Machine Learning II Final Report

Hsueh-Yi Lu

Master of Data Science, The George Washington University

DATS 6203_11: Machine Learning II

Dr. Admir Jafari

December 13, 2022

Introduction

For our project, we are working on breast cancer image classification using various deep learning models and different training skills. Invasive ductal carcinoma (IDC) is one of the most common types of breast cancer. It's malicious and able to form metastases which makes it especially dangerous. Often a biopsy is done to remove small tissue samples. Then a pathologist must decide whether a patient has IDC, another type of breast cancer or is healthy. In addition, sick cells need to be located to find out how advanced the disease is, and which grade should be assigned. This must be done manually and is a time-consuming process. Furthermore, the decision depends on the expertise of the pathologist and his or her equipment. Therefore, deep learning could be of great help to automatically detect and locate tumor tissue cells and to speed up the process. To exploit the full potential, one could build a pipeline using massive amounts of tissue image data of various hospitals that were evaluated by different experts. This way one would be able to overcome the dependence on the pathologist which would be especially useful in regions where no experts are available.

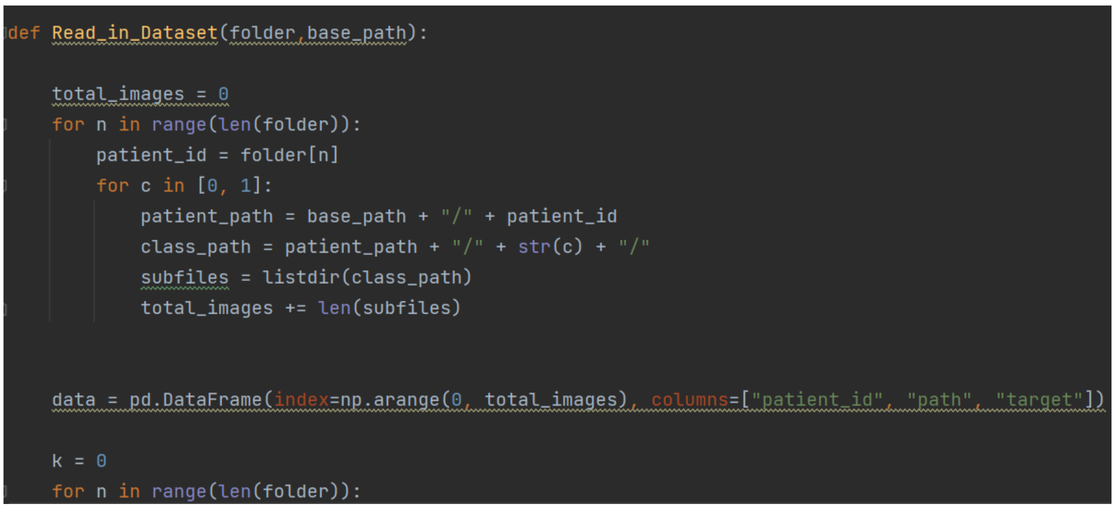
In the project, we used two pre-trained convolutional neural networks to detect IDC in the tissue slice images. To avoid the overfitting issue and improve the model performance, we implemented two methods. One is data augmentation method and the other is learning rate search method. In the data augmentation, methods in the transformer were used to increase the diversity of the images and a general adversarial network was implemented to standardize the image. In the training process, the cyclical learning rate (CLR) search method was used when training the Resnet18 and the VGG16 framework. The best performance is the accuracy 0.85 on the test set on Resnet18 with CLR search.

The major works can be divided into several pieces: Dataset pre-process, Classification model design, improving training of model, and testing model then make analysis of results.

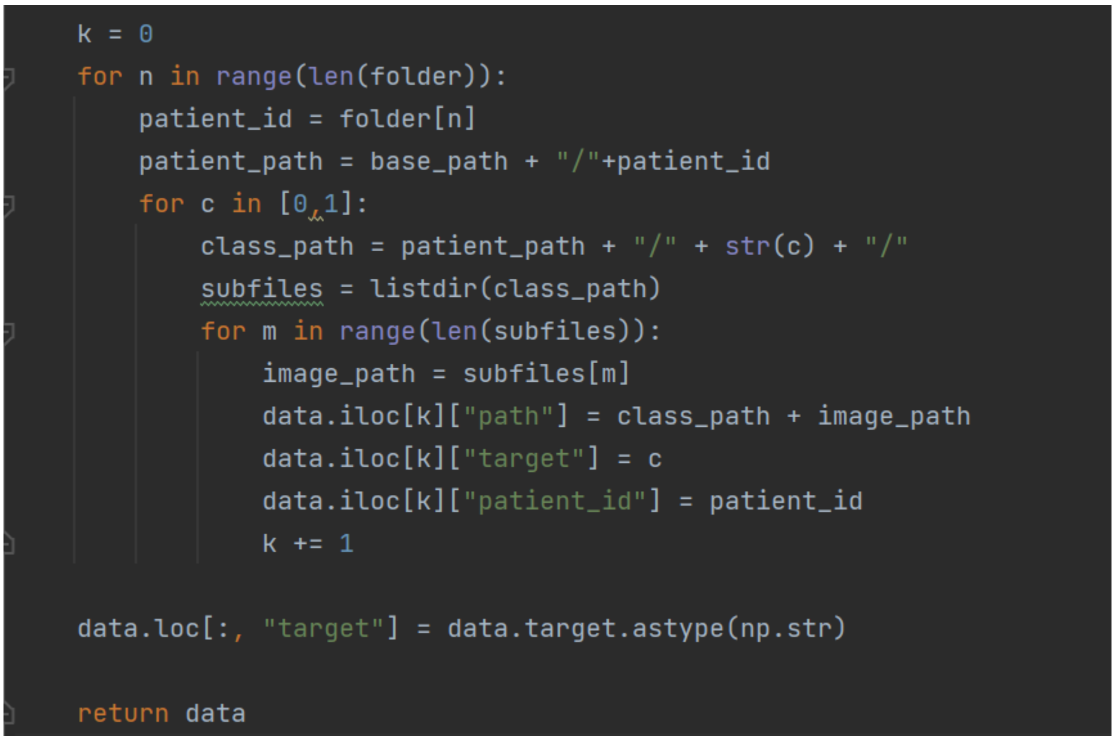
Works I have done

The major works can be divided into several pieces: Dataset pre-process, Classification model design, testing model, and plot analysis of results. Most works are worked together by the whole team.

For the very beginning of the project, there is not much data cleaning work to do. Transforming the image data into a loadable dataset that fit the pre-trained model is our major work. By the same time, doing some simple EDA to look inside the dataset.

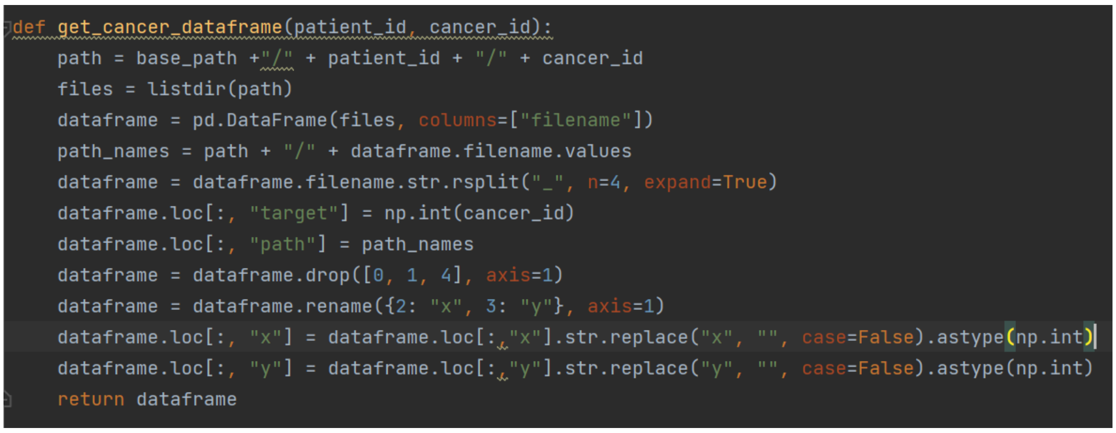


```python
def Read_in_Dataset(folder,base_path):

    total_images = 0
    for n in range(len(folder)):
        patient_id = folder[n]
        for c in [0, 1]:
            patient_path = base_path + "/" + patient_id
            class_path = patient_path + "/" + str(c) + "/"
            subfiles = listdir(class_path)
            total_images += len(subfiles)


    data = pd.DataFrame(index=np.arange(0, total_images), columns=["patient_id", "path", "target"])

    k = 0
    for n in range(len(folder)):
```



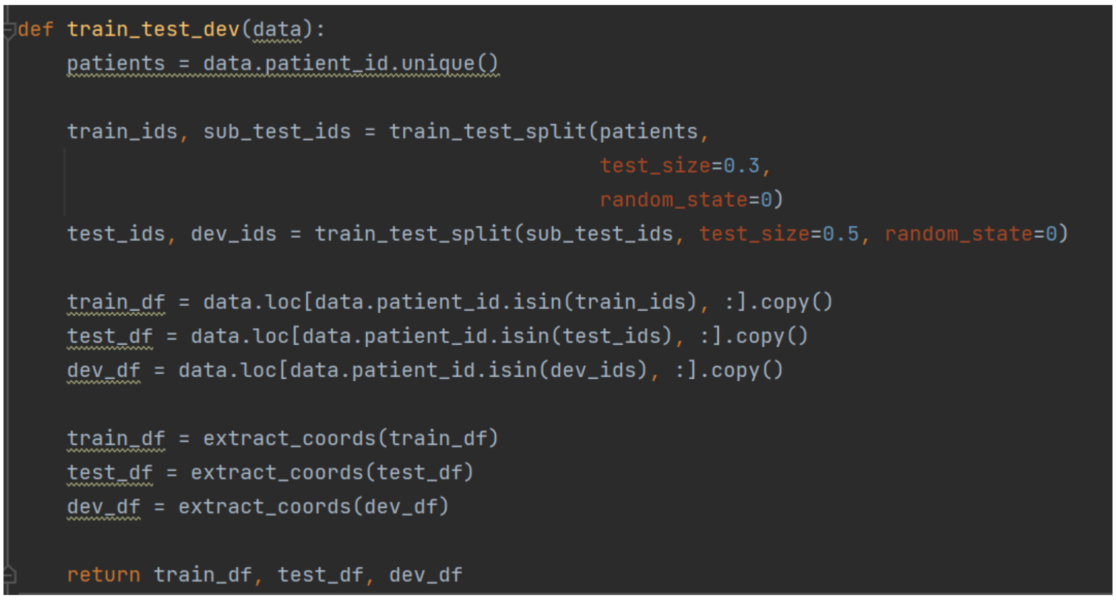
```python
    k = 0
    for n in range(len(folder)):
        patient_id = folder[n]
        patient_path = base_path + "/"+patient_id
        for c in [0,1]:
            class_path = patient_path + "/" + str(c) + "/"
            subfiles = listdir(class_path)
            for m in range(len(subfiles)):
                image_path = subfiles[m]
                data.iloc[k]["path"] = class_path + image_path
                data.iloc[k]["target"] = c
                data.iloc[k]["patient_id"] = patient_id
                k += 1

    data.loc[:, "target"] = data.target.astype(np.str)

    return data
```

Figure 1 Buildup original dataset



```python
def get_cancer_dataframe(patient_id, cancer_id):
    path = base_path +"/" + patient_id + "/" + cancer_id
    files = listdir(path)
    dataframe = pd.DataFrame(files, columns=["filename"])
    path_names = path + "/" + dataframe.filename.values
    dataframe = dataframe.filename.str.rsplit("_", n=4, expand=True)
    dataframe.loc[:, "target"] = np.int(cancer_id)
    dataframe.loc[:, "path"] = path_names
    dataframe = dataframe.drop([0, 1, 4], axis=1)
    dataframe = dataframe.rename({2: "x", 3: "y"}, axis=1)
    dataframe.loc[:, "x"] = dataframe.loc[:,"x"].str.replace("x", "", case=False).astype(np.int)
    dataframe.loc[:, "y"] = dataframe.loc[:,"y"].str.replace("y", "", case=False).astype(np.int)
    return dataframe
```

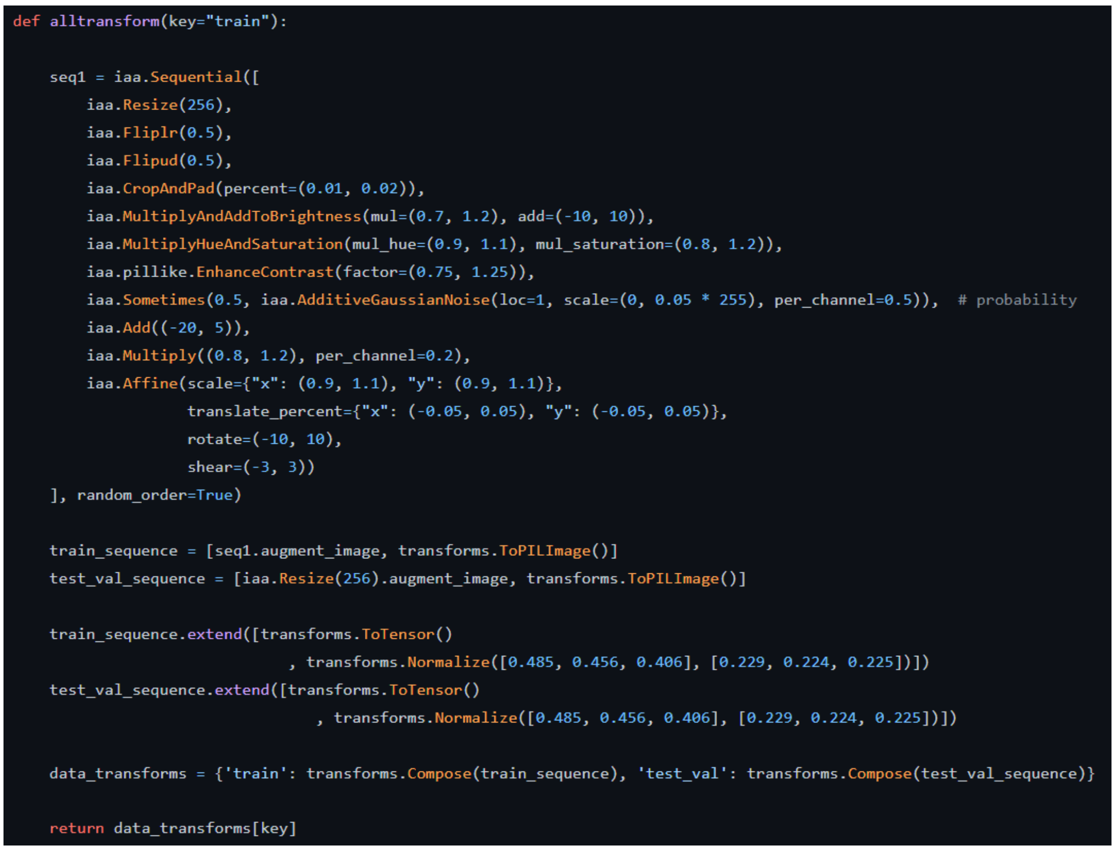
Figure 2 Part of data preprocessing



```python
def train_test_dev(data):
    patients = data.patient_id.unique()

    train_ids, sub_test_ids = train_test_split(patients,
                                               test_size=0.3,
                                               random_state=0)
    test_ids, dev_ids = train_test_split(sub_test_ids, test_size=0.5, random_state=0)

    train_df = data.loc[data.patient_id.isin(train_ids), :].copy()
    test_df = data.loc[data.patient_id.isin(test_ids), :].copy()
    dev_df = data.loc[data.patient_id.isin(dev_ids), :].copy()

    train_df = extract_coords(train_df)
    test_df = extract_coords(test_df)
    dev_df = extract_coords(dev_df)

    return train_df, test_df, dev_df
```

Figure 3 Part of data preprocessing

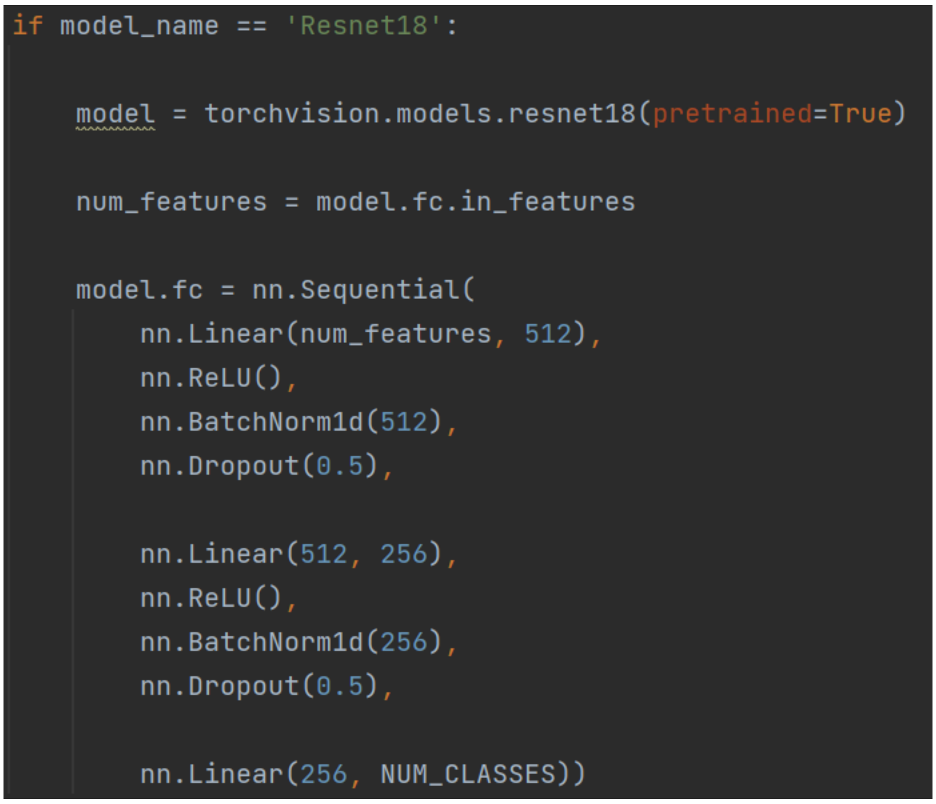
Before passing the data into our training loop, we added some data augmentation base on the imgaug package.



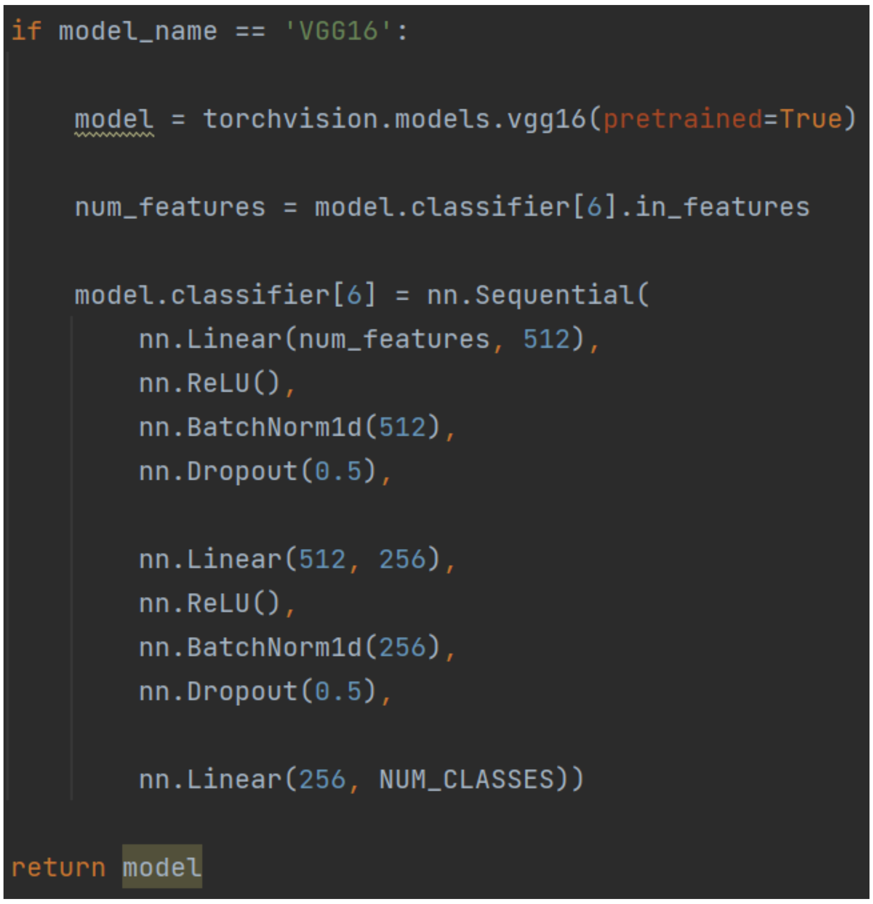
```python
def alltransform(key="train"):

    seq1 = iaa.Sequential([
        iaa.Resize(256),
        iaa.Fliplr(0.5),
        iaa.Flipud(0.5),
        iaa.CropAndPad(percent=(0.01, 0.02)),
        iaa.MultiplyAndAddToBrightness(mul=(0.7, 1.2), add=(-10, 10)),
        iaa.MultiplyHueAndSaturation(mul_hue=(0.9, 1.1), mul_saturation=(0.8, 1.2)),
        iaa.pillike.EnhanceContrast(factor=(0.75, 1.25)),
        iaa.Sometimes(0.5, iaa.AdditiveGaussianNoise(loc=1, scale=(0, 0.05 * 255), per_channel=0.5)),  # probability
        iaa.Add((-20, 5)),
        iaa.Multiply((0.8, 1.2), per_channel=0.2),
        iaa.Affine(scale={"x": (0.9, 1.1), "y": (0.9, 1.1)},
                   translate_percent={"x": (-0.05, 0.05), "y": (-0.05, 0.05)},
                   rotate=(-10, 10),
                   shear=(-3, 3))
    ], random_order=True)

    train_sequence = [seq1.augment_image, transforms.ToPILImage()]
    test_val_sequence = [iaa.Resize(256).augment_image, transforms.ToPILImage()]

    train_sequence.extend([transforms.ToTensor()
                           , transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])
    test_val_sequence.extend([transforms.ToTensor()
                              , transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

    data_transforms = {'train': transforms.Compose(train_sequence), 'test_val': transforms.Compose(test_val_sequence)}

    return data_transforms[key]
```

Figure 4 Data augmentation

After applying augmentation, we redesigned the pretrained models, VGG-16 and RESNET18, and let the output layer fit our expectation.



```
if model_name == 'Resnet18':

    model = torchvision.models.resnet18(pretrained=True)

    num_features = model.fc.in_features

    model.fc = nn.Sequential(
        nn.Linear(num_features, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Dropout(0.5),

        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Dropout(0.5),

        nn.Linear(256, NUM_CLASSES))
```



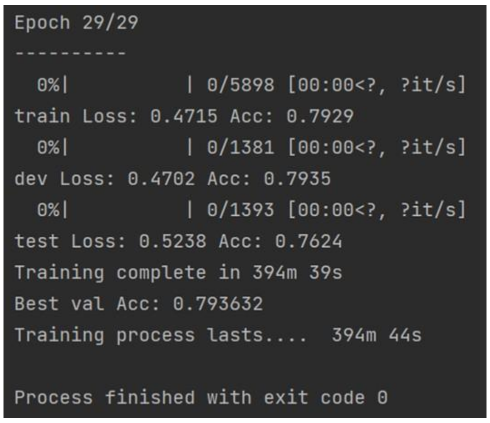
```
if model_name == 'VGG16':

    model = torchvision.models.vgg16(pretrained=True)

    num_features = model.classifier[6].in_features

    model.classifier[6] = nn.Sequential(
        nn.Linear(num_features, 512),
        nn.ReLU(),
        nn.BatchNorm1d(512),
        nn.Dropout(0.5),

        nn.Linear(512, 256),
        nn.ReLU(),
        nn.BatchNorm1d(256),
        nn.Dropout(0.5),

        nn.Linear(256, NUM_CLASSES))

return model
```

Figure 5 Model design

Next step, I was responding to the VGG-16 training and testing. Finding the best parameters set for fitting our data into VGG-16 which got the best accuracy.

Results

Eventually, while setting epoch = 30, learning rate = 0.006, batch size = 32, Cyclical learning rate = True, the VGG-16 model got a 0.79 accuracy.



```
Epoch 29/29
----------
  0%|          | 0/5898 [00:00<?, ?it/s]
train Loss: 0.4715 Acc: 0.7929
  0%|          | 0/1381 [00:00<?, ?it/s]
dev Loss: 0.4702 Acc: 0.7935
  0%|          | 0/1393 [00:00<?, ?it/s]
test Loss: 0.5238 Acc: 0.7624
Training complete in 394m 39s
Best val Acc: 0.793632
Training process lasts....  394m 44s

Process finished with exit code 0
```

Figure 6 Training result of VGG-16

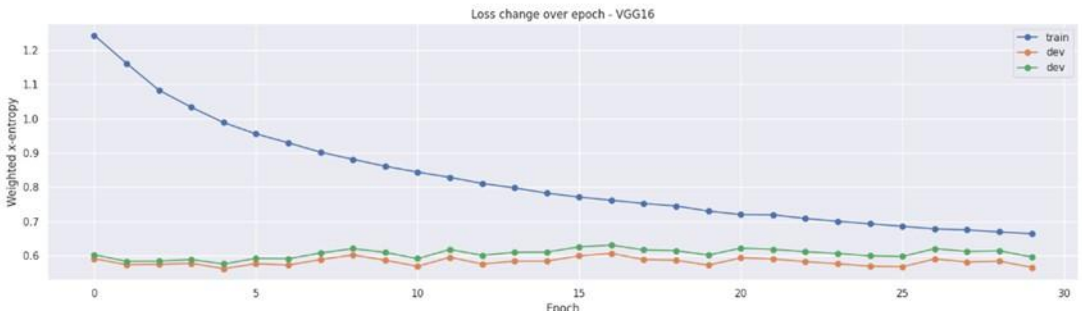


Figure7 Loss change over epoch

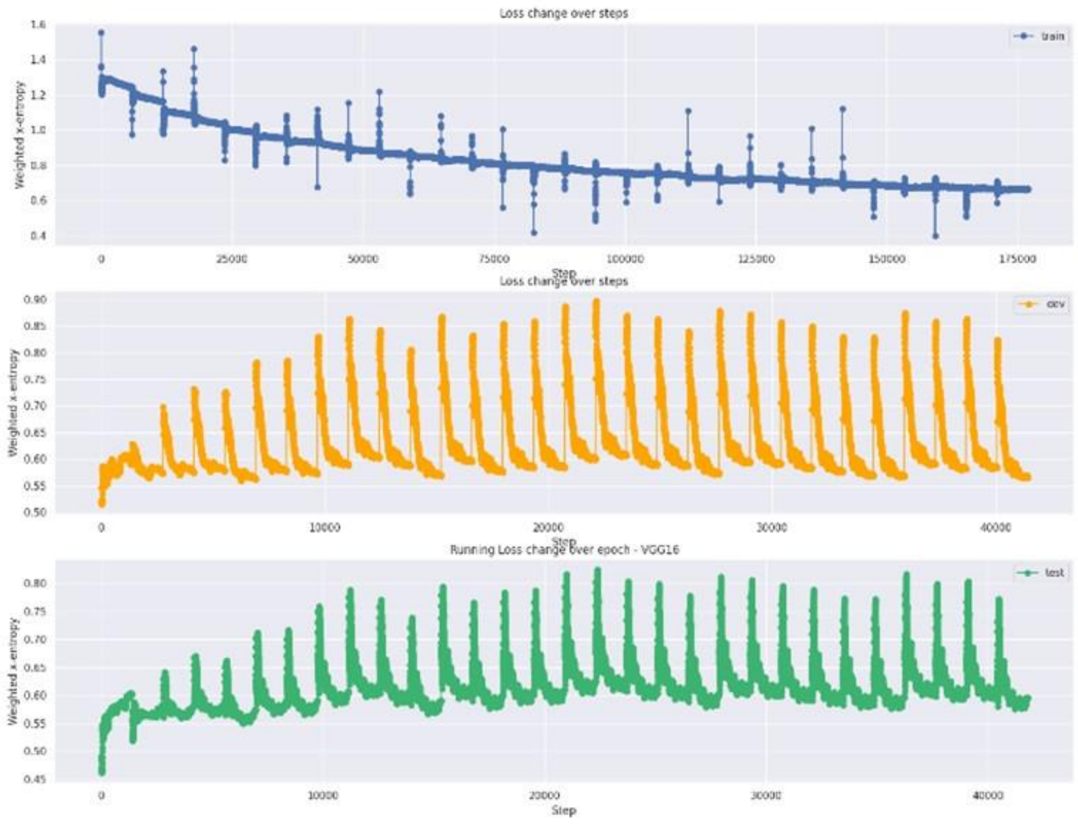


Figure 8 Loss change oversteps

For the final conclusion we got our best four experiment, which are RESNET18 and VGG-16 with CLR on or off. The final model we selected is the RESNET18 with applying CLR skill, getting a 0.85 accuracy rate and having a shortest training time.

Model	CRL	Validation Accuracy	Training Time
ResNet18	Yes	0.85%	265m51s
ResNet18	No	0.77%	270m41s
VGG16	Yes	0.79%	390m44s
VGG16	No	0.73%	393m43s

Figure 9 Four models we tried for our project

Summary

The whole process of dealing with this project is not too difficult, however, the major problem I faced is scheduling the time of the model training. Training a model such large really takes time and each time I changed the parameters I need to train the model again, which probably take six hours. Hence, scheduled the training time properly that match the due date is the thing I should pay attention to next time.

Percentage of codes: 70 %

Reference

[1]Convolutional neural networks: an overview and application in radiology: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9#Sec18

[2]Breast Cancer Notebook:https://www.kaggle.com/code/allunia/breast-cancer#Exploratory-analysis-

[3]Breast Histopathology Images: https://www.kaggle.com/datasets/paultimothymooney/breast-histopathology-images

[4]Cyclic Learning rate :https://towardsdatascience.com/super-convergence-with-cyclical-learning-rates-in-tensorflow-c1932b858252

[5]ResNet and VGG:https://stats.stackexchange.com/questions/280179/why-is-resnet-faster-than-vgg

[6] sklearn.utils.class_weight.compute_class_weight https://scikit-learn.org/stable/modules/generated/sklearn.utils.class_weight.compute_class_weight.html