CV Project on Image Classification using Fastai.

Project by:

Maaz Ansari J002

Harsh Gupta J017

Riddhi Mehta J030

Table Of Contents

Introduction to Fastai:

Fastai is a deep learning library which provides practitioners with high-level components that can quickly and easily provide state-of-the-art results in standard deep learning domains, and provides researchers with low-level components that can be mixed and matched to build new approaches.

The fastai library simplifies training fast and accurate neural nets using modern best practices.

So fastai provides a single DataLoaders class which automatically constructs validation and training data loaders with these details already handled. This helps practitioners ensure that they do not make mistakes such as failing to include a validation set. Since the training set and validation set are integrated into a single class, fastai is able, by default, always to display metrics during training using the validation set.

In addition, fastai also allows to perform data augmentation using get_transforms() and normalization of data.

Transfer learning is critically important for training models quickly, accurately, and cheaply, but the details matter a great deal. Fastai automatically provides transfer learning optimised batch-normalization training, layer freezing, and discriminative learning rates.

In general, the library's use of integrated defaults means it requires fewer lines of code from the user to re-specify information or merely to connect components. As a result, every line of user code tends to be more likely to be meaningful, and easier to read.

According to the 2019 Kaggle ML & DS Survey (**https://www.kaggle.com/c/kaggle-survey-2019**), 10% of data scientists in the Kaggle community are already using fastai. Many researchers are using fastai to support their work.

Data:

The dataset images are 3-meter spatial resolution, and each is labelled with whether an oil palm plantation appears in the image (0 for no plantation, 1 for any presence of a plantation).

We have 768 satellite images of oil palm plantations and 2257 satellite images of no oil palm plantations.

You can have a look at the dataset here.

	
Example of a Satellite image with Oil palm plantation.	Example of a Satellite image with no Oil palm plantation.

Aim:

The aim of the project is to create a model that predicts the presence of oil palm plantations in satellite imagery.

Now, Why Oil Palms?

Deforestation through oil palm plantation growth represents an agricultural trend with large economic and environmental impacts. From shampoo to donuts and ice cream, oil palm is present in many everyday products—but many have never heard of it explicitly! Because oil palm grows only in tropical environments, the crop's expansion has led to deforestation, increased carbon emissions, and biodiversity loss, while at the same time providing many valuable jobs.

With the economic livelihoods of millions and the ecosystems of the tropics at stake, how might we work towards affordable, timely, and scalable ways to address the expansion and management of oil palm throughout the world?

Implementation Flow Chart:

1. Importing Libraries and Reading the Data
2. Specifying parameter values for fastai
3. Compiling images and loading it in ImageDataBunch function
4. Data Augmentation and Data normalization
5. Creating CNN Model and Plotting Learning Rate
6. Fitting one cycle with 5 epochs and Learning rate as 0.1
7. Confusion matrix of Validation Data
8. Saving the model
9. Unfreeze the CNN Layers
10. Saving the new model (If the accuracy is better)
11. Prediction on Test Data

Implementation (Code with Explanation):

Reading Train Data from specified path and listing its contents

```
[ ] labels = os.listdir('/content/drive/Project_Data_Train')
    print("No. of labels: {}".format(len(labels)))
    print("-----------------")
    for label in labels:
        print("{}, {} files".format(label, len(os.listdir('/content/drive/Project_Data_Train/'+label))))
```

```
No. of labels: 2
-----------------
0, 2257 files
1, 768 files
```

Specifying parameter values for "fastai"

```
[ ] path = '/content/drive/Project_Data_Train'
    size = 256
    bs = 256
```

Compiling images and loading it in ImageDataBunch function

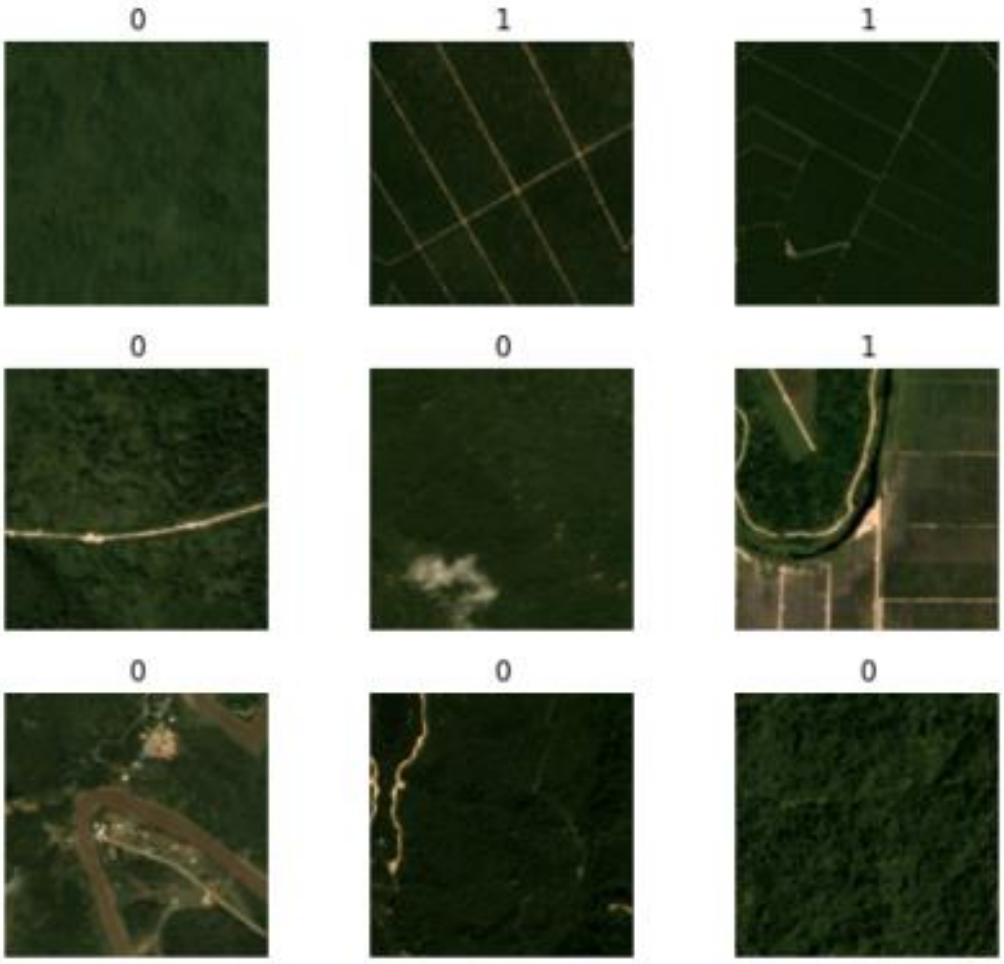
```
[ ] data = ImageDataBunch.from_folder(path, ds_tfms=get_transforms(do_flip=True, flip_vert=True),
                                      valid_pct=0.2, size=size, bs=bs)

[ ] data.normalize(imagenet_stats)
```

- Fastai is very handy when it comes to reading the images, data augmentation and data normalization. We will use the ImageDataBunch.from_folder() function to create an object that contains our image data.
- We don't even need to worry about train_test split, that is automatically split by Fastai, what we need to do is input the batch size for each iteration and the size of inputs.
- For a better model we can even add some data augmentation, get_transforms() gives us a basic set of data augmentations.
- Parameters do_flip and flip_vert vertically flips the images so that the angle of image does not matter.
- Valid_pct allows us to control the percent of train-test spilt we desire.
- We can normalize data (substract the mean, divide by std) for the GPU to work better, ImageNet stats helps us to perform the same. It basically decreases computation time and reduces heavy numbers into a more consistent and normalized format.

Lets have a look at our data. The two classes look really identical, lets see how well our model classifies it!

```
data.show_batch(rows=3, figsize=(7,6))
```



- We can make a CNN model in Fastai with the help of the function create_cnn.
- In create_cnn, we specify our data and, model you wish to use, metrics and callback function. In metrics we have used accuracy, we can even use error_rate, mean_squared_error, mean_absolute_error or root_mean_squared_error.
- Callback function ShowGraph returns a graph which is very useful to check whether the model is improving or not.

Plotting Learning Rate

We require the learning rate graph to visually decide which learning rate has to be used in order to train the images.

```
[ ] learner = create_cnn(data, models.resnet18, metrics=[accuracy], callback_fns=ShowGraph)
```

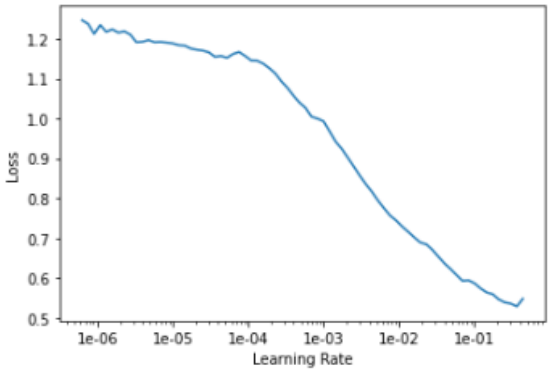
```
[ ] learner.lr_find()
    learner.recorder.plot()
```

66.67% [2/3 18:03<09:01]

epoch	train_loss	valid_loss	accuracy	time
0	1.166954	#na#		09:17
1	0.592619	#na#		08:45

37.84% [14/37 03:19<05:28 1.5724]

```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```

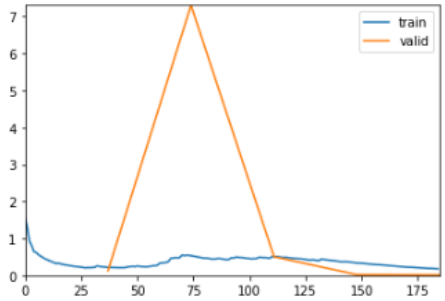


Fitting one cycle with 5 epochs.

- We have used the learning rate of = **1e-01** as we can see in the graph above.
- Since the data is heavy, and image sizes are also significantly large, we observe that every epoch takes an approx. 10 minutes.
- Overall, the model has trained for **50 minutes**.

```
[ ]  learner.fit_one_cycle(5, max_lr=slice(1e-1))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.223235	0.122588	0.988333	10:34
1	0.541890	7.306499	0.655000	09:35
2	0.503985	0.501260	0.986667	09:42
3	0.341435	0.017828	0.996667	09:37
4	0.175752	0.011502	0.996667	09:39

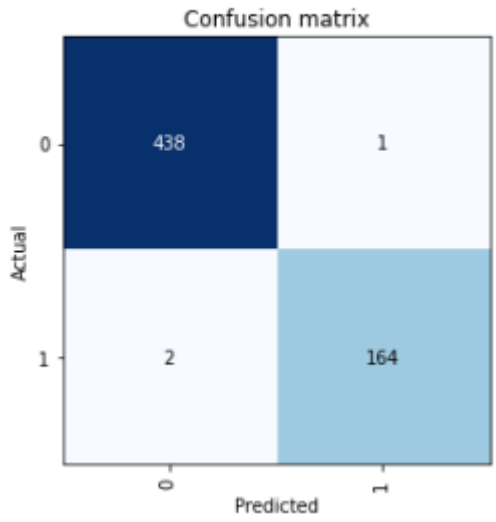


It took approximatiely 10minutes per epoch, nevertheless we can observe that in the fifth epoch our accuracy is 99.6% and validation loss as 0.011, which is pretty impressive!

To check our top losses, confusion matrix, wrong classifications and and to review our model, we use ClassificationInterpretation.from_learner().

```
interp = ClassificationInterpretation.from_learner(learner)
interp.plot_top_losses(3, figsize=(15,11))
```

```
[ ]  interp.plot_confusion_matrix()
```



As we can see in the confusion matrix, total only three images were misclassified.
In this image there are no oil palm plantations but our model predicted it the opposite.

In these two images our model predicted there are no oil palm plantations, which is not the case hence they are misclassified. Oil palm plantations are present only in a small area of the image which may have led to the model incorrectly classifying them.

If the accuracy of the model is not high or the loss is not low enough, then we can use the unfreeze() function to improve the model.
So the main difference is:

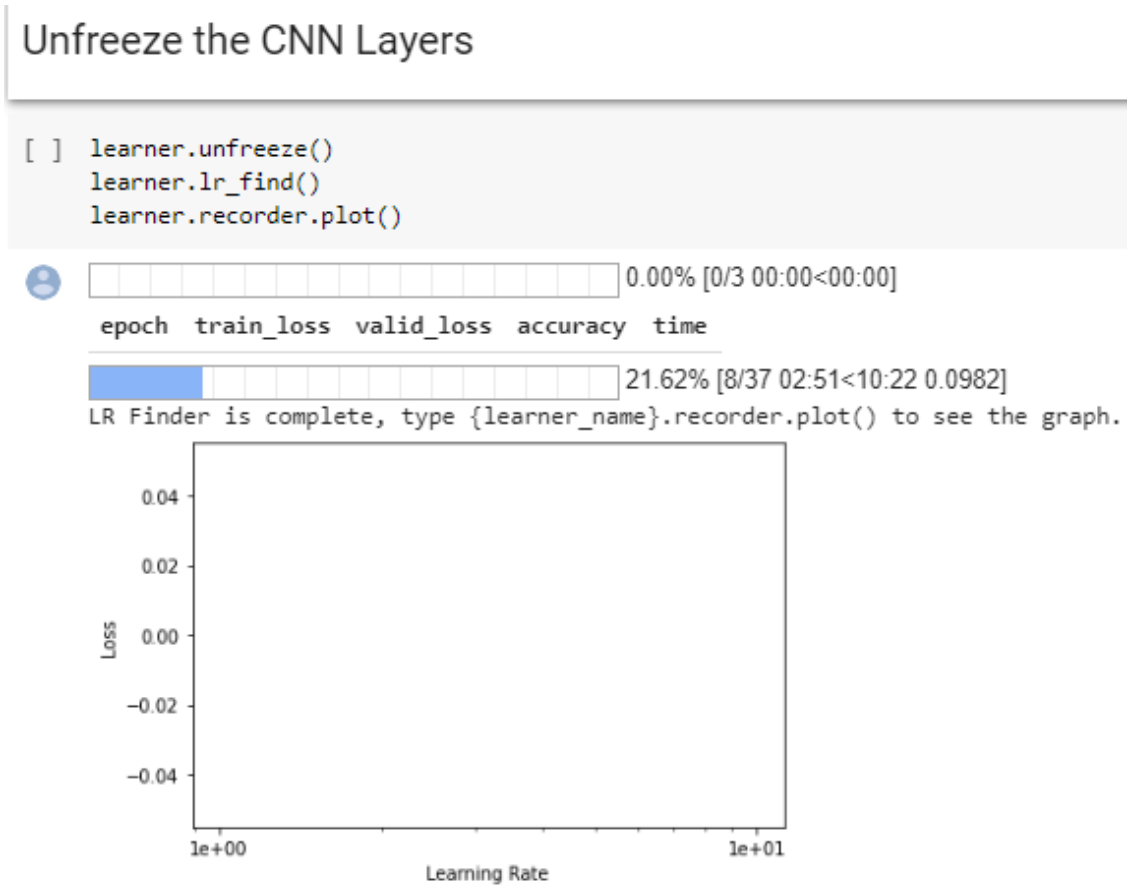
> Before Unfreezing

* Training = CNN Layers (Frozen) + FC Layers (Training)

> After Unfreezing

* Training = CNN Layers (Training) + FC Layers (Training)

But in our case the model already has an accuracy of 99.6% and the loss is 0.01, There is no need to unfreeze because the accuracy has already reached its peak, as well as the loss being so low of value 0.01 **cannot** be plotted on the graph. Hence, we do not even obtain the _NEW model weights_.



Creating a user defined function to predict new images and create confusion matrix of them.

```
[ ]  def predict_img(imagelist):

       y_pred = []
       y_true = []
       for img_path in imagelist:

         img = im.open_image(img_path)
         (pred_class,pred_label,__) = learner.predict(img)
         y_pred.append(int(pred_label))
         y_true.append(int(img_path.split('/')[-1].split('_')[0]))

       return y_pred, y_true
```

```
[ ]  y_pred,y_true = predict_img(pathlist)
     y_pred
```

```
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1]
```

```
[ ]  print("Confusion matrix for test dataset is as follows:", end = "\n\n")
     print(confusion_matrix(y_true, y_pred))
```

```
Confusion matrix for test dataset is as follows:

[[5 0]
 [0 5]]
```

We observe that the model is able to predict the test data with an accuracy of 100%.

Conclusion:

As shown in the results above, the model has an extremely high accuracy, both on training and testing. The classification problem was solved easily with the help of FastAI and it's supporting vision methods, without the need of creating a customised neural network. Data augmentation, splitting and processing were faster and easier with the help of FastAI. Hence, overall FastAI has eased the creation of the model and given a higher accuracy of predictions.

Benefits of FastAI include:

- Much less code to write for most common tasks.
- Very easy to implement.
- Normally faster to train.
- Yields higher accuracy.
- Easier to understand.

Limitations of FastaAI that jump to mind:

- Not much documentation
- Relies on pytorch, which doesn’t have such mature production (mobile or high scalability server) capabilities compared to tensorflow
- Pytorch doesn’t run on as many devices yet (e.g Google’s TPU)
- Not supported by as big an organization as tf
- Some parts still missing or incomplete (e.g object localization APIs)