CMPE 258 Deep Learning

Homework - 1

Github Link: https://github.com/ShreyasKulkarni19/Covid19_training_with_inference/tree/main

Problem Statement

The task was to select an image classification dataset for training, modify the model architecture or training configurations to achieve better performance, evaluate performance across various configurations and visualize them using matplotlib, perform model inference optimization and evaluate the latency or the computational cost.

Dataset

The dataset for this assignment was taken from Kaggle. It is a Covid-19 dataset which consists of 3 classes of images. The classes are Normal, Covid and Pneumonia. The dataset consists of 317 images out of which 251 were used for training and 66 were used for testing.

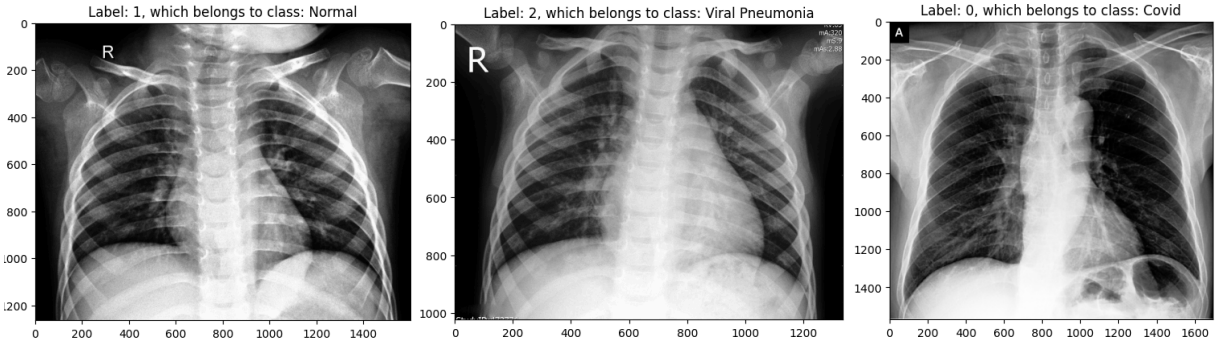

Fig. 1 Above images show the three classes from the dataset

Data Preprocessing

The training and testing datasets were used for image augmentation including resizing and horizontal flipping. These augmented images were then converted to tensors and normalized.

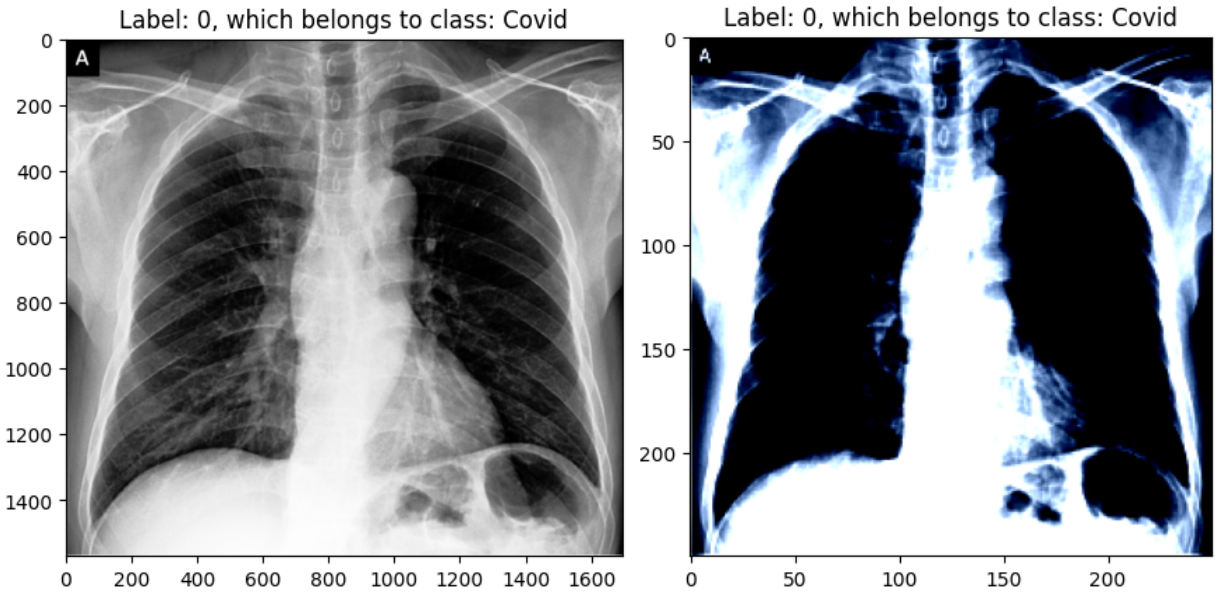

Fig. 2 The image on the left is before transformation and the image on the right is after transformation

Methodology

1. DataLoader

The Dataset retrieves our dataset's features and labels one sample at a time. While training a model, we typically want to pass samples in "mini batches", reshuffle the data at every epoch to reduce model overfitting, and use Python's multiprocessing to speed up data retrieval. DataLoader is an iterable that abstracts this complexity for us in an easy API. The batch size for both training and validation dataloader is 8.

2. Model Architecture

```
Covid19Model0(
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dropout1): Dropout(p=0.5, inplace=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dropout2): Dropout(p=0.5, inplace=False)
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dropout3): Dropout(p=0.5, inplace=False)
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (linear1): Linear(in_features=123008, out_features=256, bias=True)
  (dropout4): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=256, out_features=128, bias=True)
  (dropout5): Dropout(p=0.5, inplace=False)
  (linear3): Linear(in_features=128, out_features=64, bias=True)
  (dropout6): Dropout(p=0.5, inplace=False)
  (linear4): Linear(in_features=64, out_features=3, bias=True)
)
```

Fig. 3 Image showing model architecture

The first convolution takes a 3 channel input and outputs 32 feature maps. Similarly the second and third convolutional layers are taken in 32 and 64 feature maps and output 64 and 128 feature maps respectively. We use the 2D max pooling layer to reduce the spatial dimensions of the feature maps by taking the maximum value within a 2x2 window. The network consists of four linear layers which are fully connected. Layer 1 flattens the outputs of the convolutional layers and feeds it into the fully connected layer with 256 neurons. Linear 2 takes the output of linear 1 and feeds it into the 128 neurons of layer 3 and layer 3 feeds its input to the 64 neurons in layer 4. Finally, layer 4 outputs with 3 neurons which correspond to the 3 classes we are interested in.

3. Training and Evaluation

The training and evaluation function takes in the model, training and validation dataloaders, loss criterion, optimizer and the number of epochs as parameters. We use 50 epochs to train the model and save the model at each epoch to check if the model is overfitting and use the best model with minimal cost function. We use the cross entropy loss function and Adam as our optimizer function.

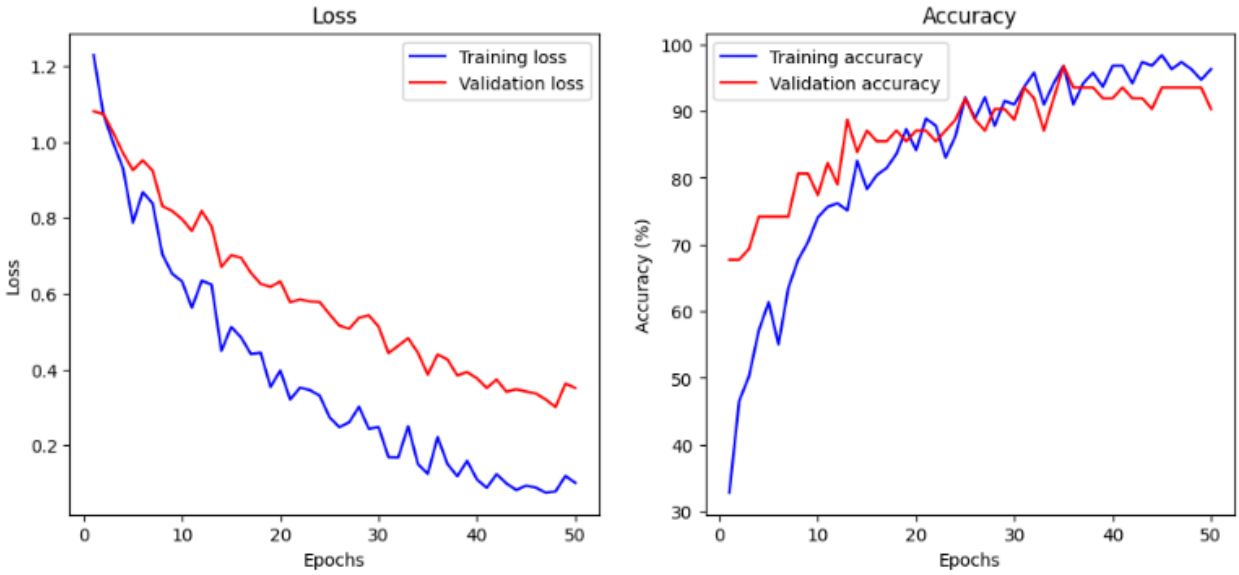

Fig. 4 Graph showing training accuracy and loss of model over 50 epochs

a. Hyperparameter Tuning

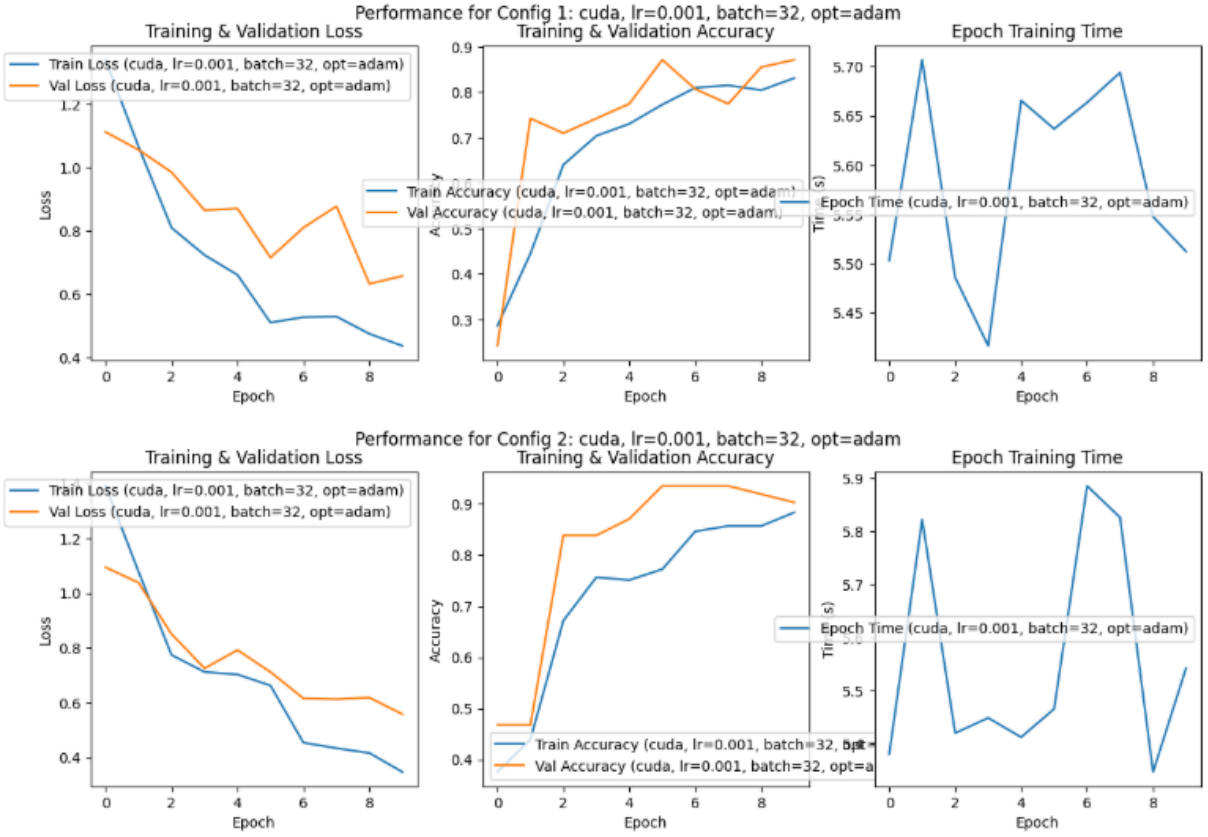

Fig. 5 Evaluating accuracies with variation in learning rate for adam optimizer

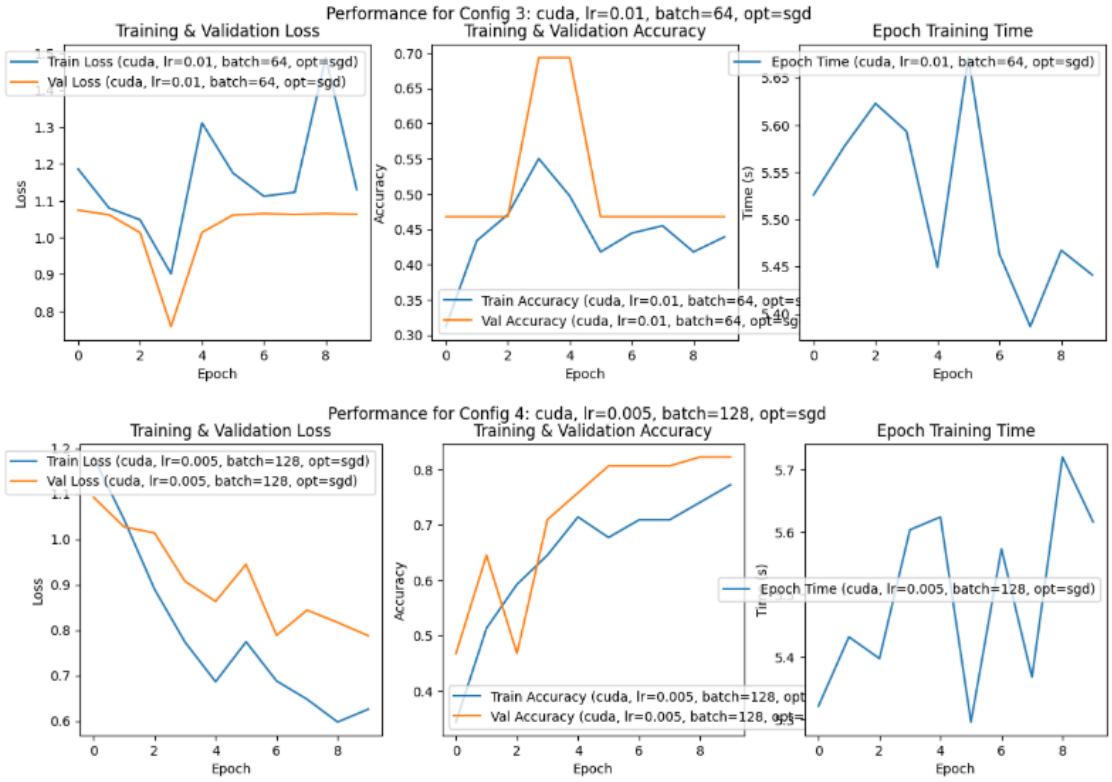

Fig. 6 Evaluating accuracies with variation in learning rate for for SGD optimizer

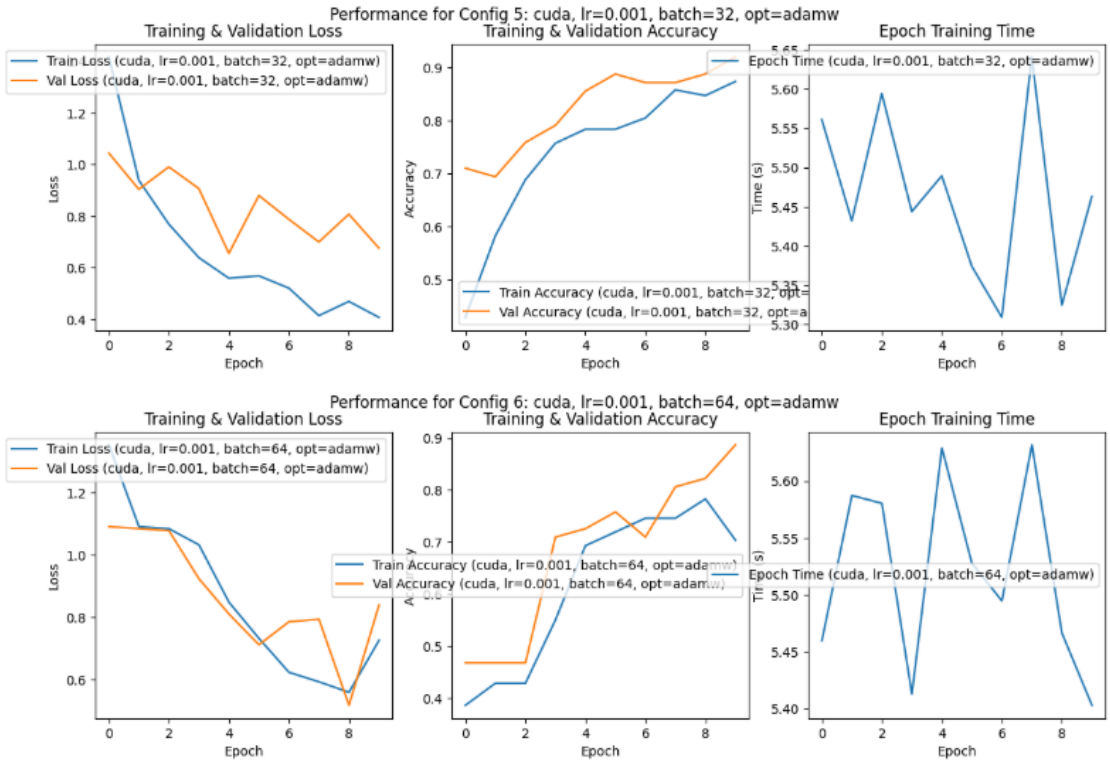

Fig. 7 Evaluating accuracies with variation in learning rate for for AdamW optimizer

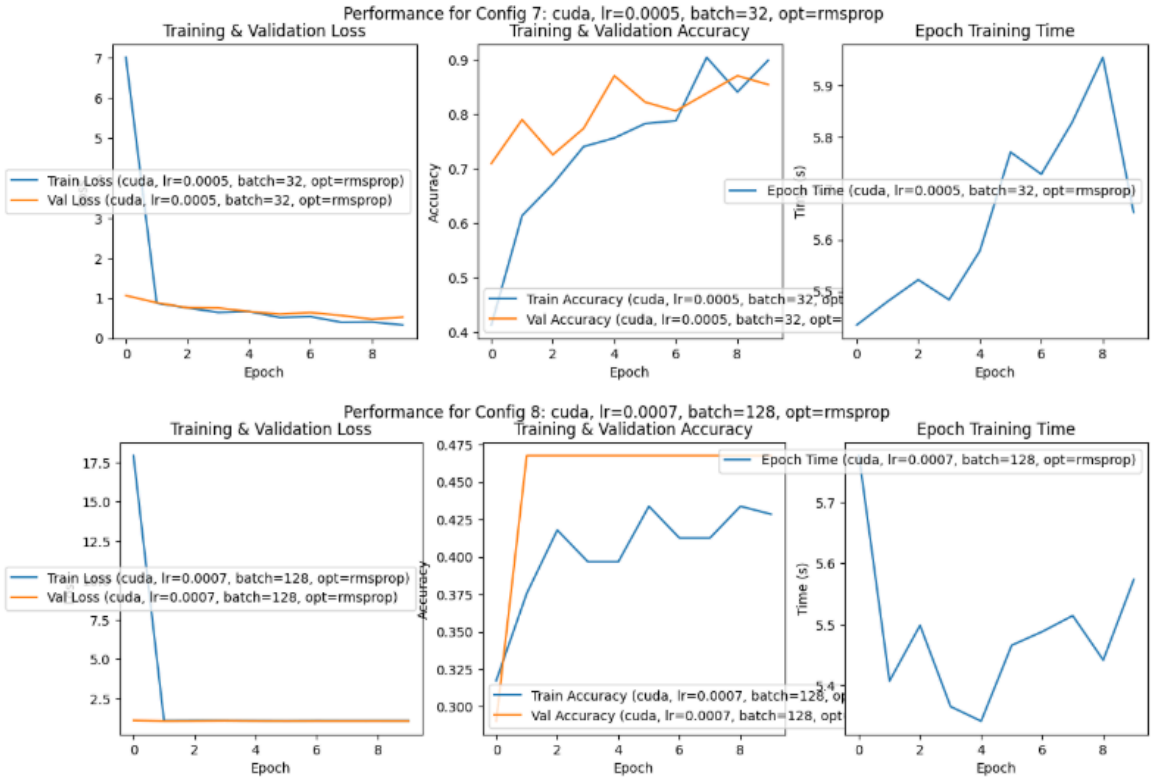

Fig. 8 Evaluating accuracies with variation in learning rate for for RMSProp optimizer

During the hyperparameter tuning, we evaluated the performance of the model across various model configurations like learning rate and optimizer. Based on the results shown above, we understand that the model performed better using Adam optimizer. Therefore we finalized Adam as our optimizer.

4. Testing

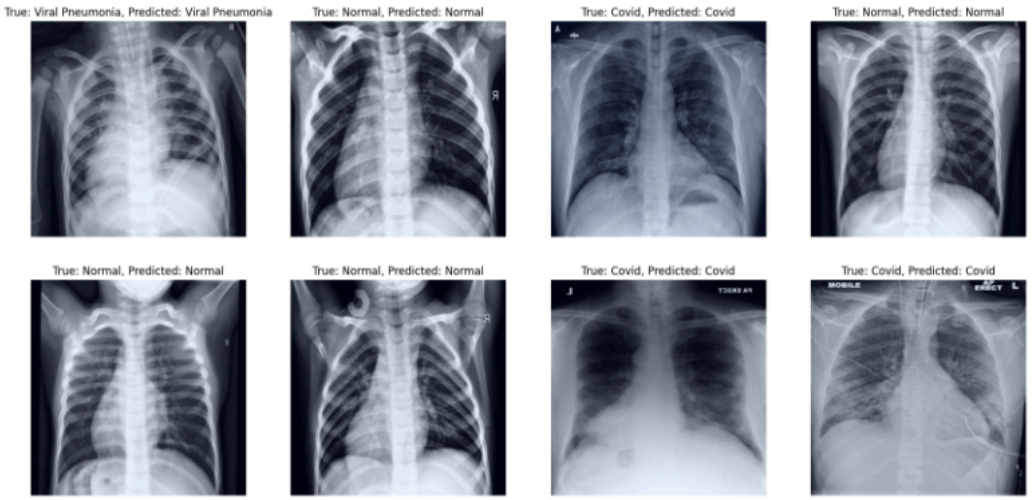

Fig. 9 Predicted Images

We used 66 images for testing like mentioned earlier in this report. These were not previously seen by the model. The image above shows the predicted images and each image has the original label and the predicted label. We may notice that all the images were correctly predicted and this proves that our model is doing well on unseen data.

Inference

During inference, we make use of the same model from our training. We save the model as a '.pth' file. In this assignment, we are using ONNX for inference. To begin with, we convert our model into ONNX format which takes in parameters like model name, ONNX model path, sample input tensor and store the trained parameters weights inside the model file. Given the image path, we apply the same transformations that we used during the training. We apply horizontal flip and resize the image to convert it into a tensor and then normalize it. We then add the batch dimension to the image tensor using the 'unsqueeze' function. Finally we use the ONNX Runtime to load the ONNX model and pass the input data as a parameter while running the inference. The inference model will output in a numerical format where '0', '1', and '2' represent the three classes we are trying to predict.

```
import onnxruntime as ort
import numpy as np

# Load the ONNX model
session = ort.InferenceSession(onnx_model_path)

# Prepare the input data
input_name = session.get_inputs()[0].name
input_data = image_tensor.numpy()  # Convert to NumPy array if necessary

# Run inference
output = session.run(None, {input_name: input_data})

print(output)

output = np.argmax(output, axis=2)

# Display the result
print("Inference output:", output)
```

```
[array([[ 2.7040148 ,  0.05735902, -1.1582744 ]], dtype=float32)]
Inference output: [[0]]
```

Fig. 10 Inference