Jonah Buchanan

4/24/20

Binary Classification of Chest X-Rays

1. Abstract

The goal of this project is to train a Deep Learning model to learn to identify whether an x-ray image of the lungs is normal or has pneumonia. The model's performance will be evaluated by the accuracy of correctly labeled images in a test set of x-ray images. The project will implement many different Deep Learning techniques such as regularization, optimization, different architectures, and hyperparameters to try and create a model with the best performance.

2. Introduction

This project will implement a supervised learning model to accomplish the goal stated in the abstract. The learning task is a binary classification since there are two types of images. The model will classify each image as either a normal image or an image with pneumonia.

3. The Problem

3.1 Representation and Organization of Data

The data was obtained from a data set on Kaggle named "Chest X-Ray Images (Pneumonia)". The data is represented as x-ray images stored as jpeg files. The data is stored in three separate folders: test, train, and val. Each folder has two subfolders labeled: NORMAL, PNEUMONIA. The NORMAL subfolder contains images of the lungs without pneumonia and the PNEUMONIA subfolder contains images of the lungs with pneumonia. The train folder is used to train the model. The val folder is used as the validation set in the training of the model. The test set is used to test the model learned from the training set.

3.2 Augmenting the Data

There is one issue with the data set. There are a lot more normal images than pneumonia images. This can affect the model to not generalize to unseen images. In order to even out the data, I augmented the data using a Keras function called ImageDataGenerator class and a method called flow_from_directory. This method takes a data set and returns batches of the augmented data. By doing this to the data, we avoid overfitting the training data and our model should generalize to the test set better.

4. Learning Components

Now that we have our data generated, we can create our model. In this project, we will generate several models so this section will describe how to define, train, and evaluate a generic model to perform binary classification.

4.1 Defining the Model

We will use Keras to create a sequential model. Then we will add Convolutional layers with “relu” as the activation function. You can add Pooling layers as well. Once you have the architecture you want, you must add flatten the data then add two Dense layers that result in a prediction for each input image.

Then we compile the model with binary cross entropy since we are performing a binary classification. Then you can add the optimizer of your choice, in this project we started with a default optimizer of Adadelta.

4.2 Training and Testing the Model

Once you create your model, you just need to set the epochs and batch size. Then you can use fit_generator to train your data model. Then use evaluate generator with your test data set to test the model.

The above process can be used to generate your own model. You can try different regularization methods, optimization functions, architectures, and hyperparameters. Fine tuning these areas of the model can lead to a better model.

5. Learning Approach

5.1 Performance of Models

Model	Test Accuracy
Model 1	91.98718070983887
Model 2	88.94230723381042
Model 3	90.06410241127014
Model 4	92.62820482254028
Model 5	88.78205418586731
Model 6	90.70512652397156
Model 7	89.26281929016113
Model 8	85.09615659713745
Model 9	87.0192289352417

5.2 Learning Approach of Model 4

I will describe the learning approach of model 4 which had the highest test accuracy of the 9 models.

5.2.1 Model

Model 4 replicates the LeNet-5 architecture. First there is a convolutional layer with a kernel size of (3, 3) and an activation function of “relu”. Then we have a pooling layer with the max pooling function and a pool size of (2, 2). Then we have another convolution layer with a kernel size of (3, 3) and a activation function of “relu”. Then we have another pooling layer with the max pooling function and a pool size of (2, 2). Then we flatten the data and have a Dense layer with 128 units and an activation function of “relu”. Next, we have another Dense layer with 1 unit and sigmoid as the activation function. Finally, we compile the model with a loss of binary cross entropy, RMSprop as the optimizer, and accuracy as the metric.

5.2.2 Training the Model

Now that we have created the model, we will train the model with the augmented data set described above, 10 epochs, and a batch size of 32. Using fit_generator, set steps_per_epoch to the number of training data divided by the batch size. This size is recommended by the Keras documentation. Also set the validation steps to the number of validation images divided by the batch size. This is also the recommended way to set the value according to the Keras documentation.

5.3.3 Testing the Model

Finally, we can test the model by using evaluate_generator on the test set. We can also plot the loss function to see how the model learned over time.

6. Conclusion

6.1 Performance

This project has developed a model that can predict if a chest x-ray image is normal or has pneumonia with 92.6% accuracy. Although this model has good performance, I would not recommend using this model alone to base if a patient has pneumonia or not. I would not recommend the model because 8% is still a high amount of error, especially when dealing with people’s lives. I would suggest using the model to assist doctors in determining whether a patient has pneumonia or not.

6.2 Limitations

I think the project was limited by the data set. Augmenting the data is a good technique but if we had more pneumonia images, we might have been able to create a better model.

6.3 Future Work

I would recommend continuing to try different architectures. I did not try every regularizer or optimizer. Also, you could try testing models on non-augmented data to see if augmenting the data truly made an impact.