Deep Learning

Programming Exercise 3

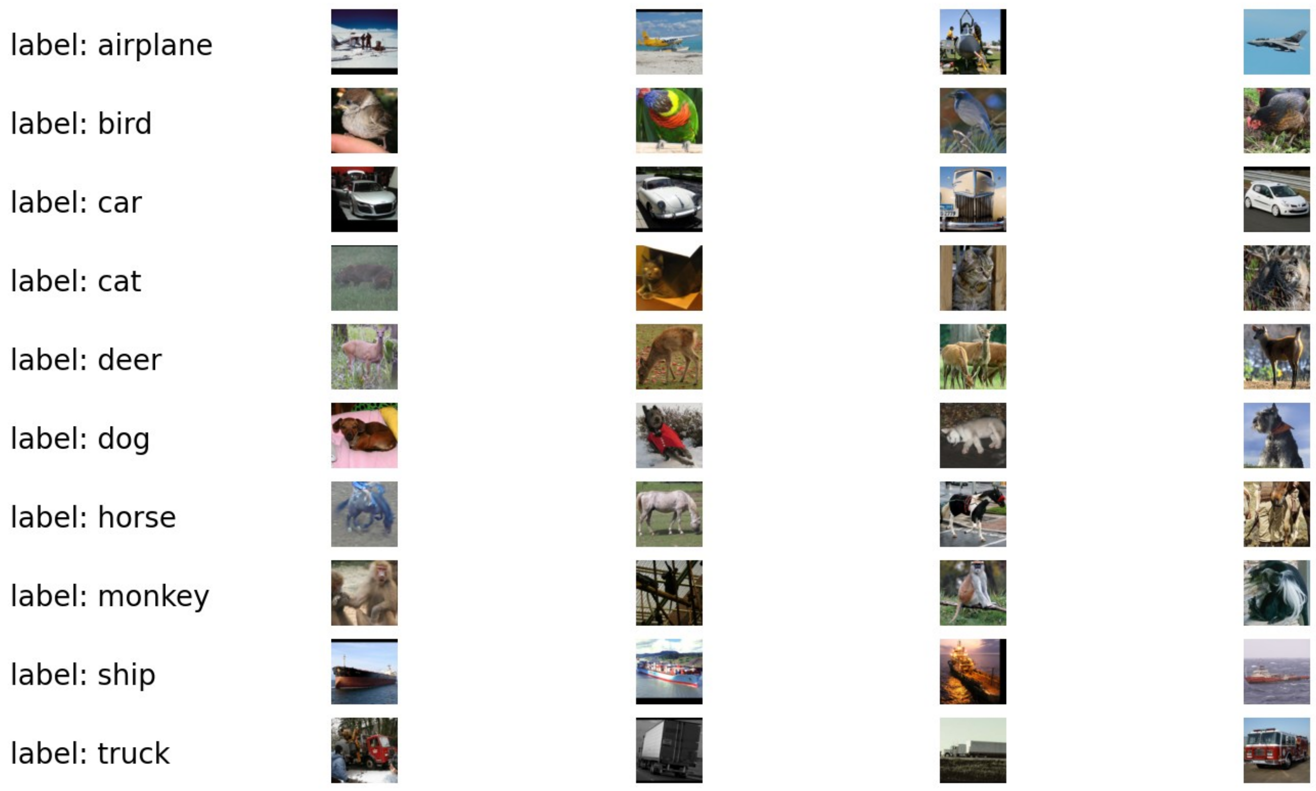
Ofer Idan, 038467668

Ron Ferens, 037222825

Table of Contents

Part 1 - Visualize the Data

The following are 40 (10x4) samples showing 4 different examples from each class in the ***STL-10*** training dataset:



Part 2 - Classification with Various Networks

In this section we have tested the performance of different networks:

1. Logistic regression over flattened version of the images
2. Fully-connected NN with at least 3 hidden layers over flattened version of the images followed by a classification layer
3. CNN with at least two convolution layers and two pooling layers followed by two fully connected layers and a classification layer
4. A fixed pre-trained MobileNetV2 as feature extractor followed by two fully connected layers and an additional classification layer
5. A learned pre-trained MobileNetV2 as feature extractor followed by two fully connected layers and an additional classification layer

For each network, we have performed an extensive ablation study using different configurations of hyperparameters such as: number of hidden layers s and their size, optimizer, batch size, learning rate, dropout rate etc.

2.1 Data Augmentation

In order to increase the model's performance and its robustness, we have utilized several pythorch integrated augmentations:

- Color jitter
- Random horizontal flip
- Random rotation

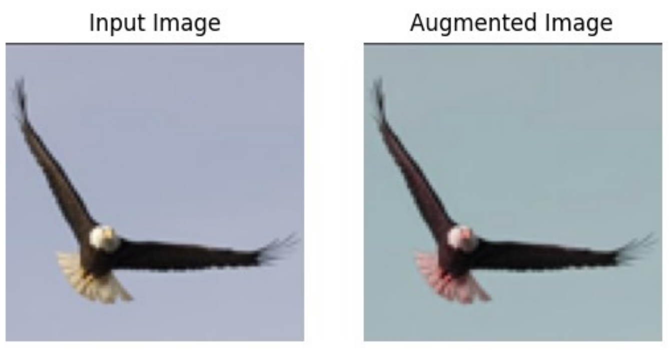
2.1.1 Color Jitter Augmentation

Reference: torchvision.transforms.ColorJitter

Definition: Randomly change the brightness, contrast, saturation and hue of an image.

The purpose of using the color jitter augmentation is to make the model more robust to changes in the lighting of the input image during inference.

When applied, the augmentation will modify the input image's brightness, saturation, contrast and hue.



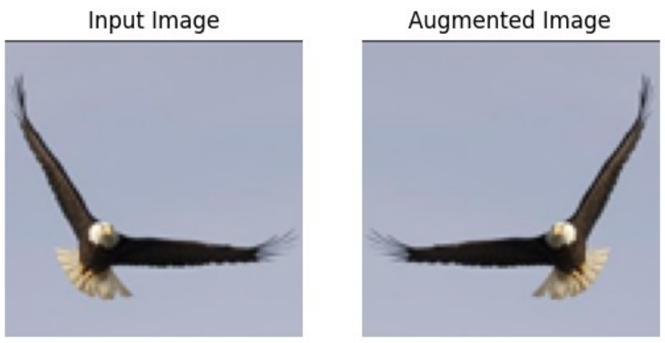
2.1.2 Random Horizontal Flip Augmentation

Reference: torchvision.transforms.RandomHorizontalFlip

Definition: Horizontally flip the given image randomly with a given probability.

The purpose of using the random horizontal flip augmentation is to make the model agnostic to the orientation of the object within the input image.

We have used a probability of p=0.5.



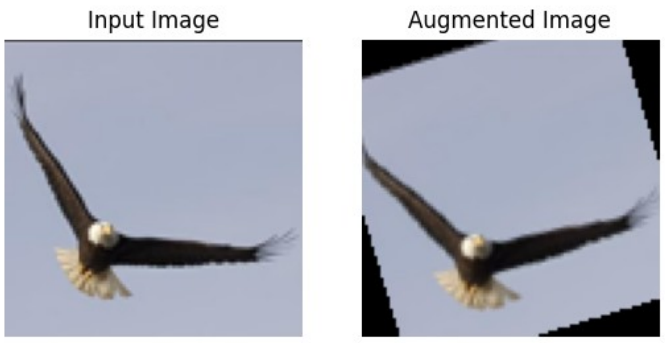
2.1.2 Random Rotation Augmentation

Reference: torchvision.transforms.RandomRotation

Definition: Rotate the image by angle.

Same as the random horizontal flip, the purpose of using the random rotation augmentation is to make the model agnostic to the orientation of the object within the input image.

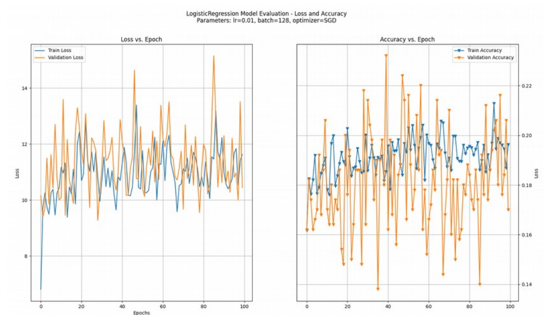
We have limited the augmentation to apply random rotation within the range of [0, 20] degrees

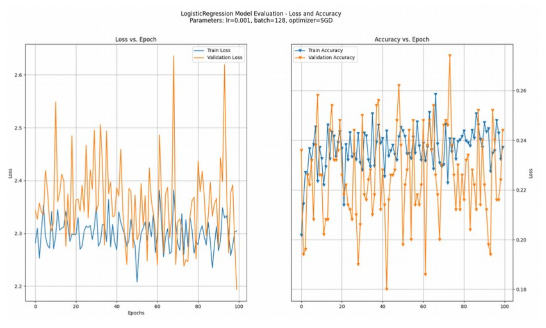


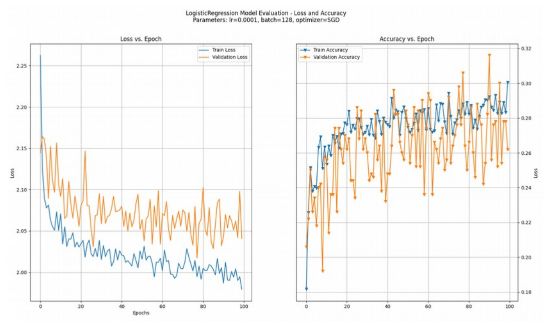
2.2. Logistic Regression Model

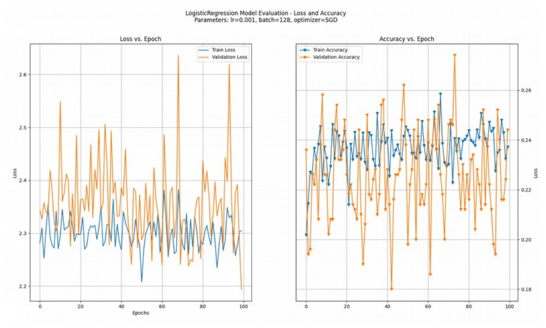
2.2.3 Learning Rate

First, in order to determine the best learning rate for the model, we have tested the model performance with the following rates: 0.01, 0.001, 0.0001 and 1e-5.









As expected, when applying high learning rate (0.01 and 0.001) the model doesn’t converge. On other other hand, when setting a low learning rate (1e-5) the model does converge, but in a very small pace.

As seen above, **the model performs best when the learning rate is set to 1e-4**.

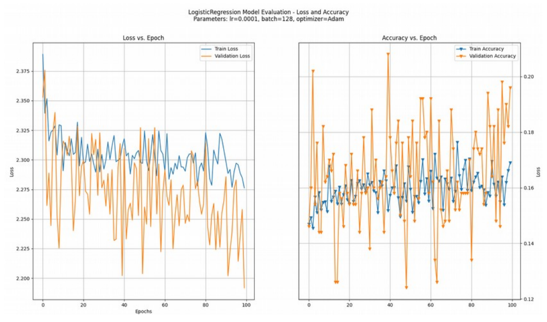
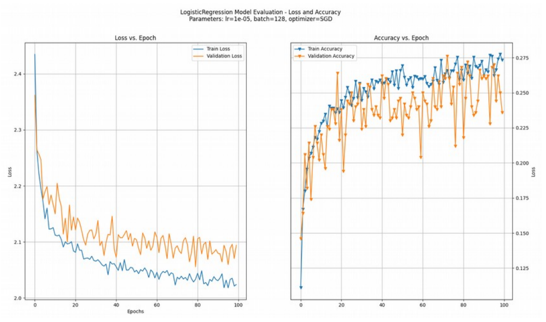
2.2.1 Batch Size

We have tested, different batch sizes: 64, 128, 256 and 512.

We found that optimal results achieved when using batch size of 128. Using batch sizes smaller or equal to 64 yield low performance. When using big batches, the model didn’t gain any gain in performance, hence **we selected 128 as our best batch size**.

2.2.2 Optimizer

To evaluate the impact of the optimizer type, used for training the data, we have tested ‘Adam’ and ‘SGD’ optimizers.

Adam Optimizer	SGD Optimizer
	

We can clearly seen that for the logistic regression model we found that the Stochastic Gradient Decent (SGD) optimizer outperformed Adam.

2.3. Fully-connected NN Model

2.3.1 Learning Rate and Batch size

Same as performed for the logistic regression model, we have evaluated the model performance when using different learning rate and batch size:

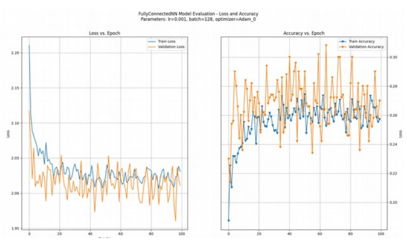
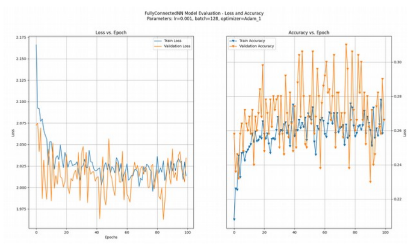
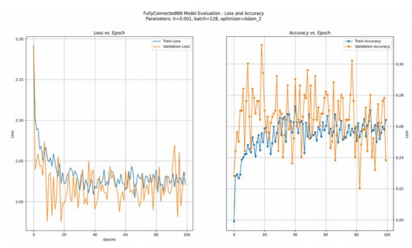
- Learning rate values: 0.01, 0.001, 0.0001 and 1e-5
- Batch sizes: 8, 16, 32, 64, 128 and 256

We have found the when using **learning rate of 0.001 and batch size of 128** the model yield the best overall performance.

2.3.2 Hidden Layers

In this section we have evaluated the model performance when setting different number of hidden layer and dimensions:

- **Three hidden layers** with the following dimensions: 512, 256, 128
- **Five hidden layers** with the following dimensions: 512, 512, 256, 256, 128
- **Seven hidden layers** with the following dimensions: 1024, 1024, 512, 512, 256, 256, 128

Three Hidden Layers	Five Hidden Layers	Seven Hidden Layers
		

As seen above, the **3 hidden layers** model outperforms the other configurations.

2.4. Convolutional Neural Network Model

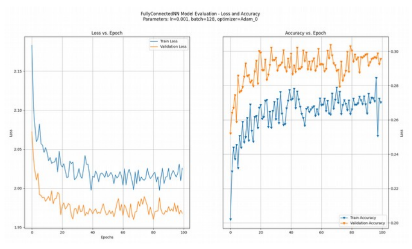
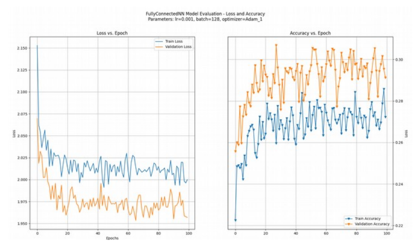
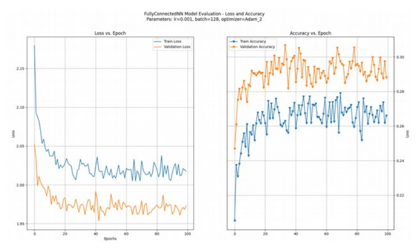
This model consist of two convolution layers and two pooling layers followed by two fully connected layers and a classification layer. Batch normalization where applied to the convolutional layers and dropout to the fully connected layers.
As feature extractor, we have implemented a custom CNN with 4 convolutional layers with the following number of kernels: 256, 512, 512 and 1000. Following each convolutional layer, we have applied batch normalization, ReLU activation function and a 2x2 MaxPooling operation.

2.4.1 Learning Rate and Batch size

As before, we have found the when using **learning rate of 0.001 and batch size of 128** the model yield the best overall performance.

2.4.2 Dropout Rate

As guided, we have applied dropout to the fully connected layers. The following experiments shows the model performance when applying different dropout rate and when disabling it:

No Dropout	Dropout p=0.2	Dropout p=0.5
		

The dropout layers are used as a regularization mechanism, hence when disabled we see the model suffers from higher overfitting. When setting a dropout rate of p=0.5, we can see that the regularization is too high and the model performance is decreasing.
To gain from the dropout regularization efficiently it is recommended to set a lower rate, such as p=0.2.

2.5. CNN with a Pre-Trained MobileNetV2 Model

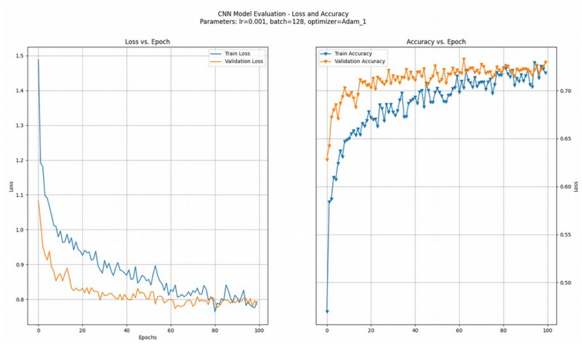
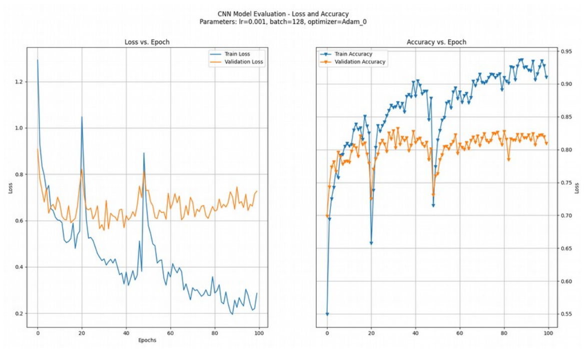
The last tested models are based on a pre-trained MobileNetV2 network used as feature extractor followed by two fully connected layers and an additional classification layer.

2.5.1 Learning Rate and Batch size

We have found the when using **learning rate of 0.001 and batch size of 128** the model yield the best overall performance.

2.5.2 Freezing the Feature Extractor

In this experiment we have evaluated the impact of updating the entire network during training or freezing the pre-trained feature extractor (MobileNetV2 network). The following graphs show the results of the two experiment:

Updating the Entire Network	Freezing the Feature-Extractor
	

When training the entire network, the model performance is better. We should note that the loaded MobileNetV2 feature-extractor was trained on ImageNet. We can conclude that the STL data distribution might require additional fine-tuning in order to boost the model’s performance.

2.6. Learned Parameters

2.6.1 Logistic Regression Model

The learned parameters in the logistic regression model consist of the weights and biases components.
The model's input is a flatten version of a image of size 64x64x3 pixels. Hence, the number of parameters are:
<input_size> * <num_of_classes> + <1 * num_of_classes> = (64*64*3) * 10 + 1 * 10 = **122,890**

2.6.2 CNN Model

The learned parameters in the CNN model consist of the ones in the 2D convolutional-based feature-extractor and the fully connected layers following it.
The model's input is a 2D image of size 64x64x3 pixels. As mentioned before, for each convolutional layer we have applied batch normalization, ReLU activation function and a MaxPooling layers. Both the activation and pooling layers don't have learned parameters, but each Batch Normalization layer add 2 learnable parameters (the learned mean and STD).
To calculate the number of parameters in each convolutional layer, we will use the formulas on the right:

1. Input layer:
   $\underbrace{(3*3*3)*256}_{Kernel\,Weights} + \underbrace{256}_{Kernel\,Biases} + \underbrace{(2*256)}_{Batch\,Normalization} = 7680$
2. Hidden layer #1:
   (3*3*256) * 512 + 512 + (2*512) = **1,181,184**
3. Hidden layer #2:
   (3*3*512) * 512 + 512 + (2*512) = **2,360,832**
4. Hidden layer #3:
   (3*3*512) * 1000 + 1000 + (2*1000) = **4,611,000**

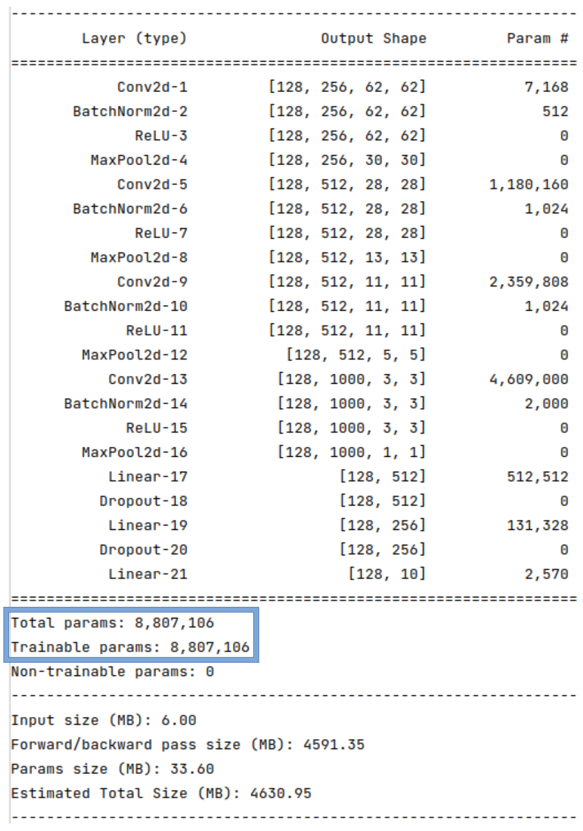
In addition to the CNN feature-extractor, we will calculate the number of feature learned by the fully connected layers:

1. Fully connected layer #1:
   (<input_size> + 1) * <num_weights> = (1000 + 1) * 512 = **512,512**
2. Fully connected layer #2:
   (512 + 1) * 256 = **131,328**
3. Output layer:
   (256 + 1) * 10 = **2570**

The total number of learned parameters in the CNN model is:
7,680+1,181,184+2,360,832+4,611,000+512,512+131,328+2570=<u>**8,807,106**</u>

To verify, we can use pytorch’s network summary function:

We see that indeed that number of trained parameters matches the number we have calculated.



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1        [128, 256, 62, 62]           7,168
       BatchNorm2d-2        [128, 256, 62, 62]             512
              ReLU-3        [128, 256, 62, 62]               0
         MaxPool2d-4        [128, 256, 30, 30]               0
            Conv2d-5        [128, 512, 28, 28]       1,180,160
       BatchNorm2d-6        [128, 512, 28, 28]           1,024
              ReLU-7        [128, 512, 28, 28]               0
         MaxPool2d-8        [128, 512, 13, 13]               0
            Conv2d-9        [128, 512, 11, 11]       2,359,808
      BatchNorm2d-10        [128, 512, 11, 11]           1,024
             ReLU-11        [128, 512, 11, 11]               0
        MaxPool2d-12          [128, 512, 5, 5]               0
           Conv2d-13         [128, 1000, 3, 3]       4,609,000
      BatchNorm2d-14         [128, 1000, 3, 3]           2,000
             ReLU-15         [128, 1000, 3, 3]               0
        MaxPool2d-16         [128, 1000, 1, 1]               0
           Linear-17                [128, 512]         512,512
          Dropout-18                [128, 512]               0
           Linear-19                [128, 256]         131,328
          Dropout-20                [128, 256]               0
           Linear-21                 [128, 10]           2,570
================================================================
Total params: 8,807,106
Trainable params: 8,807,106
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 6.00
Forward/backward pass size (MB): 4591.35
Params size (MB): 33.60
Estimated Total Size (MB): 4630.95
----------------------------------------------------------------
```

Part 3 – Testing the models

Based on the experiments detailed in the previous sections, we report the models performance when applied the best configurations:

Model	Test Loss	Test Accuracy
Logistic regression over flattened version of the images	1.981	0.302
Fully-connected NN with 3 hidden layers over flattened version of the images followed by a classification layer	1.993	0.351
CNN with 4 convolution layers followed by two fully connected layers and a classification layer	0.950	0.705
A fixed pre-trained MobileNetV2 as feature extractor followed by two fully connected layers and an additional classification layer	0.772	0.732
A learned pre-trained MobileNetV2 as feature extractor followed by two fully connected layers and an additional classification layer	0.589	0.834

Part 4 - Running the Code

- The entire code is placed within the *main.py* and consists of three different sections
    - *Part 1 - Visualize the Data*
    - *Part 2 – Train a model and run validation*
    - *Part 3 – Test a selected model*

- In order to run a specific part, you should comment out all others