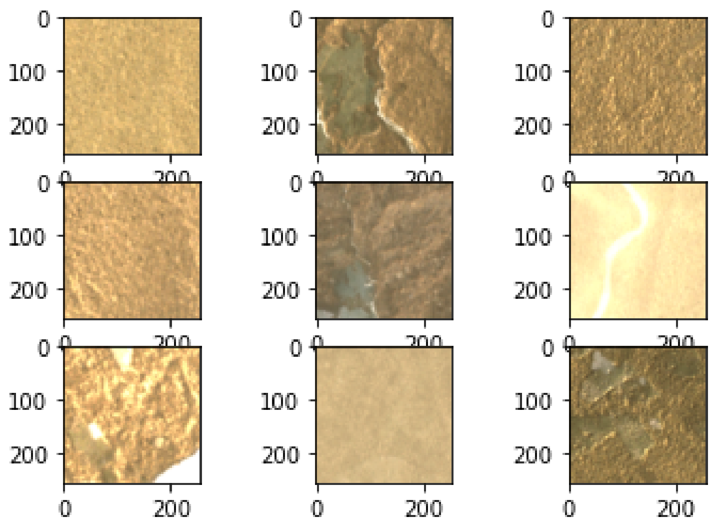
Report of the Mandatory Assignement 1

Lina Saba

22 February 2022, 23 March 2022



Preview of the data

Contents

1 Introduction

This report is a detailed explanation of the steps I followed during the first mandatory assignment.

This assignment's goal is to apply the notions we worked on such as develop a model using an algorithm of optimisation to reduce the errors between the model and the data. We will need to predict the labels of our images from a resnet-18 model.

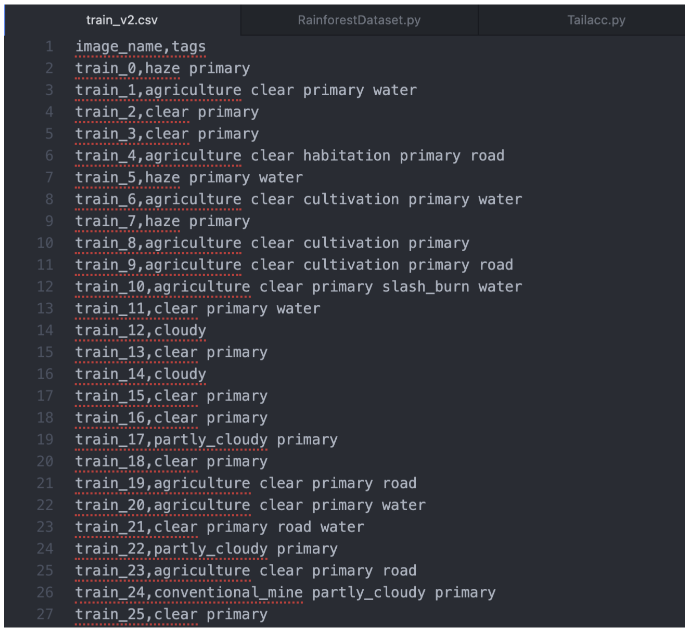
Working with a custom loss is the challenge of this work.

2 Task 1:

In order to get to coding, I started by uploading the dataset by following the commands of this document *Additional commands for running files on the server.* I got access to the ML Nodes by going to this website `https://www.uio.no/tjenester/it/forskning/kompetansehuber/uio-ai-hub-node-project/it-resources/ml-nodes/#toc2` and login in order to register to AI HUB provides resources and services for machine learning and deep learning tasks at UiO. Once, my demand was accepted, I could type this command on my terminal and type my code to get the rainforest.tar zipped folder in my computer.

After zipping the folder, we get:

- train_v2.csv : a csvfile defining the names of each image without the '.tif' extension and their predefined labels. For instance; [train_0] for the 1st imagename and [haze primary] for the labels of this image.



```
image_name,tags
train_0,haze primary
train_1,agriculture clear primary water
train_2,clear primary
train_3,clear primary
train_4,agriculture clear habitation primary road
train_5,haze primary water
train_6,agriculture clear cultivation primary water
train_7,haze primary
train_8,agriculture clear cultivation primary
train_9,agriculture clear cultivation primary road
train_10,agriculture clear primary slash_burn water
train_11,clear primary water
train_12,cloudy
train_13,clear primary
train_14,cloudy
train_15,clear primary
train_16,clear primary
train_17,partly_cloudy primary
train_18,clear primary
train_19,agriculture clear primary road
train_20,agriculture clear primary water
train_21,clear primary road water
train_22,partly_cloudy primary
train_23,agriculture clear primary road
train_24,conventional_mine partly_cloudy primary
train_25,clear primary
```

Figure 1: csv data

Each image will have potentially one or more atmospheric label. In order to save the labels used in each image I thought about using the split function so it's easier to use them for the classes later.

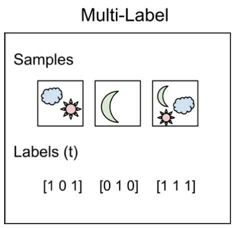


Figure 2: Multi-Labels data

- a folder with 40478 .tif photos forming the dataset.

1. Let's start by defining the function _init_() :

```
def __init__(self, root_dir, trvaltest, transform):
    """
    This function's goal is to define how we initialise the data
    At first we create two lists image_paths and split_labels to store the path labels of
    Second we binarise the multi-labels from the string
    Third we make the fit_transform of the multi-labels which will be stored to be used fo
    Last we define the train and test split if it s 0 then train else it s 1
    """
    self.image_paths = []
    self.split_labels = []
    self.transform = transform
    #https://www.w3schools.com/python/pandas/pandas_csv.asp
    #https://datatofish.com/convert-pandas-dataframe-to-list/
    root_cc =  root_dir+ "/train_v2.csv"
    root_csv = pd.read_csv(root_cc)
    #root_csv = root_csv.iloc[:100]
    liste = root_csv.values.tolist()
    for line in liste :
        image_path = root_dir + "/train-tif-v2/" +line[0]+".tif"
        self.image_paths.append(image_path)
        #https://www.w3schools.com/python/ref_string_split.asp
        self.split_labels.append(line[1].split())
    #https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MultiLabelBin
    classes,_ = get_classes_list()
    mlb = MultiLabelBinarizer(classes=classes)
    #https://scikit-learn.org/stable/modules/preprocessing.html
    y = mlb.fit_transform(self.split_labels)
    #https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_
    X_train, X_test, y_train, y_test = train_test_split(self.image_paths, y, test_size=0.33
    if trvaltest == 0 :
        self.image_paths = X_train
        self.split_labels = y_train
    else:
        self.image_paths = X_test
        self.split_labels = y_test
```

2. Next function to run is the __getitem__ This function's goal is to load the image, label and filename from an index

```
    def __getitem__(self, idx):
        """
        This function get the label and filename and load the image from file after transformi
        """
        #https://www.geeksforgeeks.org/python-pil-image-open-method/
        image = Image.open(self.image_paths[idx])
        label = self.split_labels[idx]
        #print(label)
        #print(image)
        if self.transform:
            #https://pytorch.org/vision/stable/transforms.html
            image = self.transform(image)
        else:
            image = torch.transforms.ToTensor()(image)
        sample = {'image': image, 'label': self.split_labels, 'filename': self.image_paths[idx
        return sample
```

3. After that I completed the function **runstuff** to run **RainforestDataset** and get the data set and load.

```
    image_datasets={}
    root = '/itf-fi-ml/shared/IN5400/2022_mandatory1/'
    image_datasets['train']= RainforestDataset(root,0,data_transforms['train'])
    image_datasets['val']= RainforestDataset(root,1,data_transforms['val'])
    # Dataloaders
    #Exercises week 3 part 1: Implement a dense neural network
    dataloaders = {}
    dataloaders['train'] =  torch.utils.data.DataLoader(image_datasets['train'], batch_size=co
    dataloaders['val'] = torch.utils.data.DataLoader(image_datasets['val'], batch_size=config[
```

4. Create a Resnet-18 model and drop it last layer and create a new fully-connected layer : Knowing that :

pretrained_net.fc.in_features = 512 and numcl = 17

```
    pretrained_net.fc = torch.nn.Linear(512, 17)
```

5. Minimizing 17 separate binary classifiers with a loss function ?
Our main goal is to define a function that will calculate the difference the predictions of our model and the ground truth of the data. But what is the most appropriate loss function for our example? Let's focus on the data type. Here, we have limited categories as the prediction is to tell whether the images' labels are 'agriculture' , 'clear' , 'primary' , 'water' for instance. So the problem is a classification one. And the most appropriate loss function within classification data is the **BCEWithLogitsLoss**. **BCEWithLogitsLoss** is Binary Cross Entropy loss with logits. It's the stable version of **BCE**, where we add a sigmoid layer before calculating its BCELoss.
This is the formula behind the python program :

$$BCE = -\frac{1}{N}\sum_{c=0}^{N}(y_c \log(y_c) + (1 - y_c)\log(1 - y_c)) \tag{1}$$

```
class yourloss(nn.modules.loss._Loss):

    def __init__(self, reduction: str = 'mean') -> None:
        super(yourloss, self).__init__()
        self.loss = nn.BCEWithLogitsLoss()
        pass
    def forward(self, input_: Tensor, target: Tensor) -> Tensor:
        #https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html
        BCE_loss = self.loss(input_.float(), target.float())
        return BCE_loss
```

6. APM is the average precision measure. Precision measures how accurate is your predictions.

$$precision = \frac{TP}{TP + FP} \tag{2}$$

with TP = True Positive and FP = False Positive
So first, we collect all the predictions and use them in the sklearn suitable function; average_precision_score. The concat_labels and concat_pred are 100*17 arrays that'll stock the information we need to calculate the average precision for each class separately.

```
for c in range(numcl):
    avgprecs[c] = average_precision_score(concat_labels[:, c], concat_pred[:, c])
    avgprecs[c] = np.nan_to_num(avgprecs[c])
```

So concat_labels[:, c] equals the concat_labels of the class c. This is what will be used for all the 17 classes. After that we can just run the mean of these measures with an already-done script :

```
avgperfmeasure = np.mean(perfmeasure)
```

7. **Results** We can after initialising and completing all needed functions plot the MAP and loss curves by using this code :

```
#plot the train and test:
plt.plot(range(num_epochs), trainlosses)
plt.plot(range(num_epochs), testlosses)
plt.legend(['train_loss', 'test_loss'])
plt.title('Loss per epoch ')
plt.savefig('loss_curve.pdf')
plt.close()
```

The result is :

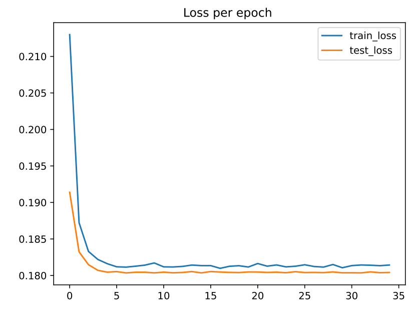


Figure 3: Loss per epoch Curve

```
#plot the Mean Average precision:
plt.plot(range(num_epochs), mean_avg_precs)
plt.title("Mean Average precision over epochs")
plt.xlabel("Epochs")
plt.ylabel("Mean Average precision")
plt.savefig("Mean_Average_precision.pdf")
plt.close()
```

The result is :

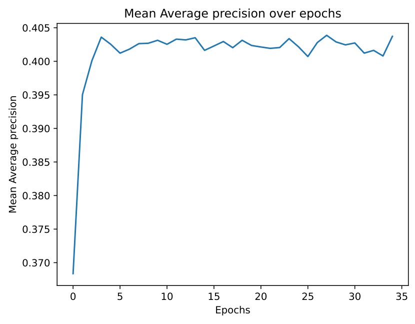


Figure 4: Mean Average precision over epochs

3 Task 2:

Compute for each of the 17 classes the accuracy of predictions in the upper tail, for 10 to 20 values of t from t = 0 if classification threshold is zero, or from t = 0.5 if classification threshold is 0.5, until t = max f(x). Each t separates a percentage of the validation data. So we define such as t is between 0.5 and the maximum of sample_scores.

$$\text{Tailacc}(t) = \frac{1}{\sum_{i=1}^{n} I[f(x_i) > t]} \sum_{i=1}^{n} I[f(x_i) = y_i] I[f(x_i) > t], t > 0$$

Figure 5: Tailacc Function

```
def Tailacc(predictions, labels, t):
    """
    Create an average over all 17 classes for 10 to 20  @values of t.
    It'll show how the accuracy changes as the threshold changes
    #another method
    for c in range(numcl):
        pred = predictions[:,c]
        lab = labels[:,c]
        pred_thresholded = pred[pred>t]
        labels = labels[pred >t]
    return np.sum(pred_thresholded == labels)
    """
    #stores the indice of prediction
    ind = np.where(predictions > t)[0]
    if len(ind) == 0:
        return(labels[np.argmax(predictions)])
    tail_labels = labels[ind]
    return np.mean(tail_labels)
```

After defining evenly spaced values of t for the function Tailacc, we plot it this way :

```
#plot tailacc
plt.plot(t_values, mean_tailaccuracies)
plt.xlabel("t")
plt.ylabel("accuracy")
plt.title("Tailaccuracies")
plt.savefig("Tailacc.pdf")
plt.close()
```

The result is :

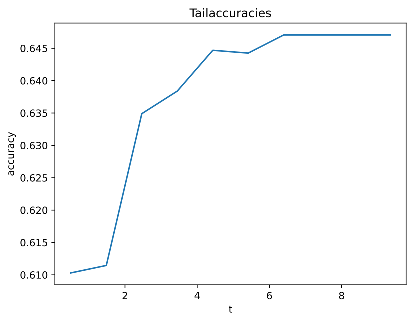


Figure 6: Tailaccuracies

Let's plot the top-10 ranked images and the bottom- 10 ranked images : (the code is in the file *TopAndBottom.py*)

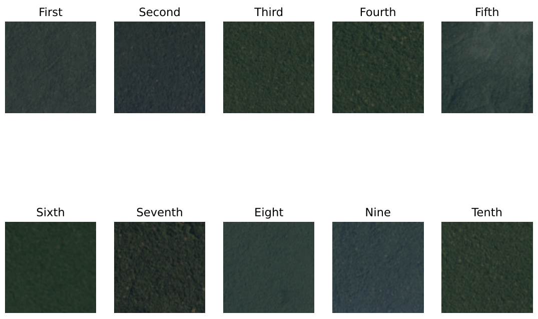


Figure 7: top-10 ranked images

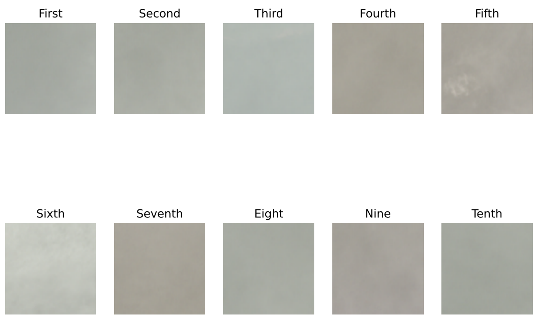


Figure 8: bottom-10 ranked images

The top-50 highest scoring images for a given class looks so well when the ranking measure (average precision) is not perfect, why? To answer to this question; I believe the 10 first images are the one with the highest score therefore they had a huge impact on the final scores, even though the measure isn't necessarily high, still the prediction is going to be good because only good-predicted images were used within these 50. If random images were chosen, I wouldn't get this good predictions results and so on average I would have had a less precise measure in the end.

The Tailacc(t) accuracy increases as we look at the more top-ranked results (by increasing the value of t)

4 Task 3:

Using the TwoNetwork class, use two pre-trained networks resnet-18, one to handle the rgb and the other to handle the near infrared. Concatenate the high level features, and feed these features into a final linear layer.

```
class TwoNetworks(nn.Module):
    '''
    This class takes two pretrained networks,
    concatenates the high-level features before feeding these into
    a linear layer.

    functions: forward
    '''
    def __init__(self, pretrained_net1, pretrained_net2):
        super(TwoNetworks, self).__init__()

        _, num_classes = get_classes_list()
        self.pretrained_net1 = nn.Sequential(*list(pretrained_net1.modules())[:-1])
        self.pretrained_net2 = nn.Sequential(*list(pretrained_net2.modules())[:-1])
        self.fc = nn.Linear(1024, num_classes)
    def forward(self, inputs1, inputs2):
        inputs1 = inputs1[:, 0:3, :, :]
        inputs2 = inputs2[:, 3:4, :, :]
        outputs1 = self.pretrained_net1(input1)
        outputs2 = self.pretrained_net2(input2)
        output = self.linear(torch.cat((outputs1 , outputs2, self.fc), dim=1))
        return output
```

In order to see the results of TwoNetworks Class , this code should be run in *train_pytorch_in5400_studentversion1.py* :

```
# how we do with the TwoNestworks :
pretrained_net1 = models.resnet18(pretrained=True)
pretrained_net2 = models.resnet18(pretrained=True)
model = TwoNetworks(pretrained_net1=pretrained_net1, pretrained_net2=pretrained_net2)
model = model.to(config['device'])
someoptimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
somelr_scheduler = torch.optim.lr_scheduler.StepLR(someoptimizer, step_size=1, gamma=0.1, last_
```

5 Task 4:

Initializing the weights of the additional channel using kaimnghe, concatenating them to the original weights, using new weight as the weight for the first convolutional layer.

```
if weight_init is not None:
    current_weights = pretrained_net.conv1.weight
    new_weights = torch.zeros(64, 1, 7, 7)
    if weight_init == "kaiminghe":
        nn.init.kaiming_normal_(new_weights)
    weights = torch.cat((current_weights, new_weights), 1)
    pretrained_net.conv1.weight = nn.Parameter(weights)
```

I thank Mr. Ghadi al Hajj for his help and follow-up during this project.

References

[1] https://gombru.github.io/2018/05/23/cross_entropy_loss/

[2] https://towardsdatascience.com/importance-of-loss-function-in-machine-learning-eddaaec69519

[3] https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173

[4] https://medium.com/analytics-vidhya/how-to-add-additional-layers-in-a-pre-trained-model-using-