Report of Tuberculosis X-Ray Image Classification

Last Name : Giri

First Name : Surya

CWID : 10475010

Course : EE628WS - Data Acquisition, Modeling and Analysis: Deep Learning

Purpose : Final Project

1. Introduction

Image classification is a Machine Learning Problem that is tasked with classifying images into classes. It is a problem that can be pretty effectively solved with neural networks, which is a part of Deep Learning, a subset of Artificial Intelligence. Machines can recognize and extract features from images thanks to deep learning. This implies that they can learn the characteristics to look for in images by analyzing a large number of them. Thus, these filters don't need to be manually modelled by programmers.

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. Pre-trained models are frequently utilized as the foundation for deep learning tasks in computer vision and natural language processing because they save both time and money compared to developing neural network models from scratch and because they perform vastly better on related tasks.

Mycobacterium tuberculosis, a species of bacteria, are the cause of tuberculosis, which most frequently affects the lungs. People with lung TB cough, sneeze, or spit, which causes TB to spread via the air. Just a few bacteria must be inhaled for someone to become ill. A person with advanced tuberculosis will have infected lungs and a cavity forming, as shown on a chest X-ray. Thus, we can use various image classification techniques to detect whether a person has tuberculosis or not. This is done by learning what tuberculosis affected lung x-rays look like using convolutional neural networks (CNN) to learn from tuberculosis affected and normal x-ray images. The model can then predict whether a person has tuberculosis or not by taking their x-ray as input and making a prediction. The more accurate the model, the better are the results.

2. Problem Formulation

Binary Classification:

Our problem is a binary image classification problem, meaning a classification model predicts whether an image belongs to either of two labels (1 or 0). In our case, the two target labels are 'Normal' and 'Tuberculosis'; where Normal is represented by 0 and Tuberculosis is represented by 1.

Cross-Entropy Loss:

To make this prediction, we train the model on the training set (80% in this project) to learn what images belong which one of the two labels (Normal and Tuberculosis). 10% of the data is used for validation while training and the remaining 10% is used for testing the predictions. The prediction is improved using a loss function which learnings and improves the difference between predicted and real results. However, since in our case the dataset is imbalanced, we use sample weighting for weights needed by the loss cross-entropy loss function. The cross entropy function is given by:

$$\mathcal{L}^{w}_{cross-entropy}(x) = -(w_p y \log(f(x)) + w_n(1-y)\log(1-f(x))).$$

Where, x is the input (images) and y is the target (labels Normal and Tuberculosis).

Sample Weighting

Since the dataset is imbalanced, meaning, there are significantly more images for normal than for tuberculosis. This will negatively affect our model if we use it as is since the contribution from tuberculosis will be less. Tuberculosis contribution is vital since it will train the model to identify tuberculosis images. Thus, we'll make contributions equal by doing, i.e.:

$$w_{pos} \times freq_p = w_{neg} \times freq_n,$$

This can be done by making:

$$w_{pos} = freq_{neg}$$

$$w_{neg} = freq_{pos}$$

(Reference: https://www.kaggle.com/code/sanphats/microcalcification-weighting-loss-dnn)

Output and Classification Report:

Due to the imbalance and sample weighting, we will train the model really well to predict only one class. Since our problem is binary classification, if we predict one of the classes well, then the other class can automatically be predicted as it will be the opposite of the predicted class, meaning anything that does not belong to the predicted label will belong to the other label. And because the contributions from both classes during training is equal, training for only label is equivalent This is especially true for Transfer Learning, where prediction of label can achieve really good accuracy with ease. Therefore, we only have one neuron in the dense output layer. (Reference: https://stackoverflow.com/questions/48218185/why-my-deep-learning-model-always-give-

an-output-of-1-class). The number of classes will still remain two and the output is in the form of a label prediction, i.e. 1 (Tuberculosis) or 0 (Normal). The classification report gives us a few evaluation metrics such as accuracy, f1 score.

3. Method

a. Load Dataset:
   Our dataset is in divided as metadata and images. The images and metadata have two categories: Normal and Tuberculosis. We simply import the metadata to start reading the actual images from the folders. There are 4200 images in our dataset.
b. Manipulate Data:
   The metadata is manipulated a bit for personal convenience and understanding. A column name is changed: File Name -> Image and a new column is added that indicates the state of the person whose x-ray the image belongs to and becomes the target label -> 'tb_status' (short for tuberculosis status).
c. Create Directories
   We create directories to store the normal and tuberculosis images for training, testing and validation. The code includes two functions, depending on whether the code is being run on Google Colab or Jupyter.
d. Split Data
   We split the data (images) into 3 categories: training, validation and testing. The ratio is 80:10:10. The training set needs to be larger because that is where the model learns and validation and testing are used to check the predictions. Validation set is used to evaluate predictions during training and testing set is used to evaluate predictions during testing.
e. Analyze Data
   The dataset was visualized and counted and it was observed that there is an imbalance in the dataset, i.e., there were more normal images than there were tuberculosis images. Out of 4200 images, only 700 images are Tuberculosis images, which is significantly less compared to 3500 Normal images. This means that only 16.67% of the database is Tuberculosis images and the rest 83.33% is Normal images.
f. Handle Data Imbalance
   The imbalance in the dataset must be handled since a tuberculosis classification model that is trained on very less tuberculosis images and significantly more normal images will be undertrained to classify tuberculosis images. We do this by using the concept of calculating sample weighting for loss where the contribution from both classes is the same. This makes sure that even though there is tuberculosis images in our training set, the contribution from it is the same as that of normal images during training.
g. Data/Image Preprocessing
   We do some image augmentation by passing the images through ImageDataGenerator. This augmentation allows us to take care of rotated or tilted images, images with inconsistent colors, RGB color scheme, set the pixel size etc.

h. Transfer Learning

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Fig 1. DenseNet architectures

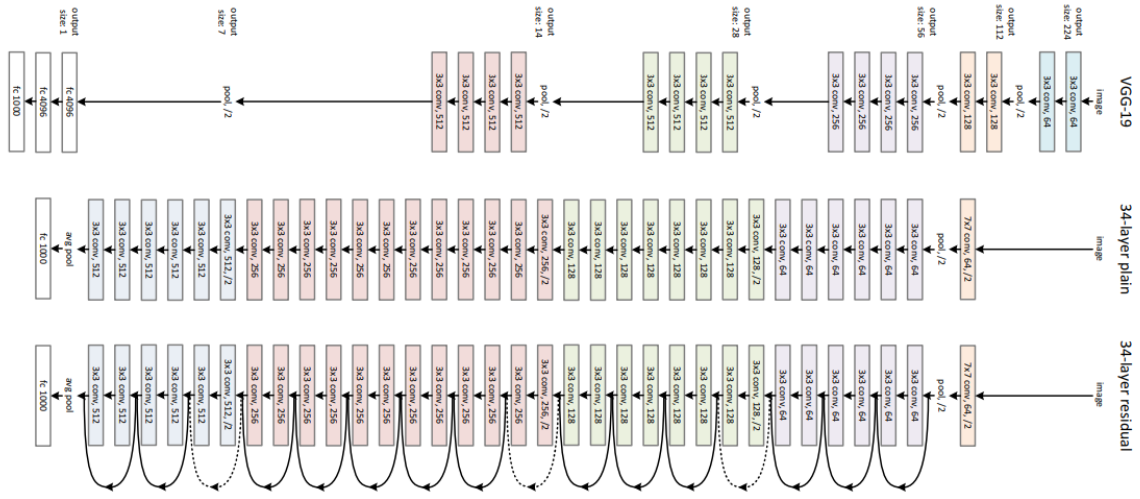

Fig 2. ResNet 34 architecture

i. DenseNet121

The table above (Fig 1.) provides a summary of the architecture used to build the ImageNet database. The number of pixels shifted across the input matrix called the stride. The filters are moved 'n' pixels at a time when the stride parameter is set to 'n' (the default value is 1). When analyzing the table using the DenseNet-121 architecture, we can observe that each dense block has a different number of layers (repetition), each

of which contains two convolutions: a bottleneck layer that is 1x1 in size and a convolution layer that is 3x3 in size. DenseNet-121 has the following layers:

- 1 7x7 Convolution
- 58 3x3 Convolution
- 61 1x1 Convolution
- 4 AvgPool
- 1 Fully Connected Layer

In the code, the final output shape is given as 1 since the ImageDataGenerator takes care of the classes as long as the shape is right, when number of classes is given as input. The number of classes is given as 2, and the output shape is 1 as the data is coupled together.

ii. DenseNet169

One of the models in the DenseNet family created for picture classification is the densenet-169 model. The size and precision of the densenet-169 model are the primary differences. Compared to the densenet-121 model's around 31MB size, the densenet-169 is bigger at just about 55MB. The writers changed them from their original Torch training into Caffe* format. On the ImageNet picture database, all of the DenseNet models have undergone pretraining. The input for the model is a blob made up of a single image that is 1x3x224x224 in BGR order. Before sending the picture blob into the network, the BGR mean values need to be removed as follows: [103.94, 116.78, 123.68]. Values must also be split by 0.017.

The object classifier output that is typical for the 1000 different classifications that correspond to those in the ImageNet database is what densenet-169 produces.

iii. ResNet50

ResNet-34 (Fig 2.), a version of the original ResNet design, which included 34 weighted layers. By utilizing the idea of shortcut connections, it offered a creative solution for expanding the number of convolutional layers in a CNN without encountering the vanishing gradient issue. A shortcut link turns a conventional network into a residual network by "skipping over" some layers.

The VGG neural networks (VGG-16 and VGG-19) served as the foundation for the regular network; each convolutional network has a 33 filter. A ResNet, on the other hand, is simpler and contains fewer filters than a VGGNet. In comparison to a VGG-19 Network's 19.6 billion FLOPs, a 34-layer ResNet can perform at 3.6 billion FLOPs, and a smaller 18-layer ResNet can accomplish 1.8 billion FLOPs.

The ResNet architecture abides by two fundamental design principles. First, regardless of the size of the output feature map, there are the same number of filters in each layer. Second, to preserve the time complexity of each layer even if the size of the feature map is half, it has twice as many filters.

i. Prediction and Evaluation

We make prediction using predict_generator() of our models, which allows us to evaluate the predictions by feeding the results of the function to sklearn's classification_report() functions. This gives us evaluation metrics for our predictions, which includes precision, recall, f1-score, support, accuracy, macro average and weighted average. We save the results of the predictions in a csv file that can be easily accessed from the root directory of the project. The results of the models are as follows:

```
1]: y_test = test_df['tb_status']
    print(classification_report(y_test, y_pred1))
    print("Accuracy of the Model 1:",accuracy_score(y_test, y_p

                  precision    recall  f1-score   support

             0.0       0.97      0.99      0.98       350
             1.0       0.95      0.87      0.91        70

        accuracy                           0.97       420
       macro avg       0.96      0.93      0.95       420
    weighted avg       0.97      0.97      0.97       420

    Accuracy of the Model 1: 97.14285714285714 %

2]: print(classification_report(y_test, y_pred2))
    print("Accuracy of the Model 2:",accuracy_score(y_test, y_p

                  precision    recall  f1-score   support

             0.0       1.00      0.29      0.45       350
             1.0       0.22      1.00      0.36        70

        accuracy                           0.41       420
       macro avg       0.61      0.64      0.40       420
    weighted avg       0.87      0.41      0.43       420

    Accuracy of the Model 2: 40.714285714285715 %

3]: print(classification_report(y_test, y_pred3))
    print("Accuracy of the Model 3:",accuracy_score(y_test, y_p

                  precision    recall  f1-score   support

             0.0       0.86      1.00      0.92       350
             1.0       1.00      0.16      0.27        70

        accuracy                           0.86       420
       macro avg       0.93      0.58      0.60       420
    weighted avg       0.88      0.86      0.81       420

    Accuracy of the Model 3: 85.95238095238096 %
```

Fig 3. Classification report for DenseNet121, DenseNet169 and ResNet50 respesctively.

```
result_densenet121 = pd.read_csv('predictions1.csv')
result_densenet121
```

	Image	Predictions
0	Normal-2072.png	Normal
1	Normal-1695.png	Normal
2	Normal-377.png	Normal
3	Normal-1896.png	Normal
4	Normal-3227.png	Normal
...	...	...
415	Normal-2109.png	Normal
416	Normal-3309.png	Normal
417	Normal-1053.png	Normal
418	Normal-2757.png	Normal
419	Normal-2565.png	Normal

420 rows × 2 columns

Fig 4. Predictions made by DenseNet121

```
result_densenet169 = pd.read_csv('predictions2.csv')
result_densenet169
```

	Image	Predictions
0	Normal-2072.png	Normal
1	Normal-1695.png	Tuberculosis
2	Normal-377.png	Tuberculosis
3	Normal-1896.png	Tuberculosis
4	Normal-3227.png	Tuberculosis
...	...	...
415	Normal-2109.png	Tuberculosis
416	Normal-3309.png	Tuberculosis
417	Normal-1053.png	Normal
418	Normal-2757.png	Tuberculosis
419	Normal-2565.png	Tuberculosis

420 rows × 2 columns

Fig 5. Predictions made by DenseNet169

```
result_resnet50 = pd.read_csv('predictions3.csv')
result_resnet50
```

	Image	Predictions
0	Normal-2072.png	Normal
1	Normal-1695.png	Normal
2	Normal-377.png	Normal
3	Normal-1896.png	Normal
4	Normal-3227.png	Normal
...	...	...
415	Normal-2109.png	Normal
416	Normal-3309.png	Normal
417	Normal-1053.png	Normal
418	Normal-2757.png	Normal
419	Normal-2565.png	Normal

420 rows × 2 columns

Fig 6. Fig 4. Predictions made by ResNet50

4. Experiments

The experiments and trials that were conducted during this project included a variety of changes, from the directory structuring, model comparison and implementation of original classification models such as Logistic Regression, albeit to poor results. The experiments and observations are as follows:

a. Data Manipulation: Changing columns names does not affect the predictions at all as long as proper structure is maintained, and hence can be freely explored without dire consequences.

b. Creating Directories: Directories work differently depending on the platform and the python notebook execution environment. While this project was conducted on a Window's system, there are two functions to create directories, one for Google Colab and one for Jupyter. The two python notebook environments access directories differently, specifically in the usage of '/' in Colab to separate folders and '\' to separate folders on Window's (Jupyter).

c. Splitting Data: Splitting data and generating the images in this project is taken care of using ImageDataGenerator module from Tensorflow. Furthermore, we specifically use flow_from_dataframe() module instead of flow_from_directory(). We use this to do the data augmentation we mentioned before, which is what allows us to get such good results even with base models of ResNet50 and DenseNet121. However, images can also be converted to their RGB values using Image.convert() module of Pillow. This then allows us to put it into a numpy array and freely use it for whatever model we like.

d. Models: Base DenseNet121 consistently gives good results on this data even without the need for callbacks. ResNet shows great results during training however, overall estimation still puts it below DenseNet, implying it can be further optimized. DenseNet169 shows the least amount of improvement even with hyperparameter tuning. This might be a result of too many Dense layers. Further investigation on this topic is required.

5. Conclusion

During the drafting of this report, the results observed from the three models: DenseNet121, DenseNet169 and ResNet50 were compared, and it was observed that the base model of DenseNet121 gives us the best results, with an accuracy score of 97.14%, followed by ResNet50 giving us an accuracy of 85.95% and finally the worst accuracy achieved by DenseNet169. When comparing the base models, DenseNet121 seems to be the best suited model for image classification of this dataset, i.e., Tuberculosis (TB) Chest X-ray Database. ResNet also shows very promising results during the training phase, therefore more ResNet models can be explored in the future.