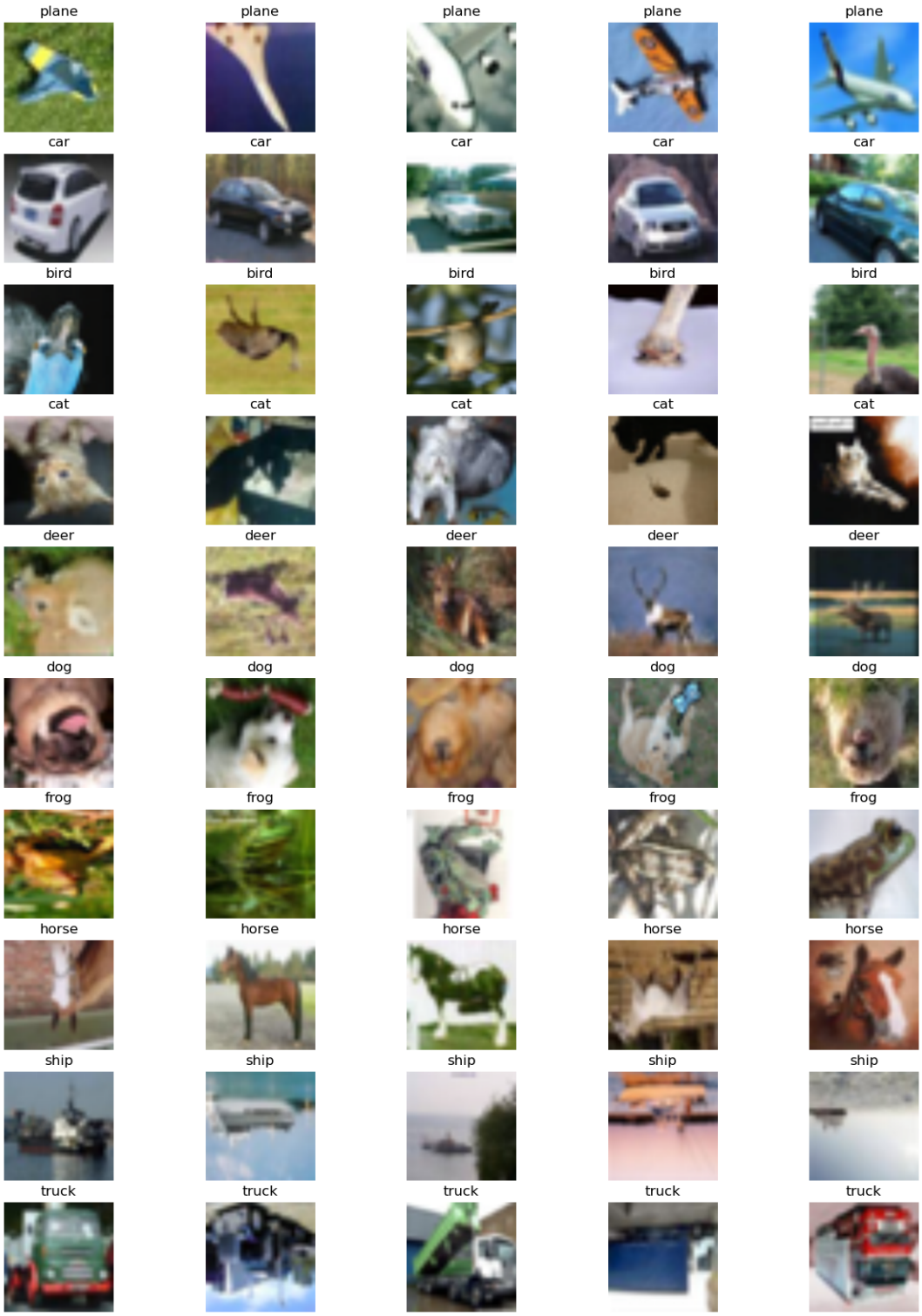


Figure 5: Sample images for each class in validation dataset

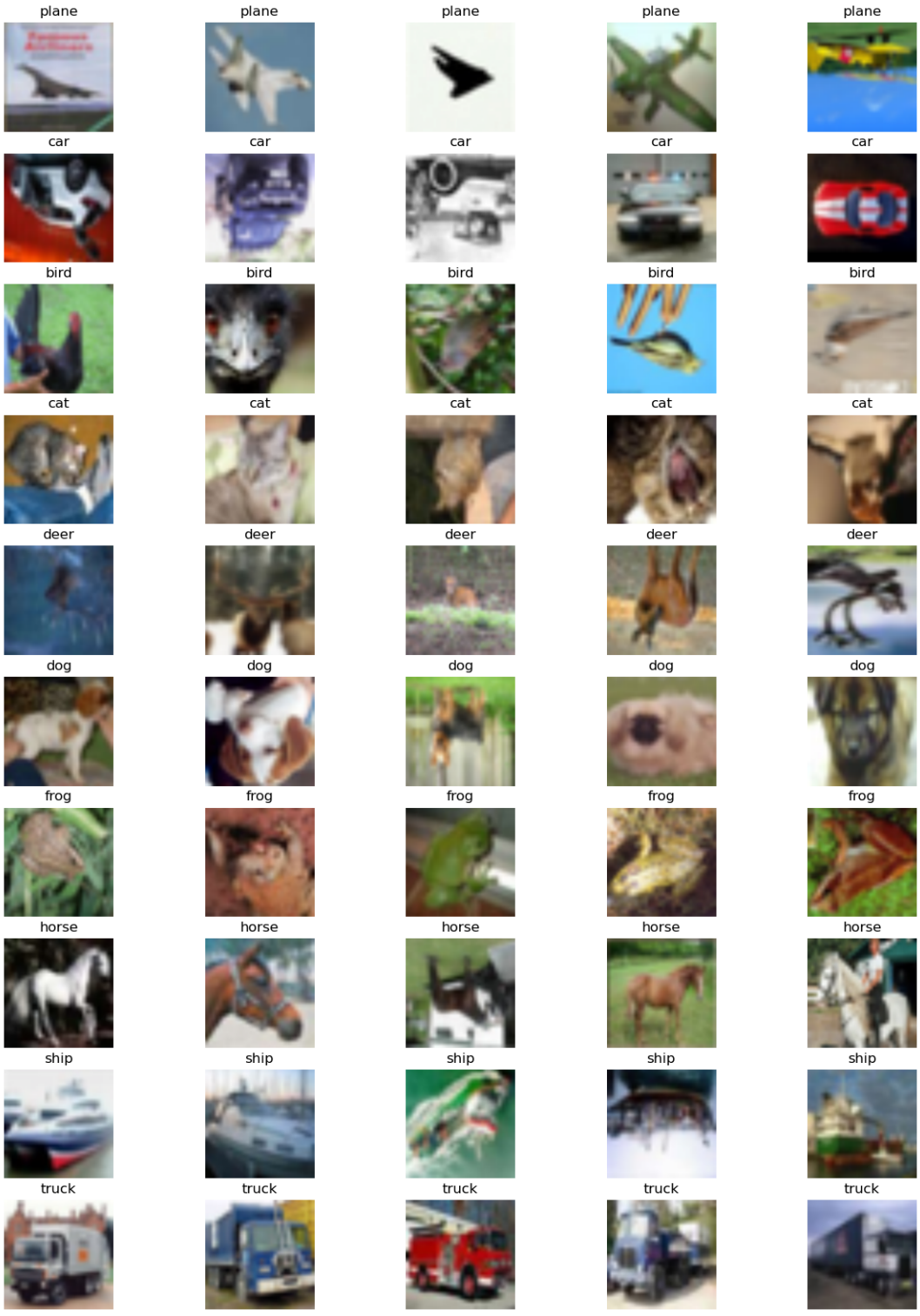


Figure 6: Sample images for each class in test dataset:

4 Base Model Details

The complete details of the base models that I have chosen is provided in Tables 3 and 4. The reason for choosing these models is because these are standard baseline image prediction networks. These are easily available in pytorch and configuring them is easy. The code that I referred also used these models (DenseNet, VGG19 and ResNet) and I chose another popular model MobileNetV2 as the problem requires me to have an additional model. The reasons for choosing the parameters in Table 3 is because these are standard values. I tried varying the optimizer to SGD, but did not achieve good performance. I kept the batch size 128 and number of epochs 5 after slight experimentation to get accuracy of about 80% to 90% so that I can test the ensemble part effectively.

Model	**Batch Size**	**Learning Rate**	**Optimizer**	**Epochs**
DenseNet	128	0.001	Adam	5
VGG19	128	0.001	Adam	5
ResNet	128	0.001	Adam	5
MobileNetV2	128	0.001	Adam	5

Table 3: Summary of Model Configurations for Training

Property	**DenseNet**	**VGG19**	**ResNet**	**MobileNetV2**
Architecture	Dense Blocks	Sequential Layers	Residual Blocks	Depthwise Separable Convolutions
Number of Layers	201 layers	19 layers	152 layers	Lightweight (53 layers)
Pretrained	Yes	Yes	Yes	Yes
Input Channels	3 (CIFAR-10) or 1 (MNIST)	3 (CIFAR-10) or 1 (MNIST)	3 (CIFAR-10) or 1 (MNIST)	3 (CIFAR-10) or 1 (MNIST)
MNIST adjustments	Conv2d, Pool0 as Identity	Conv2d adjustments, removed some layers	Conv2d adjustment, removed maxpool	Conv2d adjustment for MNIST
Final Layer	Linear(1920, 10) (adjusted)	Linear(4096, 10) (adjusted)	Linear(2048, 10) (adjusted)	Linear(1280, 10) (adjusted)
Model Size	Large (71 MB)	Very Large (533 MB)	Large (223 MB)	Lightweight (8.8 MB)
Training Speed	Slower due to depth (396 s)	Moderate (257 s)	Slower due to depth (306 s)	Fast due to lightweight nature (155 s)
Main Feature	Dense connectivity between layers	Deep sequential layers	Residual connections	Efficient model with depthwise separable convolutions

Table 4: Comparison of DenseNet, VGG19, ResNet, and MobileNetV2 Architectures in pytorch. Note: The MNIST adjustments is optional. Note: The model size (in MB) and training time (in seconds) is reported based on my experiments.

5 Experiments

5.1 Baseline Training and Testing

In the .ipynb notebook, I have neatly defined functions to perform data loading, exploratory data analysis, training, calculating validation loss, testing. I have also neatly defined classes for each of the chosen base models which makes it easy to implement and define. Once, we do that, we can simply call the auxillary function train_and_save_model() which will perform training. Then, we can call the test_model() function which will perform testing process and return the average_loss, accuracy, correct_examples, correct_labels, wrong_examples, true_labels_for_wrong, focal_preds (predicted values for wrong examples). Table 5 shows the statistics after executing training and testing for the 4 baseline models.

Metric	**DenseNet**	**VGG19**	**ResNet**	**MobileNetV2**
Training Loss	0.234115	0.462180	0.324185	0.279558
Training Accuracy (%)	91.8350	85.1300	88.7325	90.4700
Average Epoch Training Time (s)	66.66	44.98	54.01	25.41
Total Training Time (s)	396.01	257.45	306.81	155.93
Validation Loss	0.358907	0.596272	0.444142	0.306337
Validation Accuracy (%)	88.0100	80.4800	84.6300	89.8000
Test Loss	0.401965	0.620401	0.445471	0.333988
Test Accuracy (%)	87.3500	79.9500	84.6500	88.8100

Table 5: Comparison of Training, Validation, and Test Metrics for DenseNet, VGG19, ResNet, and MobileNetV2.

An interesting thing to note is how MobileNetV2 performed the best on test data despite having the lowest training time and lowest model size. This is due to its feature of depth wise separable convolutions. Clearly for this dataset, MobileNetV2 is the best model. During our ensemble experiments, we should ideally find this model in the best ensemble team.

The training loss and accuracy curves for these 4 models are shown in figures 7,8,9 and 10. Note that these are not smooth as we have chosen very few epochs. Also, they start from high accuracies as these are already pre-trained models. We are fine-tuning them for CIFAR datasets.

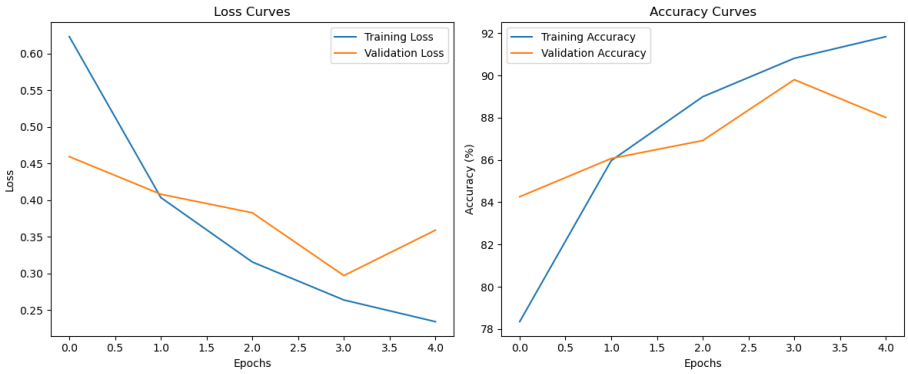


Figure 7: Accuracy and Loss curves for DenseNet

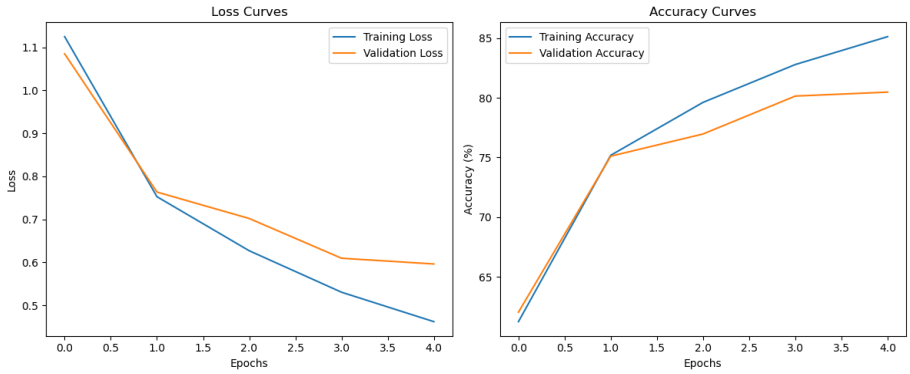


Figure 8: Accuracy and Loss curves for Vgg19

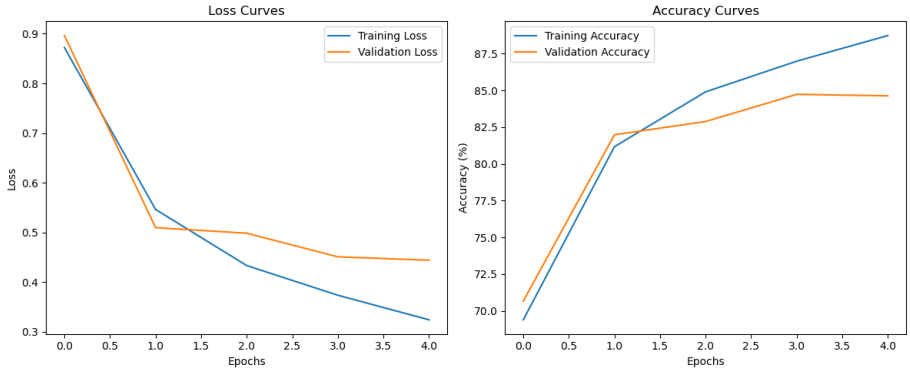


Figure 9: Accuracy and Loss curves for ResNet

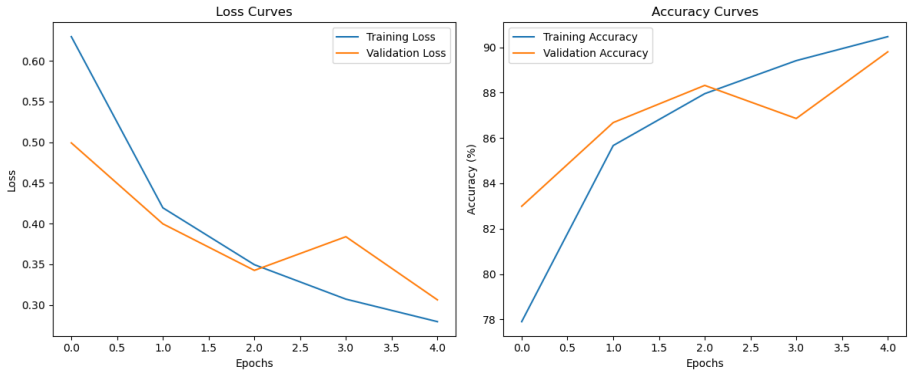
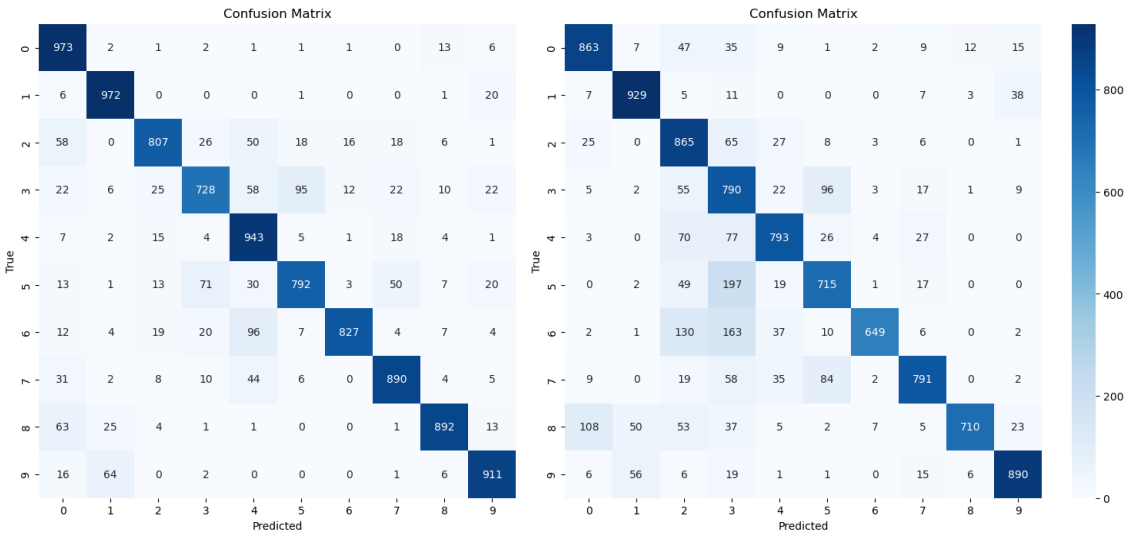
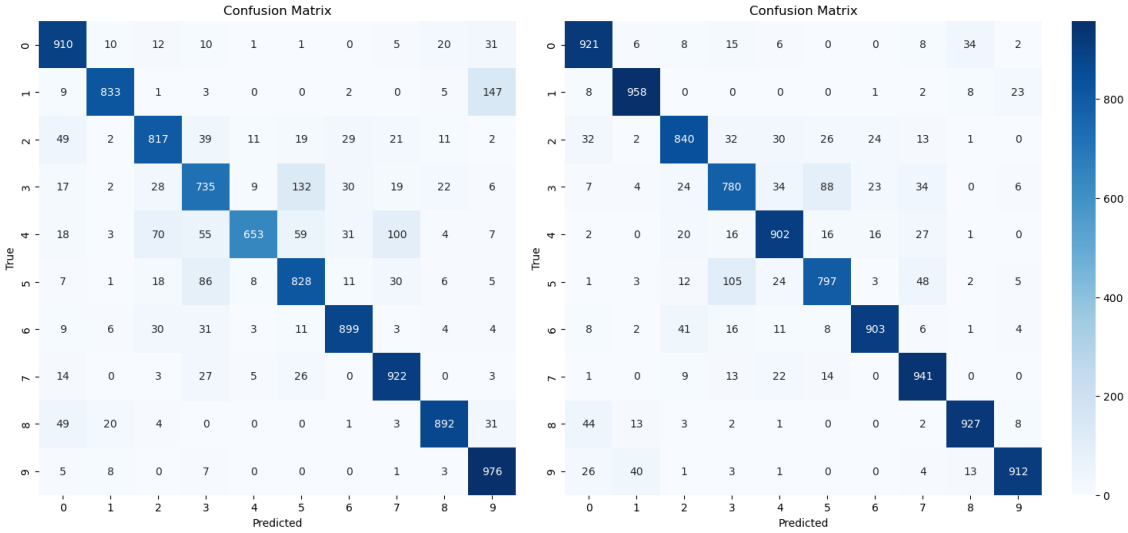


Figure 10: Accuracy and Loss curves for MobileNetV2

Figures 11, 12, 13 and 14 show the confusion matrices for these base models for the test dataset. It seems that prediction of class 5 (dog) is difficult especially for VGG19. Class 5 is predicted as class 3 (cat). It is because these images are similar. Another interesting fact is how Resnet frequently misclassified Class 1 (car) as class 9 (Truck). This is also because the images for these classes are similar.



Figures 11 and 12: Confusion matrix for test set for DenseNet and Vgg19



Figures 13 and 14: Confusion matrix for test set for ResNet and MobileNetV2

5.2 Focal Model Negative Examples Testing

VGG19 performed the worst among the 4 models (Table 5). Let us keep this model as the focal model for our experiments. The test_model function will return the wrongly predicted examples, the focal model predictions, and the true labels for focal prediction. We will utilize this data (wrongly predicted examples) to test the other 3 models. We can run the pytorch model(input) to get the output. Then, we can check if each of the model predicted correctly or not. We can then store the statistics and perform analysis. Table 6 provides the statistics for this experiment. We are storing the data where some examples are predicted correctly by only 1 model, some examples predicted correctly by 2 models, some predicted correctly by all 3, and some which were predicted wrong by all models.

Scenario	**Count**
Model 1 Correct, 2 and 3 Wrong	138
Model 2 Correct, 1 and 3 Wrong	117
Model 3 Correct, 1 and 2 Wrong	111
Model 1 and 2 Correct, 3 Wrong	137
Model 1 and 3 Correct, 2 Wrong	226
Model 2 and 3 Correct, 1 Wrong	165
All Models Correct	813
All Models Wrong	298
Total wrong examples in focal model (Vgg19)	**2005**

Table 6: Different Scenarios for Models 1 (DenseNet), 2 (ResNet), and 3(MobileNetV2).

From Table 6, we can see that 2005 images were wrongly predicted by focal model. Now, on passing these 2005 images as dataset for testing the other 3 models, we can see that 813 were predicted correctly by the 3 models. Hence, if we use an ensemble model of all these 4 models and use majority voting, we will improve our prediction for these images. However, we have 298 images that were wrongly predicted by all the 3 models (and also the focal model). These are hard negatives and will fail in the ensemble model. A good ensemble must minimize this number.

If we go back to Table 5, we can see the accuracies for each model and use this to draw some analysis. Model 3 (MobileNetV2) has the highest test accuracy and we can see from Table 6 that the number of examples where model 3 is involved is high (226, 165). Hence, an ideal ensemble must include this member.

Note that the uniquely correct predictions in Table 6 (138, 117 and 111) are a small portion. This shows us that blindly including all the ensembles is not a good idea in cases where resources are low as it does not increase the total accuracy by a large percentage. We must wisely choose particular models, and the method of ensemnle (majority voting / weighted majority etc) also matters.

Figures 15, 16, 17, 18, 19, 20, 21 and 22 show the images for all the scenarios in Table 6.

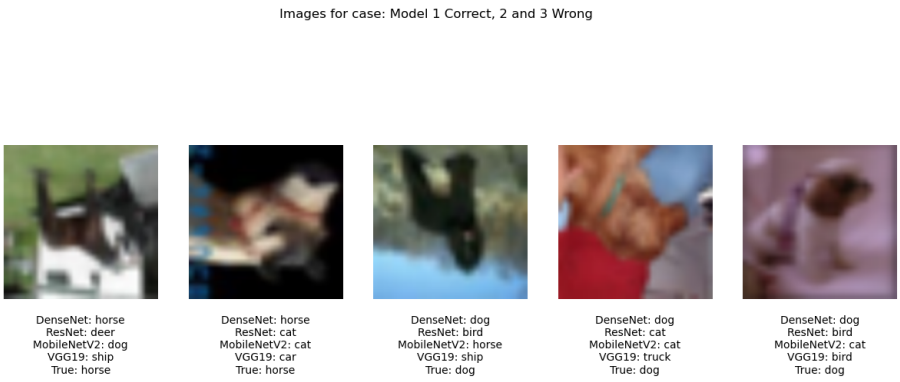


Figure 15: Model 1 Correct, 2 and 3 Wrong - Number of images: 138

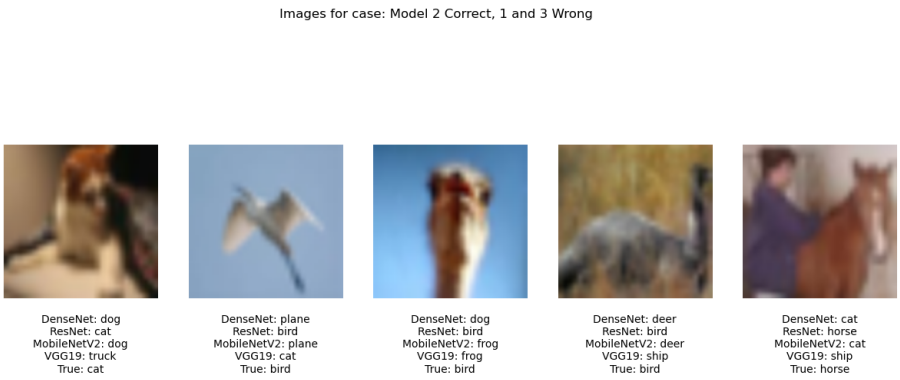


Figure 16: Case: Model 2 Correct, 1 and 3 Wrong - Number of images: 117

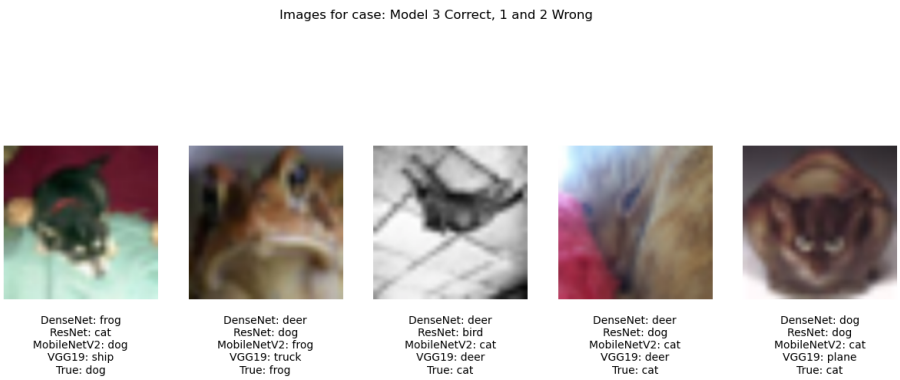


Figure 17: Case: Model 3 Correct, 1 and 2 Wrong - Number of images: 111

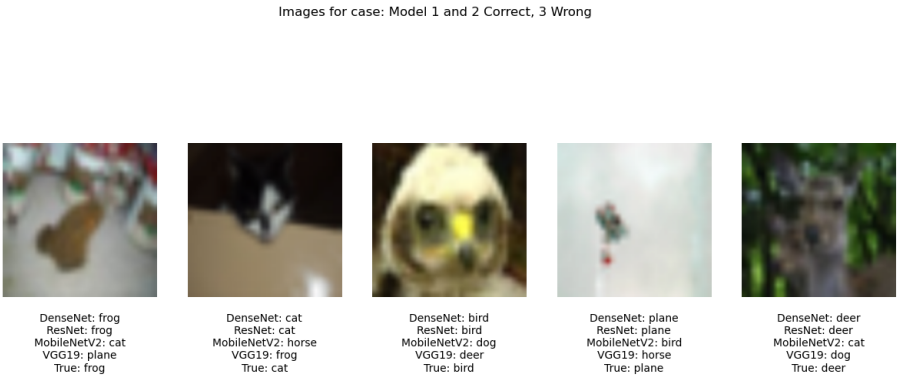


Figure 18: Case: Model 1 and 2 Correct, 3 Wrong - Number of images: 137

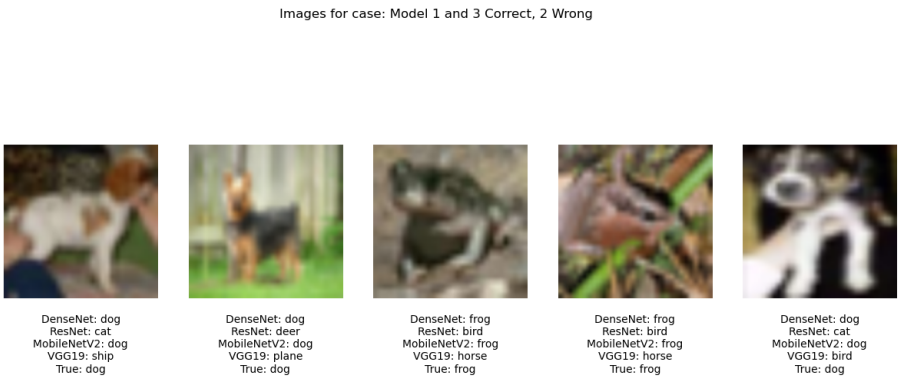


Figure 19: Case: Model 1 and 3 Correct, 2 Wrong - Number of images: 226

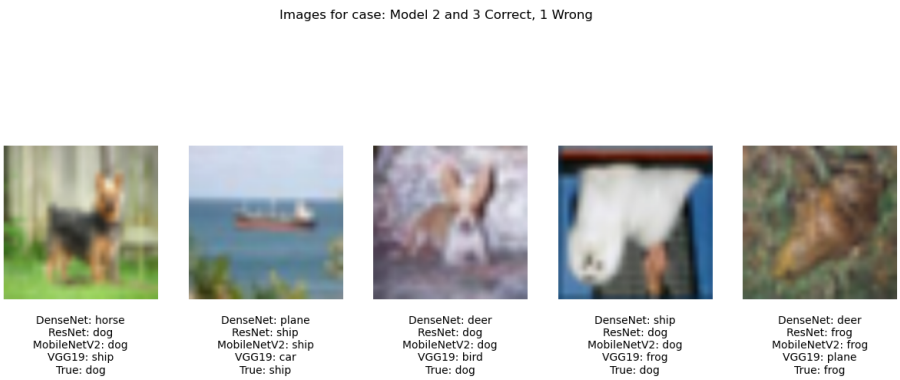


Figure 20: Case: Model 2 and 3 Correct, 1 Wrong - Number of images: 165

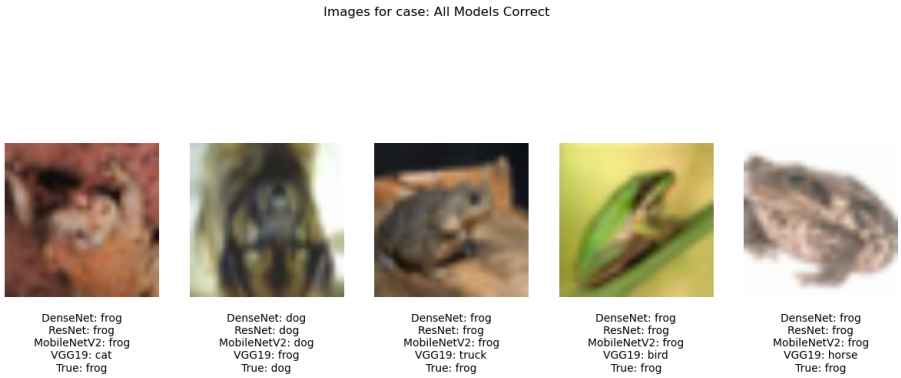


Figure 21: Case: All Models Correct - Number of images: 813

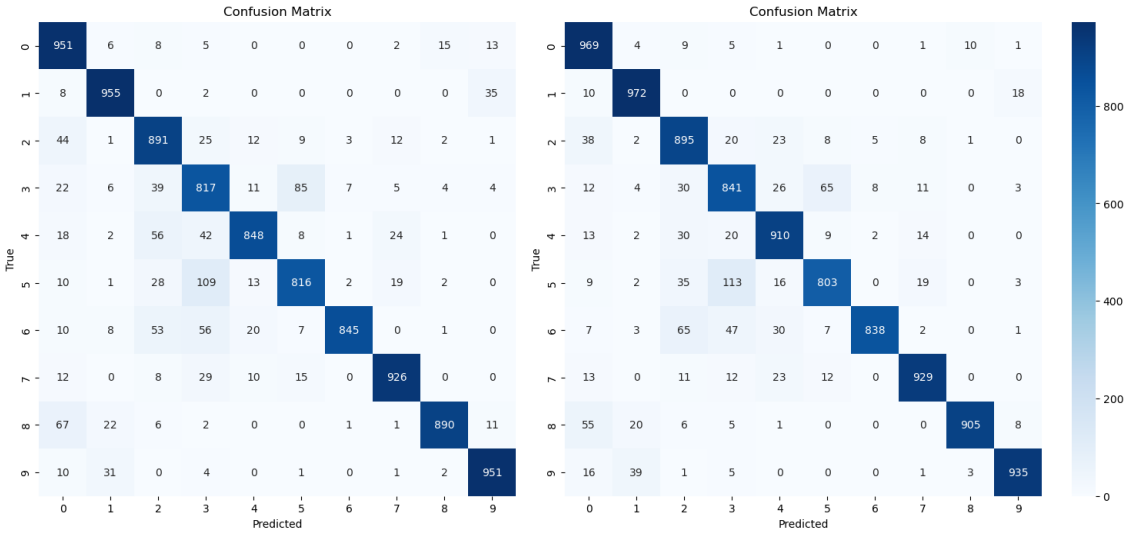
Figure 22: Case: All Models Wrong - Number of images: 298

Some interesting observations from these images are that from Figure 21, Vgg19 fails to correctly identify Frog images. From Figure 22, we can see that all models fail as it is difficult to independently identify the objects (difficult image segmentation).

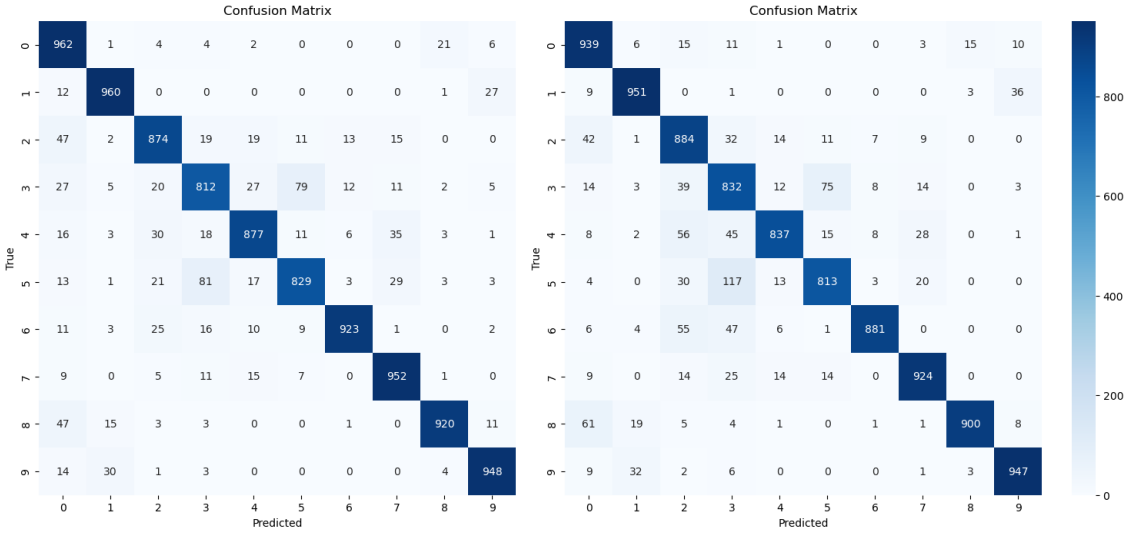
5.3 Ensemble Testing

There are 4 possible combinations of ensembles if we choose 3 out of the 4 models - 123, 124, 134 and 234. I have used majority voting method and tested for all these 4 combinations using the same test data that I used for baseline models and have recorded the reuslts in Table 7. The confusion matrices for all these 4 ensembles are in Figures 23, 24, 25 and 26.

- Ensemble 123 (DenseNet, VGG19, ResNet)
- Ensemble 124 (DenseNet, VGG19, MobileNetV2)
- Ensemble 134 (DenseNet, ResNet, MobileNetV2)
- Ensemble 234 (VGG19, ResNet, MobileNetV2)



Figures 23 and 24: Confusion matrix for Ensemble 123 and Ensemble 124



Figures 25 and 26: Confusion matrix for Ensemble 134 and Ensemble 234

Model / Ensemble	Test Accuracy (%)
DenseNet	87.3500
VGG19	79.9500
ResNet	84.6500
MobileNetV2	88.8100
Ensemble 123 (DenseNet, VGG19, ResNet)	88.9000
Ensemble 124 (DenseNet, VGG19, MobileNetV2)	89.9700
Ensemble 134 (DenseNet, ResNet, MobileNetV2)	90.5700
Ensemble 234 (VGG19, ResNet, MobileNetV2)	89.0800

Table 7: Test Accuracies for Individual Models and Ensembles.

From Table 7, we can draw several inferences.

- Any combination of 3 model ensemble will outperform any single model. All the ensembles have higher accuracy when compared to individual models.
- We can see VGG19 alone is the worst performer. Excluding this from the ensemble gives us the highest accuracy ensemble (ensemble 134). However, we cannot generalize a fact that excluding lowest individual model from an ensemble will lead to high accuracies. For this experiment, it shows this behaviour.
- Excluding second worst performer (ResNet) from the ensemble gives the second best ensemble (ensemble 124). Again, this cannot be generalized.
- Similarly, excluding the best individual model (MobileNetV2) gives us the worst performing ensemble (ensemble 123). Excluding the second best (DenseNet) model gives us second worst ensemble (ensemble 234). Again, we cannot generalize this for other datasets / other methods of voting.
- The best ensemble team is Ensemble 134 (DenseNet, ResNet, MobileNetV2). Is it a high quality? In terms of accuracy, yes. But in terms of efficiency not that much. If we take ResNet, accuracy increased by approximately 6%. But for MobileNetV2, accuracy increased by only about 1.2%. So, MobileNetV2 is better off alone in case there is resource constratints that prevent using all 3 models.
- However, for VGG19, Ensemble 124 (DenseNet, VGG19, MobileNetV2) is a high quality ensemble. This is because if we compare to standalone VGG19, then the accuracy increased by almost 10%! In this case, using an ensemble is beneficial.
- We need to remember that choosing an ensemble depends on the use case, the datasets, the models chosen and the method of voting etc. For real world use cases, extensive experimentation on all these factors is necessary to determine the best ensemble team.

6 Optional Part: Using a second Dataset - MNIST

6.1 Experiment Details

I had done MNIST classification for HW2, hence, I decided to use this dataset. However, I faced significant challenges

- The accuracies were high for base models, even for just 1 epoch, I got 95% accuracy
- The ensembles did not perform very well, I observed just 1% accuracy increase
- Hence, I decided to pollute the data by adding noise, mislabelling the data and some data transformation to get low accuracies
- However, after this process, I received mixed results as shown in Tables 8 and 9.

6.2 Experiments

I have put the dataset description, confusion matrices, etc in the appendix as this was optional. I have only described the results related to the experiments of base models and ensembles in this section. Table 8 describes the training and test accuracies for 1 epoch, for learning rate = 0.0001 (other hyper-parameters constant). I chose these values in order to attempt to get less accuracy for base model and high accuracy for ensemble. Even for 2 epochs, accuracies shot to 90% even after data contamination.

Metric	DenseNet	VGG19	ResNet	MobileNetV2
Training Loss	1.187508	1.172306	1.159877	1.601177
Training Accuracy (%)	71.0062	72.1500	71.7375	50.8771
Validation Loss	1.226497	1.208148	1.223145	1.458906
Validation Accuracy (%)	77.1167	77.6083	76.6417	68.7083
Total Training Time (s)	72.01	25.41	59.66	20.93
Test Loss	1.040335	1.031675	1.032475	1.226693
Test Accuracy (%)	76.9800	77.1700	76.5500	68.8700

Table 8: Training, Validation, and Test Metrics for DenseNet, VGG19, ResNet, and MobileNetV2.

Model / Ensemble	Test Accuracy (%)
DenseNet	76.9800
VGG19	77.1700
ResNet	76.5500
MobileNetV2	68.8700
Ensemble 123 (DenseNet, VGG19, ResNet)	77.8200
Ensemble 124 (DenseNet, VGG19, MobileNetV2)	77.1900
Ensemble 134 (DenseNet, ResNet, MobileNetV2)	76.7700
Ensemble 234 (VGG19, ResNet, MobileNetV2)	77.0600

Table 9: Test Accuracies for Individual Models and Ensembles.

Table 9 shows the accuracy across the individual models. However there are differences when we compare to the CIFAR 10 table (7).

Observations:

- We can see the base models have different accuracy rankings for MNIST. VGG19 is the best here (was worst for CIFAR-10).
- MobileNetV2 is no longer the best individual model for MNIST. Hence this affects the ensemble team with the highest accuracy.
- We can see ensemble team 123 without MobileNetV2 having the highest accuracy.
- We can also observe that DenseNet alone has much better accuracy than the ensemble 134. This shows that there is potential for decreased accuracy in an ensemble when compared to individual models of the ensemble. This is likely due to less accurate ResNet and MobileNetV2 out-voting the correct predictions made by DenseNet. Hence we need to be careful about the choice of ensembles.

Let us combine both the tables 7 and 9 to get a better view.

Model / Ensemble	**Test Accuracy (%)**
DenseNet_MNIST	76.9800
VGG19_MNIST	77.1700
ResNet_MNIST	76.5500
MobileNetV2_MNIST	68.8700
Ensemble 123 (DenseNet_MNIST, VGG19_MNIST, ResNet_MNIST)	77.8200
Ensemble 124 (DenseNet_MNIST, VGG19_MNIST, MobileNetV2_MNIST)	77.1900
Ensemble 134 (DenseNet_MNIST, ResNet_MNIST, MobileNetV2_MNIST)	76.7700
Ensemble 234 (VGG19_MNIST, ResNet_MNIST, MobileNetV2_MNIST)	77.0600
DenseNet_CIFAR10	87.3500
VGG19_CIFAR10	79.9500
ResNet_CIFAR10	84.6500
MobileNetV2_CIFAR10	88.8100
Ensemble 123 (DenseNe_CIFAR10, VGG19_CIFAR10, ResNet_CIFAR10)	88.9000
Ensemble 124 (DenseNet_CIFAR10, VGG19_CIFAR10, MobileNetV2_CIFAR10)	89.9700
Ensemble 134 (DenseNet_CIFAR10, ResNet_CIFAR10, MobileNetV2_CIFAR10)	90.5700
Ensemble 234 (VGG19_CIFAR10, ResNet_CIFAR10, MobileNetV2_CIFAR10)	89.0800

Table 10: Test Accuracies for Individual Models and Ensembles on MNIST and CIFAR-10.

For CIFAR-10, 134 is the best team, for MNIST, 123 is the best team. **Thus, we can see that there are different best ensemble teams for different datasets.** Dataset is an important deciding factor for the choice of ensemble. There are several other factors such as method of ensemble (majority voting / weighted voting) etc that should also be considered when building an ensemble for a real world use case.

7 Conclusion

- I learnt many things about ensemble learning from doing this assignment
- I also learnt a lot about image classification, data pre-processing by working with 2 datasets.
- Any combination of 'n' model ensemble will not necessarily outperform any single model. All the ensembles may not have higher accuracies when compared to individual models in majority voting method (Table 10).
- I observed that excluding the worst individual performer generally gives the ensemble with a better accuracy. However, there is no solid proof and for a third dataset, this inference may fail.
- The quality depends upon how much the individual model benefits from being in the ensemble. If the ensemble has very high accuracy compared to the individual model, then it is a good high quality ensemble. If there is low accuracy gain, then it may not be efficient to use that ensemble in a case when resources are limited.
- The best ensemble team may not necessarily be the highest quality ensemble. Quality depends on how effective the ensemble is in increasing accuracy.
- Dataset is an important deciding factor for the choice of ensemble. There can be different best ensemble teams for different datasets.
- Different base models can behave differently for different datasets which affect the ensemble accuracy and choice of models for the ensemble.
- There are several other factors such as method of ensemble (majority voting / weighted voting) etc that should also be considered when building an ensemble for a real world use case.

8 Workflow diagram

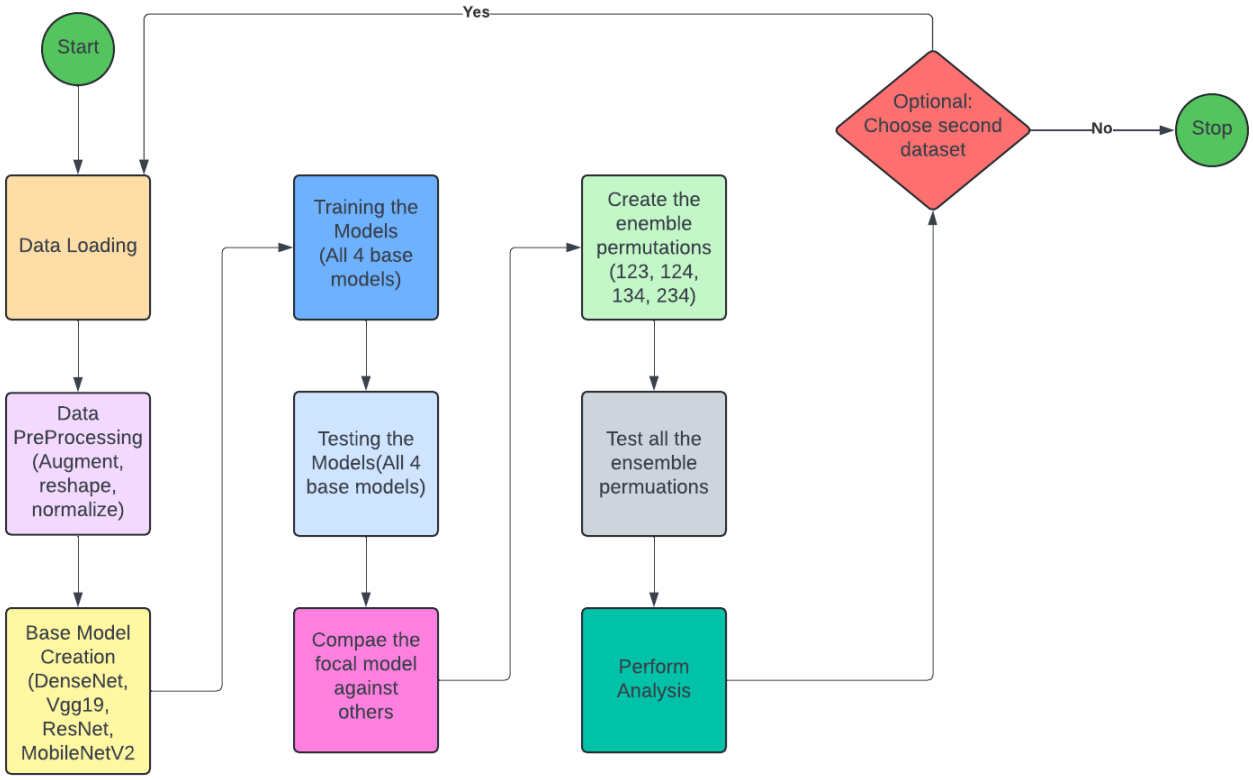


Figure 27: Workflow diagram

9 Screenshots of Execution

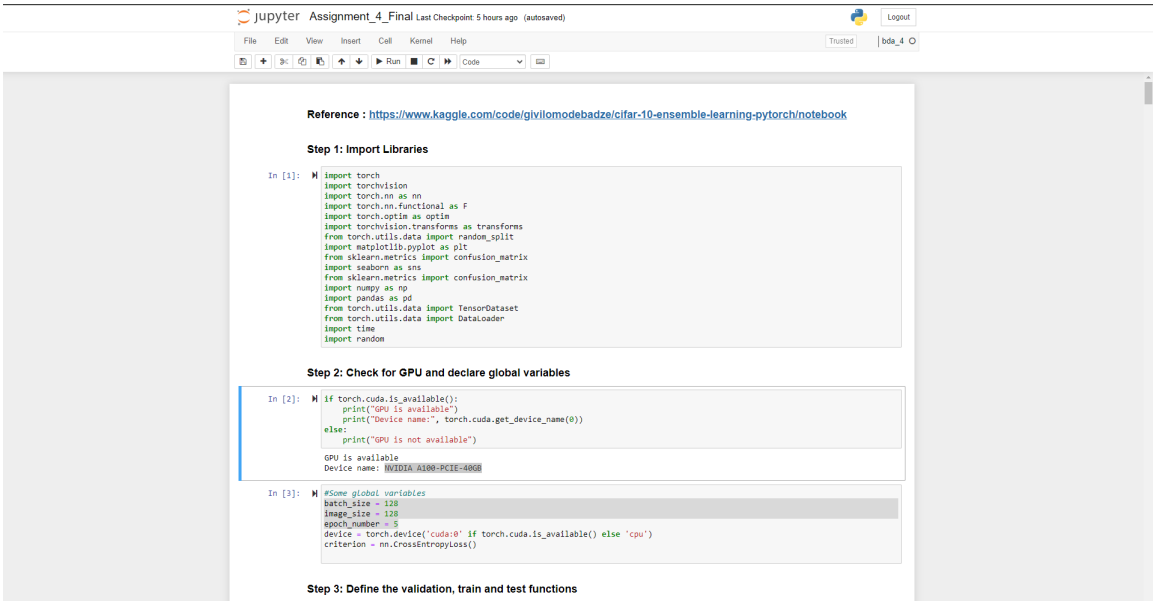


Figure 28: Jupyter notebook + Checking for GPU

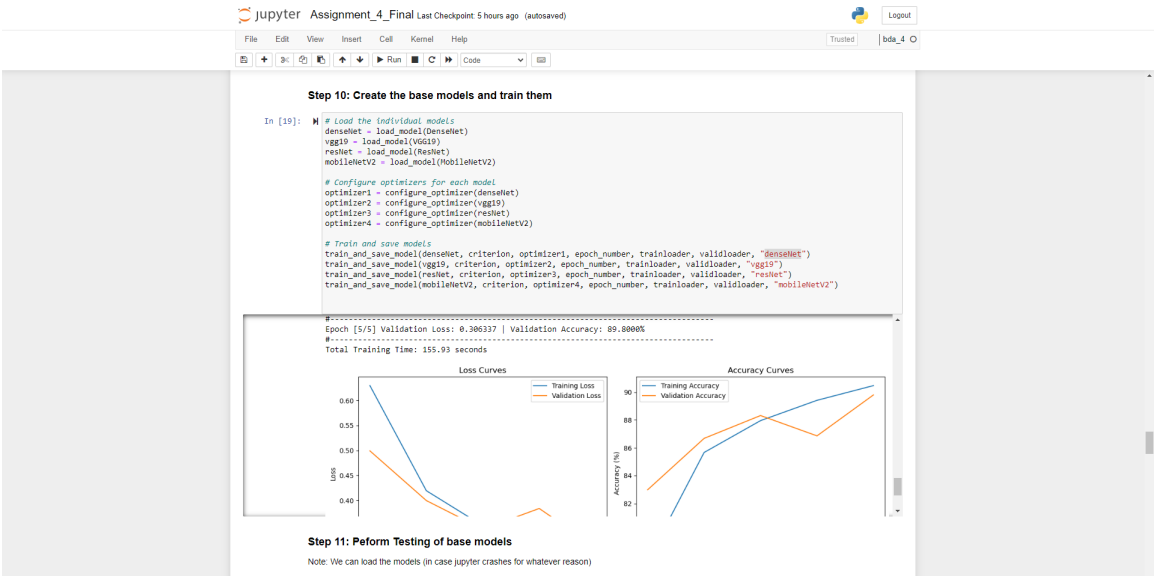


Figure 29: Training Base Models

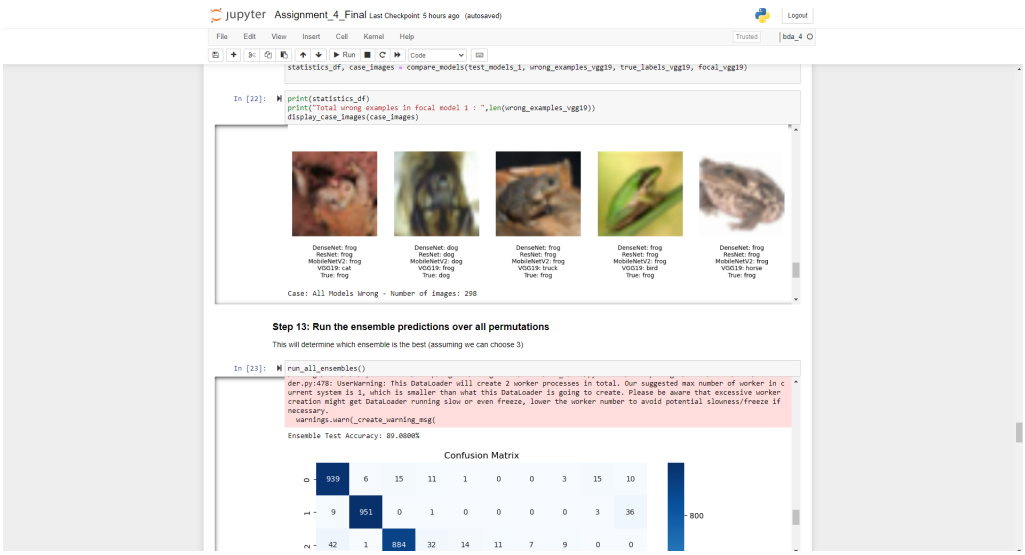


Figure 30: Performing required problem tasks

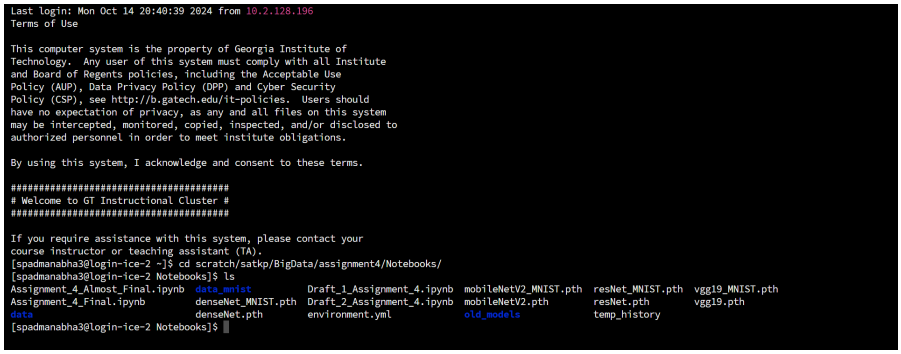


Figure 31: PACE Directory structure with models

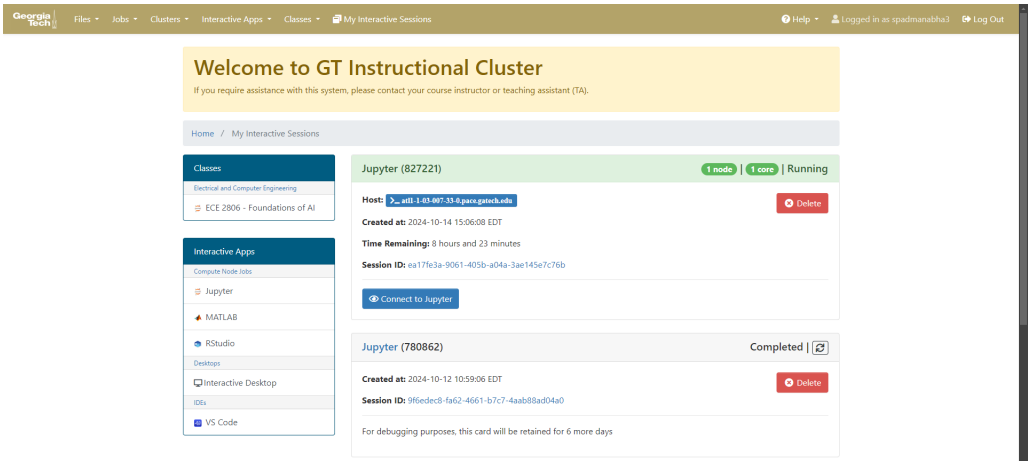


Figure 32: Setting up Jupyter Node

10 References

[1] https://www.kaggle.com/code/givilomodebadze/cifar-10-ensemble-learning-pytorch/notebook¶

[2] https://discuss.pytorch.org/t/majority-voting/207260

[3] https://www.kaggle.com/code/sharif485/pytorch-majority-voting-based-classification

[4] https://discuss.pytorch.org/t/simple-way-to-inverse-transform-normalization/4821/18

[5] https://pytorch.org/

[6] https://github.com/rasbt/mlxtend/blob/master/docs/sources/user_guide/classifier/EnsembleVoteClassifier.ipynb

[7] https://github.com/rnoxy/cifar10-cnn/blob/master/Ensemble_learning.ipynb

[8] https://scikit-learn.org/1.5/modules/generated/sklearn.metrics.confusion_matrix.html

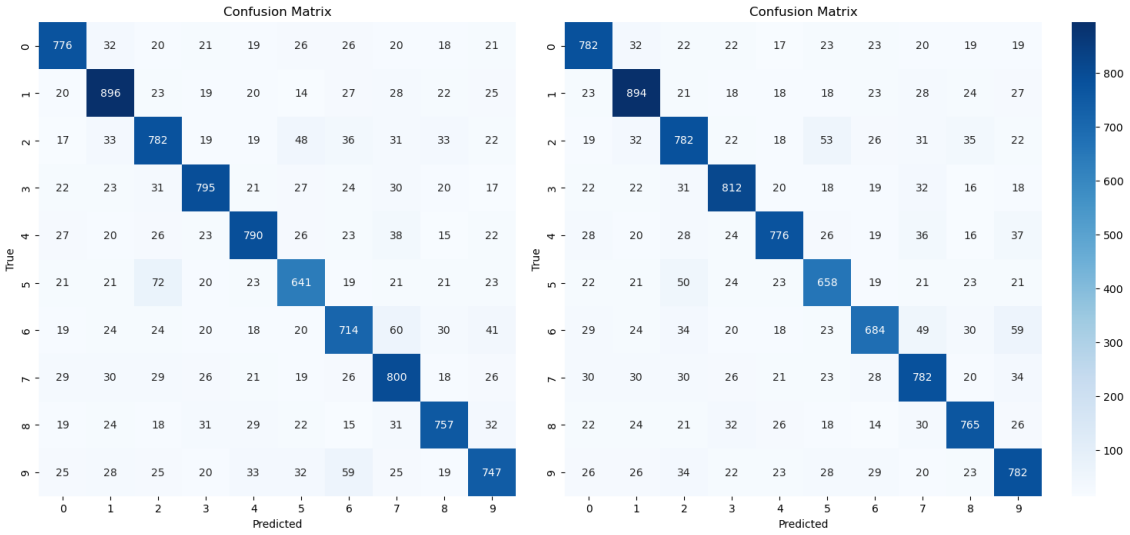
[9] Previous Homework 2

[10] CS6220 Lecture Slides

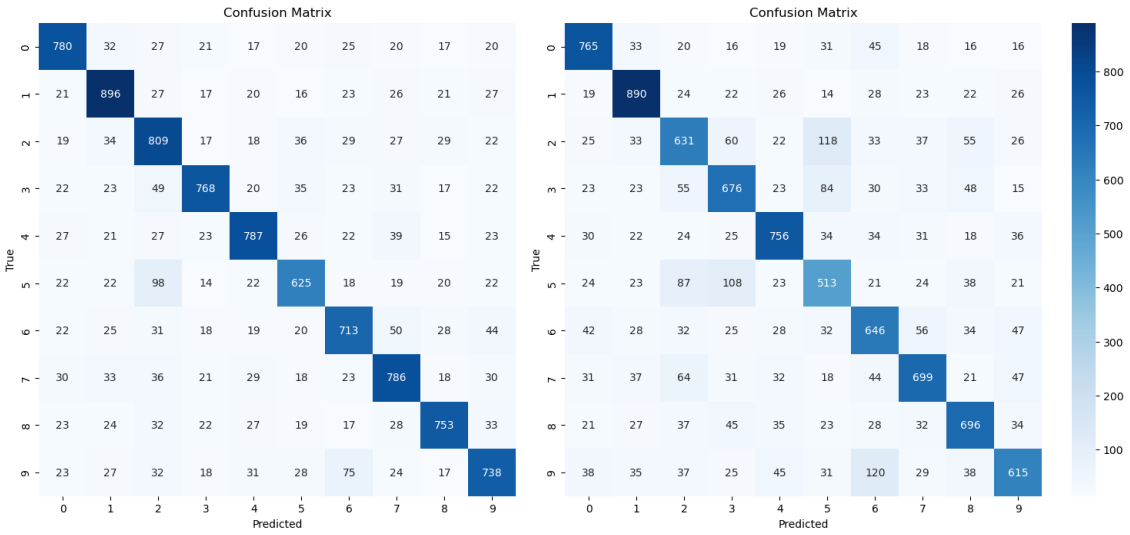
[11] https://pytorch.org/vision/stable/models.html

11 Appendix

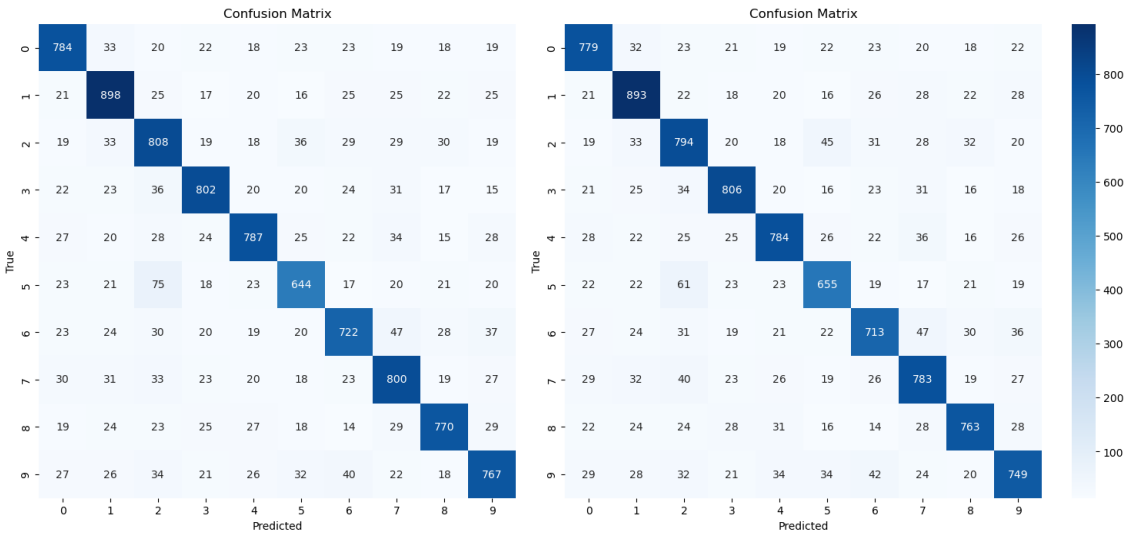
11.1 Confusion Matrices for MNIST Experiments



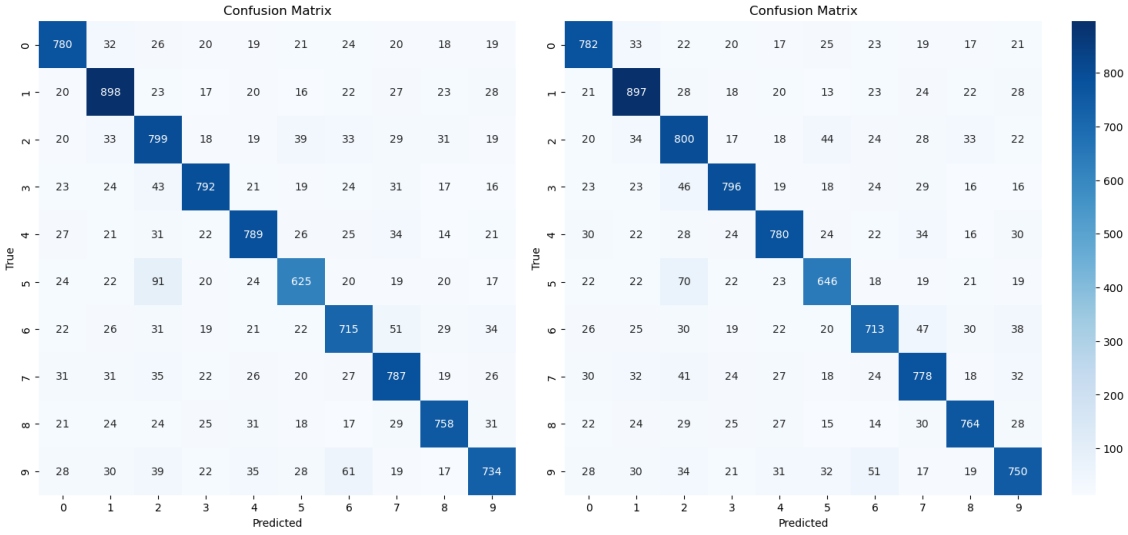
Figures 33 and 34: Confusion matrix for test set for DenseNet and Vgg19 (MNIST)



Figures 35 and 36: Confusion matrix for test set for ResNet and MobileNetV2 (MNIST)



Figures 37 and 38: Confusion matrix for Ensemble 123 and Ensemble 124 (MNIST)



Figures 39 and 40: Confusion matrix for Ensemble 134 and Ensemble 234 (MNIST)

11.2 MNIST Data EDA

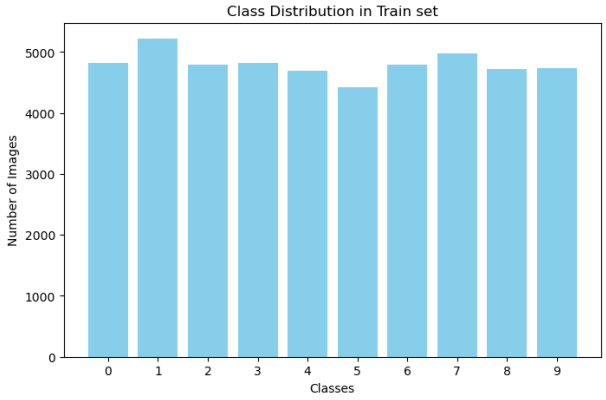


Figure 41: Train Dataset Distribution (MNIST)

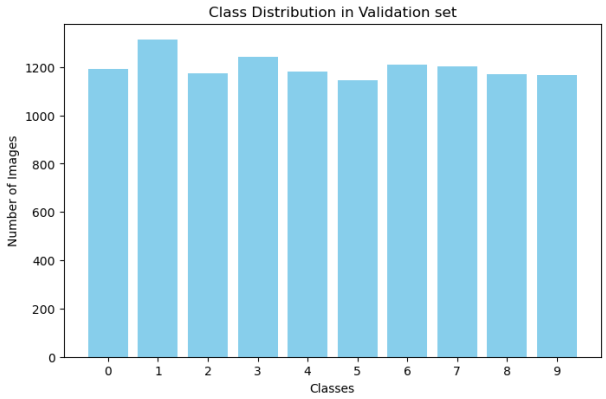


Figure 42: Validation Dataset Distribution (MNIST)

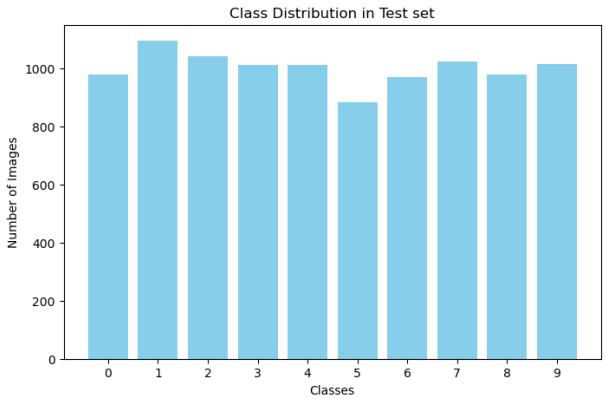


Figure 43: Test Dataset Distribution (MNIST)

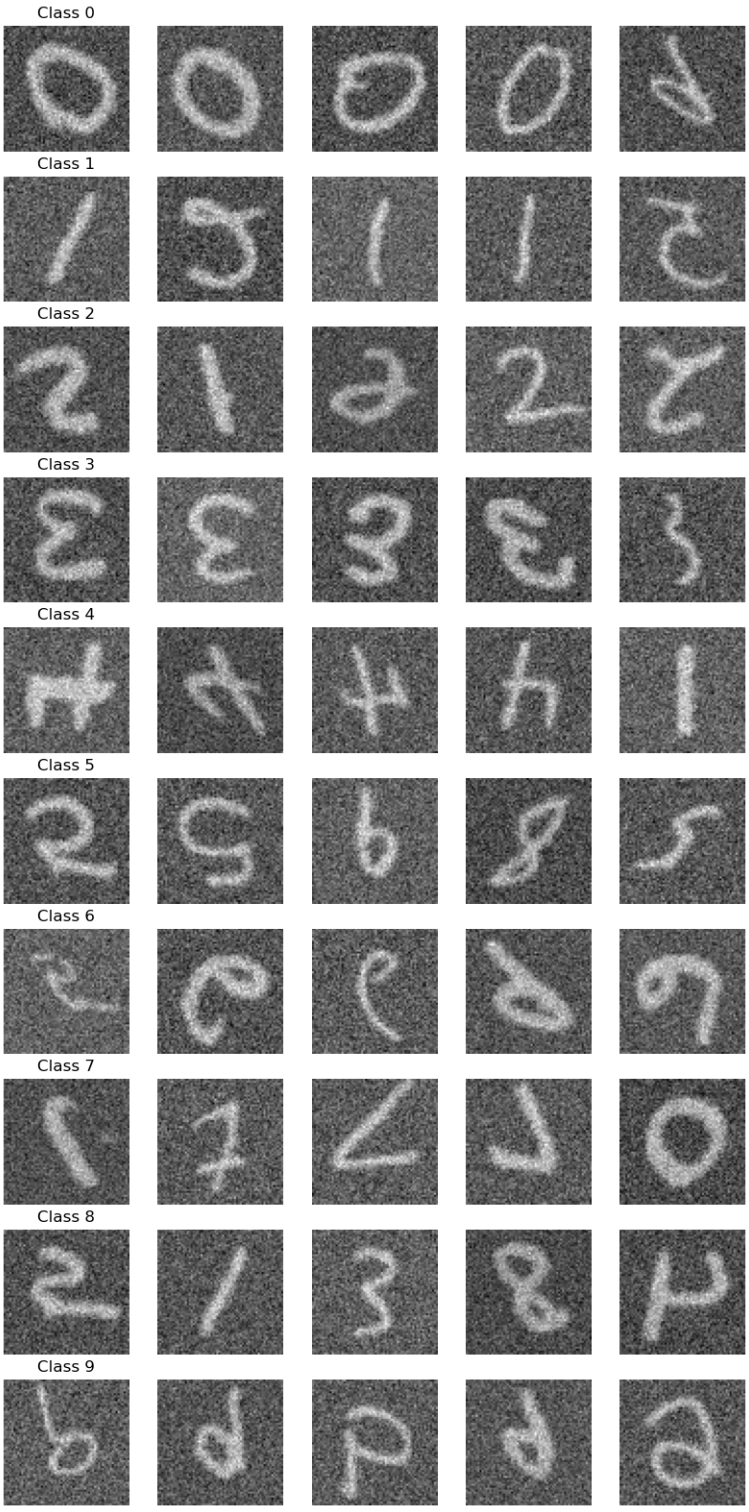


Figure 44: Sample images for each class in train dataset (MNIST)

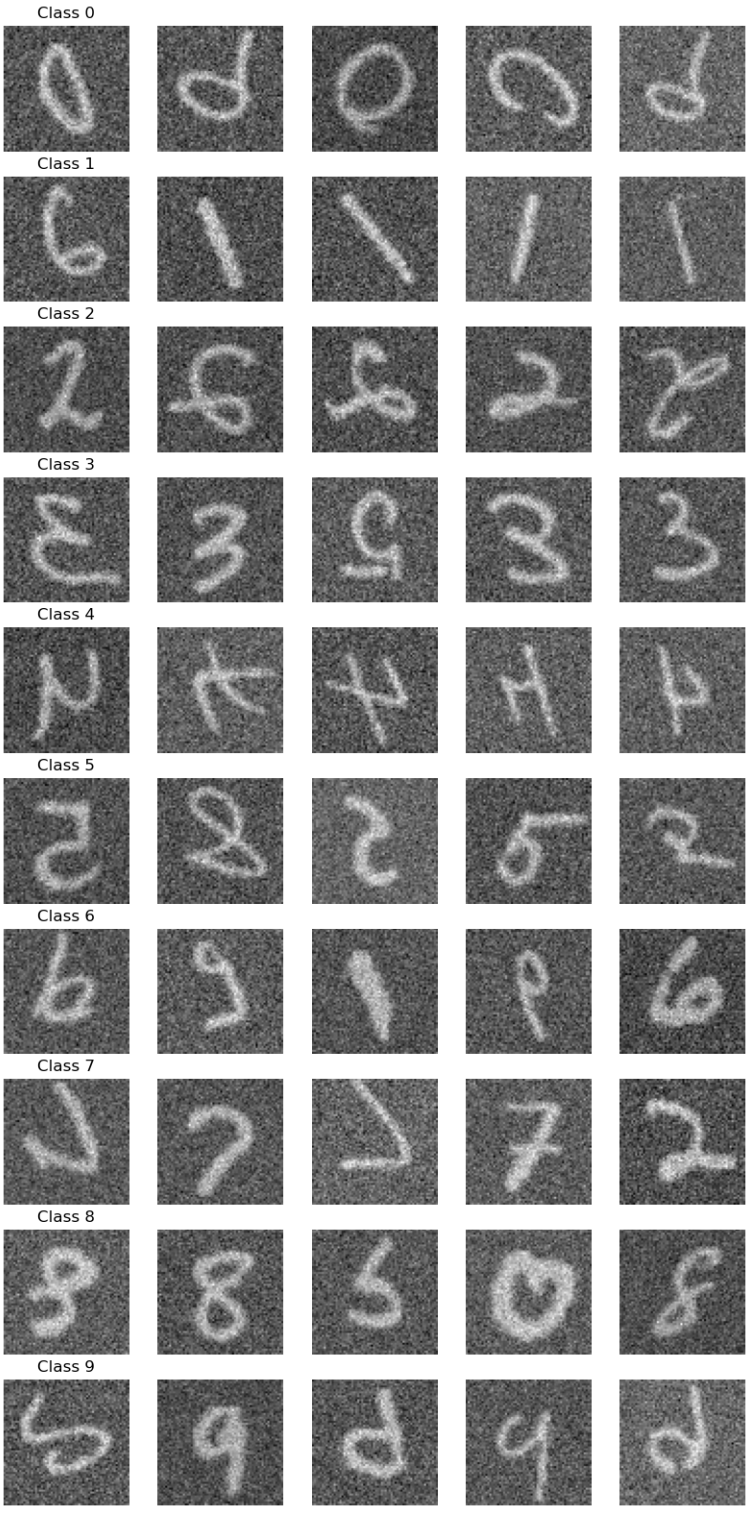


Figure 45: Sample images for each class in validation dataset (MNIST)

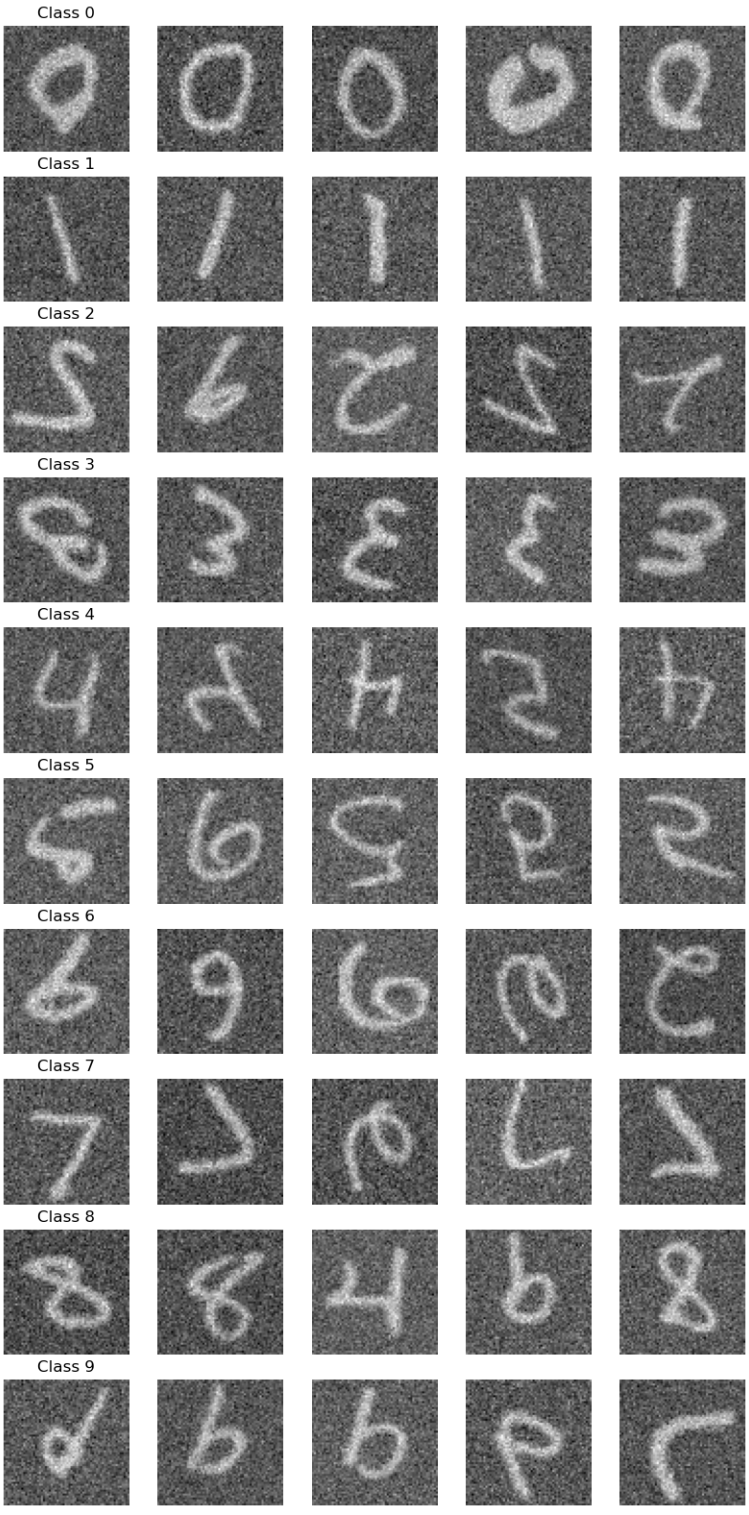


Figure 46: Sample images for each class in test dataset: (MNIST)