Topics in Smart Health Informatics

Assignment 3

Komal Barge (1095709)

Given dataset has 1500 CT scan images of various parts such as abdomen, chest, and head which we have treated as 0, 1, 2 respectively. This assignment is completely based on feature extraction from convolutional neural network and modelling it in simple classification techniques like K nearest neighbor and Random forest. The real quality of machine learning model comes from extensive feature engineering than from the modeling technique itself. While specific machine learning methods may work best sometimes, but still features are the critical component of any image classification task.

This CT scan image data has dimensions of 64x64 and it is loaded into python notebook using PIL.Image library. By using simple String.split technique, we figured out the labels for each image and resized an image into 32x32 dimensions according to given task. Our dataset is kind of a small dataset, so to introduce the variability in the dataset, we have used imageDataGenerator. Dataset might have images taken in a limited set of conditions, but we might fall short in a variety of conditions that we do not account for. So, to get more data, we have done some minor alterations to existing dataset using data augmentation. These alterations include flipping the image horizontally, scaling, rotating, and zooming in etc. These are some preprocessing steps that we have used for preparing the dataset.

The main convolutional neural network that we have modelled has 3 Conv2D, 1 BatchNormalization, 3 MaxPooling2D, 1 Dropout and 2 Dense layers. We have used 'adam' as an optimizer and 'categorical_crossentropy' as a loss function because of 3 class labels. You can see the model used for extracting features in image 1.

Layer (type)	Output Shape	Param #
conv2d_input (InputLayer)	[(None, 32, 32, 1)]	0
conv2d (Conv2D)	(None, 30, 30, 25)	250
batch_normalization (BatchNo	(None, 30, 30, 25)	100
max_pooling2d (MaxPooling2D)	(None, 15, 15, 25)	0
dropout (Dropout)	(None, 15, 15, 25)	0
conv2d_1 (Conv2D)	(None, 14, 14, 16)	1616
max_pooling2d_1 (MaxPooling2	(None, 7, 7, 16)	0
conv2d_2 (Conv2D)	(None, 6, 6, 8)	520
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 8)	0
flatten (Flatten)	(None, 288)	0
extractedFeature (Dense)	(None, 50)	14450

Total params: 16,936
Trainable params: 16,886
Non-trainable params: 50

Image 1: Extracted feature CNN architecture

After training the CNN model with augmented images to get the updated weights and extract the features from intermediate dense layer with feature size 50. So, we extracted the features for original images for training and testing. We have trained our algorithms on the original images' features and augmented images' features.

Therefore, there are two ways to train our algorithm, either with data augmented images or original images.

1) **Extracting features with data augmented images with KNN and Random forest model:**
   If we use K nearest neighbor algorithm (KNN) on these extracted features, we get optimal k value as 5 which gives accuracy performance around 34.5%. you can see the variation in performance metrics with respect to k values of [3,5,7,9] in image 2 which is clearly underfitting the model.

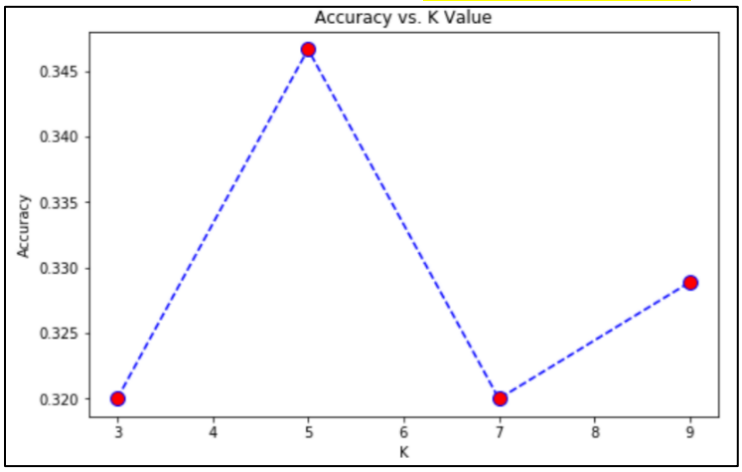

Image 2. Variation in performance with given k values.

We are training our KNN model using augmented images features(extracted from CNN architecture) which I think is very complex for such a simple algorithm to understand, that's why it is giving a poor performance on testing images (image 3) as well as training images (image 4).

```
showClassficationResults(y_test,loaded_knn_pred)
```

	precision	recall	f1-score	support
0.0	0.34	0.48	0.40	148
1.0	0.30	0.33	0.31	141
2.0	0.46	0.24	0.32	161
accuracy			0.35	450
macro avg	0.36	0.35	0.34	450
weighted avg	0.37	0.35	0.34	450

```
[[71 55 22]
 [71 46 24]
 [68 54 39]]
```

Image 3. Confusion matrix for augmented images in KNN algorithm.

```
trainpred = bestModel.predict(train_features)
accuracy_score(y_train, trainpred)
```

```
0.5676190476190476
```

Image 4. KNN training accuracy for augmented images.

If we use the Random forest algorithm with randomized search on the same augmented images where we have tuned 3 hyperparameters i.e. max features, number of trees and maximum depth in tree. We face the problem of overfitting, where training accuracy is around 99.71% (image 5) and testing accuracy is approximately 33% (image 6) even after cross validation in the search.

As, the dataset is small, and we use data augmentation, we get many degrees of freedom in model selection like lots of features, tuning of many hyperparameters. So, we might over-fit because of the cross-validation measure as the model is tuned in ways that might utilize this random variation rather than in ways that really do improve performance, and that's why we ended up with a model that is performing poorly.

```
predrfc = rf_random.predict(train_features)
accuracy_score(y_train, predrfc)
```

```
0.9971428571428571
```

Image 5. Random forest training accuracy for augmented images.

```
showClassficationResults(y_test,loaded_rfc_pred)
```

	precision	recall	f1-score	support
0.0	0.32	0.37	0.34	148
1.0	0.29	0.38	0.33	141
2.0	0.42	0.24	0.30	161
accuracy			0.33	450
macro avg	0.34	0.33	0.32	450
weighted avg	0.35	0.33	0.32	450

```
[[55 69 24]
 [58 54 29]
 [59 64 38]]
```

Image 6. Confusion matrix for augmented images in Random forest algorithm.

Thus, I think that this technique of extracting features from augmented images might be complex and not required. As, we must classify our original images, we should extract features from original images instead of augmented images which we can see in section 2.

2) Extracting features with original images with KNN and Random forest model:

If we try to train our KNN model on these extracted features from original images, with the same settings as in section 1, the only thing that has been changed is we have extracted features for original images from trained CNN model on augmented images.

So, after training the images in KNN model with optimal k values as [3, 5, 7, 9], we always get 100% accuracy on all the K values for both training as well as testing dataset (image 7). You can see the performance for all the k values in image 8.

```
showClassficationResults(y_test,loaded_knn_pred)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       148
         1.0       1.00      1.00      1.00       141
         2.0       1.00      1.00      1.00       161

    accuracy                           1.00       450
   macro avg       1.00      1.00      1.00       450
weighted avg       1.00      1.00      1.00       450


[[148   0   0]
 [  0 141   0]
 [  0   0 161]]
```

Image 7. Confusion matrix for original images in KNN algorithm.

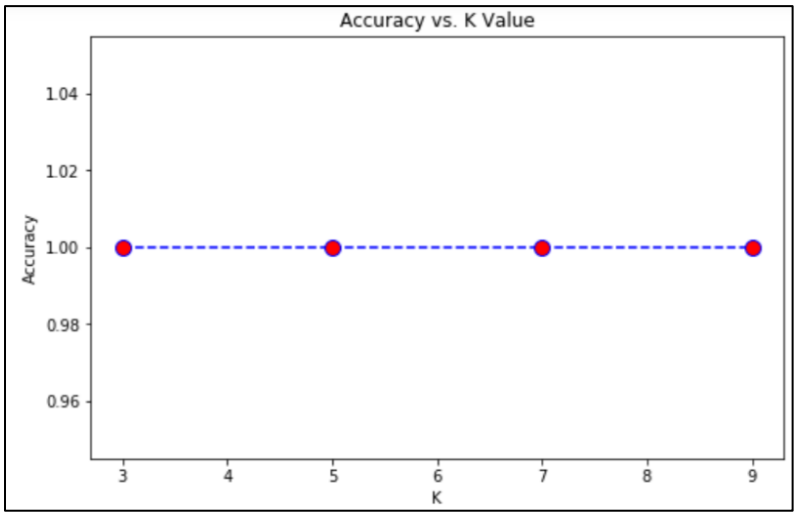

Image 8. Variation in performance with given k values on original images.

After training these images in Random forest method and tuning the 3 hyperparameters using randomized search with cross validation same as section 1 (image 9), we get 100% accuracy for both training and testing original images (image 11).

```
rf_random.best_params_
```

```
{'n_estimators': 50, 'max_features': 'sqrt', 'max_depth': 90}
```

Image 9. Optimal values of hyperparameters for Random Forest algorithm.

We have tuned the hyperparameters like number of trees in the forest of which final optimal value is 50, maximum depth of the tree having 90 as optimal value, and the number of features to be considered when looking for the best split is sqrt of the features. So, the overall hyperparameters combinations handled in algorithm can be seen in image 10.

```
param_distributions={'max_depth': [10, 20, 30, 40, 50, 60,
                                   70, 80, 90, 100, 110],
                     'max_features': ['auto', 'sqrt'],
                     'n_estimators': [20, 30, 40, 50, 60, 70,
                                      80, 90]},
```

Image 10. Combinations handled for hyperparameter tuning in Random Forest algorithm.

```
showClassficationResults(y_test,loaded_rfc_pred)
```

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	148
1.0	1.00	1.00	1.00	141
2.0	1.00	1.00	1.00	161
accuracy			1.00	450
macro avg	1.00	1.00	1.00	450
weighted avg	1.00	1.00	1.00	450

```
[[148   0   0]
 [  0 141   0]
 [  0   0 161]]
```

Image 11. Confusion matrix for original images in Random Forest algorithm.

If we look at the data, it seems to be well balanced labelwise. But, when you look at the real images in the dataset, you can see that it is sort of an easy and small dataset and some images feels like have been duplicated. That is why, I suppose we are getting a higher accuracy, because most examples in testing dataset might be already there in training dataset. Getting a higher testing accuracy does not really mean that this model would work great on unobserved images which proves from augmented images accuracy. This model wouldn't necessarily generalise to new datasets. So, I think to build a better model with good performance, we need large distinct images dataset with sufficient information on each ouput label.