Project Final Report CS230

Title: Image Recognition Algorithm for Packed Products

Students: Asherin George (asherin), Cynthia Brosque (cbrosque), Luka Salamunic (lsalamun)

GitHub link: https://github.com/Asherin7/CS230-Grocery-Product-Classification

Abstract

We develop an image recognition software for packed products. We built a dataset of 8,920 snapshots of packed products of size 4,032 x 3,024 x 3, which we labeled according to the product that was photographed. Each captioned image contains one product out of 165 possible products (i.e. classes). We implemented a shallow neural network (**"SNN"**) and a convolutional neural network ("**CNN**") with fully connected softmax functions as outputs. Our algorithms recognizes products with high precision (>99.5% accuracy on a 1,338 sample test set for the SNN and CNN models). The algorithms are robust to highly similar products (e.g. same brand, different flavor), as well as products taken from different angles and zooms. We benchmarked these results to a K-Nearest Neighbour model ("**KNN**"), which achieved a marginally lower performance.

I. Introduction

The "multiple choice problem" is an issue every diet-conscious supermarket client faces. Deciding –for example– the optimal cereal to buy would require analyzing flavor, nutritional content, ingredients and price and applying a formula weighting each of these factors. In a supermarket that sells easily, e.g. over 100 different cereals, this is virtually impossible. So, consumers end up using heuristics –simple, efficient rules which people often use to form judgments and make decisions–. At the end, the typical consumer will base its decision upon precedent consumption, recommendations from others, branding and marketing, which we deem suboptimal.

We propose a solution to the multiple choice problem. We develop an image recognition software for packed products. The software serves as input for a smart-choice food app that will provide product recommendations to users that are looking to optimize their consumption decisions. The application will enable users to buy products that helps them achieve specific "goals" (e.g. lose weight), while avoiding products that do not pass their personal "filters" (e.g. only eat kosher). The fully-functional application will receive multi-class images (i.e. images multiple classes) and will rank products based on users' profile, goals and filters.

For the scope of this class, we designed an image recognition software for single captioned products. We built a proprietary dataset of 8,920 snapshots of packed products. Each captioned image contains one product out of 165 possible products (i.e. classes) which is compressed (from 4,032 x 3,024 x 3 to 256 x 256 x 3) and vectorized (from 256 x 256 x 3 to a vector of length 196,608) to be used as input. Our models comprised a SNN and a CNN with fully connected softmax functions as outputs. Our output is a number which reflects the predicted class (i.e. unique product class).

Despite the extra difficulty imposed by training and testing highly similar products (e.g. same brand, different flavors) with differences in angles and zooms, the algorithms were able to achieve an accuracy of >99.5% on a 1,338 sample test set. The SNN (CNN) reaches 99% accuracy on train set at 7th (12th) epoch and 99% accuracy on dev test at 9th (5th) epoch. Our benchmark KNN model achieved 99.4% accuracy, comparable to SNN (CNN) test set accuracy of 99.6% (100.0%).

II. Related Work

Related work is scarce. Despite expecting several open source architectures to recognize packed products, we had difficulties finding one that served our business problem. One relevant one was the empirical research on grocery recognition by Hoffman, S. C. and Thiagarajan, D. (2016) (https://vision.cornell.edu/se3/wp-content/uploads/2016/09/Report.pdf). In this study, the authors apply deep CNNs to recognize objects in the wild using images of the object taken under ideal conditions to train. Deep learning models include CIFAR-10 Network and Inception V3.

Another relevant contribution is the work done by Wisdom D'Almeida on Transfer Learning (https://github.com/wisdal/Image-classification-transfer-learning). Using Inception V3, D'Almeida develops an

image categorization algorithm to help retailers reduce human effort in the inventory management process of warehouses and retail outlets. Accuracy on his test set was over 85%.

III. Dataset and Features

Lacking a correctly labeled open source dataset to address our problem, we were compelled to create our own proprietary dataset. The dataset comprises snapshots taken by our iPhones of packed products at five supermarkets: four Safeways near Palo Alto, SF and one H-E-B in Austin, TX. Between 40 and 60 snapshots were taken per packed product using iPhone's burst function. Pictures were taken from different angles, zooms and lighting (lighting depended on products' positions on the supermarket shelves).

165 different classes (i.e. product distinct types) encompass our dataset. 70 out of 165 classes are cereal boxes taken for our first iteration / milestone. The other 95 classes were added as a suggestion from our project advisor to increase the difficulty of the overall problem. These are composed of 19 subsets of highly similar products, which we handpicked with the intention of challenging our algorithm as much as possible. Figure 1 shows examples of similar products used for training and testing purposes:

Figure 1: Sample images from 5 different classes

The pre-processed dataset comprises 8,920 snapshots of size 4,032 x 3,024 x 3, which we labeled according to the product photographed. Each image contains a single product captioned, which was done on purpose to test the power of the algorithm in single-image detection only. The collected images were converted to a h5 dataset for ease of transport and convenience. Images were compressed to 256 x 256 x 3 (RGB) for ease of computation and flattened to vectors of length 196,608. Hence, our input contains 196,608 features, which is normalized by dividing each RGB (i.e. pixel) by 255. Data was divided as follows: 70% train set (6,244 images), 15% dev set (1,338 images) and 15% test set (1,338 images).

IV. Methods and Hyperparameter Tuning

Our main models comprise a SNN and a CNN with one layer of 165 neurons with softmax activations for our output. Our underlying rationale for the SNN was to test the learning capacity of neurons when number of neurons matched the number of classes. We expected each of the neurons to be "assigned" the role of learning a specific class:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 256, 256, 3)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 196608)            0
_________________________________________________________________
fc (Dense)                   (None, 165)               32440485
=================================================================
Total params: 32,440,485
Trainable params: 32,440,485
Non-trainable params: 0
_________________________________________________________________
```

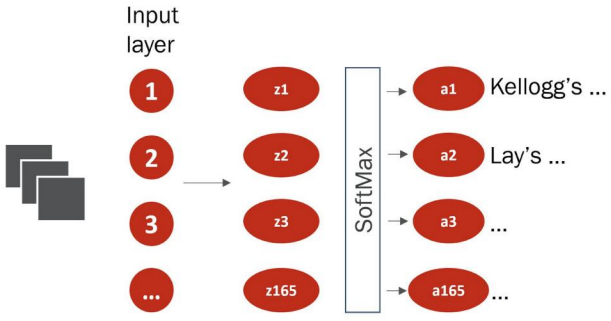


Figure 2: SNN overview and diagram

Our CNN comprises a deeper neural network with dropouts of 50% as regularization component. The CNN ends with a fully connected layer and a softmax output representing each class:

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	(None, 256, 256, 3)	0
conv0 (Conv2D)	(None, 250, 250, 32)	4736
bn0 (BatchNormalization)	(None, 250, 250, 32)	128
activation_7 (Activation)	(None, 250, 250, 32)	0
max_pool0 (MaxPooling2D)	(None, 62, 62, 32)	0
dropout_7 (Dropout)	(None, 62, 62, 32)	0
conv1 (Conv2D)	(None, 58, 58, 128)	102528
bn1 (BatchNormalization)	(None, 58, 58, 128)	512
activation_8 (Activation)	(None, 58, 58, 128)	0
max_pool1 (MaxPooling2D)	(None, 29, 29, 128)	0
dropout_8 (Dropout)	(None, 29, 29, 128)	0
flatten_4 (Flatten)	(None, 107648)	0
fc (Dense)	(None, 165)	17762085

Total params: 17,869,989
Trainable params: 17,869,669
Non-trainable params: 320

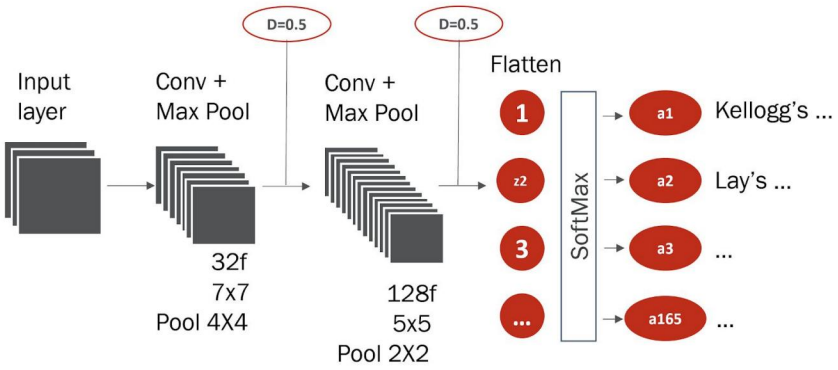


Figure 3: Baseline model overview and diagram

As for the optimization algorithm, we decided upon an Adam optimizer with regularization parameters beta_1 = 0.9, beta_2 = 0.999 and eps = 1e-10 for the SNN and CNN. Loss function used was categorical cross-entropy loss. The justification towards using cross-entropy loss is that softmax assigns probabilities for each class, which requires a loss function that penalizes a higher probability on y_pred_class when y_true_class=0, and vice versa.

$$P_j^{(i)} = \frac{e^{y_j^{(i)}}}{\sum_{j'} e^{y_j^{(i)}}} \qquad L^{(i)} = -\log P_{j_{\text{correct}}^{(i)}}^{(i)}$$

Training was performed on mini-batches to allow the loss function to converge without hitting any saddle points. Besides betas from Adam optimizer, tuneable hyperparameters for this model include: 1) learning rate and 2) mini-batch size. Our optimized hyperparameters for the SNN (CNN) comprise mini-batches of 512 (32) samples each. Mini-batch size of 512 (32) coupled with a learning rate of 0.00005 (0.00001) gave us optimal accuracy and loss, as it is shown in the results section.

To benchmark our neural networks, we implemented a KNN with K=1. Illustratively, we have:

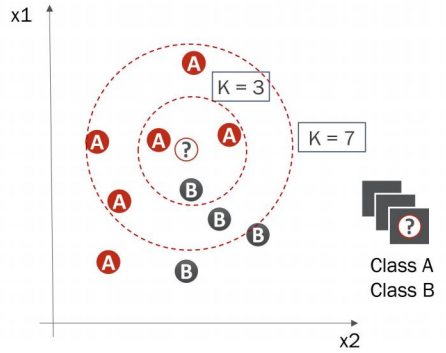


Figure 4: Illustrative KNN

Our rationale on using a KNN as benchmark is that our commercial application consists in classifying exact same products, not types or categories of products. Thus, classes should -by definition- only contain exact same boxes. So, despite changes in angles, zoom or lighting, or even despite having other very similar

products in the training set (e.g. Figure 1), a simple supervised algorithm such as KNN should provide decent outcomes. We used K=1 and Euclidean distance function for our KNN:

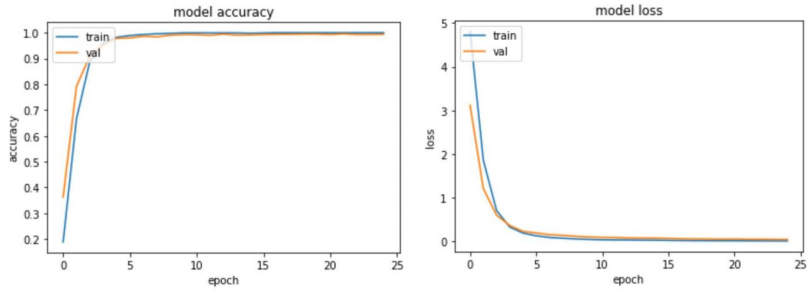
$$dist\,(A,B) = \sqrt{\frac{\sum_{i=1}^{m}(x_i - y_i)^2}{m}}$$

V. Experiments, Results and Discussion

Our SNN quickly converged to desired outcomes rapidly:

```
Train on 6244 samples, validate on 1338 samples
Epoch 1/25
6244/6244 [==============================] - 746s 120ms/step - loss: 4.8099 - acc: 0.1885 - val_loss: 3.1173 - val_acc: 0.3625
Epoch 2/25
6244/6244 [==============================] - 715s 115ms/step - loss: 1.8622 - acc: 0.6666 - val_loss: 1.2122 - val_acc: 0.7922
Epoch 3/25
6244/6244 [==============================] - 606s 97ms/step - loss: 0.7086 - acc: 0.8889 - val_loss: 0.6020 - val_acc: 0.9096
Epoch 4/25
6244/6244 [==============================] - 658s 105ms/step - loss: 0.3296 - acc: 0.9611 - val_loss: 0.3653 - val_acc: 0.9529

Epoch 25/25
6244/6244 [==============================] - 646s 103ms/step - loss: 0.0114 - acc: 1.0000 - val_loss: 0.0463 - val_acc: 0.9940
```



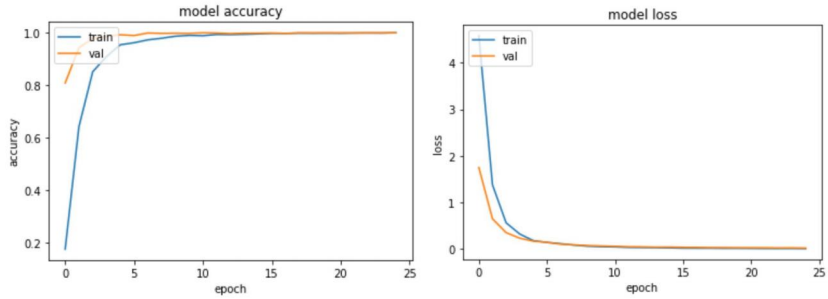
Figures 5, 6 and 7: Accuracy and model loss function per epoch for SNN

Results exceed our initial expectations. The SNN reaches 99% accuracy on train set at 7th epoch and 99% accuracy on dev test at 9th epoch. The loss functions decreases monotonically reaching J_train = 0.0114 and J_dev = 0.0463 by epoch 25. Test set J_test = 0.0461 and accuracy = 99.6%. Each epoch took approximately 650 seconds to train on a Macbook Pro.

Our CNN delivered marginally lower training accuracy, but even better dev/test performance, hence higher bias and lower variance, as seen below:

```
Train on 6244 samples, validate on 1338 samples
Epoch 1/25
6244/6244 [==============================] - 44s 7ms/step - loss: 4.5826 - acc: 0.1742 - val_loss: 1.7502 - val_acc: 0.8079
Epoch 2/25
6244/6244 [==============================] - 39s 6ms/step - loss: 1.3772 - acc: 0.6417 - val_loss: 0.6468 - val_acc: 0.9425
Epoch 3/25
6244/6244 [==============================] - 39s 6ms/step - loss: 0.5616 - acc: 0.8507 - val_loss: 0.3498 - val_acc: 0.9776
Epoch 4/25
6244/6244 [==============================] - 39s 6ms/step - loss: 0.3198 - acc: 0.9089 - val_loss: 0.2295 - val_acc: 0.9865

Epoch 25/25
6244/6244 [==============================] - 39s 6ms/step - loss: 0.0064 - acc: 0.9994 - val_loss: 0.0184 - val_acc: 0.9993
```



Figures 8, 9 and 10: Accuracy and model loss function per epoch for CNN

The CNN reaches 99% accuracy on train set at 12th epoch and 99% accuracy on dev test at 5th epoch. The loss functions decreases monotonically reaching J_train = 0.0064 and J_dev = 0.0184 by epoch 25. Test set J_test = 0.0093 and accuracy = 100%. Impressively, none of the 1,338 images were misclassified. Each epoch took approximately 40 seconds to train on AWS servers.

KNN algorithm achieved 99.4% accuracy on the test set, meaning it misclassified 8 images out of 1,338. We explain such accurate results (in the SNN, CNN and KNN) due to the seemingly large differences between two distinct products when analyzed on a N-dimensional space (i.e. 256 x 256 x 3 = 196,608 dimensions). In other words, it is way more probable than e.g. two random images of Froot Loops are closer on an N-dimensional space than an image of Froot Loops and e.g. Frosted Flakes. We also attribute the success of our results to the capacity our models have in constraining the degrees of freedom, which despite overfitting, helps the model learn how to recognize products of the same exact type.

VI. Conclusions and Future Work

A shallow neural networks (SNN) and a convolutional neural networks (CNN) with 165 neurons activated by softmax functions are able to almost perfectly classify 165 different types of packed products from a sample of 8,920 images (6244/1,338/1,338 split). The algorithms are robust to highly similar products as well as products taken from different angles and zooms. A K-Nearest Neighbour model (KNN) achieves a similar performance, though it is not as commercially applicable as it does not have trainable parameters, so it cannot run fast enough (e.g. it takes hours to predict).

Model	Train Accuracy	Sample (#)	Test Accuracy	Sample (#)	AUC ROC score
SNN	0.9875	6244	0.9858	1338	0.9899
CNN	0.9994	6244	1.0	1338	1.0
KNN			0.994	1338	-

Figures 11: Summary results on train and test sets

These results takes us one step closer of building a smart-choice food app that will provide product recommendations to users that are looking to optimize their consumption decisions. As future work, we plan to keep increasing the problem difficulty by taking pictures from the web or from different environments, like street markets instead of supermarkets where lighting is very much controlled. We also plan to build an image detection algorithm that will enable us to classify pictures that include more than one product. We also plan to test multi-class algorithms and optical character recognition (OCR) algorithm to benchmark its performances with the ones obtained. Finally, we plan to expand our training set to 20,000 packed products, with the intention of launching a prototype in the near future.

VII. Contributions

Asherin George (asherin): Main coder, model selection, hyperparameter tuning, data preprocessing.
Cynthia Brosque (cbrosque): Product manager, team coordinador, poster design and execution, architecture selection, secondary coder, data preprocessing.
Luka Salamunic (lsalamun): Idea proposal, commercial application, reports drafting, architecture selection, secondary coder, data preprocessing.

Each member contributed equally on the creation and execution of this project. We want to thank Ahmadreza Momeni for his extraordinary feedback and contributions to the several iterations of this project. It was an exciting learning experience!