Scene Classification

Jiaoyue Sun

Introduction

The project is about classifying Scenes around the globe into one of six possible classes using Deep Neural Networks. Saurav trained CNN networks from scratch and, also experimented with multiple pre-trained networks. I tried to see if data augmentation of these CNN models would perform better. Boyuan tried to see if pytorch would give us a better score.

Description and Details of individual work

Saurav used five different pretrained CNN networks including VGG16, VGG19, ResNet152, InceptionV3, InceptionResV2, which were trained using the ImageNet dataset. Performance for each of these models were compared with each other both before and after data augmentation.

In this case, the code specifying the percentage of the image to shift as 0.1 the height of the image and 0.1 the width of the image. Both horizontal and vertical positive and negative shifts probably make sense for the chosen photograph, but in some cases, the replicated pixels at the edge of the image may not make sense to a model. The flip augmentation is specified by a boolean horizontal_flip or vertical_flip argument to the ImageDataGenerator class constructor. Only horizontal flips are considered in this model. Random rotations via the rotation_range argument are considered, with rotations to the image between 0 and 180 degrees. Other augmentations like zoom and shear are not used in this project. A sample code of data augmentation can be seen below:

```
# This will do preprocessing and realtime data augmentation:
datagen = ImageDataGenerator(
    featurewise_center=False,  # set input mean to 0 over the dataset
    samplewise_center=False,  # set each sample mean to 0
    featurewise_std_normalization=False,  # divide inputs by std of the dataset
    samplewise_std_normalization=False,  # divide each input by its std
    zca_whitening=False,  # apply ZCA whitening
    zca_epsilon=1e-06,  # epsilon for ZCA whitening
    rotation_range=0,  # randomly rotate images in the range (degrees, 0 to 180)
    width_shift_range=0.1,  # randomly shift images horizontally (fraction of total width),
    height_shift_range=0.1,   # randomly shift images vertically (fraction of total height)
    shear_range=0.,  # set range for random shear
    zoom_range=0.,  # set range for random zoom
    channel_shift_range=0.,  # set range for random channel shifts
    fill_mode='nearest',    # set mode for filling points outside the input boundaries
    cval=0.,  # value used for fill_mode = "constant"
    horizontal_flip=True,  # randomly flip images
    vertical_flip=False,  # randomly flip images
    rescale=None,   # set rescaling factor (applied before any other transformation)
    preprocessing_function=None,   # set function that will be applied on each input
    data_format=None,   # image data format, either "channels_first" or "channels_last"
    validation_split=0.0)   # fraction of images reserved for validation

datagen.fit(X_train)
# Fit the model on the batches generated by datagen.flow().
model.fit_generator(datagen.flow(X_train, y_train,batch_size=512),
                    epochs=35,validation_data=(X_CV,y_CV),workers=4)
```

Results and Summary

To measure the performance of model Accuracy, Cohen Kappa Score and Macro F1 Score were used. Since our dataset is balanced, Accuracy Score was considered as the prime performance indicator. The comparison of the scores before and after data augmentation are shown below:

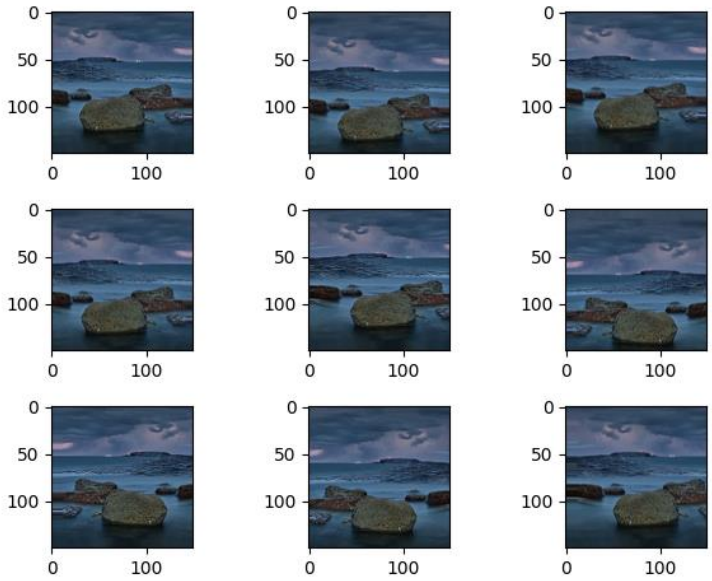
Model	Accuracy	Cohen Kappa Score	Macro F1 Score
VGG16	88.87 %	0.87	0.89
VGG19	87.73 %	0.85	0.88
ResNet152V2	82.40 %	0.79	0.82
InceptionV3	83.33 %	0.79	0.83
InceptionResV2	84.60 %	0.82	0.85

Table 1: Scores before data augmentation

Model	Accuracy	Cohen Kappa Score	Macro F1 Score
VGG16	89.43 %	0.87	0.90
VGG19	88.73 %	0.86	0.89
ResNet152V2	87.27 %	0.84	0.88
InceptionV3	84.20 %	0.81	0.85
InceptionResV2	83.99 %	0.81	0.85

Table 2: Scores after data augmentation

The models do not appear to perform better after data augmentation. Balanced data may not be improved as much as unbalanced data by using data augmentation. One example of the images augmented can be seen below:



From this example, we can see that the images did not change much due to the nature of these images. This could be one of the reasons why the score after data augmentation did not improve much.

Percentage of the outside code

30%

References

https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/
https://www.kaggle.com/puneet6060/intel-image-classification