Assignment 2019 - Street View House Numbers

Sangita Sriram
D17129392
DT228A

I. Introduction

This assignment aims to perform image classification, by making use of the Street View House Numbers dataset. It is a real-world image dataset obtained from house numbers in Google Street View images, of size 32x32. It has totally 10 classes, representing each digit.

This assignment was performed using Jupyter notebook and the model was implemented using Keras library.

II. Data Preparation

First, the dataset is downloaded from the URL directly and then loaded into the dictionary using loadmat function, which is used to read .mat files. The .mat files have 2 variables: X which is a 4-D matrix containing the images, and y which is a vector of class labels. Thus the images and class labels are split as x_train, y_train and x_test and y_test variables, from the train and test datasets.

Then, the train and test datasets are transposed, by converting it from (width, height, channels, size) to (size, width, height, channels).

Then, the x_train and x_test variables are normalized by converting to float and dividing by 255. The meaning behind this is, usually 32-bit precision is used, when training a neural network, finally at one point the training data is converted to 32-bit floats. And for the division by number 255, it is the maximum value of an input byte, before the conversion to float32 happens. Thus, this ensures that the input features are always scaled between 0.0 and 1.0. It is usually preferred to have input features around that scale so that the learning rate and other hyperparameters, work well in a stabilized manner, and so that the cost takes reasonable values.

The y_train and y_test variables are converted as categorial class labels, as they have a finite number of classes from 1 to 10.

III. Modelling

Keras library is used for modelling this dataset. I have made use of a Sequential model for this purpose of the assignment. A sequential keras model is the simplest of all, it is basically a linear stack of layers built on top of one other.

Usually first, the sequential model is created and then the layers are added in the order of computation we wish to perform.

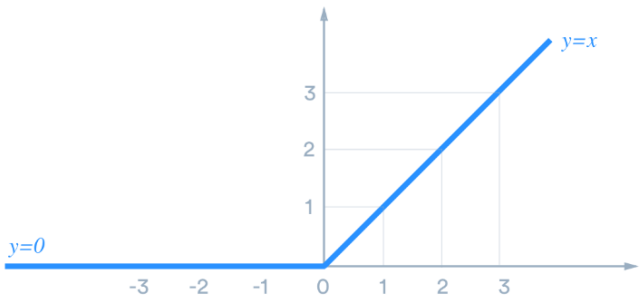
Conv2D is used here, which is a 2D convolution layer. This layer creates a convolution kernel that is convolved with the input layer to produce a tensor or array of outputs. It takes the following as input parameters:

Only the first layer requires to know the shape of the input, and that is why the input shape is mentioned in the first layer itself. It consists of an input_shape argument. This is the shape tuple (a tuple of integers or None entries, where None indicates that any positive integer may be expected).

An input parameter filter is mentioned, which means the number of output filters that has to be produced at the end of convolution. The first layer learns 64 filters. Finally, the last layer learns 128 filters.

The input parameter kernel_size indicates an integer or tuple of two integers, specifying the height and width of the 2D convolution window. It can also be a single integer, to specify the same value for all of the dimensions.

An activation function is also mentioned, which can be softmax, rectifier, tanh and sigmoid. For this assignment, I have used the ReLU activation function which stands for rectified linear unit. Mathematically, it is defined as y = max(0, x). This is the visual representation of this function is: [4]



ReLU is linear for all values that are positive, and it is zero zero for every negative value. It essentially means the following:

- It is cheaper to compute as there is not much complicated math involved. The models thus take lesser time to train and run.
- It usually begins the convergence faster. When x gets large, Linearity will mean that the slope doesn't plateau, or suffer at the same point or saturate for a long time. The vanishing gradient problem suffered by other activation functions like sigmoid or tanh, is not a problem compared to ReLU.
- Since ReLU is zero for all of the negative inputs, it is likely to not be activated much, which means it is sparsely activated.

The sparsity aspect of ReLU results in concise models that have better predictive capabilities and lesser issues of

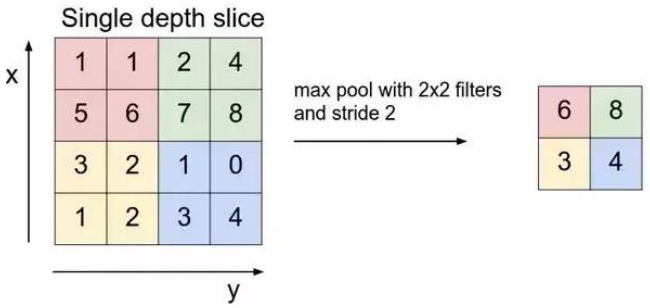
XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

overfitting/noise. In this type of sparse network, it is considered to be more likely that neurons actually process meaningful aspects of the problem.

The next layer has a maxpooling 2d layer. Max pooling is a sample-based discretization process to down-sample the input feature, resulting in a reduce of its dimensionality and making space for assumptions to be made about features contained in the sub-regions that have been selected.

This is done to help over-fitting by providing an overall abstracted representation. It also reduces the computational cost by reducing the number of parameters to learn. A filter of pool size (2, 2), means we will have a stride of 2 (meaning that the (dx, dy) for stepping over our input, will be (2, 2)) and will not overlap regions.

An example representation for pooling is as shown below: [2]



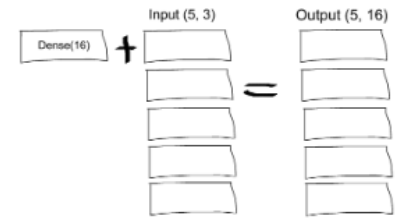
The next layer Flattens the network. Flattening a tensor means, removing all of the dimensions except for one. A flatten operation on a tensor, actually reshapes it, to have a shape that is equal to the number of elements contained in the tensor.

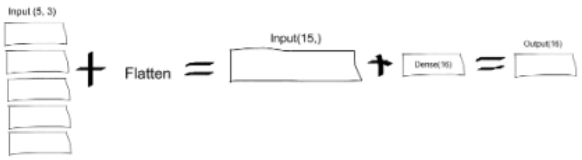
Finally, the last layer implements the Dense function. It performs the operation:

output = activation(dot(input, kernel))

where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer.

A figurative example to show what happens if only Dense function is used vs. when Flatten and Dense function is used: [3]





We compile a model, to configure the learning process, which is done via the model.compile method. It has three arguments:

- An optimizer
- A loss function
- A list of metrics

The model is then ready to be trained using the fit function, with the x_train and y_train variables. Input paramer batch_size is an Integer or NULL value. representing the number of samples per gradient update.

There is another input parameter, known as an epoch, which can be explained as consisting of one full training cycle on the training set. Once every sample in the set is seen, the second epoch starts again - marking the beginning of the next epoch cycle.

A batch means that we update once at the end of the epoch (after every sample is seen) and online that you update after each sample.

IV. RESULTS

Result from the epochs carried out by the model:

```
model_history = model.fit(X_train, y_train, batch_size=128, epochs=5, validation_split = 0.1)
```

```
WARNING:tensorflow:From C:\Users\student\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 65931 samples, validate on 7326 samples
Epoch 1/5
65931/65931 [==============================] - 244s 4ms/step - loss: 1.2244 - acc: 0.6031 - val_loss: 0.7338 - val_acc: 0.7806
Epoch 2/5
65931/65931 [==============================] - 247s 4ms/step - loss: 0.5568 - acc: 0.8449 - val_loss: 0.5561 - val_acc: 0.8399
Epoch 3/5
65931/65931 [==============================] - 246s 4ms/step - loss: 0.4698 - acc: 0.8713 - val_loss: 0.5289 - val_acc: 0.8456
Epoch 4/5
65931/65931 [==============================] - 245s 4ms/step - loss: 0.4136 - acc: 0.8854 - val_loss: 0.8018 - val_acc: 0.7595
Epoch 5/5
65931/65931 [==============================] - 246s 4ms/step - loss: 0.3723 - acc: 0.8960 - val_loss: 0.5094 - val_acc: 0.8492
```

Evaluation result of Accuracy:

```
#Evaluating the model against the test datasets and finding the accuracy of the model.

score = model.evaluate(X_test, y_test, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

```
Test score: 0.5975283736382515
Test accuracy: 0.8349339274738783
```

We can see that a good accuracy of 83% has been reached by this Sequential model, by evaluating the trained model against x_test and y_test test datasets.

Testing the model:

A random number was assigned to a variable i, to predict the actual number in the random numbered image and to compare the result of the prediction with the actual number in the image was performed:

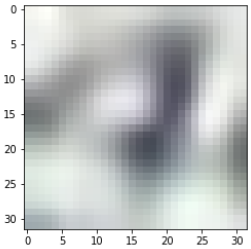
```
i = 13180
res = model.predict(X_test[i][None,:,:])
print("Image", i)
print(f"Model says it is a {np.argmax(res)} while it is a {np.argmax(y_test[i])}")
print("Stats are", np.array(res))
plt.imshow(X_test[i])
```

```
Image 13180
Model says it is a 4 while it is a 4
Stats are [[0.0002 0.0620 0.0650 0.1761 0.3786 0.0001 0.0051 0.2061 0.0438 0.0526
  0.0105]]
```

Here, a random image's number was chosen which actually had the number 4 and our model correctly predicted the value as 4.

```
Image 13180
Model says it is a 4 while it is a 4
Stats are [[0.0002 0.0620 0.0650 0.1761 0.3786 0.0001 0.0051 0.2061 0.0438 0.0526
  0.0105]]

<matplotlib.image.AxesImage at 0x11a061d54e0>
```



Result of test(FileName) function:

The aim of this function is to use the saved trained model and pass an image containing a number and to see if our model is able to predict the number correctly or not.

An image of number 2 was passed to the function:

The trained model by being called by the test() function, gave the output as [2], representing the class label of the image 2 correctly.

```
#Calling in the first function test(FileName)
#Any image name can be passed here. Better prediction is obtained when image has just a single number
#Output is the class label of the number

img = 'number2.jpg'
test(img)
```

```
[2]
```

V. Conclusions

Thus, we could see that the model used in this assignment reached a decent accuracy of 83% and was able correctly identify a test image passed to the model that was trained previously.

References

[1] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* 2011.

[2] https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks

[3] https://stackoverflow.com/questions/43237124/role-of-flatten-in-keras

[4] https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7