CS 760: Machine Learning Fall 2020

COVID-19 Diagnosis based on CT Scans

Authors: Yuchen Zeng, Zhuoyan Xu

We applied several machine learning methods on the scanned CT chest images to explore the. relationship between positiveness of COVID-19 and chest CT.

We considered several competitive machine learning approaches, including Logistic Regression, K Nearest Neighbors, Naive Bayes, Decision Tree, Random Forest, Support Vector Machine and Convolutional Neural Network. We trained these models with raw image data and make predictions. We implemented the transfer learning on our own dataset. We used ResNet-18 architecture and personalized our own fully connection layer at the end. We froze the parameters in previous layers] only trained the personalized layers. In our result, we have SVM achieve the best performance with test accuracy 89%. The ResNet-18 has the second high test accuracy of 81.33%.

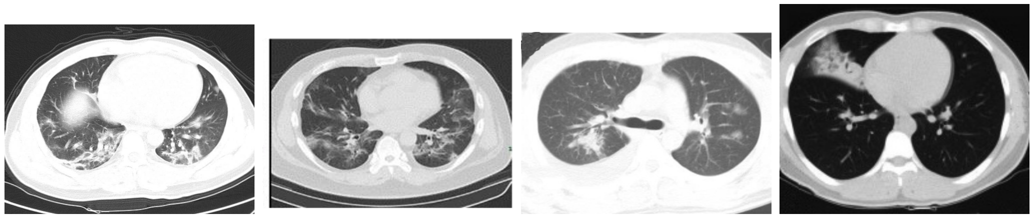
To combine the feature extraction advantage of convolutional neural networks and the interpretation of other machine learning methods, we used simple CNN to process images and transfer them into vectors representing the image characteristics. Then we applied several machine learning models to the image features. The advantage of this idea is instead of using raw image pixel value for classification, we implemented our machine learning algorithms on feature vectors extracted by pre-trained Convolutional Neural Network layers. This can reduce the feature dimension significantly, hence reduce the time and space complexity, free up more resources for more analysis. We aim to improve the accuracy of prediction and explore the explanation and interpretation as well.

We make use of the high accuracy from neural networks along with the interpretation from other machine learning models to better help make decisions and explain the relationship between COVID results and scanned images from a different perspective. We hope our model provides some insights into potential applications of machine learning methods in this specific COVID-19 situation.

1 Introduction

After China reported its first case of COVID-19 in December 2019, COVID-19 spread rapidly and became a global issue of public health since January 2020. According to the New York Times, 55.4 million people were tested positive for COVID-19, and 1.33 million people died as a result. COVID-19 is caused by the coronavirus SARS-CoV-2. The elderly and people with serious basic medical diseases (such as heart disease, lung disease or diabetes) seem to be more likely to have more serious complications from COVID-19. It is believed that COVID-19 is mainly spread through close contact between people, including people who are physically close to each other (within about 6 feet). People who are infected but have no symptoms can also spread the virus to other people.

As specific drugs for COVID-19 are not available, there is an urgent need to detect the disease early and isolate the infected person immediately. A single-slice of Computed tomography (CT) contains enough clinical information for accurate diagnosis decision-making. Bilateral changes in the chest CT scans can be observed from the clinical reports of infected people[1]. Due to its high sensitivity, chest CT has been used as an alternative tool to detect COVID-19 infection[2].



(a) The CT scan of a COVID-19 patient. (b) The CT scan of a COVID-19 patient. (c) The CT scan of a healthy person. (d) The CT scan of a healthy person.

Figure 1: Examples of CT scans from COVID-CT dataset.

This work is to develop a classifier for accurate diagnosis of COVID-19 based on the CT scans. We train the classifier based on three popular machine learning methods: random forest, SVM and Neural network. Since neural network is the most commonly used method for image processing, we train two Neural network models: simple convolutional neural network and ResNet-18 to perform COVID-19 CT classification. Out of consideration of computation complexity, we use pre-trained ResNet-18 to improve the efficiency of ResNet-18. Finally, we perform numerical evaluations of the selected methods on the CT dataset, comparing and contrasting their performance of both accuracy and efficiency.

The highest accuracy we obtain is 89%, which is achieved by SVM classifier. The accuracy of our SVM classifier is higher than the baseline method *self-Tran+DenseNet-169* which is proposed by He et al.[3].

Contributions of this paper are summarized as follows. The rest of the paper is organized as follows. Section 2 discusses the related work on COVID-19 diagnosis based on CT scans. Section 3 introduce the dataset we analyze in details. We present the basic ideas of the selected methods Section 4, describe the experiment setting, report and discuss the empirical results in Section 5. We summarize the conclusions of this work, and discuss interesting future follow-up work in Section 6. The implementation details are provided in Appendix.

2 Related work

As coronavirus caused a global epidemic problem that spread quickly, there has been a fair amount of work in diagnosis of COVID-19[3, 4, 5, 6, 7, 1]. Diagnosis of COVID-19 is typically associated with symptoms of pneumonia, computed tomography (CT) scans[3, 7], and chest X-ray[4, 5, 6]. Our work is closely related to He et al.[3], Xu et al.[1] and Singh et al.[7] which also used CT scans to perform diagnosis of COVID-19. Specifically, we use the same dataset[8] as He et al.[3].

He et al.[3] combined many popular models in deep learning with transfer learning to train the data. He et al. proposed *Self-Trans*, a self-supervised transfer learning approach where contrastive self-supervised learning is integrated into transfer learning process to adjust the network weights pretrained on source data. The highest accuracy 86% is achieved by combining Self-Tran and DenseNet-160.

Xu et al.[1] evaluated two convolutional neural networks (CNN) three-dimensional classification models. Ont was ResNet-based network and another model was based on ResNet-based network structure by concatenating the location attention mechanism in the full-connection layer. It achieves an overall accuracy of 86.7%.

Singh et al.[7] used a CNN whose initial parameters of CNN are tuned using multiobjective differential evolution (MODE). The highest accuracy of the proposed approach achieves above 92%.

Ardakani et al.[9] implemented several well-known CNN architectures such as AlexNet, VGG, ResNet to predict on CT scans. The best performance is achieved by ResNet101. They demonstrated in their study that the ResNet-101 can be considered as a promising model to characterize and diagnose COVID-19 infections.

In contrast, our work is not restricted to deep learning methods, but also consider other competitive machine learning approaches. We consider other benchmark machine learning methods: Random Forest and Support Vector Machine. We try to improved the accuracy of prediction and find explanation and interpretaion as well. We compare these methods in the Section 5.

3 Dataset

Data provided by Zhao et al.[8] has 746 CT images, containing clinical findings of COVID-19 from 216 patients. The images in the dataset consist of 349 CT scans that are positive for COVID-19 and 397 CT scans are negative for COVID-19. Figure 1 contains some sample images from the dataset. The size of images are different. The minimum width is 115 pixels, and the minimum height is 61 pixels.

4 Approach

In this section, we will describe which approaches we utilize to train our data and make classification.

4.1 Random forest

Random forests or random decision forests are one of the most widely used machine learning algorithms for regression and classification. Random forests are essentially a *bootstrap*, or *bagging* of *decision trees.*

Suppose the data matrix $\boldsymbol{X} = (\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top)^\top \in \mathbb{R}^{n \times d}$. A decision tree is a flowchart-like tree structure where the branch represents a decision rule, and each leaf node represents the outcome. A decision tree makes predictions based on series of questions. In practice, we train the decision tree by choosing the most informative feature via mutual information or other criterion in each internal node, and split the data according to this feature.

However, decision trees is sensitive to the data. Thus they can be quite biased, and tend to overfit. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Random forests generally outperform decision trees.

4.2 Neural network

The idea of neural networks comes from the biological brains and neurons. Each neuron receives information from several previous neurons. In the human body, eyes and ears receive information, then the neurons transfer the information to the brain to make decisions. A node without an activation function is known as a perceptron.

One layer consists of multiple nodes. A standard neural network contains several layers: one input layer, one output layer, and several hidden layers. The input layer receives the feature vectors from each observation, while the output layer outputs the prediction probability of each class of label, and the hidden layer contains the intermediate results. The number of nodes in the input is the number of training samples in each batch, the number of nodes in the output layer is the number of labels, the number of nodes in hidden layers and the number of hidden layers is determined by users based on the empirical performance. During the training period, the weights and bias are updated to minimize the selected objective function such as cross entropy via stochastic gradient descent.

A convolutional neural network adds several other layers especially for image processing. An image contains pixels represented by value. A convolutional layer can extract certain features in an image, such as eyes, mouth, ears of a dog. The word convolution comes from the functional analysis, it has an integral expression that can be used for computing the sum of two random variables. The convolution in CNN conducts the similiar steps, it has several feature detectors, each detector walk through the image and compute the sum of the product of values stored in corresponding pixels.

We train the CT datasets with two CNN models: simple CNN and ResNet.

4.2.1 Simple CNN

We build one simple CNN model with three convolutional layers to train the data. To avoid the large parameter dominate the updates, we use the batch normalization in two convolutional layers. The max-pooling layers are introduced to help with local invariance. They introduce no parameters. The architecture of this model is shown as Figure 2.

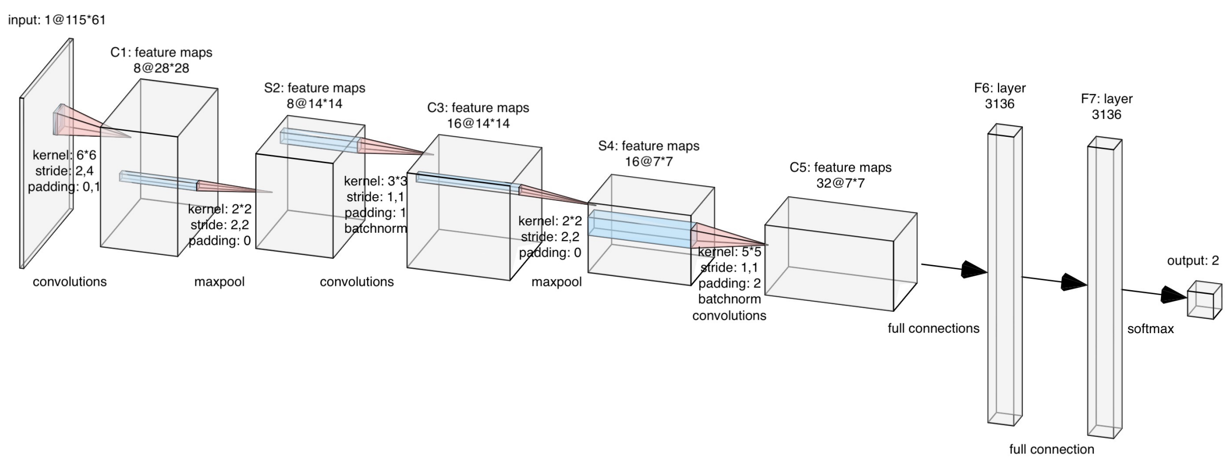


Figure 2: Architecture of the simple CNN we trained.

4.2.2 Transfer Learning + ResNets

The accuracy of the simple CNN we build is 70.67%, which is far less than the accuracy achieved by the baseline method He et al.[3]. We need a CNN model with a deeper architecture to improve the accuracy of classification. Deeper architecture significantly improves the accuracy over training set, with a cost of computation complexity. However, the heavy computation burden may make these model infeasible. We use transfer learning and ResNets to solve this issue.

ResNets is a CNN architecture proposed by He et al.[10]. The key idea of ResNets is that it allows to skip connections such as layers that are not useful. Therefore, ResNets allow us to implement significantly deep architectures. Transfer learning[11] is a technique proposed to mitigate the computation burden. The basic idea is to store the knowledge gained in one training task and apply it to another. The knowledge in fearture extraction layers of a well-trained model will help to extract features of an image on a new dataset. The features include the ears or mouths of a dog or cat in simple classification tasks. We stored the parameters in convolutional layers of a trained model and assigned them to our constructed models. We froze the parameters in those layers in training task and only update the last few fully connection layers.

4.3 Support vector machines (SVMs)

Support vector machines (SVMs) are considered as one of the best "out-of-box" classifier. Before SVM was proposed, perceptron is one of the most popular classification method. However, perceptron need high computation budget, and may have multiple solutions. The motivation of SVM is to solve the problems perceptron have when data is linearly separable.

Let $\boldsymbol{X} = (\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top)^\top \in \mathbb{R}^{n \times d}$ denote the sample matrix, where $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ are individual samples. $\boldsymbol{y} = (y_1, \ldots, y_n)$ are the labels, where $y_i \in \{0, 1\}$ for $i = 1, \ldots, n$. The training of a SVM classifier involves finding a hyperplane

$$\mathcal{H}_{\boldsymbol{\beta},\beta_0} = \{i \in \{1, \ldots, n\} : y_i(\langle \boldsymbol{x}_i, \boldsymbol{\beta} \rangle + \beta_0) \geq 1\}, \ \|\boldsymbol{\beta}\| < m,$$

to separate the training samples from two different class with the largest margin m [12]. To maximize the margin m:

$$m = \min_{i=1,\ldots,n} \operatorname{dist}(\boldsymbol{x}_i, \mathcal{H}_{(\tilde{\boldsymbol{\beta}},\tilde{\beta}_0)}).$$

However, the requirement that the data is linear separable have been too strong, leading to many possible relaxations, e.g. soft-margin SVM. Soft-margin SVM state a preference for margins that classify the training data correctly, but soften the constraints to allow for non-separable data with a penalty $(\varepsilon_1, \ldots, \varepsilon_n) \in \mathbb{R}^n$ proportional to the amount by which the example is misclassified. Thus the optimization problem becomes:

$$\begin{aligned} &\min_{\boldsymbol{\beta},\beta_0,\varepsilon_1,\ldots,\varepsilon_n} \frac{\|\boldsymbol{\beta}\|^2}{2} + C\sum_{i=1}^n \varepsilon_i \\ \text{s.t. } &y_i(\langle \boldsymbol{x}_i, \boldsymbol{\beta} \rangle + \beta_0) \geq 1 - \varepsilon_i \qquad i = 1, \ldots, n \\ &\varepsilon_i \geq 0 \qquad\qquad\qquad\qquad\quad i = 1, \ldots, n \end{aligned}$$

In practice, C is determined by cross validation.

A more advanced tool is kernel, which transforms the data to be linear separable. Suppose the transformation $\psi : \mathbb{R}^d \to$ Hilbert space H transforms the data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ to linear separable data $\psi(\boldsymbol{x}_1), \ldots, \psi(\boldsymbol{x}_n)$, then the primal optimization problem becomes:

$$\begin{aligned} &\min_{\boldsymbol{\beta} \in H, \beta_0 \in \mathbb{R}} \frac{\|\boldsymbol{\beta}\|_H^2}{2} + C\sum_{i=1}^n \varepsilon_i \\ \text{s.t. } &y_i(\langle \psi(\boldsymbol{x}_i), \boldsymbol{\beta} \rangle_H + \beta_0) \geq 1 - \varepsilon_i \quad i = 1, \ldots, n \\ &\varepsilon_i \geq 0 \qquad\qquad\qquad\qquad\qquad\quad i = 1, \ldots, n \end{aligned}$$

The dual problem is

$$\begin{aligned} &\max_{\boldsymbol{\lambda} \in \mathbb{R}^n} -\frac{1}{2}\sum_i \sum_j \lambda_i \lambda_j \langle \psi(\boldsymbol{x}_i), \psi(\boldsymbol{x}_j) \rangle_H + \sum_{i=1}^n \lambda_i \\ \text{s.t. } &\sum_{i=1}^n \lambda_i y_i = 0, \ \lambda_i \geq 0 \qquad\qquad\qquad\qquad i = 1, \ldots, n. \end{aligned}$$

With kernel $K = \langle \psi(\cdot), \psi(\cdot) \rangle_H$, the dual problem can be represented as

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^n} \; & -\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{i=1}^n \lambda_i \\ \text{s.t.} \; & \sum_{i=1}^n \lambda_i y_i = 0, \; \lambda_i \geq 0 \qquad\qquad i = 1, \ldots, n. \end{aligned}$$

Actually, given K, ψ is unnecessary for us to solve the dual problem. In practice, we select the kernel K and then solve the dual problem to get SVM classifier.

4.4 Simple CNN as feature extractor

We also consider using simple CNN to process images and transfer them into vectors representing the image characteristics, then we use other machine learning methods in previous sections on the feature vectors. The main idea is similar to transfer learning, the convolutional layer is suitable for extracting features in images. Ideally, applying previous machine learning methods on these vectors can give higher accuracy and better interpretation than applying them on raw images.

We use the CNN architecture in 4.2.1 as feature extractor, we add one layer with 10 nodes right before the output layer. Once we finish the training, we feed all images to the architecture and get the intermediate result outputted by the second to last layer. Then we have the new dataset with each observation has one corresponding feature vector.

5 Experiments

5.1 Main result on raw images

We resize all images to size 115×61, and divide them into training set and testing set at a ratio of 9:1. The packages we use are listed below:

- file reading: `pandas, skimage, shutil, os`;
- data processing: `pandas, numpy, random, sklearn, PIL`;
- benchmarking: `time`;
- plot: `matplotlib`;
- classifer: `sklearn, torch, torchvision`.

We provide more details of how we use these packages in Appendix. We train the methods we introduced in last section, and the settings are described below.

Random Forest: We flatten the image arrays to vectors of length 7015. Our random forests generate 100 decision trees, whose sample size is the same as the original sample size but the samples are drawn with replacement. Our criteria of choosing features in each internal node is Gini impurity. Gini impurity is also known as the total decrease in node impurity. This is how much the model fit or accuracy decreases when you drop a variable. The test accuracy of our random forest is around 80%.

Simple CNN: The learning rate, batch size and number of epochs we select empirically are 0.05, 8 and 20, respectively. The objective function we use here is cross entropy. All the activation functions except the output layer are ReLU. The test accuracy we obtain from simple CNN is 70.67%.

Transfer Learning+ResNet: We use a pre-trained ResNet-18[13] and only train the last hidden layer with activation function ReLu and output layer. To avoid overfitting, we drop feature maps with a probability of 0.5. The accuracy of the training set and test set are 82.116% and 81.33%, respectively, indicating that there is almost no overfitting.

Support Vector Machine: We flatten the image matrices to vectors of length $115 \times 61 = 7015$. We use 3-fold cross-validated grid search to determine the kernel from two alternative kernels, radial basis function kernel $\exp\left(-\gamma\|\boldsymbol{x} - \boldsymbol{x'}\|^2\right)$ and linear kernel $\boldsymbol{x}^\top \boldsymbol{x'}$, the $\gamma \in \{0.001, 0.0001\}$ in radial basis function kernel and the regularization coefficient $C \in \{1, 10, 100, 1000\}$. Therefore, we choose radial basis function kernel with $\gamma = 0.001$, $C = 10$. The hyperparameters we obtain are $C = 10, \gamma = 0.001$, and selected kernel is radial basis function kernel. The test accuracy our SVM classifier we obtain is 89%.

We summarize the empirical performance of the selected approaches as Table 2. Both SVM and Random Forest achieves 100% accuracy on training set. However, the test accuracy of random forest is only 77%, which is far less than 100%, thus we can conclude that the random forest suffers from serious over-fitting. The highest accuracy is achieved by SVM, which is 89%, higher than the baseline method He et al.[3]. SVM also stand out for its computation efficiency. The execution time of SVM is only 0.87 minutes, far less than another competitive model: ResNet-18, which takes 12.58 minutes to train the model.

Model	Train Accuracy	Test Accuracy	Execution time (min)
SVM	**100%**	**89%**	0.87
Random Forest	**100%**	77%	**0.04**
Simple CNN	81.222%	70.67%	4.61
ResNet-18	82.116%	81.33%	12.58
He et al.[3]	-	86%	-

Table 1: Comparison of SVM, random forest, simple CNN, ResNet-18 and the baseline method He et al.[3].

5.2 Results on extracted feature by simple CNN

We use the idea in 4.4, we apply several machine learning methods on the extracted feature vectors, we also search for the best parameters for each model and make predictions. We divided the datasets to 5 fold and compute the mean cross validation accuracy. The results are shown below:

Model	Train Accuracy	Test Accuracy	K-fold accuracy
Logistic Regression	99%	59%	97.52%
Decision Tree	98.14%	60%	97.37%
Random Forest	98.45%	57%	97.83%
K Nearest Neighbors	98.61%	58%	98.45%
Naive Bayes	98.60%	57%	98.45%
SVM	98.45%	63%	98.76%

Table 2: Comparison of the baseline method on extracted features.

The SVM gave the best test accuracy so we draw the hyperplane on the 2 dimension plot to show the region of data in 2 classes:

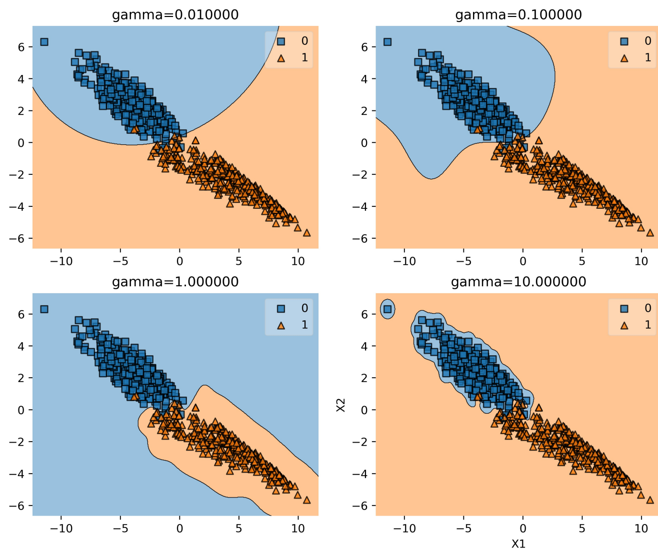


Figure 3: SVM with different γ in Gaussian kernel

The plot shows a fairly clear hyperplane to separate the two regions. We conclude the SVM can separate the data well on a higher dimensional space.

6 Conclusions and future work

We utilize the high accuracy from neural network along with the interpretation from machine learning models to better help make decisions and explain the relationship between COVID results and scanned images from a different perspective.

To summarize, SVM is the best model among the selected models to diagnose COVID-19, with the highest test accuracy of 89% and high computation efficiency. The ResNet architecture has the second high test accuracy, which can be improved if we tuned the layers and nodes to make them more suitable for the data.

The model on raw images works pretty well on the test dataset. However, the models on feature vectors have high train accuracy but lower test accuracy. The main reason is the potential overfitting problem when we fed the extracted features to our model. The CNN extract the main features but neglect some other important factors of the CT images. The focus on minor features may lose the overall characteristics of the images, we need to be careful when we use this idea in image classification.

In this work, the methods with raw dataset have generally better performance than those on extracted features. In our future work, we can explore more on how to use CNN architecture extract features well and combine them with our other machine learning models.

We will also focus on the feature importance generated by machine learning methods. We will explore which feature can best represent the CT image and help distinguish the Covid and Non-Covid results.

References

[1] Xiaowei Xu, Xiangao Jiang, Chunlian Ma, Peng Du, Xukun Li, Shuangzhi Lv, Liang Yu, Yanfei Chen, Junwei Su, Guanjing Lang, Yongtao Li, Hong Zhao, Kaijin Xu, Lingxiang Ruan, and Wei Wu. Deep learning system to screen coronavirus disease 2019 pneumonia, 2020.

[2] Xingzhi Xie, Zheng Zhong, Wei Zhao, Chao Zheng, Fei Wang, and Jun Liu. Chest ct for typical coronavirus disease 2019 (covid-19) pneumonia: Relationship to negative rt-pcr testing. *Radiology*, 2020.

[3] Xuehai He, Xingyi Yang, Shanghang Zhang, Jinyu Zhao, Yichen Zhang, Eric Xing, and Pengtao Xie. Sample-efficient deep learning for covid-19 diagnosis based on ct scans. *medRxiv*, 2020.

[4] Şaban Öztürk, Umut Özkaya, and Mücahid Barstuğan. Classification of coronavirus (covid-19) from x-ray and ct images using shrunken features. *International Journal of Imaging Systems and Technology*.

[5] Terry Gao. Chest x-ray image analysis and classification for covid-19 pneumonia detection using deep cnn. *medRxiv*, 2020.

[6] Asmaa Abbas, Mohammed Abdelsamea, and Mohamed Gaber. Classification of covid-19 in chest x-ray images using detrac deep convolutional neural network. *medRxiv*, 2020.

[7] Dilbag Singh, Vijay Kumar, Vaishali, and Manjit Kaur. Classification of covid-19 patients from chest ct images using multi-objective differential evolution–based convolutional neural networks. *European Journal of Clinical Microbiology & Infectious Diseases*, 39(7):1379–1389, 2020.

[8] Jinyu Zhao, Yichen Zhang, Xuehai He, and Pengtao Xie. Covid-ct-dataset: a ct scan dataset about covid-19. *arXiv preprint arXiv:2003.13865*, 2020.

[9] Ali Abbasian Ardakani, Alireza Rajabzadeh Kanafi, U Rajendra Acharya, Nazanin Khadem, and Afshin Mohammadi. Application of deep learning technique to manage covid-19 in routine clinical practice using ct images: Results of 10 convolutional neural networks. *Computers in Biology and Medicine*, page 103795, 2020.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[12] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg, 1995.

[13] Mujadded Al Rabbani Alif, Sabbir Ahmed, and Muhammad Hasan. Isolated bangla handwritten character recognition with convolutional neural network. pages 1–6, 12 2017.

A Appendix: Data processing Python source code

```python
# pandas: plot the accuracy results, read files, save the data, etc
import pandas as pd

# os: make sure the code can be executed for all operating systems
import os

# random: draw random samples
```

```python
import random

# imread: read and display images
from skimage.io import imread

# train_test_split: create validation set
from sklearn.model_selection import train_test_split

# pyplot: plot
import matplotlib.pyplot as plt

# Image: process images such as resize/clip the images
from PIL import Image

# shutil: remove an entire directory
import shutil

# load the images' names and label, and save them into a dataframe
# use filter to skip the flies such as DS_Store and other caches
covidImgs = list(filter(lambda x: len(x.split(".")) > 1, os.listdir(os.path.join("dataset",
    "CT", "CT_COVID"))))
healthImgs = list(filter(lambda x: len(x.split(".")) > 1, os.listdir(os.path.join("dataset",
     "CT", "CT_NonCOVID"))))
# 1: True; 0: False
ctLabel = pd.DataFrame({"Image": covidImgs + healthImgs, "Covid": [1]*len(covidImgs) + [0]*
    len(healthImgs)})
ctLabel

# create the training set and test set
train_name, test_name, train_y, test_y = train_test_split(ctLabel.Image, ctLabel.Covid,
    test_size = 0.1)
# create the validation set
train_name, val_name, train_y, val_y = train_test_split(train_name, train_y, test_size =
    0.1)

def resizeImage(folder):

    """
    Resize all the images in the folder to (115, 61).

    Parameter
    ---------
    folder: the folder to save the resized images: "train", "test" and "validation".

    Return
    ------
    a list of image arrays
    """

    if folder == 'train':
        x, y = train_name, train_y
    elif folder == 'test':
        x, y = test_name, test_y
    else:
        x, y = val_name, val_y

    imgs = []
    new_dir = os.path.join("dataset", "CT", folder)

    # remove previous data, recreate the train/val/test dataset
    if os.path.exists(new_dir) and os.path.isdir(new_dir):
        shutil.rmtree(new_dir)
        os.makedirs(new_dir)

```

```
    for name, covid in zip(x, y):
        original_folder = "CT_COVID" if covid else "CT_NonCOVID"
        img_path = os.path.join("dataset", "CT", original_folder, name)
        img = Image.open(img_path).convert('RGB').resize((115, 61)).convert('L')
        new_path = os.path.join(new_dir, name)
        img.save(new_path, "JPEG", optimize=True)
        img.close()

        ###################################################################
        ###### read the image arrays (not necessary for this function) #####
        ###### can be removed ##############################################
        ###################################################################
        img = imread(new_path, as_gray=True)
        # convert the type of pixel to float32
        img = img.astype('float32')
        # normalize the pixel values
        img /= 255.0
        # append the image into the list
        imgs.append(img)

    return imgs

    train_x, test_x, val_x = resizeImage("train"), resizeImage("test"), resizeImage("
    validation")

    # display the CT images
for i in random.choices(range(len(train_x)), k=4):
    plt.imshow(train_x[i], cmap='gray')
    plt.show()

# write the data into csv files
pd.DataFrame({"name":train_name , "covid":train_y}).to_csv('train.csv', index=False)

pd.DataFrame({"name":test_name , "covid":test_y}).to_csv('test.csv', index=False)

pd.DataFrame({"name":val_name , "covid":val_y}).to_csv('validation.csv', index=False)
```

B Appendix: Random forest implementation with Python

```
# os: make sure the code related to path can work for all operating systems
import os

# time: benchmarking
import time

# sklearn: report the classification accuracy
import sklearn

# RandomForestClassifier: the random forest classifier
from sklearn.ensemble import RandomForestClassifier

# GridSearchCV: select the hyperparameter
from sklearn.model_selection import GridSearchCV

# pandas: plot the accuracy results, read files, save the data, etc
import pandas as pd

# numpy: process arrays
import numpy as np

# os: make sure the code can be executed for all operating systems
```

```python
import os

# random: draw random samples
import random

# imread: read and display images
from skimage.io import imread

# train_test_split: create validation set
from sklearn.model_selection import train_test_split

# pyplot: plot
import matplotlib.pyplot as plt

# Image: process images such as resize/clip the images
from PIL import Image

# shutil: remove an entire directory
import shutil

# time: benchmarking
import time

DIMENSION = (115, 61)
IMG_PATH = os.path.join("dataset", "CT")

# Importing Train and Test datasets
train_data = pd.read_csv("train.csv")
test_data = pd.read_csv("test.csv")
val_data = pd.read_csv("validation.csv")

def readImage(folder):
    """
    Get the image data array of all images in one folder.

    Parameters
    ----------
    folder: the name of the folder: "train", "test", "validation".

    Returns
    -------
    A list of image arrays and the labels.
    """

    data = pd.read_csv(folder + ".csv")
    x, y = data.name, data.covid

    imgs = []
    new_dir = os.path.join(IMG_PATH, folder)

    for name, covid in zip(x, y):
        img_path = os.path.join(new_dir, name)
        img = imread(img_path, as_gray=True).flatten()
        # convert the type of pixel to float32
        img = img.astype('float32')
        # normalize the pixel values
        img /= 255.0
        # append the image into the list
        imgs.append(img)

    return imgs, y

# load the datasets
train_x, train_y = readImage("train")
```

```
test_x, test_y = readImage("test")
val_x, val_y = readImage("validation")

# ================== Using Random Forest without hyper paramter tuning and clustering ===================
start = time.time()
rf = RandomForestClassifier(n_estimators = 100)
rf.fit(train_x+val_x, np.concatenate((train_y.values, val_y.values)))
# rf.fit(train_x, train_y)

print("Training data metrics:")
print(sklearn.metrics.classification_report(y_true = train_y, y_pred = rf.predict(train_x)))

print("Validation data metrics:")
print(sklearn.metrics.classification_report(y_true = val_y, y_pred = rf.predict(val_x)))

# Predictions on testset
    # test data metrics
print("Test data metrics:")
print(sklearn.metrics.classification_report(y_true = test_y, y_pred = rf.predict(test_x)))
end = time.time()
print("Time elapsed: %.2f min" % ((end-start)/60))
```

C Appendix: Simple CNN implementation with Python

```
# In[1]:


# pandas: plot the accuracy results, read files, save the data, etc
import pandas as pd

# os: make sure the code can be executed for all operating systems
import os

# random: draw random samples
import random

# imread: read and display images
from skimage.io import imread

# train_test_split: create validation set
from sklearn.model_selection import train_test_split

# pyplot: plot
import matplotlib.pyplot as plt

# Image: process images such as resize/clip the images
from PIL import Image

# shutil: remove an entire directory
import shutil

# time: benchmarking
import time

# pytorch libraries and modules
import torch

# numpy: process the data
import numpy as np

```

```
from torch.autograd import Variable
from torch.nn import Linear, ReLU, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module,
    Softmax, BatchNorm2d, Dropout
from torch.utils.data import Dataset
from torchvision import transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
from torch.optim import Adam, SGD


# In[2]:


#-------------------------
### SETTINGS
#-------------------------

# Hyperparameters
RANDOM_SEED = 1
LEARNING_RATE = 0.05
BATCH_SIZE = 8
NUM_EPOCHS = 20

# Architecture
NUM_CLASSES = 2
DIMENSION = (115, 61)

# Other
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
IMG_PATH = os.path.join("dataset", "CT")
NUM_WORKERS = 0


# In[3]:


def readImage(folder):
    """
    Get the image data array of all images in one folder.

    Parameters
    ----------
    folder: the name of the folder: "train", "test", "validation".

    Returns
    -------
    A list of image arrays and the labels.
    """

    data = pd.read_csv(folder + ".csv")
    x, y = data.name, data.covid

    imgs = []
    new_dir = os.path.join(IMG_PATH, folder)

    for name, covid in zip(x, y):
        img_path = os.path.join(new_dir, name)
        img = imread(img_path, as_gray=True)
        # convert the type of pixel to float32
        img = img.astype('float32')
        # normalize the pixel values
        img /= 255.0
        # append the image into the list
        imgs.append(img)
```

```

    return imgs, y

# load the datasets
train_x, train_y = readImage("train")
test_x, test_y = readImage("test")
val_x, val_y = readImage("validation")


# In[4]:


plt.imshow(train_x[random.randrange(len(train_x))], cmap='gray')
plt.show()


# In[5]:


class CTDataset(Dataset):
    """Custom Dataset for loading CT images"""

    def __init__(self, csv_path, img_dir, transform=None):

        df = pd.read_csv(csv_path)
        self.img_dir = [img_dir] * len(df['name'].values)
        self.img_names = df['name'].values
        self.y = df['covid'].values
        self.transform = transform

    def __getitem__(self, index):
        img = Image.open(os.path.join(self.img_dir[index],
                                      self.img_names[index]))
#         img = imread(os.path.join(self.img_dir[index],
#                                   self.img_names[index]), as_gray=True).astype('
    float32')
        if self.transform is not None:
            img = self.transform(img)

        label = self.y[index]
        return img, label

    def __len__(self):
        return self.y.shape[0]

    # concatenate two datasets
    def __add__(self, newDataset):
        self.img_names = np.concatenate((self.img_names, newDataset.img_names))
        self.img_dir += newDataset.img_dir
        self.y = np.concatenate((self.y, newDataset.y))
        return self


# In[6]:


# Note that transforms.ToTensor()
# already divides pixels by 255. internally

custom_transform = transforms.Compose([#transforms.Lambda(lambda x: x/255.),
                                       transforms.ToTensor()])


train_dataset = CTDataset(csv_path='train.csv',
```

```
                                     img_dir=os.path.join("dataset", "CT", "train"),
                                     transform=custom_transform)

valid_dataset = CTDataset(csv_path='validation.csv',
                                     img_dir=os.path.join("dataset", "CT", "validation"),
                                     transform=custom_transform)

# this method don't need validation dataset

train_loader = DataLoader(dataset= train_dataset+valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True,
                          num_workers=NUM_WORKERS)


valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=False,
                          num_workers=NUM_WORKERS)

test_dataset = CTDataset(csv_path='test.csv',
                                    img_dir=os.path.join("dataset", "CT", "test"),
                                    transform=custom_transform)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=BATCH_SIZE,
                         shuffle=False,
                         num_workers=NUM_WORKERS)


# In[7]:


# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break

# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break


# In[8]:


# This cell just checks if the dataset can be loaded correctly.

torch.manual_seed(0)

num_epochs = 2
for epoch in range(num_epochs):

    for batch_idx, (x, y) in enumerate(train_loader):

        print('Epoch:', epoch+1, end='')
        print(' | Batch index:', batch_idx, end='')
        print(' | Batch size:', y.size()[0])

        x = x.to(DEVICE)
        y = y.to(DEVICE)
```

```
        print('break minibatch for-loop')
        break


# # Multilayer Perceptron Model

# In[9]:


################################
### NO NEED TO CHANGE THIS CELL
################################

def compute_epoch_loss(model, data_loader):
    model.eval()
    curr_loss, num_examples = 0., 0
    with torch.no_grad():
        for features, targets in data_loader:
            features = features.to(DEVICE)
            targets = targets.to(DEVICE)
            logits, probas = model(features)
            loss = F.cross_entropy(logits, targets, reduction='sum')
            num_examples += targets.size(0)
            curr_loss += loss

        curr_loss = curr_loss / num_examples
        return curr_loss


def compute_accuracy(model, data_loader, device):
    model.eval()
    correct_pred, num_examples = 0, 0
    for i, (features, targets) in enumerate(data_loader):

        features = features.to(device)
        targets = targets.to(device)

        logits, probas = model(features)
        _, predicted_labels = torch.max(probas, 1)
        num_examples += targets.size(0)
        correct_pred += (predicted_labels == targets).sum()
    return correct_pred.float()/num_examples * 100


# In[10]:


class ConvNet(torch.nn.Module):

    def __init__(self, num_classes):
        super(ConvNet, self).__init__()

        self.num_classes = num_classes

        ### Layers: ADD ADDITIONAL LAYERS BELOW IF YOU LIKE

        # 115*61*1 => 28*28*8
        self.conv_1 = torch.nn.Conv2d(in_channels=1, out_channels=8, kernel_size=(6,6),
    stride=(2,4), padding=(0,1))

        # 28*28*8 => 14*14*8
        self.pool_1 = torch.nn.MaxPool2d(kernel_size=(2,2), stride=(2,2), padding=0)

```

```
        # 14*14*8 => 14*14*16
        self.conv_2 = torch.nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(3,3),
    stride=(1,1), padding=1)

        # 14*14*16 => 7*7*16
        self.pool_2 = torch.nn.MaxPool2d(kernel_size=(2,2), stride=(2,2), padding=0)

        self.bn_2 = torch.nn.BatchNorm2d(16)

        # 7*7*16 => 7*7*32
        self.conv_3 = torch.nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride
    =1, padding=2)

        self.bn_3 = torch.nn.BatchNorm2d(32)

        # Multilayer perceptron
        self.linear_1 = torch.nn.Linear(7*7*32, 7*7*64)
        self.bn_l1 = torch.nn.BatchNorm1d(7*7*64)
        self.linear_2 = torch.nn.Linear(7*7*64, 7*7*64)
        self.bn_l2 = torch.nn.BatchNorm1d(7*7*64)
        self.linear_out = torch.nn.Linear(7*7*64, num_classes)


    def forward(self, x):

        ### MAKE SURE YOU CONNECT THE LAYERS PROPERLY IF YOU CHANGED
        ### ANYTHNG IN THE __init__ METHOD ABOVE
        out = self.conv_1(x)
        out = F.relu(out)
        out = self.pool_1(out)

        out = self.conv_2(out)
        out = self.bn_2(out)
        # out = F.dropout(out, p=0.2, training=self.training)
        out = F.relu(out)
        out = self.pool_2(out)

        out = self.conv_3(out)
        out = self.bn_3(out)
        # out = F.dropout(out, p=0.2, training=self.training)
        out = F.relu(out)

        out = self.linear_1(out.view(-1, 7*7*32))
        out = self.bn_l1(out)
        out = F.relu(out)
        # out = F.dropout(out, p=0.2, training=self.training)

        out = self.linear_2(out)
        out = self.bn_l2(out)
        out = F.relu(out)
        # out = F.dropout(out, p=0.2, training=self.training)

        logits = self.linear_out(out)
        probas = F.softmax(logits, dim=1)
        return logits, probas


#################################
### Model Initialization     ###
#################################


# the random seed makes sure that the random weight initialization
# in the model is always the same.
```

```python
# In practice, some weights don't work well, and we may also want
# to try different random seeds. In this homework, this is not
# necessary.
torch.manual_seed(RANDOM_SEED)

### IF YOU CHANGED THE ARCHITECTURE ABOVE, MAKE SURE YOU
### ACCOUNT FOR IT VIA THE PARAMETERS BELOW. I.e., if you
### added a second hidden layer, you may want to add a
### hidden_2 parameter here. Also you may want to play
### with the number of hidden units.

model = ConvNet(NUM_CLASSES)
model = model.to(DEVICE)


### For this homework, do not change the optimizer. However, you
### likely want to experiment with the learning rate!
optimizer = torch.optim.SGD(model.parameters(), lr=LEARNING_RATE)


# In[11]:


###############################
### NO NEED TO CHANGE THIS CELL
###############################

def train(model, train_loader, test_loader):

    minibatch_cost, epoch_cost = [], []
    start_time = time.time()
    for epoch in range(NUM_EPOCHS):

        model.train()
        for batch_idx, (features, targets) in enumerate(train_loader):

            features = features.to(DEVICE)
            targets = targets.to(DEVICE)

            ### FORWARD AND BACK PROP
            logits, probas = model(features)
            cost = F.cross_entropy(logits, targets)
            optimizer.zero_grad()

            cost.backward()
            minibatch_cost.append(cost)

            ### UPDATE MODEL PARAMETERS
            optimizer.step()

            ### LOGGING
            if not batch_idx % 150:
                print ('Epoch: %03d/%03d | Batch %04d/%04d | Cost: %.4f'
                       %(epoch+1, NUM_EPOCHS, batch_idx,
                         len(train_loader), cost))


        with torch.set_grad_enabled(False): # save memory during inference
            print('Epoch: %03d/%03d | Train: %.3f%%' % (
                  epoch+1, NUM_EPOCHS,
                  compute_accuracy(model, train_loader, device=DEVICE)))

            cost = compute_epoch_loss(model, train_loader)
            epoch_cost.append(cost)
```

```

        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))

    print('Total Training Time: %.2f min' % ((time.time() - start_time)/60))


    with torch.set_grad_enabled(False): # save memory during inference
        print('Test accuracy: %.2f%%' % (compute_accuracy(model, test_loader, device=DEVICE)
    ))

    print('Total Time: %.2f min' % ((time.time() - start_time)/60))

    return minibatch_cost, epoch_cost


minibatch_cost, epoch_cost = train(model, train_loader, test_loader)


plt.plot(range(len(minibatch_cost)), minibatch_cost)
plt.ylabel('Cross Entropy')
plt.xlabel('Minibatch')
plt.show()

plt.plot(range(len(epoch_cost)), epoch_cost)
plt.ylabel('Cross Entropy')
plt.xlabel('Epoch')
plt.show()
```

D Appendix: ResNet implementation with Python

```
# In[1]:


# pandas: plot the accuracy results, read files, save the data, etc
import pandas as pd

# os: make sure the code can be executed for all operating systems
import os

# random: draw random samples
import random

# imread: read and display images
from skimage.io import imread

# train_test_split: create validation set
from sklearn.model_selection import train_test_split

# pyplot: plot
import matplotlib.pyplot as plt

# Image: process images such as resize/clip the images
from PIL import Image

# shutil: remove an entire directory
import shutil

# time: benchmarking
import time

# pytorch libraries and modules
```

```
import torch

# numpy: process the data
import numpy as np

from torch.autograd import Variable
from torch.nn import Linear, ReLU, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module, Softmax, BatchNorm2d, Dropout
from torch.utils.data import Dataset
from torchvision import transforms
from torch.utils.data import DataLoader
import torch.nn.functional as F
from torch.optim import Adam, SGD


# In[2]:


#-------------------------
### SETTINGS
#-------------------------

# Hyperparameters
RANDOM_SEED = 1
LEARNING_RATE = 0.05
BATCH_SIZE = 8
NUM_EPOCHS = 20

# Architecture
NUM_FEATURES = 32*32
NUM_CLASSES = 2
DIMENSION = (115, 61)

# Other
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
IMG_PATH = os.path.join("dataset", "CT")
NUM_WORKERS = 0


# # Loading the data

# In[3]:


class CTDataset(Dataset):
    """Custom Dataset for loading CT images"""

    def __init__(self, csv_path, img_dir, transform=None):

        df = pd.read_csv(csv_path)
        self.img_dir = [img_dir] * len(df['name'].values)
        self.img_names = df['name'].values
        self.y = df['covid'].values
        self.transform = transform

    def __getitem__(self, index):
        img = Image.open(os.path.join(self.img_dir[index],
                                      self.img_names[index]))
#          img = imread(os.path.join(self.img_dir[index],
#                                        self.img_names[index]), as_gray=True).astype('float32')
        if self.transform is not None:
            img = self.transform(img)

```

```
        label = self.y[index]
        return img, label

    def __len__(self):
        return self.y.shape[0]

    # concatenate two datasets
    def __add__(self, newDataset):
        self.img_names = np.concatenate((self.img_names, newDataset.img_names))
        self.img_dir += newDataset.img_dir
        self.y = np.concatenate((self.y, newDataset.y))
        return self


# In[4]:


# Note that transforms.ToTensor()
# already divides pixels by 255. internally

custom_transform = transforms.Compose([#transforms.Lambda(lambda x: x/255.),
                                       transforms.ToTensor()])


train_dataset = CTDataset(csv_path='train.csv',
                                 img_dir=os.path.join("dataset", "CT", "train"),
                                 transform=custom_transform)

valid_dataset = CTDataset(csv_path='validation.csv',
                                 img_dir=os.path.join("dataset", "CT", "validation"),
                                 transform=custom_transform)

# this method don't need validation dataset

train_loader = DataLoader(dataset= train_dataset+valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=True,
                          num_workers=NUM_WORKERS)


valid_loader = DataLoader(dataset=valid_dataset,
                          batch_size=BATCH_SIZE,
                          shuffle=False,
                          num_workers=NUM_WORKERS)

test_dataset = CTDataset(csv_path='test.csv',
                                img_dir=os.path.join("dataset", "CT", "test"),
                                transform=custom_transform)

test_loader = DataLoader(dataset=test_dataset,
                         batch_size=BATCH_SIZE,
                         shuffle=False,
                         num_workers=NUM_WORKERS)


# In[5]:


# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break
```

```
# Checking the dataset
for images, labels in train_loader:
    print('Image batch dimensions:', images.shape)
    print('Image label dimensions:', labels.shape)
    break


# In[6]:


# This cell just checks if the dataset can be loaded correctly.

torch.manual_seed(0)

num_epochs = 2
for epoch in range(num_epochs):

    for batch_idx, (x, y) in enumerate(train_loader):

        print('Epoch:', epoch+1, end='')
        print(' | Batch index:', batch_idx, end='')
        print(' | Batch size:', y.size()[0])

        x = x.to(DEVICE)
        y = y.to(DEVICE)

        print('break minibatch for-loop')
        break


# # ResNet

# In[7]:


def compute_epoch_loss(model, data_loader):
    model.eval()
    curr_loss, num_examples = 0., 0
    with torch.no_grad():
        for features, targets in data_loader:
            features = features.to(DEVICE)
            targets = targets.to(DEVICE)
            logits = model(features)
            loss = F.cross_entropy(logits, targets, reduction='sum')
            num_examples += targets.size(0)
            curr_loss += loss

        curr_loss = curr_loss / num_examples
        return curr_loss

def compute_accuracy(model, data_loader):
    model.eval()
    correct_pred, num_examples = 0, 0
    for i, (features, targets) in enumerate(data_loader):

        features = features.to(DEVICE)
        targets = targets.to(DEVICE)

        logits = model(features)
        _, predicted_labels = torch.max(logits, 1)
        num_examples += targets.size(0)
        correct_pred += (predicted_labels == targets).sum()
    return correct_pred.float()/num_examples * 100

```

```


# In[8]:


model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet18', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet34', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet50', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet101', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.6.0', 'resnet152', pretrained=True)
model.eval()


# In[9]:


# keep the pretrained layers, don't update them
for parameter in model.parameters():
    parameter.requires_grad = False


# In[10]:


model.conv1 = torch.nn.Conv2d(1,64,kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=
    False)
model.fc = torch.nn.Sequential(
           torch.nn.Linear(in_features=512, out_features=100, bias=True),
           torch.nn.ReLU(inplace=True),
           torch.nn.Dropout(p=0.5, inplace=False),
           torch.nn.Linear(in_features=100, out_features= NUM_CLASSES, bias=True))


# In[15]:


# instances where you want to save and load your neural networks across different devices.
model = model.to(DEVICE)

# torch.optim is a package implementing various optimization algorithms.
optimizer = torch.optim.Adam([
                 {'params': model.conv1.parameters()},
                 {'params': model.fc.parameters()}
             ])


# In[16]:


def train(model, train_loader, test_loader, NUM_EPOCHS):

    minibatch_cost, epoch_cost = [], []
    start_time = time.time()
    for epoch in range(NUM_EPOCHS):

        model.train()
        for batch_idx, (features, targets) in enumerate(train_loader):

            features = features.to(DEVICE)
            targets = targets.to(DEVICE)

            ### FORWARD AND BACK PROP
            logits = model(features)
```

```
            cost = F.cross_entropy(logits, targets)
            optimizer.zero_grad()

            cost.backward()
            minibatch_cost.append(cost)

            ### UPDATE MODEL PARAMETERS
            optimizer.step()

            ### LOGGING
            if not batch_idx % 150:
                print ('Epoch: %03d/%03d | Batch %04d/%04d | Cost: %.4f'
                       %(epoch+1, NUM_EPOCHS, batch_idx,
                         len(train_loader), cost))


        with torch.set_grad_enabled(False): # save memory during inference
            print('Epoch: %03d/%03d | Train: %.3f%%' % (
                  epoch+1, NUM_EPOCHS,
                  compute_accuracy(model, train_loader)))

            cost = compute_epoch_loss(model, train_loader)
            epoch_cost.append(cost)

        print('Time elapsed: %.2f min' % ((time.time() - start_time)/60))

    print('Total Training Time: %.2f min' % ((time.time() - start_time)/60))


    with torch.set_grad_enabled(False): # save memory during inference
        print('Test accuracy: %.2f%%' % (compute_accuracy(model, test_loader)))

    print('Total Time: %.2f min' % ((time.time() - start_time)/60))

    return minibatch_cost, epoch_cost


# In[17]:


minibatch_cost, epoch_cost = train(model, train_loader, test_loader, NUM_EPOCHS = NUM_EPOCHS
    )

plt.plot(range(len(minibatch_cost)), minibatch_cost)
plt.ylabel('Cross Entropy')
plt.xlabel('Minibatch')
plt.show()

plt.plot(range(len(epoch_cost)), epoch_cost)
plt.ylabel('Cross Entropy')
plt.xlabel('Epoch')
plt.show()
```

E Appendix: SVM implementation with Python

```
# Path: list the files in the directory
from pathlib import Path

# os: make sure code can be executed in all operating systems
import os

```

```python
# time: benchmarking
import time

# plt: draw pictures
import matplotlib.pyplot as plt

# svm: svm classifier
from sklearn import svm, metrics

# Bunch: container object for datasets
from sklearn.utils import Bunch

# numpy: process the image arrays
import numpy as np

# pandas: process the image arrays
import pandas as pd

# GridSearchCV: Exhaustive search over specified parameter values for an estimator
# train_test_split: split the full datasets into train and test dataset, respectively
from sklearn.model_selection import GridSearchCV, train_test_split

# imread: read the images
from skimage.io import imread

# resize: resize the images
from skimage.transform import resize

# plot_decision_regions: visualize the SVM hyperplane
from mlxtend.plotting import plot_decision_regions

DIMENSION = (115, 61)
IMG_PATH = os.path.join("dataset", "CT")

def load_image_files(container_path, dimension=DIMENSION):
    """
    Load image files with categories as subfolder names
    which performs like scikit-learn sample dataset

    Parameters
    ----------
    container_path : string or unicode
        Path to the main folder holding one subfolder per category
    dimension : tuple
        size to which image are adjusted to

    Returns
    -------
    Bunch
    """

    image_dir = Path(container_path)
    folders = [directory for directory in image_dir.iterdir() if (directory.is_dir() and "
    COVID" in directory.name)]
    categories = [fo.name for fo in folders]

    descr = "A image classification dataset"
    images = []
    flat_data = []
    target = []
    for i, direc in enumerate(folders):
        for file in direc.iterdir():
            if len(file.name.split(".")) == 1 or file.name[0] == '.':
                continue
```

```
            img = imread(file, as_gray = True)
            img_resized = resize(img, dimension, anti_aliasing=True, mode='reflect')
            flat_data.append(img_resized.flatten())
            images.append(img_resized)
            target.append(i)
    flat_data = np.array(flat_data)
    target = np.array(target)
    images = np.array(images)

    return Bunch(data=flat_data,
                 target=target,
                 target_names=categories,
                 images=images,
                 DESCR=descr)

image_dataset = load_image_files(IMG_PATH)

X_train, X_test, y_train, y_test = train_test_split(
    image_dataset.data, image_dataset.target, test_size=0.1,random_state=109)

param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
svc = svm.SVC()

start = time.time()
clf1 = GridSearchCV(svc, param_grid, cv = 3)
clf1.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
print("Classification report for - \n{}:\n{}\n".format(
    clf1, metrics.classification_report(y_test, y_pred)))
print("Time elasped: %.2f min" % ((time.time()-start)/60))

clf1.best_params_

param_grid = [
  {'C': [10, 50], 'gamma': [0.002, 0.001, 0.0005], 'kernel': ['rbf']},
 ]
svc = svm.SVC()

start = time.time()
clf2 = GridSearchCV(svc, param_grid, cv = 3)
clf2.fit(X_train, y_train)
y_pred = clf2.predict(X_test)
print("Classification report for - \n{}:\n{}\n".format(
    clf2, metrics.classification_report(y_test, y_pred)))
print("Time elasped: %.2f min" % ((time.time()-start)/60))

clf2.best_params_

# train accuracy
y_pred = clf2.predict(X_train)
print("Classification report for - \n{}:\n{}\n".format(
    clf1, metrics.classification_report(y_train, y_pred)))
```