Final Report

Group Members

Yang Haocheng 1004883

Lu Mingrui 1005354

1. Problem Statement

This project aims to develop a deep learning model that can accurately classify CT scans as either COVID-19 positive or negative, using a dataset of labeled CT scans of COVID-19 and non-COVID-19 patients.it can aid healthcare professionals in accurately and quickly diagnosing COVID-19 from CT scans.

2. Dataset and Collection

We collected a dataset of COVID-19 chest CT scans from various sources, including publicly available datasets and hospitals. The dataset consists of 500 CT scans from COVID-19 positive patients, and 500 CT scans from healthy patients. The CT scans were collected from different hospitals across the world, using different CT scanner models and settings. The dataset was annotated by radiologists to indicate the presence or absence of COVID-19 infection. Based on the public data set, we constructs CT image data including COVID19 and Non-COVID19, and conducts classification experiments, uses the random forest of traditional machine learning as a preliminary naive model to classify images, and classifies medical CT images.
the source：https://github.com/UCSD-AI4H/COVID-CT/tree/master/Images-processed

3. Data Pre-processing

This code is used to split a dataset consisting of CT scans of patients into a training set and a test set. The dataset contains two classes: COVID-19 scans and non-COVID-19 scans. The code first creates two directories to store the COVID and non-COVID images separately. It then loops through each image in the original dataset and copies it to the appropriate directory while also appending its file path to the data_target list and its label to the data_label list.

Next, the code uses the train_test_split function from the sklearn library with test_size=0.3 to randomly split the data_target and data_label lists into a training set and a test set. The training set and test set are each split into two subdirectories, one for COVID images and one for non-COVID images. The images in each set are then copied to their respective subdirectories.

4. Algorithm/Model:

Data Augmentation

In this project we generate an ImageDataGenerator that reads image data, reads image files, decodes jpeg images into RGB pixel networks, converts these pixels into floating-point tensors and scales them between 0 and 1.

And we also performed data augmentation via ImageDataGenerator to make our model more robust.

ImageDataGenerator parameters are introduced as follows:
rotation_range is an angle value (in the range of 0~180), indicating the angle range of the random rotation of the image.
width_shift and height_shift are the extent to which the image is shifted horizontally or vertically (relative to the total width degree or overall height).
shear_range is the angle of the random shear transformation.
zoom_range is the range in which the image is randomly zoomed.
horizontal_flip randomly flips half of the image horizontally. If there is no horizontal asymmetry assumption (such as true images of the real world), this approach makes sense.
fill_mode is the method used to fill newly created pixels, which may come from rotation or width/height translation.

Our Model

Here we decided to use Convolutional Nerual Network to capture saptial dependency and utilize monophily of pixels. The model is structures inside `cnn.py`.

The total number of parameters to be trained is: 3453121, the size of the first layer convolution kernel is: 33, the number of convolution kernels is 32, and the activation function is RELU; the second layer is the pooling layer, pooling The size is 22, the third layer convolution kernel size is: 33, the number of convolution kernels is 64, and the activation function is RELU; the fourth layer is a pooling layer, and the pooling size is 2 2. The size of the fifth layer convolution kernel is: 33, the number of convolution kernels is 128, and the activation function is RELU; the sixth layer is the pooling layer, the pooling size is 22, and the seventh layer The size of the convolution kernel is 33, the number of convolution kernels is 128, and the activation function is RELU; the eighth layer is the pooling layer, and the pooling size is 22, and the ninth layer uses the size of the eighth layer Perform tiling, then add the DROUPOUT layer, randomly drop 50% of the convolution kernel, add a dropout layer to further improve the recognition rate, then add a fully connected layer with a length of 512, the activation function is RELU, and finally add a classification layer , the classification function is SIGMOID.

The size of filters are tuned based on the performance of our model on test set.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
dropout (Dropout)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

The optimizer that we use is RMSprop with learning_rate=1e-4.

We have tried 1e-2, 1e-3, 1e-4, 1e-5 for learning rate and find our model has a relative good performance on test set with lr=1e-4 and an acceptable time of training.

5. Evaluation Methodology:

Loss function: Cross-entropy

The cross-entropy loss function is often used in classification problems, especially when neural networks are used for classification problems, cross-entropy is often used as a loss function.

Evaluation metric of model: Accuracy on test set

The dataset is split into training set and a test set with test_size=0.3 randomly. And the effectiveness of model is reflected via the accuracy on test set.

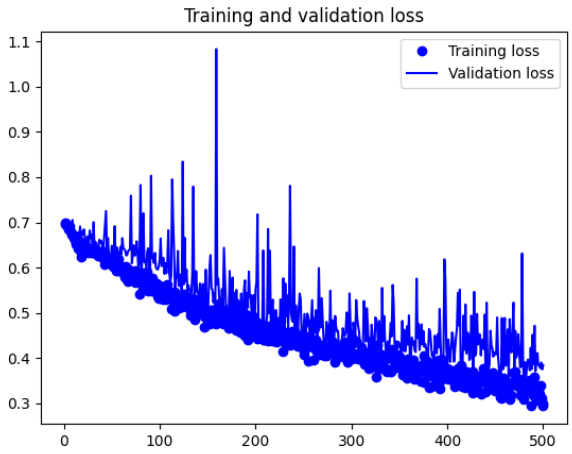
Accuracy is the easiest to understand. It refers to the samples that are correctly classified in all samples, including all sample categories.

6. Results

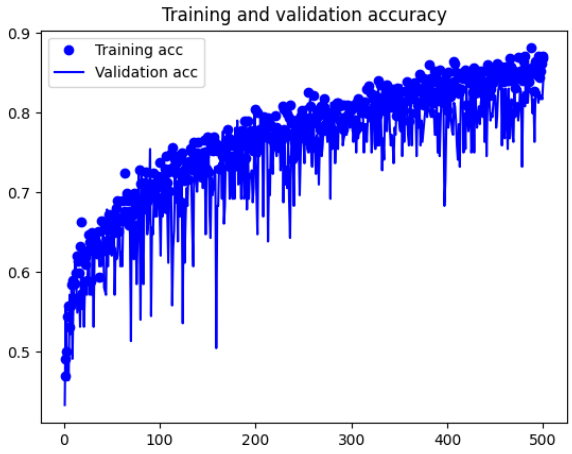
Here we use 500 epochs, and the number of CT images for each training iteration(batch size) is 50, and the number of test images is also 50 for one validation.

The training error at the beginning is relatively large. As the number of epochs gradually increases, training The error gradually decreases and tends to be stable, and the validation accuracy gradually increases.

The results of training loss and validation loss are as follows:



The results of the training set and validation set on the accuracy index are as follows:



Finally, the accuracy of our model on the test set is: **0.8393**.

7. Compare our model with existing ones

In this project we also use transfer learning where we use a pre-trained model as a starting point for a new task and compared its performance with our final model which we build and train from scratch

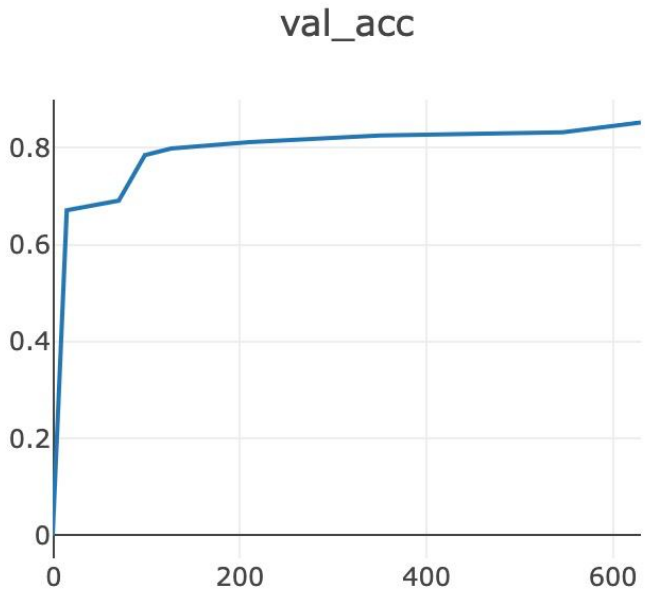
Firstly, we loaded a pre-trained model ResNet-18 with its weights which is a pre-trained convolutional neural network architecture that has shown remarkable performance on the ImageNet dataset. So it possibly has better ability to extract meaningful features from images than our final model.

Then we extracts the pre-trained model's feature extractor layers, which exclude the final classification layer. The original last layer of ResNet-18 is a fully connected layer that outputs a tensor of shape [batch_size, num_classes], where num_classes is the number of classes in the ImageNet dataset. Since our new task is binary classification task (COVID-19 vs Non-COVID-19), we need to remove this last layer and replace it with our own classification layer.

Thus we need to flatten the output of the feature extractor layers to a 1D tensor first. This is necessary because the output of the ResNet-18 feature extractor layers has a shape of [batch_size, num_channels, height, width], where num_channels, height, and width depend on the size of the input image. We need to flatten this tensor to a 1D tensor so that we can pass it through the fully-connected classification layer. Finally, we add a full-connected layer with 512 input features (which is the output size of the last convolutional layer in ResNet-18) and 2 output features (which is the number of classes in our new binary classification task). This layer is the classification layer that maps the output of the feature extractor layer to the binary classification output.

Overall, in this transfer learning we created a new model that reuses the pre-trained ResNet-18 architecture for feature extraction and replaces the final classification layer with a new binary classification.

The accuracy on test set of the model with transfer learning is **0.82666**



And when we compare the performance of this pre-trained model with our final model, the result turns out that our own model has the better performance. We think the reason is because transfer learning models require a large amount of data to learn meaningful representations. but our dataset is relatively small so ResNet-18 may not have enough data to learn good representations, and another reason we think is because ResNet-18 was pre-trained on the ImageNet dataset, which is a very large dataset of diverse natural images but our own dataset contains only COVID-19 chest CT scans which is a very different images distribution from the ImageNet dataset, hence ResNet-18 may not be able to learn meaningful representations for our specific task. So finally we still choosed our own CNN model which we build and train from scratch as our final model.

Steps to re-train the model from scratch

required libraries

`tensorflow 2.12.0`,`matplotlib`,`scikit-learn 1.2.2`,`PIL`, `shutil`

Step1

The original project folder is structured in this way:

```
.
├── dataset/
│   ├── CT_COVID
│   └── CT_NonCOVID
├── cnn.py
├── inference.py
├── data_preprocess.py
└── main.py
```

Run `data_preprocess.py` to preprocess the `dataset` and generate `train_set` and `test_set`

Detail of the code for `data_preprocess.py` can refer to '3. Data Pre-processing'

After running the file the folder will look like this:

```
.
├── dataset/
│   ├── CT_COVID
│   └── CT_NonCOVID
├── covid
├── non_covid
├── train_set/
│   ├── covid
│   └── non_covid
├── test_set/
│   ├── covid
│   └── non_covid
├── cnn.py
├── inference.py
├── data_preprocess.py
└── main.py
```

Step2

Run `main.py`

This will trigger training on `train_set` and validation on `test_set`

After the training is done the model will be saved to `savedmodel/ourmodel`

The figure for Training and validation acc/loss will also be saved as `accuracy.png` and `loss.png`

After running the file the folder will look like this:

```
.
├── dataset/
│   ├── CT_COVID
│   └── CT_NonCOVID
├── covid
├── non_covid
├── train_set/
│   ├── covid
│   └── non_covid
├── test_set/
│   ├── covid
│   └── non_covid
├── cnn.py
├── inference.py
├── data_preprocess.py
├── main.py
├── savedmodel/
│   └── ourmodel
├── accuracy.png
└── loss.png`
```

How to make prediction using our trained model

This can be done by changing the path of Input CT scan inside `inference.py` and run `inference.py`

The result will be printed.

How to recreate the exact trained model

The trained model is saved in `savedmodel/ourmodel`

This can be done by loading the trained model via `reconstructed_model = tf.keras.models.load_model("savedmodel/ourmodel")`