A Report on Chest Opacity Detection in images using a Deep Convolutional Neural Network

Gloria Ainebyona
Makerere University
College of Computing and Information Sciences
School of Computing and Informatics Technology
Student No:2100702283,Reg No:2021/HD05/2283U
gloria.ainebyona@students.mak.ac.ug

Abstract

Given deep learning as the current state of the art technology in artificial intelligence, image classification is one of its applications.
Image classification assigns labels to images based on a given set of characteristics.
In this report, I present results from a Convolutional Neural Network model that I trained to classify unknown images and results on how the model could generalise on the a dataset that was acquired under different conditions.

1 Methodology

1.1 Data preprocessing

I used the the keras preprocessing library for loading image dataset from a directory on google drive. I used this method becasuse it helps to directly feed the data into the keras preprocessing layers and generates a tf.data.Dataset from image files in a directory on google drive.
I split the known images into training samples of 523, validation samples of 92 and testing samples of 100 images.
I set the labels to “Inferred” so as to be generated in form of a directory structure. Labels were also set into an integer format. Color mode was set to grayscale, since the images provided were grayscale. The batchsize considered was 29 and so the model could load the images in batches of 29 images. Image size was set to 200*200 as this was giving better results compared to the original size of 395*488.
A seed number for image shuffling and transformations was set.
I also set the crop aspect ratio to True such that the model could resize the images without the image aspect ratio distortion.

```
ds_test=tf.keras.preprocessing.image_dataset_from_directory(test_dir,
labels='inferred',
label_mode="int",
color_mode='grayscale',
batch_size=batch_size,
 image_size = (img_height,img_width),
 crop_to_aspect_ratio=True,
  #interpolation="mitchellcubic" ,
  seed=123,

)
```

```
Found 100 files belonging to 2 classes.
```

```
ds_train=tf.keras.preprocessing.image_dataset_from_directory(train_dir,
labels='inferred',
label_mode="int",
color_mode='grayscale',
batch_size=batch_size,
 image_size = (img_height,img_width),
 shuffle=True,
 seed=123,
 validation_split=0.15,
 subset="training",
  crop_to_aspect_ratio=True,
 # interpolation="mitchellcubic",
```

1.1.1 Rescaling and normalisation

I performed rescaling on the data in order to convert the input images into a range of pixel intensity values that would fall into the normal distribution in order to make efficient computations.
I also used the AUTOTUNE function from the tf.data library in order to dynamically tune the prefetching of the data at runtime.

```
[ ]  normalization_layer = tf.keras.layers.Rescaling(1./255)

     normalized_ds = ds_train.map(lambda x, y: (normalization_layer(x), y))
```

```
[ ]  AUTOTUNE = tf.data.AUTOTUNE

     ds_train = ds_train.cache().prefetch(buffer_size=AUTOTUNE)
     ds_validation = ds_validation.cache().prefetch(buffer_size=AUTOTUNE)
```

1.1.2 Data Augmentation

I also performed data augmentation on the data so as to artificially increase the dataset for the model training.
I used the random brightness function on the training data in order to increase some bit of brightness for images that did not appear clear.
I also applied RandomFlip, RandomRotation and RandomZoom from the keras Sequental class onto the images. This would also contribute on the model's generalisation performance.

```
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   1)),
    layers.RandomRotation(0.7),
    layers.RandomZoom(0.4),
  ]
)
```

1.2 Model building

I used a sequential model which helps to build the model layer by layer.
I started with a convolution 2D with 16 nodes which I kept increasing for the next hidden layers up to 512 nodes.
A 3*3 kernel size for the filter matrix which helps to support the convolution though image smoothing and edge detection.
I set padding to "same" as this helps to preserve the original size of the image when applying a convolutional filter. It also supports the filter to perform full convolutions on the edge of the pixels.
I used an activation function 'relu' which helps to prevent oxponential growth in the computation required to operate network. I also used a MaxPooling layer which helps to remove the darker or unclear pixels from the output image of the previous layer.
I applied a dropout layer in order to prevent overfitting. This randomly drops the nodes in order to only keep the optimal nodes.
I also added a flatten layer in order to convert the output image from a 2D to 1D.
In the dense layer, I used a sigmoid activation functions as this is already normalised and it is able to make clear predictions.

```
model=keras.Sequential([
    data_augmentation,
    layers.Input((200,200,1)),
    layers.Conv2D(16,3, padding='same', activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(32,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(64,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(128,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(256,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(512,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Conv2D(512,3, padding='same',activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.4),
    layers.Flatten(),
    layers.Dense(2),
    layers.Dense(512, activation='sigmoid'),
```

1.3 Model compilation

I used RMSprop as the optimiser for minimising the loss function. This is robust in handling stochastic objectives.
Sparse categorical entropy was used to help in calculating the crossentropy loss. The model was then

compiled on the accuracy metric. Model fitting on the training data was run for 30 epochs with batch size of 29 images.
It was then evaluated on the validation data and it achieved accuracy of 0.75.

```
model.compile(
    optimizer='RMSProp',
    loss=[
        keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    ],
    metrics=["accuracy"],
)
history= model.fit(ds_train,epochs=30,batch_size=29,validation_data=ds_validation,verbose=2)
```

```
19/19 - 1s - loss: 0.4751 - accuracy: 0.7725 - val_loss: 0.4538 - val_accuracy: 0.8478 - 625ms/epoch - 33ms/step
Epoch 25/30
19/19 - 1s - loss: 0.4715 - accuracy: 0.7897 - val_loss: 2.0738 - val_accuracy: 0.5543 - 620ms/epoch - 33ms/step
Epoch 26/30
19/19 - 1s - loss: 0.4273 - accuracy: 0.8145 - val_loss: 1.3285 - val_accuracy: 0.5761 - 624ms/epoch - 33ms/step
Epoch 27/30
19/19 - 1s - loss: 0.3923 - accuracy: 0.8184 - val_loss: 1.0408 - val_accuracy: 0.6413 - 631ms/epoch - 33ms/step
Epoch 28/30
19/19 - 1s - loss: 0.4869 - accuracy: 0.7706 - val_loss: 1.3290 - val_accuracy: 0.5543 - 622ms/epoch - 33ms/step
Epoch 29/30
19/19 - 1s - loss: 0.4290 - accuracy: 0.8050 - val_loss: 0.6840 - val_accuracy: 0.7609 - 620ms/epoch - 33ms/step
Epoch 30/30
19/19 - 1s - loss: 0.4121 - accuracy: 0.8069 - val_loss: 0.7377 - val_accuracy: 0.7500 - 625ms/epoch - 33ms/step
```

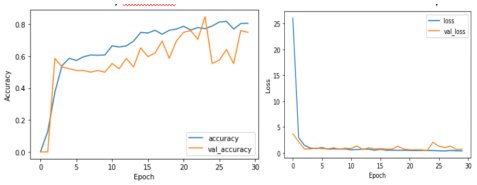
```
model.evaluate(ds_validation, verbose=2)
```

```
4/4 - 0s - loss: 0.7377 - accuracy: 0.7500 - 61ms/epoch - 15ms/step
[0.7377247214317322, 0.75]
```

2 Results

The results from the trained model were visualised as below.

2.1 Accuracy and loss curves



2.2 Model prediction on test set

The model was able to predict 50 true positives, 25 false positives, 0 false negatives and 25 true negatives. And achieved an accuracy of 0.75.

```
[ ] from sklearn.metrics._plot.confusion_matrix import confusion_matrix
    confusion_matrix(predicted_labels, correct_labels)
```

```
array([[50, 25],
       [ 0, 25]])
```

Classification report for performance of the model on the test set for dataset1 (known images)

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(
      correct_labels,
      predicted_labels,


    ))
```

```
              precision    recall  f1-score   support

           0       0.67      1.00      0.80        50
           1       1.00      0.50      0.67        50

    accuracy                           0.75       100
   macro avg       0.83      0.75      0.73       100
weighted avg       0.83      0.75      0.73       100
```

2.3 Model prediction on unknown images

The model predicted 46 true positives and 6 false negatives, with an accuracy of 0.88.

```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
confusion_matrix(predicted_labels, correct_labels)
```

```
array([[44,  0],
       [ 6,  0]])
```

```
from sklearn.metrics import classification_report
print(classification_report(
  correct_labels,
  predicted_labels,


))
```

```
              precision    recall  f1-score   support

           0       1.00      0.88      0.94        50
           1       0.00      0.00      0.00         0

    accuracy                           0.88        50
   macro avg       0.50      0.44      0.47        50
weighted avg       1.00      0.88      0.94        50
```

2.4 Model predictions on dataset 2

The model predicted 94 true positives, 62 false positives, 18 false negatives and 50 true negatives, with an accuracy of 0.64.

```
[ ] from sklearn.metrics._plot.confusion_matrix import confusion_matrix
    confusion_matrix(predicted_labels, correct_labels)
```

```
array([[94, 62],
       [18, 50]])
```

```
[ ] from sklearn.metrics import classification_report
    print(classification_report(
      correct_labels,
      predicted_labels,


    ))
```

```
                precision    recall  f1-score   support

             0       0.60      0.84      0.70       112
             1       0.74      0.45      0.56       112

      accuracy                           0.64       224
     macro avg       0.67      0.64      0.63       224
  weighted avg       0.67      0.64      0.63       224
```

I also experimented the images on transfer learning with VGG19 and Xception models. But these did not perform well as shown below.

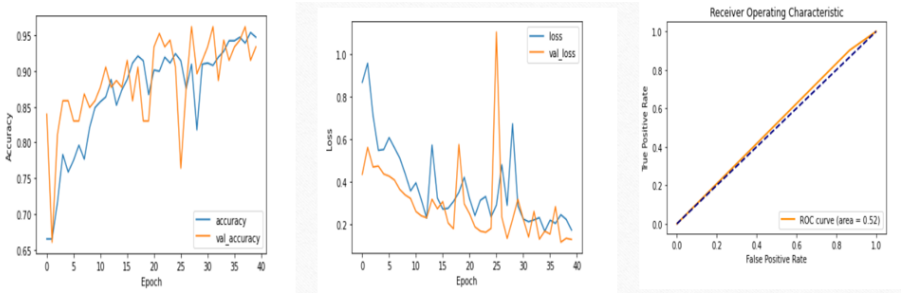


Figure 1: vgg19 accuracy, loss and ROC curves

```
print(f"Accuracy: {accuracy_score(test_ds.classes, pred_classes)}")
print(f"Precision: {precision_score(test_ds.classes, pred_classes)}")
print(f"Recall: {recall_score(test_ds.classes, pred_classes)}")
print(f"F1_Score: {f1_score(test_ds.classes, pred_classes)}")
print(f"AUC: {roc_auc_score(test_ds.classes, pred_classes)}")
```

```
Accuracy: 0.5
Precision: 0.5
Recall: 0.9107142857142857
F1_Score: 0.6455696202531646
AUC: 0.5
```

Figure 2: Xception model results.

```
Accuracy: 0.5
Precision: 0.5
Recall: 0.8839285714285714
F1_Score: 0.6387096774193548
AUC: 0.5
```

Figure 3: vgg19 model results.

3 Conclusion

The sequential model performed better than the rest of the models.
The model performance on dataset two was constantly poor. This is because the images were taken under the different conditions and so could not perform best to expectation.

References
1.Basic classification: Classify images of clothing, “Basic classification: Classify images of clothing | TensorFlow Core,” TensorFlow, 2017. https://www.tensorflow.org/tutorials/keras/classification.
2.N. Kafritsas, “Create Image Classification Models With Tensorflow in 10 minutes,” Medium, Aug. 25, 2022. https://towardsdatascience.com/create-image-classification-models-with-tensorflow-in-10-minutes-d0caef7ca011.
3.“Transfer learning with a pretrained ConvNet | TensorFlow Core,” TensorFlow. https://www.tensorflow.org/tutorials/images/transfer$_{l}earning$