Deep Learning

Food Classification using CNN

March 17, 2019

Lavanya Mandadapu

Contents

1 Introduction

The process of identifying a food item type from an image is an interesting task and also used for many applications like monitoring the food habits for a healthy lifestyle. In this document, Convolution Neural Networks for food classification is performed on food-11 dataset [1]. food-11 dataset contains a total of 16643 food images categorized into 11 categories. The 11 categories represented by food-11 dataset are Bread, Dairy products, Dessert, Egg, Fried food, Meat, Pasta, Rice, Seafood, Soup and Fruit/Vegetable. The source code and the results are available at `https://github.com/lavanya463/Food11CNN.git`.

2 Image preprocessing and Memory management

When an initial run of the food-11 dataset on a CNN, which has two conv2d and maxpool2d layers followed by a dense, is performed, insufficient memory issue was raised. In order to tackle this issue, following experiments have been performed.

- **reducing the batch size**: A varying batch size from 100 to 1 is performed with the same CNN architecture, but this does not give any solution to the problem.
- **using a smaller architecture**: Initial CNN architecture is itself a smaller one but in order to see its effect on memory issue, another CNN architecture with a single conv2d and maxpooling2d followed by dense is used. But, this does not made any effect in this case on dealing with the memory issue.
- **reducing the image size**: After looking into all the features that will cause memory issues, the original size of the image is 512×512 and is considered to be very large. So, the images are reduced to 200×200 (after trying different images dimensions) and ran on initial CNN architecture, this gave a solution to the memory issue. Therefore all the further processing was performed with the images of size $200 \times 200 \times 3$.
- **using flow_from_directory** : Even though reducing the image size gave an quick solution. The effective solution would be using a flow_from_directory method present in the keras toolkit. So, for this work flow_from_directory is used.

In this work, Image preprocessing and loading is performed using ImageDataGenerator class present in Keras. ImageDataGenerator method has many arguments that can be adjusted in order to perform cetrain transformations on the image. Among those, sample wise centering, sample wise normalization and rescaling are used.

- **sample wise centering**: By setting this argument to true, we can perform picture wise centering. This is particularly useful for visual detection and inception. By doing image centering we are making the distribution having zero mean which allows the visual inception much more easy.

- **sample wise normalization**: The main motivation in including this technique is to achieve consistency in dynamic range. While dealing with a wide range of images there can be a case of having poor contrast or glare, by performing this normalization we can deal with it.

- **zca whitening**: Even though this is not experimented in this work. The effect of zca whitening is well studied on another small set of images. By carrying out some whitening on images before feeding into a network allows us to decorrelate the data. In particular, here we are dealing with ZCA (zero-phase component analysis). Contrast to PCA, ZCA preserves the spatial information, so the whitened data is still a picture in the original space, so it can preserve the information.

3 Neural network Configuration

There are many ways to solve a image classification task. If we are intended to use standard machine learning algorithms like SVM's, first we need to generate the feature vectors from the images. But, during the process we are losing the spatial information in the images. We can still include them as a seperate feature vector, anyhow if we have a large amount of data, this process takes long and is inefficient. From recent times, CNN's are seen as an efficient technique for image classification and are considered as automatic feature extractors, preserving the spatial information, as they use adjacent pixel information to downsample the image. Henceforth, Convolution Neural Network is considered to solve our food classification problem. The final architecture contains convolution 2d layers, max pooling 2d layers and dense layers.

- **convolution 2d**: This layer is used to create feature maps by convolving input data. In our case input size is (200,200,3).

- **Max pool 2d**: The pooling function is used to reduce computational complexity and also variance in the data. Max pooling extracts the most important features in the data like edges.

- **Dropout**: Dropout is a regularization technique to reduce the overfitting in neural networks. As the name implies, dropout drops some of the units, both hidden and visible, in neural network. In our case a scale of 0.4 is used for dropout.

- **Fully connected layer**: Using the dense layer, all the neurons in the current layer are fully connected to those in the next layer. It is same as traditional MLP neural network.

- **ReLu**: Due to its benefit of sparsity, unlike sigmoid, and reduced likelihood of vanishing gradient ReLu is used in this work as a activation function.

- **softmax**: food-11 classification is a multi class problem and softmax is well known to be used as a output function for multi class problems. Additionally the soft-max layer is soft version of the max-output layer and also resilient to outliers. It outputs

a probability distribution, that is the total value of the output sums to 1. Figure 1 shows the convolution neural network architecture used in this work.

4 Neural network Training

A convolutional neural network model has been developed initially (Figure 1) to train the food-11 dataset. Originally a classic stochastic gradient descent has been used as a optimizer. After researching different optimizers that have performance impact on CNN's [2], SGD with nesterov and momentum turned out to be good for the image classification problems. The main idea behind using a momentum in neural networks is to provide a moving average of the gradient to update the weights of the network [3]. In nesterov momentum, gradients are computed taking into consideration the direction of current momentum. Initial model is tested with different number of epochs and when it reaches around 40 epochs, it is overfitting. So, the ideal number of epochs considered for initial work is 35.

In order to see the effect of an already trained model on our data. A pre-trained model Inception V3 is used. In this case the optimizer is SGD with momentum and nesterov.

Model checkpoints are made via the chekpointer callback to save the best model weights and are saved in the form of .hdf5 files.

5 Results and Analysis

In food-11 dataset, entire data is already seperated into train, validation and test folders. But, in order to train with more data, validation images have been considered in the training and test images are considered to be validation for later experiments. Initial experiments are done using the original partition of data. One important thing to note is that for all the experiments, image size of $200 \times 200 \times 3$ and the batch size of 120 is considered.

5.1 Initial implementation with SGD

This is a basic implemenation without much focussing on detailed hyperparameter consideration. The training, validation and test dataset split in this case used the original given partition. The architecture described in 1 is used and this has resulted in a test accuracy of .44 over 30 epochs running for a hour in server. When tried with more epochs or increasing the number of layers the model is overfitting. In figures 3 and 2, we can see the accuracy and loss plots of the initial implementation. In these plots, we can see after 20 epochs the model is overfitting. However, from epochs 20 to 30 the validation accuracy is fluctuating around 40 percent. In this implementation, a default sgd optimizer is used.

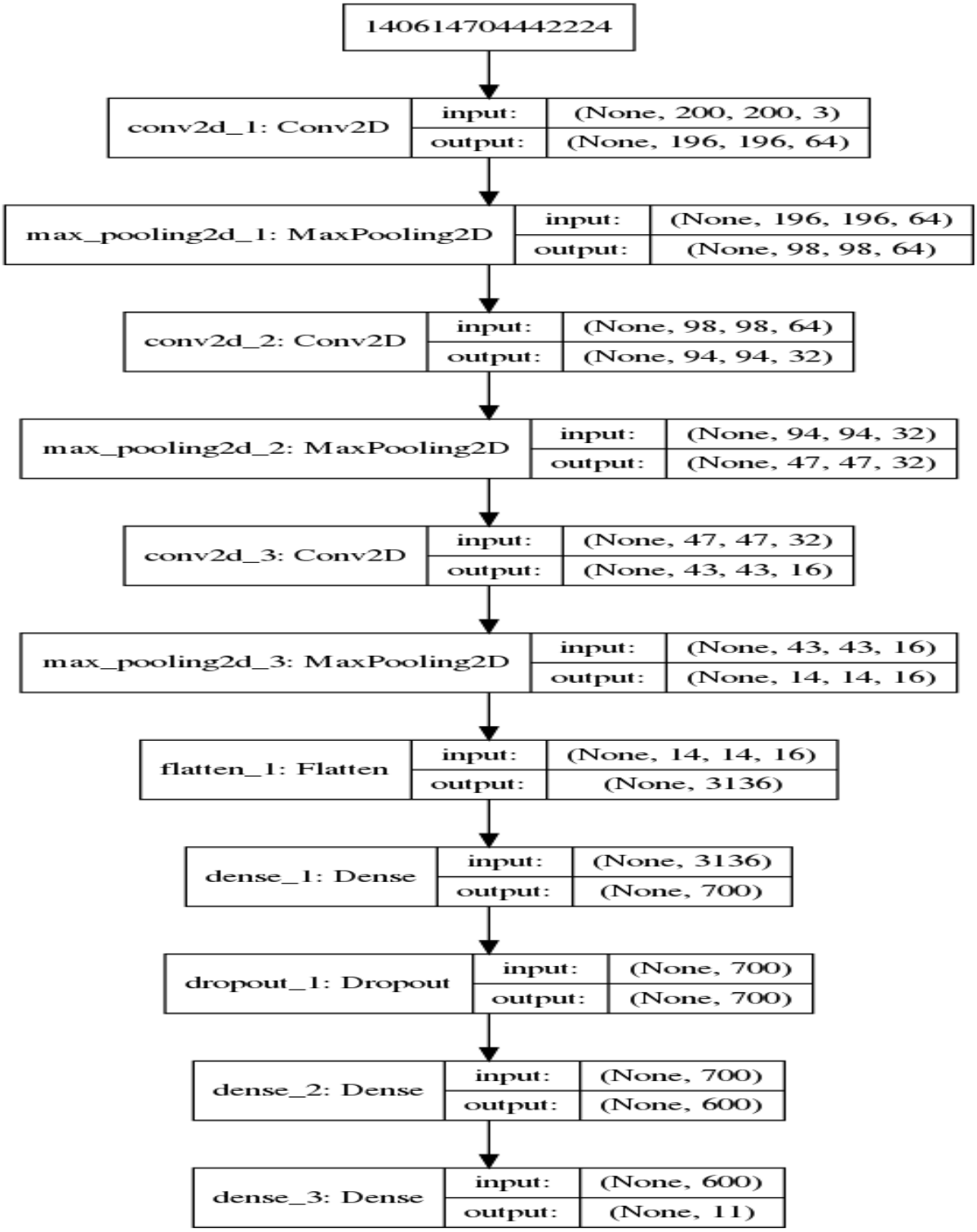


Figure 1: CNN architecture used in this work

5.2 Initial architecture with SGD and Nesterov Momentum

To know the impact of SGD on the dataset when the learning parameter, momentum and nesterov is changed. A learning parameter of 0.001 and a momentum of 0.9 making nesterov true is tested. It has made a some impact on the result. Unlike the first case when overfitting had started at the epoch 30, here there is still a room for improvement at epoch 30. When increased the epochs to 150, we can observe that the overfitting has started around 90. The accuracy is 57 percent in this case. However, the time required to complete this run with 150 epochs is more than 8 hours. In classic SGD we don't compute the exact derivate of our loss function. Instead, we're estimating it on a small batch. Which means we're not always going in the optimal direction, because our derivatives are noisy. So, using momentum which gives exponentially weighed averages can provide us a better estimate which is closer to the actual derivate. This can be seen as main reason in the increase in the accuracy in this case. In figures 4 and 5 we can see the accuracy and loss plots of this case. Below we can observe the confusion matrix of this case, we can see that in the case of dairy_product and seafood the model has mis-classified a lot, this has happened because in the images of these two, there are other classes present in them. There are also high inter-dependencies in this dataset, thats also a reason for diagonals not being clear.

5.3 Other experiments

Optimizer adam is also tested to see the effect with 30 epochs, accuracy has reached 53 percent around 25 epochs and started overfitting. As mentioned earlier, training size is increased and tested to see the effect. There is clearly a small amount of increase in the accuracy. However, the increased size is not huge enough to draw a conclusion. When tested with a pretrained model, the accuracy in epoch one reached 60 percent and then started reducing. This is because of the high complexity in the model, which making it to overfit. For larger datasets, pretrained models with fine tuning can make a big difference in learning unlike here.

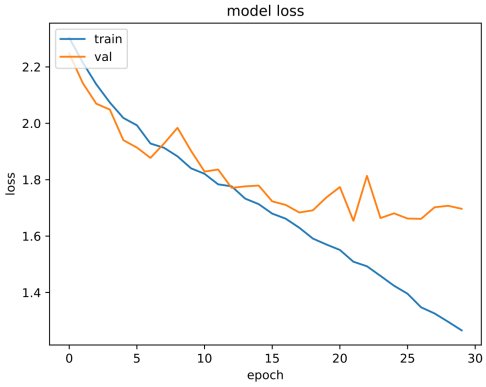


Figure 2: Loss plot for 5.1

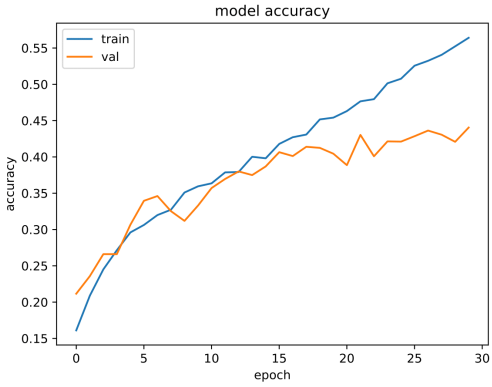


Figure 3: Accuracy plot for 5.1

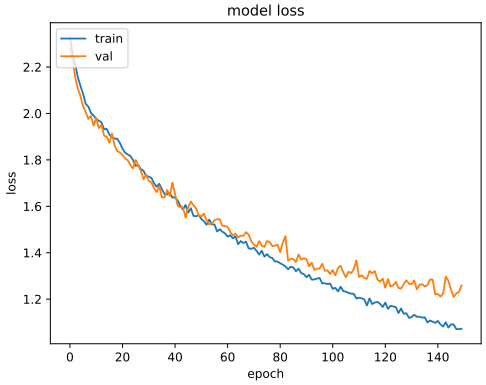


Figure 4: Loss plot for 5.2

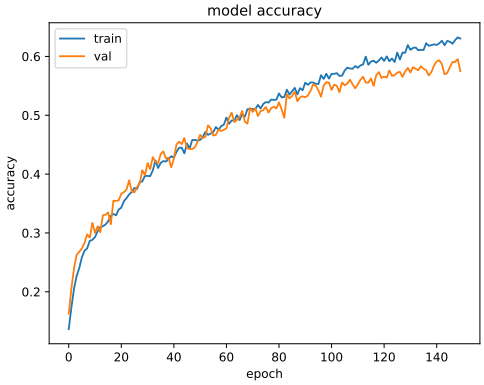
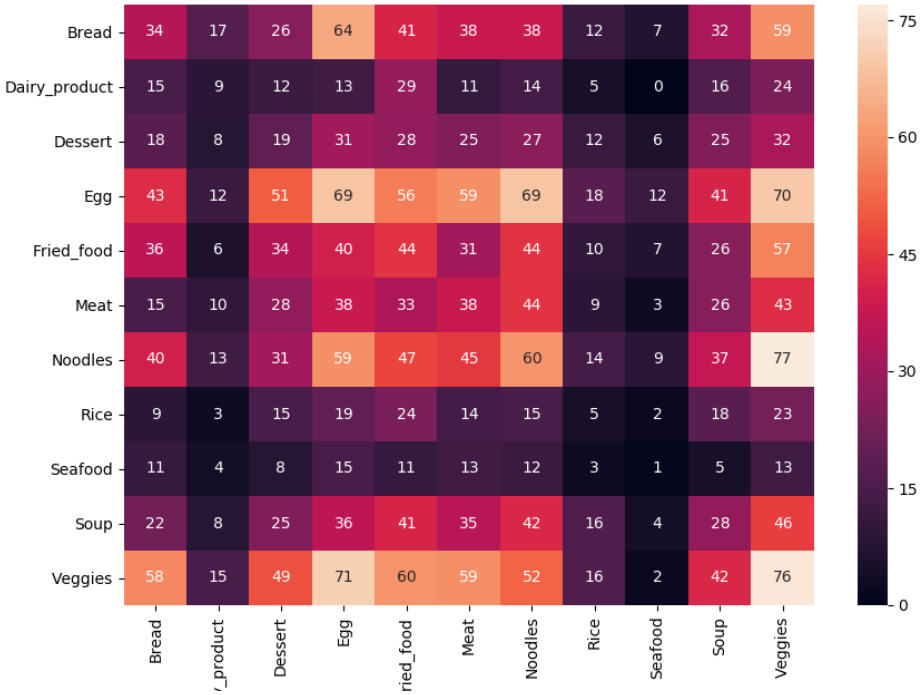


Figure 5: Accuracy plot for 5.2



6 conclusions and Future work

CNN's are good for the image classification task when we can have large data and computational resources. Our dataset Food-11 has a limited number of classes and these classes are not so well descriptive. For example, fried eggs considered to be part of the class fried_food and also egg. However, regarding food images there is a larger dataset available. It is called as food-101 and has 101 classes. A CNN based approach [4] which uses a fine tuning on Inception v3, trained on food-101 dataset using 96GB system RAM and 12 core Intel i7 processor has proved to achieve a high accuracy and has developed a web application for food classification. Dealing with food-101 can be taken as a future step. A multi level classification approach can also be developed to avoid the mis-classifications.

Finally, by performing this experiment, experience on the concepts of CNN, about its significant parameters and available libraries for image processing and classification is learned.

References

[1] Kaggle:. https://www.kaggle.com/tohidul/food11.

[2] medium article on optimizers:. https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/.

[3] distill:. https://distill.pub/2017/momentum/.

[4] David J Attokaren, Ian G Fernandes, A Sriram, YV Srinivasa Murthy, and Shashidhar G Koolagudi. Food classification from images using convolutional neural networks. In *TENCON 2017-2017 IEEE Region 10 Conference*, pages 2801–2806. IEEE, 2017.