Ship Classification with a Deep Neural Network

Jens Kobler

February 5, 2025

Figure 1: Example Ship Image from dataset.

Contents

1 Main Objective of Analysis **2**

2 Desription of Dataset **2**

3 Data Cleaning and Feature Engineering **2**

4 Deep Learning model with different hyperparameters **2**
4.1 Setup 2
4.2 Results for 30 different hyperparemter setups 3

5 Flaws in model and Revisiting Analysis **3**

A Example Images **4**

B Code **5**

1 Main Objective of Analysis

The stakeholders are harbor owners and their harbors are going through digitalization. In the past, harbor staff would always look at the ocean and check for incoming ships and classify them. The stakeholders managed to install a ship recognition system, which is able to detect ships an makes a picture of them. However, those ships should be further classified using a Deep Neural Network. The main objective of this analysis is to train a neural network to correctly classify the ship images.

2 Desription of Dataset

The dataset was downloaded from kaggle, it is called "Game of Deep Learning: Ship datasets" (Link). The dataset consists of 6252 ship images of different size. Most of the images are RGB images. Some images are not RGB. A ship image example is illustrated in Figure 1. Five different ship classes exist, namely Cargo (1), Military (2), Carrier (3), Cruise (4) and Tankers (5). In Figure 2 the class ratio is illustrated in a pie chart.

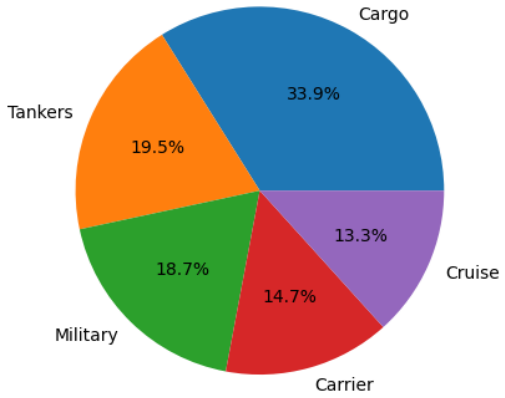


Figure 2: Ratio of ship categories in dataset.

3 Data Cleaning and Feature Engineering

The ship images which are not RGB images are dropped. This means around 100 images are dropped from the original 6252 ship images. The images are resized to one specific size. The category variable is subtracted by 1, since the deep learning model starts counting from 0.

4 Deep Learning model with different hyperparameters

4.1 Setup

A pretrained resnet model named resnet34 is taken. The images are resized to (100,100,3). As loss the cross entropy loss is used. The first 5000 images are used for training. The last 1000 images are used for testing. The batchsized is set to 50 and the data is shuffled.

4.2 Results for 30 different hyperparemter setups

In Figure 3 the final test accuracy is plotted for different learning rates (lr $\in [0.05, 0.01, 0.1]$) and epochs ($n_{epochs} \in [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]$). The test accuracy for the learning rates 0.005 and 0.01 are around 80% with number of epochs greater than 15. The green line represents the test accuracy for lr= 0.1. This learning rate is to high, the model is not able to learn a suitable representation.

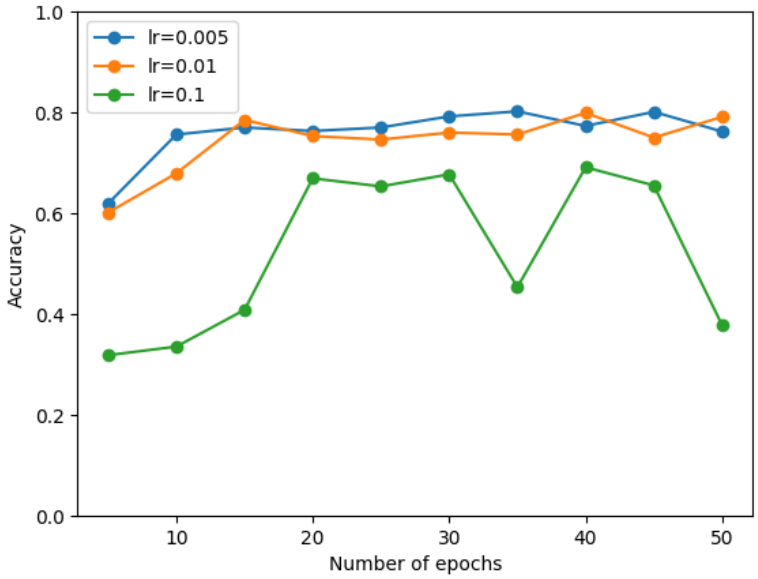


Figure 3: Results for different epochs and learning rates.

I would advice the stakeholders to use the model trained with lr=0.005 and number of epochs 30 for their harbor ship detection and classification system.

5 Flaws in model and Revisiting Analysis

A flaw of this model could be the training with unbalanced class inputs which is heavily drawn to the cargo ship class. One could balance those classes and train the neural network with the balanced dataset. In the analysis only the overall test accurcacy was examined. One could examine the accuracy per class. It would also be very interesting to study the test accuracy.

The model is able to achieve 80% test accuracy with a suitable hyperparameter choice. However, higher test accuracy results for example greater than 90% would be even better. An idea would be to try different loss functions with which the model is updated. One could also try a different pre-trained model. Batch size can be varied, as well as the shape of the resized images. Experimenting with different hyperparameter choices and trying different image pre-processing options, one could potentially achieve test accuracies greater than 90%.

Additional idea: In the training step the output vector of the model is not scaled to a probability vector (sum=1 and all values are positve) before the update step. For example, one could use the the softmax function (outputs = outputs.softmax(dim=1)) to scale the output vector of the model.

A Example Images

Figure 4: Cargo ship example.

Figure 5: Carrier ship example.

Figure 6: Military ship example.

Figure 7: Cruise ship example.

Figure 8: Tanker ship example.

B Code

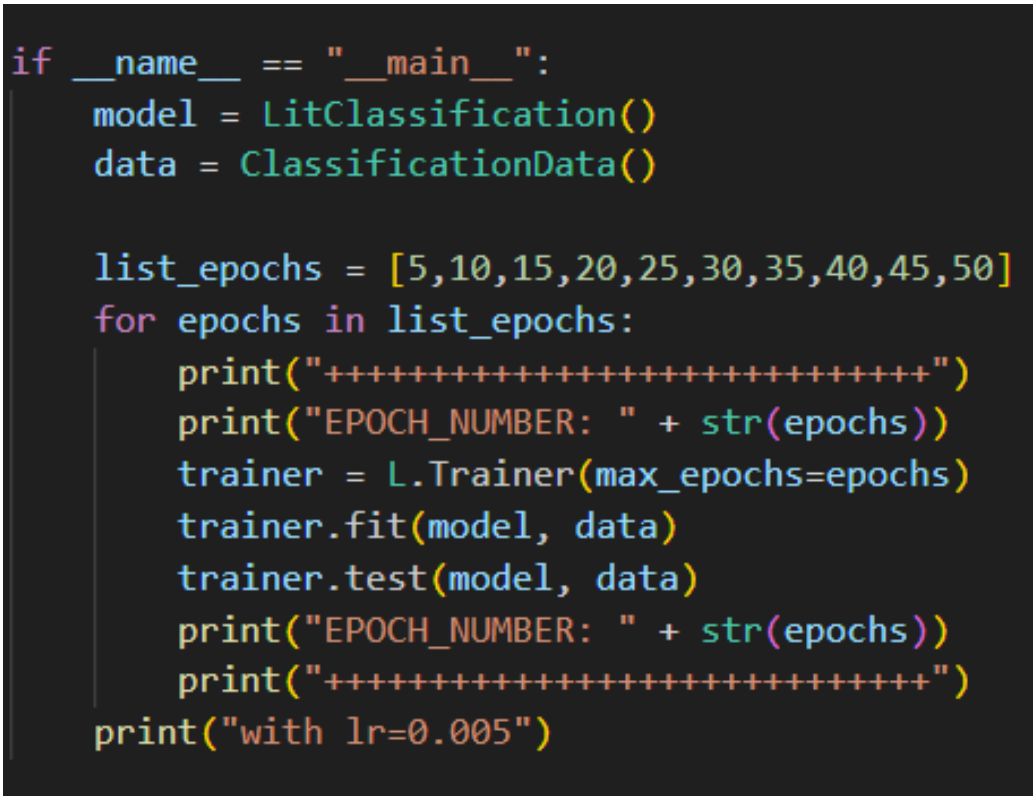
```python
import torch
from timm import create_model
from torchvision import transforms, datasets
import lightning as L
import os
import pandas as pd
from torch.utils.data import Dataset
from torchvision.io import read_image
from torchvision.transforms.functional import to_pil_image
import matplotlib.image as mpimg
from tqdm import tqdm
import torchvision

DEFAULT_TRANSFORM = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((100, 100)), # set size to 10 by 10
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

```python
class LitClassification(L.LightningModule):
    def __init__(self):
        super().__init__()
        print("Initialize LitClassification object...")
        self.model = create_model('resnet34', num_classes=5) # num_classes = 5
        self.loss_fn = torch.nn.CrossEntropyLoss()

    def configure_optimizers(self):
        return torch.optim.AdamW(self.model.parameters(),lr=0.005) # lr=0.005)  (1. lr = 0.1 2. 0.01, 3.0.005)

    def training_step(self, batch):
        images, targets = batch
        outputs = self.model(images)
        loss = self.loss_fn(outputs, targets)
        self.log("train_loss", loss)
        return loss

    def test_step(self, batch, batch_idx):
        x, y = batch
        # implement your own
        out = self.model(x)
        labels_hat = torch.argmax(out, dim=1)
        test_acc = torch.sum(y == labels_hat).item() / (len(y) * 1.0)
        # log the outputs!
        self.log_dict({'test_acc': test_acc})
```

```python
class CustomImageDataset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        print("Initialize CustomImageDataset object...")
        full_data = pd.read_csv(annotations_file)

        # delete images which are not RGB images
        drop_rows_idx = []
        for i in tqdm(range(len(full_data))):
            specific_image_file_name = full_data["image"].iloc[i]
            # print(specific_image_file_name)
            full_path = img_dir + specific_image_file_name
            img = mpimg.imread(full_path)
            if len(img.shape) != 3:
                drop_rows_idx.append(i)

        full_data.drop(drop_rows_idx, inplace=True)
        full_data.reset_index(inplace=True)
        full_data.drop(["index"], axis=1, inplace=True)
        full_data["category"] = full_data["category"] - 1 # very important to get categories between 0 and 4
        self.img_labels = full_data.head(5000) # select only 100 images
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image_tensor = read_image(img_path)
        image_pil = to_pil_image(image_tensor)
        label = self.img_labels.iloc[idx, 1]
        # print(label)
        # print(idx)
        if self.transform:
            image = self.transform(image_pil)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

```python
class CustomImageDataset_Test(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None, target_transform=None):
        print("Initialize CustomImageDataset object...")
        full_data = pd.read_csv(annotations_file)

        # delete images which are not RGB images
        drop_rows_idx = []
        for i in tqdm(range(len(full_data))):
            specific_image_file_name = full_data["image"].iloc[i]
            # print(specific_image_file_name)
            full_path = img_dir + specific_image_file_name
            img = mpimg.imread(full_path)
            if len(img.shape) != 3:
                drop_rows_idx.append(i)

        full_data.drop(drop_rows_idx, inplace=True)
        full_data.reset_index(inplace=True)
        full_data.drop(["index"], axis=1, inplace=True)
        full_data["category"] = full_data["category"] - 1 # very important to get categories between 0 and 4
        self.img_labels = full_data.tail(1000) # select only 100 images
        self.img_dir = img_dir
        self.transform = transform
        self.target_transform = target_transform

    def __len__(self):
        return len(self.img_labels)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 0])
        image_tensor = read_image(img_path)
        image_pil = to_pil_image(image_tensor)
        label = self.img_labels.iloc[idx, 1]
        # print(label)
        # print(idx)
        if self.transform:
            image = self.transform(image_pil)
        if self.target_transform:
            label = self.target_transform(label)
        return image, label
```

```
class ClassificationData(L.LightningDataModule):

    def train_dataloader(self):
        print("Run Train Data Loader")
        train_dataset = CustomImageDataset(annotations_file="../data/archive/train/train.csv", img_dir="../data/archive/train/images/", transform=DEFAULT_TRANSFORM)
        return torch.utils.data.DataLoader(train_dataset, batch_size=50, shuffle=True,  num_workers=1, persistent_workers=True) #,

    # def test_dataloader... :)
    def test_dataloader(self):
        print("Run Test Data Loader")
        test_dataset = CustomImageDataset_Test(annotations_file="../data/archive/train/train.csv", img_dir="../data/archive/train/images/", transform=DEFAULT_TRANSFORM)
        return torch.utils.data.DataLoader(test_dataset, batch_size=50, shuffle=True,  num_workers=1, persistent_workers=True) #,
```



```
if __name__ == "__main__":
    model = LitClassification()
    data = ClassificationData()

    list_epochs = [5,10,15,20,25,30,35,40,45,50]
    for epochs in list_epochs:
        print("++++++++++++++++++++++++++++++")
        print("EPOCH_NUMBER: " + str(epochs))
        trainer = L.Trainer(max_epochs=epochs)
        trainer.fit(model, data)
        trainer.test(model, data)
        print("EPOCH_NUMBER: " + str(epochs))
        print("++++++++++++++++++++++++++++++")
    print("with lr=0.005")
```