Classifying diseases from chest X-Ray scans using convolutional neural networks

1st Shaun Heffernan
Student
University of Florida
Gainesville, Florida
shaunheffernan@ufl.edu

***Abstract*—This document shows a analysis of chest Lung X-ray data. The goal is to train a convolutional neural network to classify specific diseases.**

I. Introduction

This project goes over implementation of a convolutional neural network to analyze a large dataset of Lung X-ray images to classify what type of diseases are present within the images. The goal of this is for doctors and hospitals to have more information when diagnosing a patient because some patterns with the diseases might possibly be overlooked. In this paper we will go over the architecture of the convolutional neural network, hyperparameter tuning, and the performance of this model on a held out dataset of X-Ray images.

II. Dataset and pre-processing

The dataset used is called ChestMNIST and it is based on the NIH-ChestXray14 dataset. It is made up of 112,120 X-Ray scans of various patients making up 14 distinct classes which we hope to correctly classify. The images are resized from the original 1024 x 1024 to 28 x 28, which could lead to less accurate predictions due to the loss of data from down sampling the images. This does lead to faster compute size because of the large amount of X-Ray scans and the shrunken size of each image from the original dataset. The diseases labeled in these images are Atelectasis, Cardiomegaly, Effusion, Infiltration, Mass, Nodule, Pneumonia, Pneumothorax , Consolidation, Edema (Oedema), Emphysema, Fibrosis, Pleural Thickening, and Hernia. Each X-Ray scan can have anywhere from zero diseases present to multiple diseases present in the same image, so this can be labelled as a multi-label binary-class classification task.

III. CNN architecture

For the convolutional neural network architecture, we first start with an input layer. The input layer is of shape 28x28 to ensure the data passed in is correct and matches the sizes of the images from the ChestMNIST dataset. The first convolution block is comprised of a 2D convolution which is followed by a 2D pooling and then a 0.2 dropout for regularization. The 2D convolution uses 32 filters with a kernel size of 3 because the images are small and we want to learn relations that are close together. This convolution also utilizes ReLU as the activation function to avoid the vanishing gradient problem and it uses padding same to ensure no data is lost. The pooling is used to down-sample the data so different relations can be learned in following convolutions.

For the second convolution block, a 2D convolution followed by a 2D pooling and a 0.2 dropout is used again. This 2D convolution uses 64 filters to learn more complex relations in the data then the first layer. It then has a pooling to down-sample the data for the final convolution to learn different patterns in the images. The convolution uses the same padding, kernel, and activation function for the same reasons listed in the first convolution block.

For the third convolution block, the same architecture is used as in the first two convolution blocks with one adjustment to the 2D convolution. This block utilizes 128 filters to once again find different relations in the data that are more complex and abstract than the first two convolutions with smaller filter size could not learn. The data is down-sampled twice from the original input which also allows for more abstract features to be learned across a wider area because the image is much smaller.

Then the data is flattened and goes through a dense layer of 128 filters with ReLU to extract all of the useful patterns learned in the convolutional blocks. There is then a large dropout of 0.4 to act as a regularizer and ensure the model is learning the proper weights throughout iterations. The final dense layer is of size 14 and uses sigmoid activation which is the probability of that image having each disease.

The overall architecture uses 3 similar convolution blocks with the filter count doubling each time. When paired with a max pooling layer this allows the first layer to find smaller more localized relations and the following layers to learn more abstract and farther apart relations in the images. The filter count is doubled because the dimension space is being reduced so this ensures the abstract features are able to be learned in the training process.

The model was then compiled using the binary cross-entropy loss function because each 14 classes are a binary classification problem, we want to see for each class is this disease present or not. Then the Adam optimizer is used with the learning rate tuned.

IV. Hyperparameter Tuning

For the optimizer on the convolutional neural network I chose to use the Adam optimizer because it is adaptive so it will find the best learning rates while performing gradient descent. The main hyperparameter for Adam is the initial learning rate because that is the baseline which then will get adapted while the model is learning. The default is 0.001, so I trained 5 models each on order of magnitude higher then the previous. Starting at 0.000001 and going up to 0.01, so it can cover the default learning rate while also testing new ones. For each model they were evaluated on the validation set with the metrics AUC, binary accuracy, and loss. The best performing model was the third on using the learning rate of 0.0001 and it had a binary accuracy of 0.9492, an AUC of 0.8273 and loss of 0.1628.

```
351/351 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - auc: 0.7348 - binary_accuracy: 0.9492 - loss: 0.1994
351/351 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - auc_1: 0.7401 - binary_accuracy: 0.9492 - loss: 0.2013
351/351 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - auc_2: 0.8273 - binary_accuracy: 0.9492 - loss: 0.1628
351/351 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - auc_3: 0.7892 - binary_accuracy: 0.9492 - loss: 0.1723
351/351 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - auc_4: 0.7491 - binary_accuracy: 0.9492 - loss: 0.1807
```

Fig. 1. Each models performance

V. Results

On the final model the training loss rapidly decreases for the first few epoch then slowly decreases until reaching the final loss value. For the validation loss it follows the same shape but it has spikes of increased loss on epochs where it learned bad relations. Through every epoch the validation loss is less then the training loss which is unexpected because the CNN is learning the training data so it is surprising the validation has a smaller loss. This is likely due to the heavy regularization placed on the model through various dropout layers which ensures the model is generalizable to new data, such as the validation set.

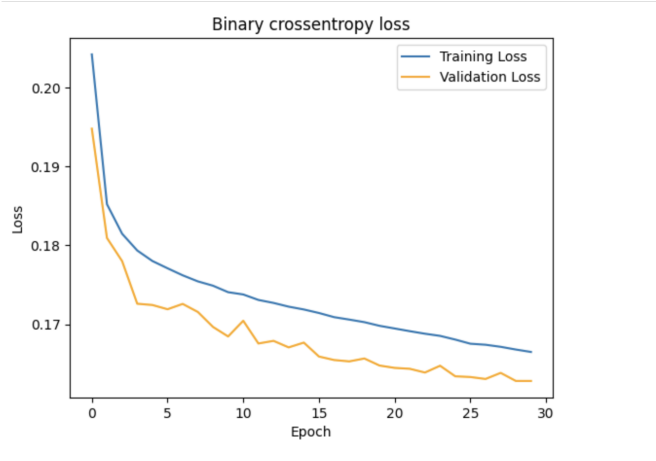

Fig. 2. Binary cross-entropy loss

On the final model the AUC increases quickly through the first few epochs and then slowly increases until the final epoch is reached. The validation AUC is higher throughout the entire training set which is due to the regularizer and generalization of the CNN. Ending with an AUC of 0.8273 for the validation set is good because it showcases that the model is able to tell the difference between positive and negative classes and isn't just consistently choosing negative. Consistently choosing negative would lead to an AUC around 0.5 and would still have a high accuracy because most images do not have diseases, but that would mean the model isn't actually learning anything of use.

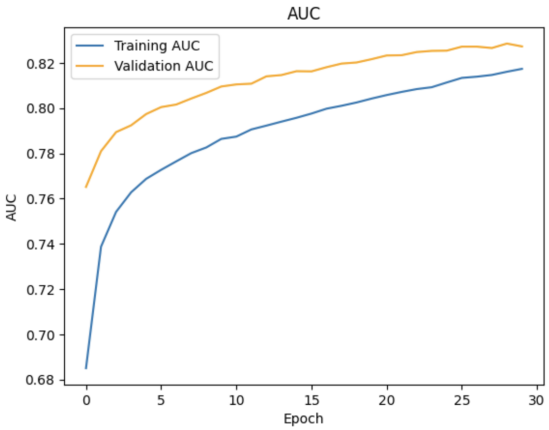

Fig. 3. AUC

On the final model the binary accuracy for the validation is consistently hovering around 0.95 and slightly increasing and decreasing through epochs. The training data has a significant jump in the first few epochs, but it isn't very large because the scale of the y axis is so small, it is just large relative to the rest of the graphs movement. It also then finds a value around 0.9485 where it slightly moves up and down.

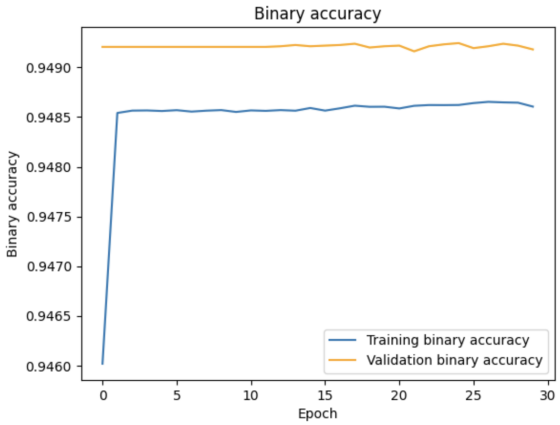

Fig. 4. Binary accuracy

The final model had a binary accuracy of 0.9475, an AUC

of 0.8235, and a loss of 0.1676 on the held out testing set. The AUC is good and shows that the model is not just predicting the dominant class each time which would be no disease for each disease. In the confusion matrices below we can see the breakdown of every classes predictions to fully evaluate if the model is accurate.

VI. Confusion Matrices

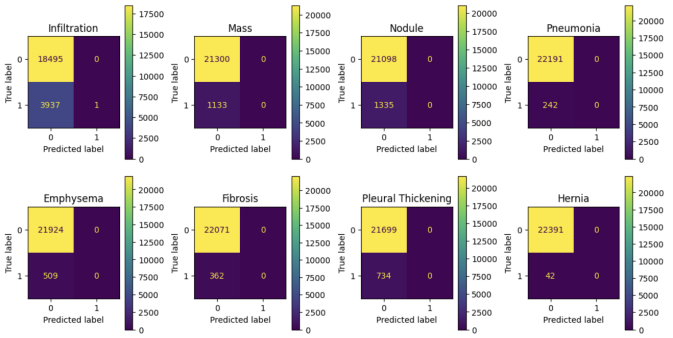

Fig. 5. Confusion Matrices

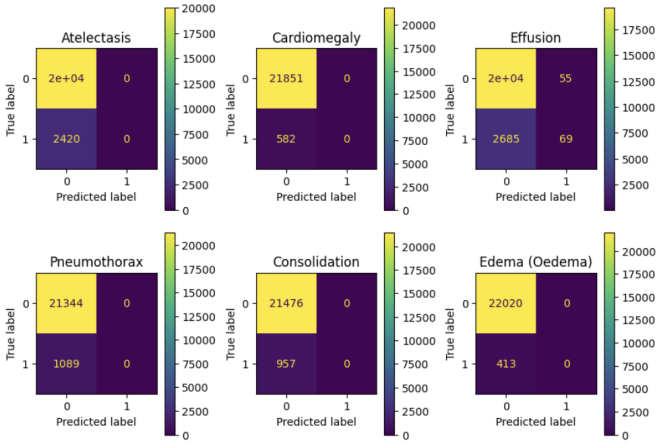

Fig. 6. Confusion Matrices

The confusions matrix show that the class imbalances heavily impacted the models performance. For almost all classes the models predicted that the disease was not present for every data sample, and this is largely due to the class imbalances. Most models had around 20,000 true negatives and and around 1,000 true positives, which means the model can still have very high accuracy even if it just predicts negative for every sample because the negative sample count is a full order of magnitude higher.

For the Infiltration confusion matrix, it was able to accurately classify 1 true positive which means that the model was learning some characteristics for the infiltration disease. But other then that it just predicted there was no disease for the rest of the test set.

For the Effusion the model was able to learn features that can predict whether effusion is present because it classified 69 true positives. But it also was the only model to have false positives with 55 false positives. This means the model started learning to predict the positive for this class but it was also misclassifying absent effusion as positive.

VII. Conclusion

The model has a very high binary accuracy, but when the confusion matrices are inspected the models performance seems very lackluster. This is because the datasets are so imbalanced that even when predicting all negative for every class, the accuracy is actually very high. Some ways to mitigate that in future model training are to implement class weights so the model can learn that the classes are imbalanced and it should take that in to account. For this problem with 14 output nodes it is harder to implement that, but a pre-processing step should be added to take into account the class imbalances when training a classifier in the future. The model still was able to learn some features and for effusion and infiltration it was able to predict the positive class. Another shortcoming of this model is the dataset which has very low quality images because 28 x 28 is only 784 pixels which is not a lot. Using the full dataset would be computational expensive, but it might create a model that could predict more true positive classes and learn the features from the images better. The AUC of 0.8273 is good and signals that the model isn't just predicting negative for each class. In production this model could be useful in some cases and it is shown to predict some true positives for the infiltration and effusion diseases. But it would be beneficial to take into account the class weights and larger datasets to create a better classifier.