Analysis of Image Classification Algorithms

Ahmed Shahabaz
U 8941-5490

I. Introduction

Image classification refers to the process of classifying input image through extracting features. Image classification on images taken in any unconstrained environment is a hard task. In this project we were given a train and test dataset which contained images from 19 classes. The images were taken in varying lighting conditions. Again the images were taken and merged from multiple data sources. Each class has at least 100 images. The Vessel has the least number of samples with 100 images. And the class with the most number of images is the Dog class with 180 images. We started by training some Deep Learning based models. Later we have shown the performance of classification for two non-Deep Learning methods/classifiers. For training and creating the deep learning architectures we have used *PyTorch* library in *Python*. And as for other approaches we will use the *sklearn* library in *Python*.

II. Performed Analysis

In this project our target was to do some analysis and comparison of different image classification algorithms while doing so the overarching goal of this project was to achieve higher accuracy on test dataset. So the performance of the algorithms was measured in terms of accuracy in the test dataset. We were given train and test dataset. Both of which contained 19 classes. The class distribution of train data of the original dataset is shown in figure [1(a)].

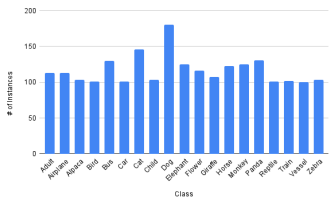
Image classification pipeline can be divided into three main steps:

- **Input pre-processing** : input normalization, resize, crop etc.
- **Feature extraction** : using CNN to get higher dimensional features.
- **Classification**: train a classifier on the extracted features.

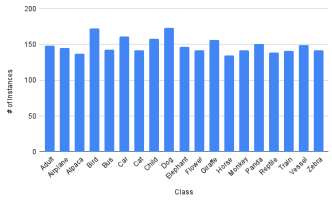
Our main work was on the last two steps of the image classification pipeline. That is coming up with a better feature extraction and then being able to train a classifier that is able to classify the the input image based on the extracted features fed/presented to it. So in terms of the feature extraction techniques and different classifier being used our approach can be divided into two parts: Deep Learning (DL) and Non-Deep Learning or Traditional approach. As for the Non-Deep Learning approaches we have used two: Support Vector Machine (SVM) and Random Forest.

We used the builtin functions available in *sklearn*. In case of Non-Deep Learning approaches we used the raw features that is the raw input (resized) image. We trained SVM for two types of kernels *RBF* and *Linear*. On the other hand Random Forest itself is a ensemble algorithm so we tried different number of ensemble ($n\_estimators$) values. We compared these methods in terms of accuracy in the test dataset. In case of non-DL approaches we didn't use any validation set rather we trained those algorithms on the whole set of original dataset. And at the end we reported only the test accuracy.

On the other hand for the DL approaches we trained a Convolutional Neural Network (as features extractor)based architecture with a Multi Layer Perceptron (MLP) as classifier. We got the based result using a pre-trained CNN while training only the classification layer instead of training the model end to end. Unlike in the case of non-DL approaches here we monitored validation loss to decide on the model performance and different training factors. But the model was never trained on the validation set. But as for the final performance measurement reported the test accuracy.



(a) Original dataset



(b) Balanced train set using torch.utils.data.WeightedRandomSampler

Fig. 1: Class instance distribution in the train dataset

As seen from the class instance distribution the original dataset was imbalanced. So in order to make the dataset balanced we used torch.utils.data.WeightedRandomSampler, a builtin functionality of PyTorch. But then we had to split the dataset into train and validation set as the original dataset given to us didn't contain any separate validation set. So we actually ended up using *torch.multinomial()* function of PyTorch to generate train set indices. The rest of the unused instances/indices of the original dataset were selected for validation set. Then using the *SubsetRandomSampler* the train and validation instances were sampled from the original dataset given the train and validation indices. Then during training the

validation loss was observed to decided on the different factors of the training such as overfitting, early stopping, comparison between algorithms before seeing the test dataset.

One potential problem with *WeightedRandomSampler* mechanism is that it addresses the issue of class imbalance through oversampling the of the classes with less number of instances. So that means the training algorithm might overfit easily to those classes. In order to make sure that doesn't happen we used data augmentation through different image transformations [2]. The image transformations made sure that each time the model sees a slightly transformed version of the original input data. This makes sure that the model learns different features of the same data through introducing different variations in the input data.

```
if mode == "train" or mode == "Train":
    transforms_list = [
    transforms.Resize((args.image_size)),
    transforms.CenterCrop(args.crop_size),
    transforms.RandomHorizontalFlip(),
    transforms.RandomVerticalFlip(),
    #transforms.RandomAutocontrast(),
    #transforms.RandomEqualize(),
    transforms.RandomApply(torch.nn.ModuleList([
        transforms.ColorJitter(),
        transforms.GaussianBlur(15),
        ]), p = 0.3),
    transforms.RandomRotation(20),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
    ]

else:
    transforms_list = [
    transforms.Resize((args.image_size)),
    transforms.CenterCrop(args.crop_size),
    transforms.ToTensor(),
    transforms.Normalize((0.485, 0.456, 0.406),(0.229, 0.224, 0.225))
    ]
```

Fig. 2: Set of image transformations

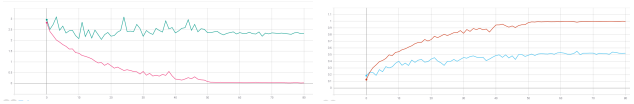
Other than using data augmentations techniques we used different regularization methods such as dropout, weight decay to address the issue of overfitting. We also monitored validation loss for implementing early stopping technique to stop training when the model overfits to the train data.

III. Results and Discussion

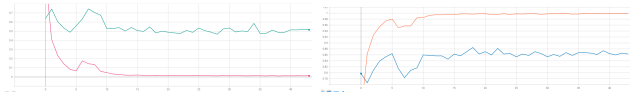
We first started with the DL approaches. Figure [3] shows the validation loss function comparison between ResNet18 [1] trained end to end with or with out imagenet pre-trained weights. In both the cases we can see that the model did overfit to the train data. Also from the loss graph in figure [3(a)] we see that after 10 epochs the validation loss kind of became stable or didn't change that much. So that gave an indication for using learning rate scheduler (changing learning rate along the way). Again from the figure [3(a)] we can see that the model overfitting to the train data. But we can see that the validation loss is much less or the accuracy is much better than in the case of training w/o imagenet weights. But seeing the model overfitting to the training data we can understand the need for a more complex model. Another take way from figure [3] is that, using pre-trained weights is better than training from scratch.

So after that we trained ResNet34, ResNet50, ResNet101 and VGG16 models. From different training approaches we found that training only the last layer or the classification layer of these models instead of training end to end. Figure [4] shows training vs validation loss and accuracy for different models trained in different ways (end to end figure [4(a),(c)] or only last layer figure [4(b),(d),(e)].

From figure [4] we see that training only the last layer ((figure [4(b)]) of ResNet34 [1] results in less overfitting than

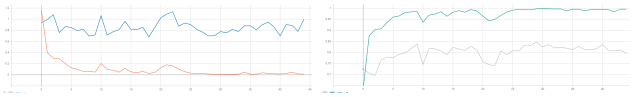


(a) Trained w/o initialization with imagenet pre-trained weights

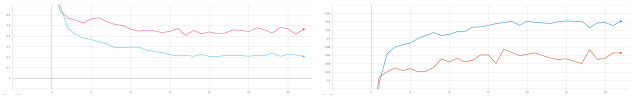


(b) Trained after initializing with imagenet pre-trained weights

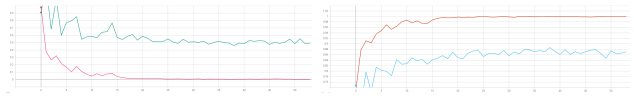
Fig. 3: Train (red line) vs Validation loss (left) & accuracy (right) graph of ResNet18 [1] training



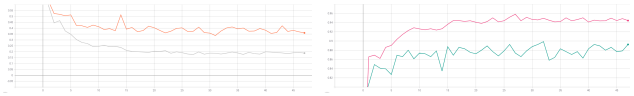
(a) ResNet34 [1] trained end to end



(b) ResNet34 [1] trained only the fully connected layer



(c) ResNet50 [1] trained end to end



(d) ResNet50 [1] trained only the fully connected layer

(e) ResNet101 [1] trained only the fully connected layer

Fig. 4: Train (red line) vs Validation loss (left) & accuracy (right) graph after initializing with pre-trained weights

training same model end to end (figure [4(a)]. We can make the same observation for ResNet50 [1] [4(c), (d)] as well. So for the ResNet101 [1] we trained only the last fully connected (dense) layer. Then we trained only the classifier layer of the VGG16 [2] architecture.

So far all the training approaches that we have talked so far used SGD optimizer. So now we experimented with ADAM optimizer. For that we trained the last layer of the VGG16 [2] architecture with ADAM optimizer. Figure [5] shows the

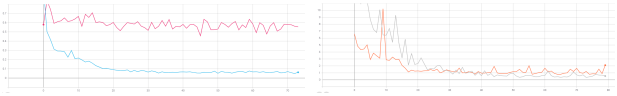


Fig. 5: Train (red line) vs Validation loss graph of the last layer of VGG16 [2] trained with SGD (left) & ADAM (right) optimizer

Model Architecture	Trained Layers	Optimizer	Learning Rate	Batch Size	LR. Scheduler	Weight Decay
VGG16	Classification	SGD	0.01	64	StepLR(optimizer, 15, 0.1)	1e-6
VGG16	Classification	ADAM	0.01	64	StepLR(optimizer, 15, 0.1)	1e-6
ResNet34	last fc.	SGD	0.01	128	StepLR(optimizer, 15, 0.1)	1e-6
ResNet50	last fc.	SGD	0.01	128	StepLR(optimizer, 15, 0.1)	1e-6
ResNet101	last fc.	SGD	0.01	128	StepLR(optimizer, 15, 0.1)	1e-6

TABLE I: Overview of the models used in ensemble

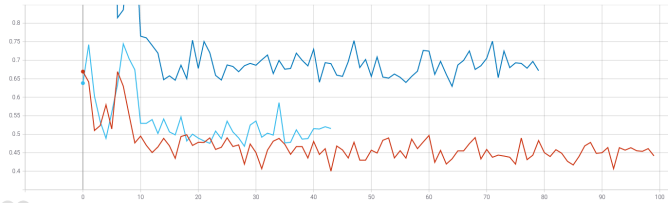


Fig. 6: Loss graph of ResNet18 [1] architecture trained with different batch size 32 (uppermost blue), 64 (middle) and 128 (lower red colored)

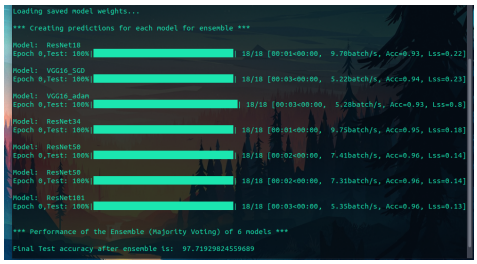


Fig. 7: Performance on Test data for each of the model individually along with the ensemble of them

comparison between the last of VGG16 trained with SGD and ADAM optimizer. From that we observe that the model least overfits to the train data when ADAM optimizer is used. But in the case of training with ADAM optimizer the validation loss doesn't go as low as in the case of SGD optimizer. So at the end we decided that SGD is best for our problem/project.

Then we experimented with different values of batch size. For that we trained ResNet18 architecture end to end with different batch size (figure [7]) where the model was initialized with imagenet pre-trained weights. We can see that model trained with batch size = 128 shows the lowest loss value. So we decided to train all the models with batch size = 128. But we had to train the VGG16 model with batch size 64 because of the memory limitation of our gpu.

At the end we took a ensemble of the 6 best (in terms of test accuracy) deep learning models to get the final prediction. We used a majority voting scheme for the ensemble method. Table [I] shows an overview of the training parameters that were used for ensemble. As we can see in the figure [7] that the ensemble method outperforms any single DL method.

As for non-DL based algorithms. We tried only two of them: Support Vector Machine (SVM) and Random Forest. As expected they were not as good as the DL based methods. We couldn't do much more analysis on the non-DL methods

Kernel	Test Accuracy
RBF	0.18070
Linear	0.1333

TABLE II: Performance of different SVM kernels

n_estimators	Test Accuracy
50	0.17543
100	0.21754
200	0.20350
300	0.22280
500	0.22807

TABLE III: Performance of RandomForest methods

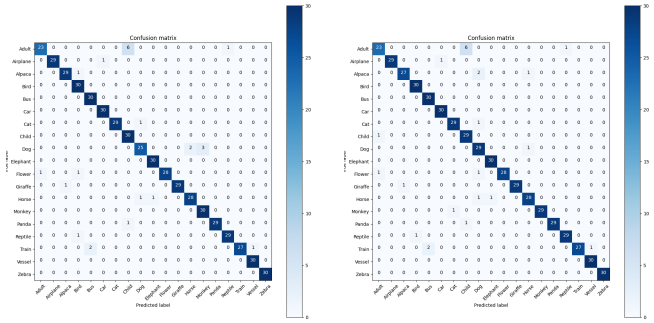
because it takes a lot of time to train on the full set of raw feature. One way to improve the time complexity of the non-DL methods would be to use PCA to reduce the feature space instead of using the whole set of raw features.

Figure [8] shows the confusion matrices of the Resnet50 model trained as described in table [I] on balanced (figure [8(a)]) and unbalanced (figure [8(b)]) dataset respectively. And the last entry of the (figure [8(c)]) show the confusion matrix for the ensemble of DL methods. All the confusion matrices that are shown in figure [8] are on test data. There is not much difference in classification performance between the model trained on balanced data and on unbalanced data. That is indicative of the fact that the original train data that we had was not too unbalanced as well as the fact that the dataset was pretty simple and small.

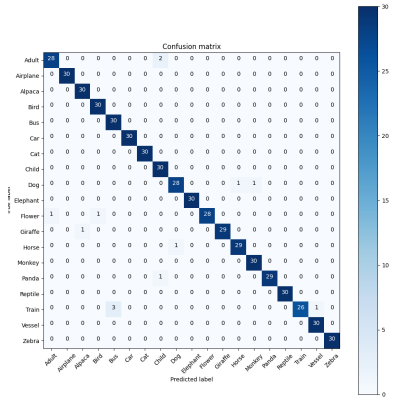
From the confusion matrices in figure [8] we can see that the models struggled mostly in classifying the data from "*Adult*" and "*Train*" class. The reason could be that the images in those two classes has a lot of variations among themselves in terms of features. For example adult is a too generalized class so is the class labeled train. There are a lot of different types of trains. As well there could be a lot of differences between two adults.

IV. Conclusion

From our analysis presented throughout this paper we see the superiority of the DL methods over the non-DL methods. Then the ensemble of DL methods is much better than any single DL method. But as we had a much simpler and small dataset we were able to train multiple DL methods. Which might not always be possible with much larger datasets. As it might take even a few days to complete training. Because of the simplicity and size of our dataset we were not able to train any of the well known model fully end to end. But as per our target we were able to get a good result on the test



(a) Confusion matrix of ResNet50 [1] trained on balanced (left) and unbalanced (right) dataset



(b) Confusion matrix of ensemble method

Fig. 8: Confusion matrices

(a) Examples of *Adult* class

(b) Examples of *Train* class

Fig. 9: Train (red line) vs Validation loss (left) & accuracy (right) graph after initializing with pre-trained weights

set and while doing so we were able to apply the learning of the course work into the project.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.