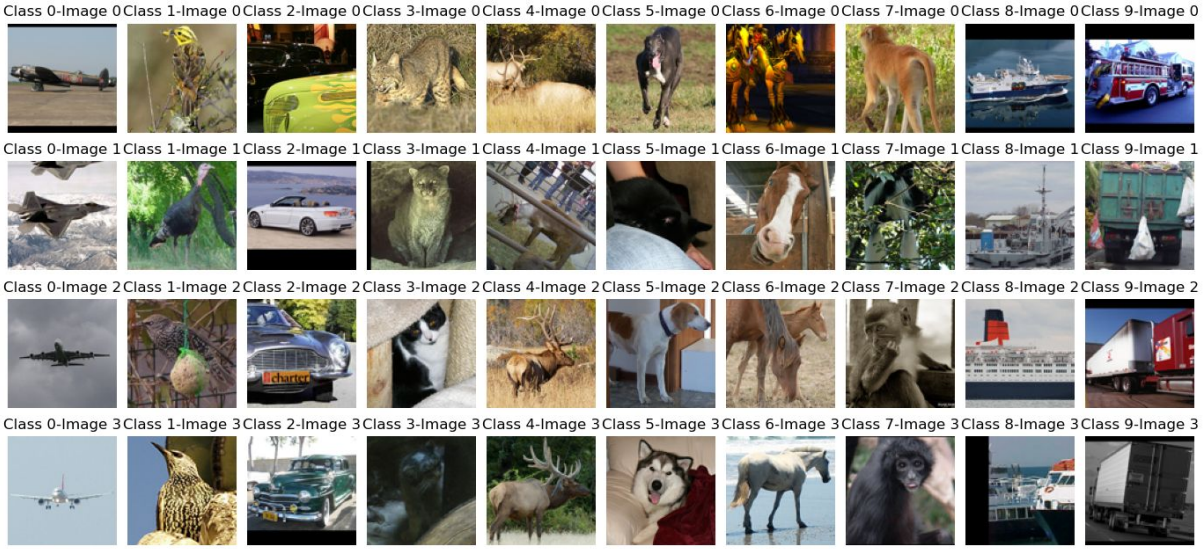
Coby Penso. 208254128

Part 1 - Visualize the data:

Visualizing the data, for every label visualize 4 images:



Part 2 - Classification with Various Networks:

In this part I implemented the following models:

Models:

1. Logistic regression over flattened version of the images
2. Fully-connected NN with 3 hidden layers over flattened version of the images. Using batch normalization and dropout to all hidden layers.
3. CNN with two convolution layers and two pooling layers. Using batch normalization to the convolution layers.
4. A pre-trained ResNet-18 as feature extractor without fine-tuning its parameters.
5. A pre-trained ResNet-18 as feature extractor with fine-tuning its parameters.

For each of the models above, the following steps made:

Steps:

1. Grid search of combinations of hyperparameters. for every set of parameters train the model for 200 epochs.
2. Save to a CSV file the table of results (test accuracy and train loss) from step 1
3. training the best model found in step 1 for 350 epochs
4. Plot the test and train loss progress

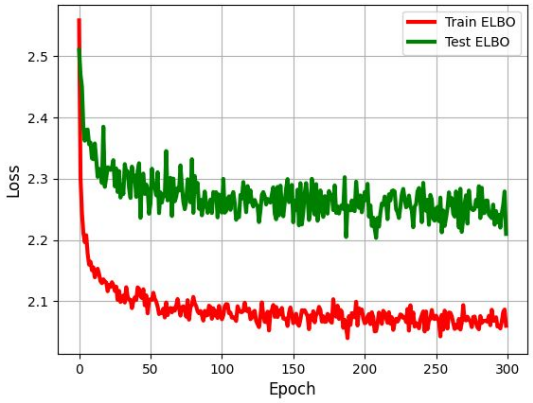
Hyperparameters results in the following attached files:

LR_hyper_param_summary.csv
FC3_hyper_param_summary.csv
CNN_hyper_param_summary.csv
ResNet18_feature_extractor_hyper_param_summary.csv
ResNet18_fine_tune_hyper_param_summary.csv

main insights: for lower Lambda coeff (small regularization) the results are better for training and validation images, probably because of the nature of the dataset the fear of overfitting is less valid in our case - enough data given the dataset's complexity.
It is known that bigger batch size leads for better results, i.e bigger batch size makes the training loss more accurate and robust. but in our case I have noticed that from batch size of 32 and above, there is no significant difference in results and performance.
In addition, lower learning rates reach more stably to the minimum and lead for better results.

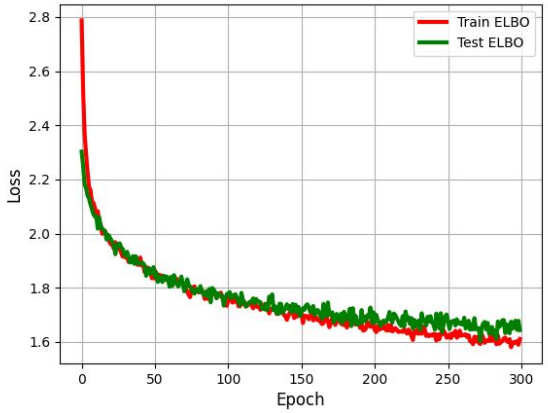
Results:

Logistic Regression - for the best model the graph tracing the loss:



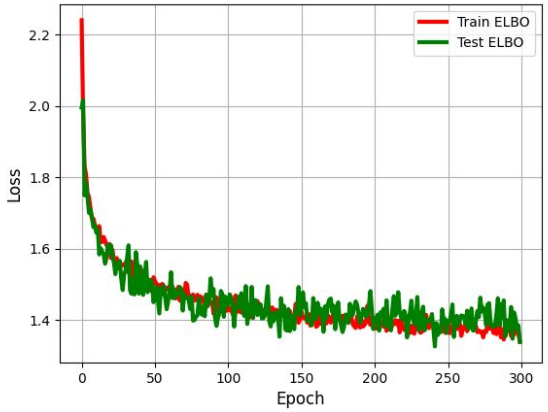
Insight: as we can see in case of logistic regression, there's a big difference between the test and train dataset performance. there's some kind of overfitting, that could be expected because the model is pretty weak for a vision dataset.

3 layers of Fully Connected - for the best model the graph tracing the loss:



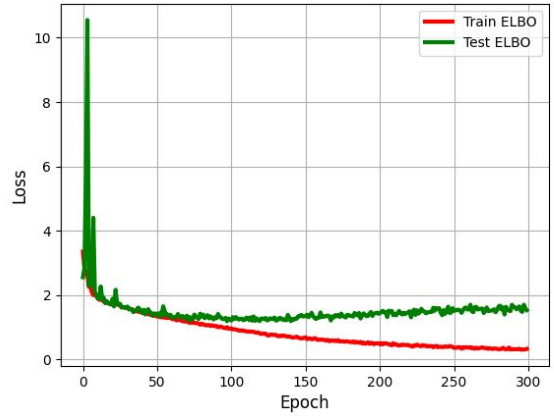
Insights: we can observe that this model didn't suffer from overfitting and achieved pretty good results considering that there is no convolutional layer.

CNN - for the best model the graph tracing the loss:



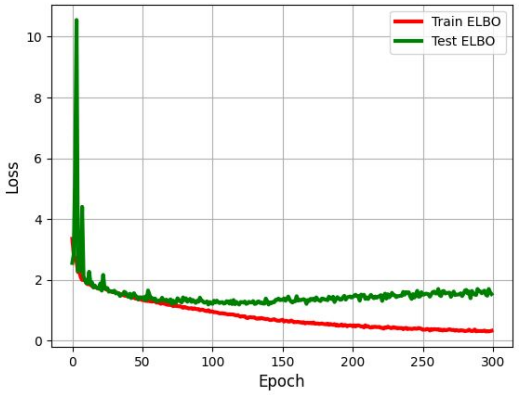
Insights: we can observe that this model didn't suffer from overfitting.
As we could predict, the results are better than the FC3 and LR, because of the convolutional layers that can capture spatial patterns and better suited for vision dataset.

ResNet18 fine tuning - for the best model the graph tracing the loss:



Insights: We can observe that the results beat previous models, as we could expect.
We took a very power model that have been trained on a bigger dataset that has the pretty much the same properties (images of objects) and then fine tuned the model to be more fitted to the STL10.

ResNet18 feature extractor - for the best model the graph tracing the loss:



Insights: Results are very close to the previous model. The difference is pretty minor in our case. Usually for a big dataset it's better to fine tune more layers of the pretrained model, In opposite to the case in which the new data is sparse and then it's better to use the pretrained model as feature extractor and learn the last layer.

Summary of accuracy:

LR - 30.325
FC3 - 44.8
CNN - 52.8375
Fine Tune - 61.775
Feature Extractor - 60.0125

Test combinations of hidden size for the ResNet18 model:

I have set the hyperparameters of the resnet18 model to be the best ones that found in the hyperparameter search.
Then, alter the hidden size of the model.
The results described in the file:

ResNet18_hidden_size_summary.csv

Insights: I have notice that for the same number of epochs. for small hiden size the model performance is a big lagging behind.
For big hidden size, it seem that the results are the same as for the small hiden size, but in my opinion, the model with bigger hidden size needs more epochs in order to learn and reach his potential.
To summarize, for 350 epochs the moderate hidden size is the one to reach the best accuracy.

Number of learnable params calculation:

The number of learnable parameters in the FC3 and CNN nets have been calculated:

FC3:

```
Layer -  model.0.weight : 100x3072 = 307200
Later -  model.0.bias : 100
Layer -  model.2.weight : 100x100 = 10000
Later -  model.2.bias : 100
Later -  model.3.weight : 100
Later -  model.3.bias : 100
Layer -  model.6.weight : 100x100 = 10000
Later -  model.6.bias : 100
Later -  model.7.weight : 100
Later -  model.7.bias : 100
Layer -  model.10.weight : 100x100 = 10000
Later -  model.10.bias : 100
Later -  model.11.weight : 100
Later -  model.11.bias : 100
Layer -  model.14.weight : 10x100 = 1000
Later -  model.14.bias : 10
Total number of learnable params in FC3:  339210
```

note: Here there are no names for the layers because they were in ModuleList.
But looking at the ModuleList it's easy to connect the index number to the layer, the fully connected layers and the batchnorm1d layers are the ones that learnable

CNN:

```
Layer -  conv1.weight : 6x3x5x5 = 450
Later -  conv1.bias : 6
Later -  bn1.weight : 6
Later -  bn1.bias : 6
Layer -  conv2.weight : 16x6x5x5 = 2400
Later -  conv2.bias : 16
Later -  bn2.weight : 16
Later -  bn2.bias : 16
Layer -  fc.weight : 10x400 = 4000
Later -  fc.bias : 10
Total number of learnable params in CNN:  6926
```

Here we can observe the name of the layer and her size(number of learnable params).
Important to notice that, the CNN net that was better in performance compared to FC3 has much less learnable parameters!

Usage:

In order to use the models trained and predict the labels on new test data, the following step should apply:

1. The script to run is: ‘Ex3_comparing_models.py’
2. Options:
    a. Specify the net to be used using: --net NET_NAME. such that NET_NAME can be LR, FC3… (use --help to get all the options for nets)
    b. Specify --hyper_parameter_search in order to search for the best set of hyperparameters for the specified net (in step a.)
    c. Specify --visualize in order to generate a grid of images from the STL10 dataset separated to classes.
    d. Specify --train_best_model in order to train the best model that is found along the hyperparam search.
    e. Specify --resnet_hidden_size_search in order to train the resnet model with different options of hidden size to find the best one (in terms of accuracy)
3. Env requirements:
    a. Numpy
    b. pandas
    c. argparse
    d. matplotlib
    e. torch
    f. torchvision