Chest X-Ray Image Classification with Deep Learning

Author: Kamile Yagci

Blog URL: https://kamileyagci.github.io/

Overview

In this study, I analyze the Chest X-ray Images of pediatric patients in order to identify whether or not they have pneumonia. I will apply Image Classification with Deep Learning using the Convolutional Neural Networks (CNN).

Business Problem

The Baylor Medical Center hired me to improve the accuracy in pneumonia diagnosis on pediatric patients. The study will use the chest X-ray Images of pediatric patients and do the image classification identifying whether the X-ray shows pneumonia or not. The outcome of this study will not only be used at Baylor Centers, but also in partner medical clinics in Africa, where the medical staff is limited. The automated identification system will provide early diagnosis of pediatric patients, so the treatment can start as soon as possible. Moreover it will decrease the human errors in pneumonia diagnosis.

Outline

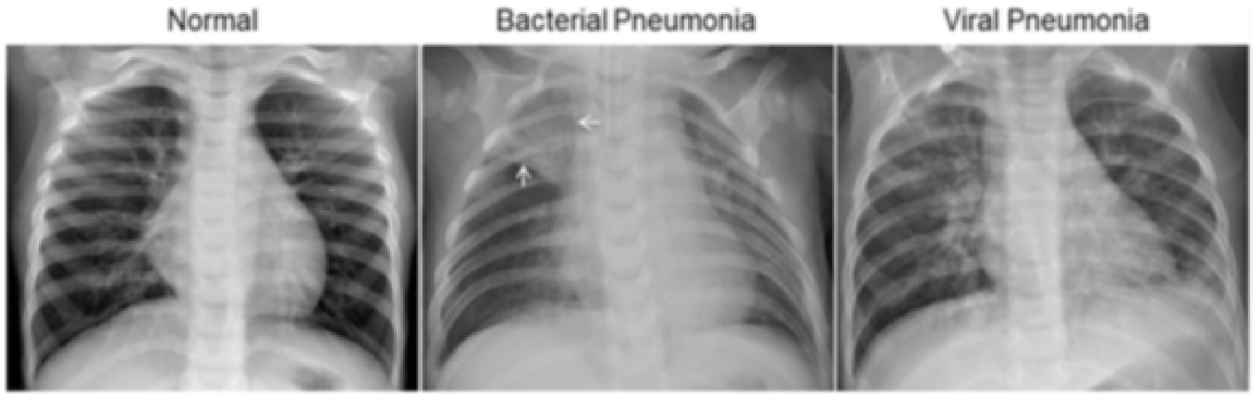
- Overview
- Business Problem
- Data
- Methods
- Analysis and Results
  - Baseline Model
  - Baseline + Regularization Model
  - Baseline + Dropout Layers Model
  - Baseline Model + Augmentation
  - New Model with more Layers and optimizer='Adam'
  - Train on whole dataset, Baseline Model with optimizer='Adam'
  - Final Model
- Conclusion
- Next Steps

Data

The dataset 'Chest X-Ray Images (Pneumonia)' is downloaded from Kaggle.

The data contains the chest x-ray images of pedicatric patients from one to five years old, collected at Guangzhou Women and Children’s Medical Center.

The diagnosis on chest x-ray images have three types: Normal, Bacterial pneumonia and Viral pneumonia. The image below show a sample for each type.



The dataset contains 5856 chest x-ray image files. They are labeled in two categories: NORMAL and PNEUMONIA. The number of NORMAL image samples is 1583, and the number of PNEUMONIA image samples is 4273. Bacterial pneumonia and Viral pneumonia samples are combined under label PNEUMONIA.

The original dataset downloaded from Kaggle distributed the data in three directories: Train, Validation and Test. However, the number of files in validation directory was small and insufficient. Thereofore, I redistributed the data with ~70% train, ~15% validation and 15% test. The table shows the number of images files in each directory per label.

	Normal	**PNEUMONIA**	**ALL**
Train	1107	2991	4098
Validation	238	641	879
Test	238	641	879
All	1583	4273	5856

Methods

In this study, I applied Image Classification with Deep Learning using the Convolutional Neural Networks (CNN) on chest x-ray images. Since the data labels has two classes, NORMAL and PNEUMONIA, this is a binary image classification.

Analysis and Results

My main challenge in this study is the computing power, since I am using an old Macbook Pro. When training with various models, I shouldn't overload my computer and need to keep the training time short. For this purpose, I train the model with the following settings:

- Use a subset of data during training: 960 images (23%) for training and 320 images (36%) for validation. I controlled the number of images by 'steps_per_epoch' parameter while training.
- Use small target_size when loading the images: (64x64)
- Use relatively small batch_size due to small dataset when loading the images: 20 (batch_size=20 yields better performance than batch_size=32 with subset data, according to my analysis)
- Run the training with small number of epochs: 30

With these parameter selections, model training takes about 10 minutes.

The whole training and testing data is used for model evaluations.

After final model is determined, I run the final model on whole dataset.

Baseline Model

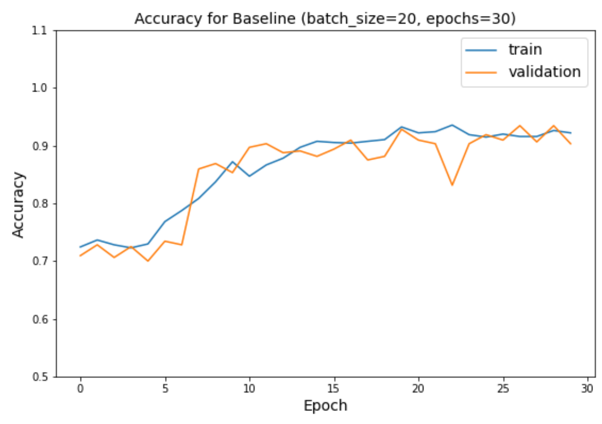
The features of my baseline model:

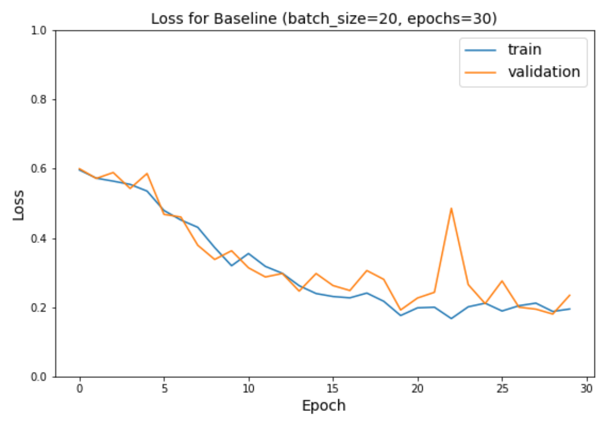
- 6 CNN layers
- 2 Dense layers
- Activation function (except out put layer): 'relu'
- Output layer activation faunction: 'sigmoid'

Compile with:

- loss='binary_crossentropy',
- optimizer= 'sgd'
- metrics='acc'

Accuracy and Loss Graphs for Training and Validation:





Evaluation Results:

	Accuracy	Loss
Train	0.9370	0.1696
Test	0.9295	0.1855

Comments:

- Overall performance is good for baseline model.
- There is a slight overfitting.

Baseline + Regularization Model

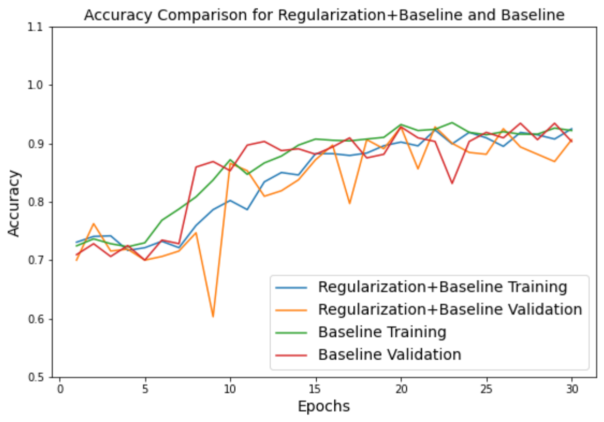
Even though, the overfiting is small when running on subset data, it can increase when running on whole data.

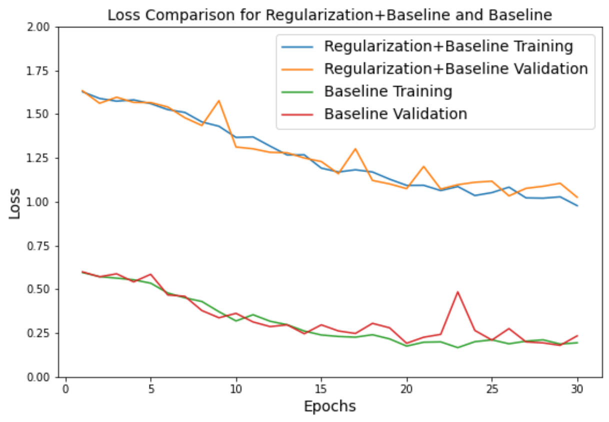
Therefore, I have added L2 Regularization to my baseline model.

Evaluation Results:

	Accuracy	Loss
Train	0.9212	0.9938
Test	0.8976	1.0141

Comparison of 'Baseline + Regularization Model' and 'Baseline Model'.





Comments:

- Regularization did improve the slight overfitting.
- The model performance with regularization is worse than the baseline model.
- The accuracy in testing decreased significantly.
- The loss values incresaed as a result of the regularization penalty on loss function.
- I will not use Regularization in my model.

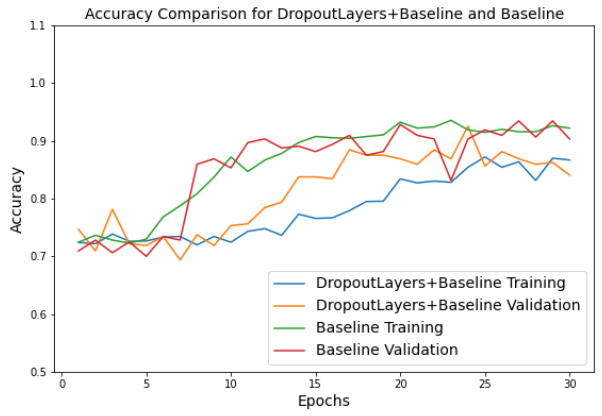
Baseline + Dropout Layers Model

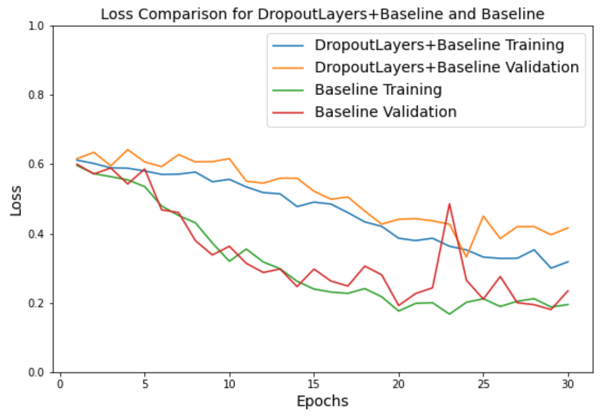
I have added Dropout Layers to the Baseline Model.

Evaluation Results:

	Accuracy	Loss
Train	0.8694	0.3846
Test	0.8464	0.4017

Comparison of 'Baseline + Dropout Layers Model' and 'Baseline Model'.





Comments:

- The result with Droput is a worse than the baseline model.
- The accuracy in training and testing decreased significantly.
- Interestingly, the validation performance looks better than training.
- I will not use Dropout Layers in my model.

Baseline Model + Augmentation

I tried different augmentation parameter for training model one by one with several values:

- horizontal_flip
- brightness_range
- width_shift_range
- height_shift_range
- rotation_range
- zoom_range
- shear_range

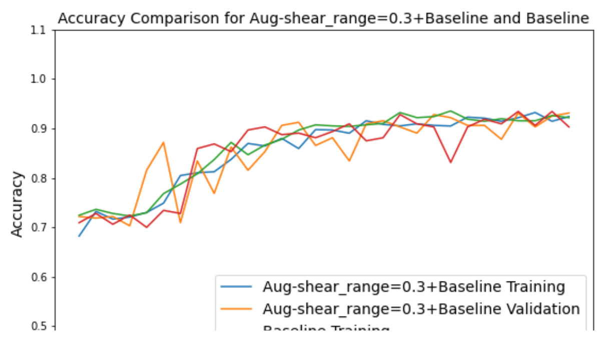
Unfortunately, the augmentation is not helping. Most of the parameters caused a lower performance. Only shear_range=0.3 keeps the performance of the baseline model.

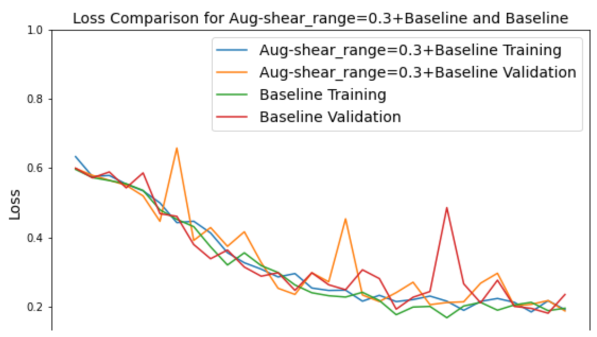
I ran the training with with shear_range=0.3 and comapred it to the baseline mode.

Evaluation Results:

	Accuracy	Loss
Train	0.9356	0.9356
Test	0.9295	0.1824

Comparison of 'Baseline + Augmentation(shear_range=0.3)' and 'Baseline Model'.





README.md

Comments:

- Augmentation with shear_range=0.3 didn't not improve the model performance. It is almost same as the baseline model results.
- I will not use augmentation in my model.

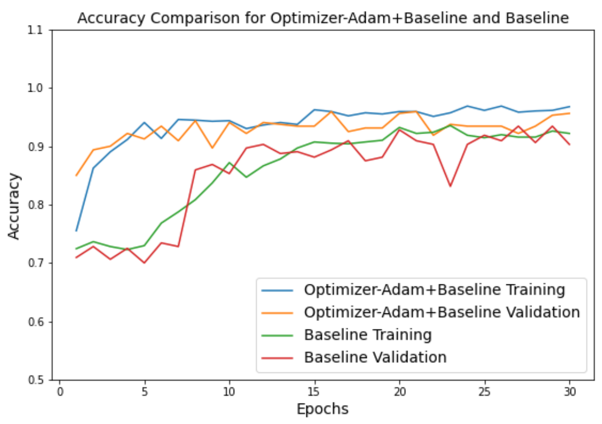
Baseline Model with Optimizer = 'Adam'

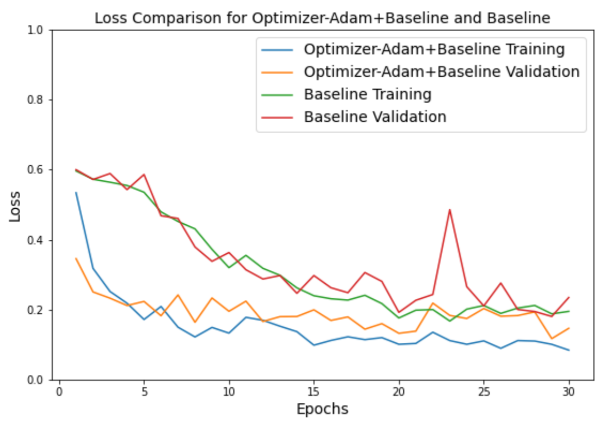
I tried a different optimizer = 'Adam' when compiling the model.

Evaluation Results:

	Accuracy	Loss
Train	0.9736	0.0774
Test	0.9545	0.1214

Comparison of 'Baseline Model with Optimizer='Adam'' and 'Baseline Model'.





Comments:

- The optimizer 'Adam' significantly improved the model performance compared to optimizer 'sgd'.
- The accuracy increased and loss decreased in training, validation and testing samples.
- There is slight overfitting, but it increases as the epoch number increases.
- I will use optimizer 'adam' for my model.

Note: I run this model with L2 regularization, but didn't improve the performance.

New Model with more Layers and optimizer='Adam'

The features of the new model:

- 8 CNN layers
- 3 Dense layers
- Activation function (except out put layer): 'relu'
- Output layer activation faunction: 'sigmoid'

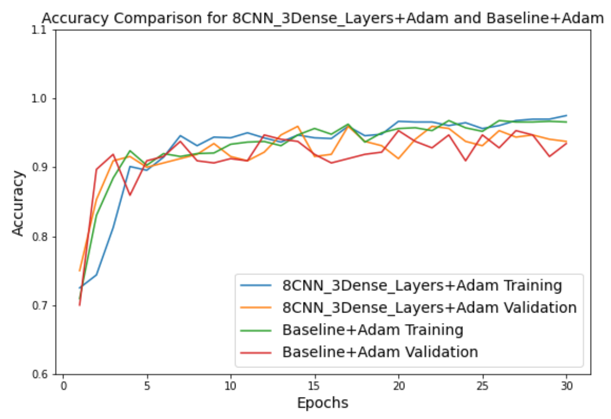
Compile with:

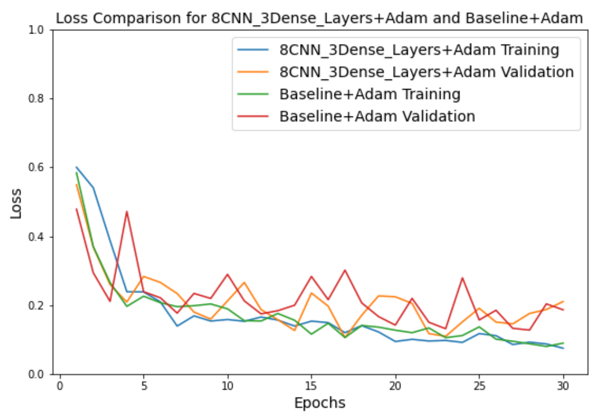
- loss='binary_crossentropy',
- optimizer= 'adam'
- metrics='acc'

Evaluation Results:

	Accuracy	Loss
Train	0.9678	0.0806
Test	0.9522	0.1536

Comparison of 'New Model (8CNN+3Dense) with optimizer='Adam'' and 'Baseline with optimizer='Adam'.





Comments:

- The new model with 8 CNN and 3 Dense layers didn't perform better than our Baseline Model.
- The testing accuracy is same, but testing loss is a bit larger.
- I will not use the new model, and keep using baseline model with optimizer='adam'.

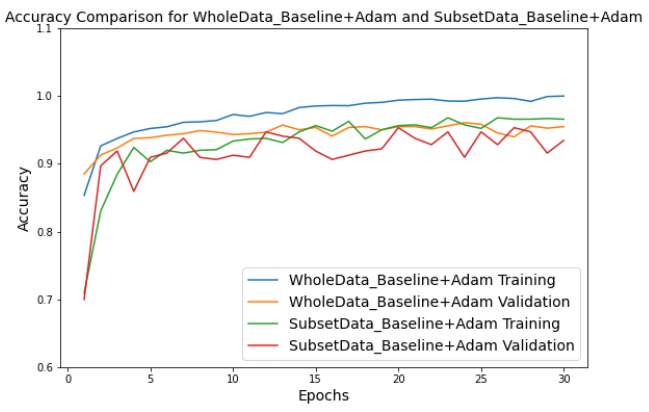
Train on whole dataset , Baseline Model with optimizer='Adam'

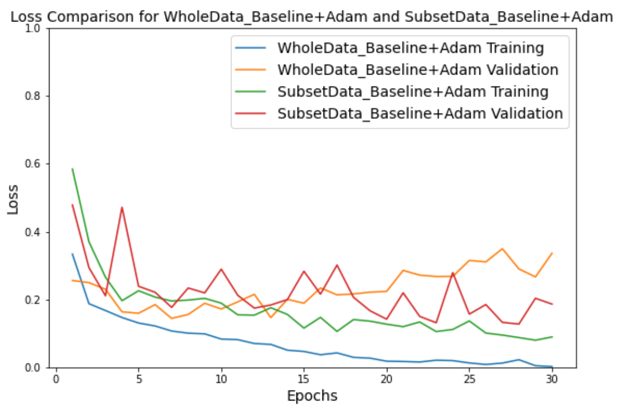
I now run baseline model with optimizer='adam' on whole training and validation data.

Evaluation Results:

	Accuracy	Loss
Train	1.000	0.0006
Test	0.9590	0.3068

Comparison of 'Whole Data on Baseline 'Adam' and 'Subset Data on Baseline 'Adam''.





Comments:

- Running on whole data did improve the model performance very little.
- The testing accuracy (0.9590) is very slightly higher than the accuracy from the subset data (0.9545).
- However, testing loss significantly increasing on whole data (0.1214 -> 0.3068).
- Large dataset increased overfitting. The training accuracy is 1.00. The overfitting is especially observed in loss curve.
- With the larger sample size, the fluctuations in the accuracy and loss curves decrease, especially on the validation data.
- I choose to train my model on subset of data, not on whole data.

** With L2 Regularization**

I trained this model with reqularization on whole data. Even though the overfitting decreased, the accuracy didn't improved.

	Accuracy	**Loss**
Train	0.9432	0.1811
Test	0.9352	0.2043

Final Model

The model with best performance is "baseline model with optimizer='adam'". It is my final model.

The structure of the model, and compile and fit information are shown below.

The features of final model:

- 6 CNN layers
- 2 Dense layers
- Activation function (except out put layer): 'relu'
- Output layer activation faunction: 'sigmoid'

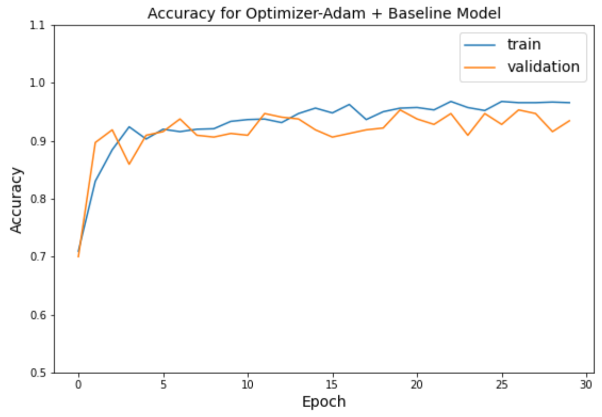
Compile with:

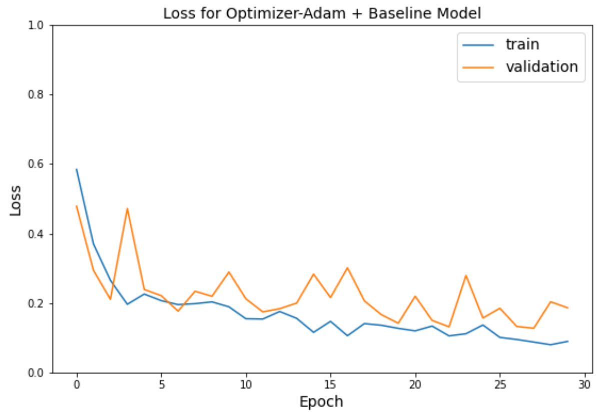
- loss='binary_crossentropy',
- optimizer= 'adam'
- metrics='acc'

Train on

- subset of data (~20% of whole data)
- epochs = 30
- batch_size = 20

Accuracy and Loss Graphs for Final Model:





Evaluation Results for Final Model:

	Accuracy	Loss
Train	0.9736	0.0774
Test	0.9545	0.1214

Conclusion

- The final model has very good performance.
- The model classifes NORMAL and PNEUMONIA chest-ray images with 95.45% accuracy.
- The overfitting on the model is small, but still exists. The L2 Regularization didn't reduce the overfitting.
- The final model uses subset of data for training.
- The model performance on subset data is comparable to whole data.
    - The accuracy is slightly better on whole data
    - The loss is larger on whole data
    - The overfitting is larger on whole data
- Due to computing power limitations, I couldn't do tuning on whole data.
- Since baseline model (with optimizer 'sgd' or 'adam') has initially good performance on subset data, there wasn't much room for improvement. However, the

performance might be different on whole data. Augmentation and new models with added layers and larger units/nodes could function better on whole data.

Next Steps

- Even though, the model performance is good. There is always a room for improvement.
- In this study, my main limitation was computing power.
    - I did most of the model training in a subset of data. (~20% of training and validation)
    - Used (64x64) images instead of (128x128) or (256x256).
    - Used batch_size=20 instead of 32, beacuse of the small sample size.
    - Ran with a small epoch number(30).
- I would like run all steps of this analysis with whole dataset on a powerful computer, or grid system. In my analysis, I did run on the whole data just once. And it didn't give the best performance, mainly due to overfitting. There might also be other issues. I didn't have power to investigate and tune the model with whole dataset.
- I would like to study augmentation more. Theoratically, it should improve the model performance. However, it didn't on my subset data. Why? Would it yield better performance in whole data? I would like to investigate this more.