School of Information Technology

AAI3001 - Computer Vision and Deep Learning

Small Project Report

Member Name	Matriculation No.
Davin Lim	2201898
Lim Jo Han Derrick	2201629

Task 1

Framework: Pytorch
Model used: EfficientNetb0

Steps to run code:

Pre Requisites

- Step 1: Navigate to the root folder “task_1” on cmd line.
- Step 2: Run pip install -r requirements.txt.

(Must be in root folder task_1 for the following to work)

- **Task01_train.py**
  - Run “**python Task01_train.py**” on cmd line or run Task01_train.py within an IDE.
- **Task01_test.py**
  - Run “**python Task01_test.py**” on cmd line or run Task01_test.py within an IDE.

Fine Tuning of last layer

Before training the EfficientNetb0 model, the default weights of the model were initialized and the last layer of the model was replaced with a Linear layer which outputs 10 classes to fit the labels of the dataset which had been pre-processed using LabelBinarizer() to return a one-hot-encoded array of the ground truth label of each sample.

Experimental parameters of the training

Random seed used:	42
Train, Validation, Test split:	0.7,0.15,0.15
Learning Rates:	[0.01,0.1]
Batch Size:	32
Criterion used:	CrossEntropyLoss
Optimizer used:	SGD
Number Of Epochs:	15

Augmentation Sets used

Transform Set	Train	Val/Test
Transform Set 1	Resize(64) RandomCrop(64) ToTensor() Normalize(mean = [0.485,0.456,0.406]	Resize(64) CenterCrop(64) ToTensor() Normalize(mean = [0.485,0.456,0.406]
Transform Set 2	Resize(64)	Resize(64)

	RandomCrop(64) RandomHorizontalFlip() RandomRotation(45) ColorJitter(0.2) ToTensor() Normalize(mean = [0.485,0.456,0.406]	CenterCrop(64) RandomRotation(45) ColorJitter(0.2) ToTensor() Normalize(mean = [0.485,0.456,0.406]
Transform Set 3	Resize(64) RandomCrop(64) RandomHorizontalFlip() RandomRotation(45) RandomAutoContrast() ToTensor() Normalize(mean = [0.485,0.456,0.406]	Resize(64) CenterCrop(64) ToTensor() Normalize(mean = [0.485,0.456,0.406]

As the dimensions of the image were of size 64 * 64, all dimensions of any resizing and crop transformations were set to be size 64 to improve train and evaluation time.

The first augmentation set was chosen to improve accuracy on images which are not centered as images for real world data can be off-centered.

The second augmentation set was chosen to improve the predictions of the model on random rotation and adjustments to the hue and sharpness of images as real world images can be less sharp than training images.

The third augmentation set was chosen to evaluate the model without altering the validation/test data after training the images on the different types of common augmentations.

Training Steps:

All combinations of transform sets with learning rates were utilized to train the model. The class accuracy and class precision of the best epochs for all experimental parameters were printed out within the console.

Model accuracy per learning rate was then compared to determine the best models per transform set. The best models of each transform set were saved to the model folder and then compared to finally determine the best model. After comparing all models, the model chosen was as follows, this model will be loaded and used for evaluation on test set:

Overall Model Chosen

Model Weight	**Model accuracy**	**Transform Set**

best_model_transform1.pth	0.97753(5dp)	1 (trg_transform1, val_transform1)

The validation accuracies and precision of the classes of the best model are as shown:

```
----------
+-------+------------------------+-------------------------+
| Class | Val Set Class Accuracy | Val Set Class Precision |
+-------+------------------------+-------------------------+
|   0   |         97.42          |          98.64          |
|   1   |         98.14          |          99.07          |
|   2   |         97.75          |          98.84          |
|   3   |         98.43          |          99.21          |
|   4   |          98.3          |          99.15          |
|   5   |         97.73          |          98.76          |
|   6   |         94.07          |          96.93          |
|   7   |         99.56          |          99.78          |
|   8   |         95.81          |          97.71          |
|   9   |         99.77          |          99.89          |
+-------+------------------------+-------------------------+
Val set Classes Average acc: 97.6993742723119
Val set Classes Average precision: 98.79762891599712
----------
```

Evaluation On Test Set

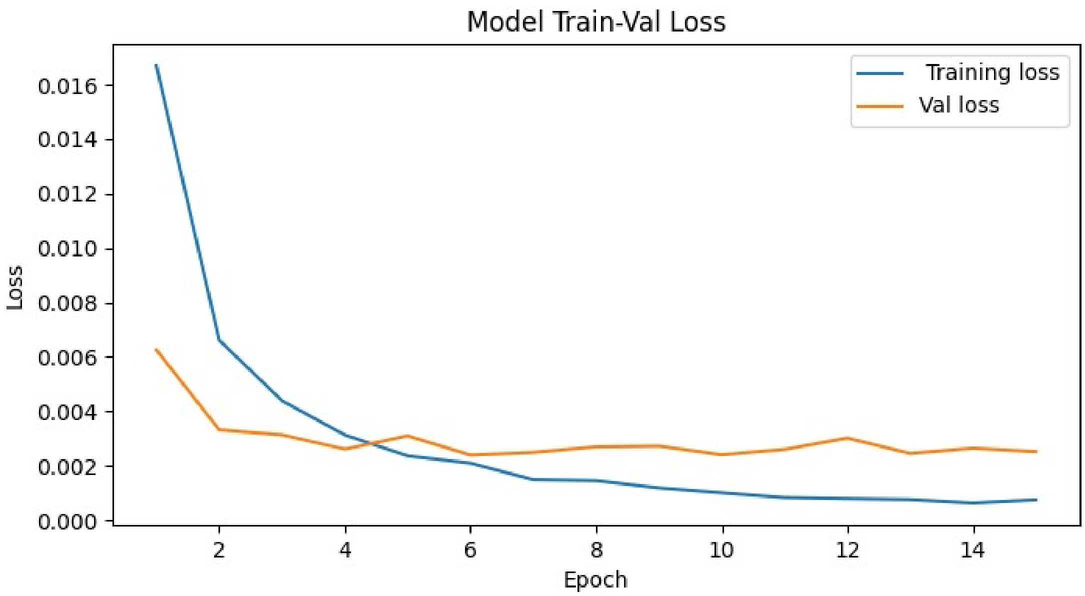
The model was loaded using the saved weights which were stored in a .pth file and evaluated on the test set in **Task01_test.py**. The results were as follows:

```
--------------Loading Model and Testing Phase--------------------
Best Model Transform IDX: 0
Best Model Acc: 0.9775308966636658
Test Set Acc: 0.98%
+-------+-------------------------+--------------------------+
| Class | Test Set Class Accuracy | Test Set Class Precision |
+-------+-------------------------+--------------------------+
|   0   |          98.0           |           99.0           |
|   1   |          99.0           |          100.0           |
|   2   |          97.0           |           99.0           |
|   3   |          98.0           |           99.0           |
|   4   |          98.0           |           99.0           |
|   5   |          91.0           |           95.0           |
|   6   |          96.0           |           98.0           |
|   7   |          99.0           |           99.0           |
|   8   |          99.0           |           99.0           |
|   9   |         100.0           |          100.0           |
+-------+-------------------------+--------------------------+
Test set Classes Average acc: 97.44839282894124
Test set Classes Average precision: 98.6564556885574
Test results saved!
```

All predictions of the test set by the best model were also saved to the "**test_predictions.txt"** file.
All results for the best model on the test set were saved to "**test_results.txt**".

Training/Validation Loss Curves

All train-val loss curves of the best models per learning rate and transform were plotted and saved to the graph folder. The figure below shows the train-val loss curve for the chosen best model.



Task 2

Framework: Pytorch
Model used: EfficientNetb0

Steps to run code:

Pre Requisites

- Step 1: Navigate to the root folder “**task_2**” on cmd line.
- Step 2: Run **pip install -r requirements.txt**.
- Step 3: Make sure that the **main.py, dataset.py, train.py** and **validate.py** is in the directory
- Step 4: Run python **main.py** on cmd line or run **main.py** within an IDE.

Setting used to define the last layer

I began loading the pre-trained EfficientNet-B0 model, which uses the default weights that was pre-trained on ImageNet. Next, I modified the model’s last classification layer with a linear layer which outputs 10 classes to fit the labels of the dataset which had been pre-processed to return a one-hot-encoded array of the ground truth label of each sample, followed by a sigmoid activation function. This allows the model to output probabilities for binary classes.

Experimental parameters of the training

Random seed used:	42
Train, Validation, Test split:	0.7,0.15,0.15
Learning Rates:	[0.001, 0.01, 0.1]
Batch Size:	32
Criterion used:	BCELoss
Optimizer used:	Adam
Number of epochs:	10

Data Augmentation used:

```
# Data Augmentations
transform1 = transforms.Compose([
    transforms.Resize(64),
    transforms.CenterCrop(64),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

Labelling Changes

```
labels = self.labels[idx]
if labels[0] == 1:      # if AnnualCrop == 1, PermanentCrop == 1
    labels[6] = 1
elif labels[6] == 1:    # if PermanentCrop == 1, AnnualCrop == 1
    labels[0] = 1
elif labels[1] == 1:    # if Forest == 1, HerbaceousVegetation == 1
    labels[2] = 1
return img, labels
```

In order to turn it into a multi-label classification, I modified the binarized labels in the CustomDataset's __getitem__ function, where:

- Set the label Annualcrop to 1 whenever PermanentCrop is 1, the other way round
- Set the label HerbaceousVegetation to 1 whenever Forest is 1.

Training Procedures

For task 2, I change the criterion from CrossEntropyLoss to BCELoss to handle multi-label classification by computing the loss for each label independently, and then summing the losses. Also, instead of Softmax activation function used in Task 1, I used sigmoid activation function to get predictions probabilities between 0 and 1.

Training Loop:

1. Set Model in Train mode
2. Batch Processing
3. Compute Predictions
4. Calculate Loss with criterion
5. Update Gradients
6. Adjust Model Parameters
7. Iterate through batch and find average train loss
8. Return train loss

Validation Loop:

1. Set Model in Eval mode
2. Set threshold to 0.6
3. Iterate through validation data
4. Generate Predictions
5. Calculate Loss with criterion
6. Apply sigmoid activation function to predictions and apply thresholding
7. Record y_true and y_pred labels
8. Concatenate all validation data y_true and y_pred
9. Compute Subset Accuracy (1 if y_pred labels match strictly with y_true labels, else 0)
10. Compute Hamming Loss (fraction of wrong labels to the total number of labels)
11. Compute Average Precision Score for each class

12. Compute Accuracy for each class
13. Return all metrics

Train Model:

1. Store the best hyperparameters, weights, epoch and the train/validation losses for the best hyperparameter.
2. Save best model at best hyperparameter, epoch and accuracy

Overall Model Chosen

Model Weight	Model Subset Accuracy	Hyperparameters
best_model.pth	0.96814 (5dp)	lr = 0.001

Evaluation on Test Set

The best model was loaded using the saved hyperparameters and weights which were stored in '**best_model.pth**' file and evaluated on the test set. The results are stored in the **test_results.txt** under the outputs folder as follows:

outputs > test_results.txt

```
Best Model Subset Accuracy: 96.8148
Test Subset Accuracy: 84.5926
Test Hamming Loss: 0.0300
Test Validation Loss: 0.0851


Test Accuracy (per class)
Class 0: 93.9753
Class 1: 98.0494
Class 2: 95.7531
Class 3: 96.5679
Class 4: 98.3457
Class 5: 98.1481
Class 6: 94.5185
Class 7: 97.7531
Class 8: 97.5309
Class 9: 99.3580

Test AP Score (per class)
Class 0: 77.2819
Class 1: 84.6746
Class 2: 84.7385
Class 3: 67.7236
Class 4: 83.7801
Class 5: 76.9138
Class 6: 78.7895
Class 7: 82.9964
Class 8: 75.9349
Class 9: 93.9902

Test Mean AP Score Over All Classes: 80.68236505612742
Test Mean Accuracy Over All Classes: 97.0
```

Training/Validation Loss Curves

All train-val loss curves of the learning rates and transform were plotted and saved to the **training_outputs** folder. The figure below shows the train-val loss curve for the chosen best model with hyperparameter learning rate 0.001 stored in the **outputs** folder.

train_val_loss_curve_0.001.png