**Visual Object Recognition Using the Fruit 360 Dataset**

Written By: Melinda Bennett

Date: July 14, 2019

**Dataset**

The dataset used in this project is a dataset of high-quality images containing 111 different types of fruits. It can be downloaded from the Kaggle website: <https://www.kaggle.com/moltean/fruits/kernels>

**Dataset Properties**

Total number of images: 75937.

Training set size: 56781 images (one fruit or vegetable per image).

Test set size: 19053 images (one fruit or vegetable per image).

Multi-fruits set size: 106 images (more than one fruit (or fruit class) per image)

Number of classes: 111 (fruits and vegetables).

Image size: 100x100 pixels.

Filename format: image\_index\_100.jpg (e.g. 32\_100.jpg) or r\_image\_index\_100.jpg (e.g. r\_32\_100.jpg) or r2\_image\_index\_100a.jpg or r3\_image\_index\_100.jpg. "r" stands for rotated fruit. "r2" means that the fruit was rotated around the 3rd axis. "100" comes from image size (100x100 pixels).

Note: Different varieties of the same fruit are stored as belonging to different classes

**Reference paper:**

The dataset was created by Mihai Oltean. Working with Horea Muresan, they published a paper which describes the training of a convolutional neural network to recognize the fruits in the dataset with an accuracy of ~ 96%.

The original paper is referenced below:

Muresan, H., Oltean, M.(2018) [*Fruit recognition from images using deep learning*](https://www.researchgate.net/publication/321475443_Fruit_recognition_from_images_using_deep_learning), Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42.

**Licenses:**

The licenses associated with this dataset are:

a) MIT License

b) Copyright (c) 2017-2019 Mihai Oltean, Horea Muresan

**Applications:**

In their paper, Muresan and Oltean discuss a variety of applications for the correct identification of fruits. One of the primary applications they envisioned is an autonomous robot that can complete complex tasks. These tasks may include aisle inspections in a store to correctly identify understocked shelves or out of place products. They chose fruits because fruits are a common product in stores. They also chose fruits that were hard to differentiate such as various species of apples (Braeburn, Red yellow, Pink Lady, etc.).

**CNN Model**

The convolutional neural network model was chosen for this dataset because this class of deep neural networks is most commonly applied to analyzing visual imagery.

The architecture of the model was based on Muresan and Oltean’s model in the paper cited above:

|  |  |  |
| --- | --- | --- |
| Layer Type | Dimensions | Output |
| Convolutional -1 | 5 x 5 x 4 | 16 |
| Max Pooling - 1 | 2 x 2 -- Stride: 2 | - |
| Convolutional - 2 | 5 x 5 x 16 | 32 |
| Max Pooling – 2 | 2 x 2 -- Stride: 2 | - |
| Convolutional - 3 | 5 x 5 x 32 | 64 |
| Max Pooling – 3 | 2 x 2 -- Stride: 2 |  |
| Convolutional - 4 | 5 x 5 x 64 | 128 |
| Max Pooling - 4 | 2 x 2 -- Stride: 2 |  |
| Flatten |  | 512 |
| Fully Connected 1 | - | 512 |
| Fully Connected 2 | - | 256 |
| Output | - | 111 |

A slight change was made in the units or output size in the first fully connected layer. Whereas the original paper had units = 1024, this model has units = 512. This change was made as the constructed model showed an output of 512 units and not 1024. The configuration of the model corresponds to configuration 9 in Muresan and Oltean’s paper.

**1. Libraries Required:**

To implement this model, the following libraries must be imported:

a) from keras.models import Sequential

The sequential model is a linear stack of layers. An instance of Sequential is created to be able to pass a list of layer instances to the model (Keras, n.d).

b) from keras.layers import Conv2D

There are four convolutional layers in this model. The Conv2D function produces a tensor of outputs (Keras, n.d). In the convolutional layer, the filter size (the dimensionality of the output space which changes with each layer), the kernel\_size (the height and width of the convolutional window (5x 5)) and the activation function are defined (ReLU is used in this model). For the first convolutional layer only, the input shape is specified. Since the size of the images in this dataset are 100 x 100 pixels, and they are in colour, the input\_shape = (100, 100, 3).

c) from keras.layers import MaxPooling2D

There are also 4 pooling layers in this model. Pooling helps reduce the size of the input image. It also helps make feature detection independent of noise and small changes like image rotation and tilting (what is known as spatial variance) (Ng, 2019). The arguments specified in this function are the pool size (i.e. the factors by which to downsize), and the strides. A stride of 2 means the filter will move 2 pixels at a time across the x and y axis of the image (Ricco, 2017). The pool size (2x2) and the number of strides to take (2) are the same in each of the 4 pooling layers.

d) from keras.layers import Flatten

There is 1 flattening layer whose purpose is to flatten the image for input into the fully connected layer.

e) from keras.layers import Dense

In this model, there are 2 fully connected layers and 1 output layer. Each of these layers calls the Dense function. The arguments are the number of units (the dimensions of the output space) and the activation function. For the 2 fully connected neural network layers, the activation function used is ReLU. For the output layer, the activation function is Softmax. The number of units specified in the output layer is 111 because this is how many classes of fruits are found in the dataset.

**2. Activation Functions**

a. ReLU

ReLU stands for Rectified Linear Units. It is used as the activation function in the hidden layers within the CNN model. The ReLU function is the simplest nonlinear activation function and tends to result in faster training for large networks (Yang, 2017).

b. Softmax

The activation function chosen in the output layer was ‘softmax’ as it is designed for multi class problems. Softmax assigns decimal probabilities to each class in a multi-class problem (Machine Learning, 2019). The decimal probabilities must add up to 1.0 which helps training converge more quickly than it otherwise would (Machine Learning, 2019). Softmax assumes that each example is a member of exactly one class and cannot simultaneously be a member of multiple classes (Machine Learning, 2019). Thus, softmax works well for images containing exactly one item and can determine the likelihood of that one item being a pear, an orange, an apple, etc. (Machine Learning, 2019).

**3. Compiling the Model**

The final step before training the model is to compile the model. Compiling the model configures the learning process. It takes three arguments: an optimizer, a loss function and a list of metrics.

a. Optimizer

The optimizer chosen was Adam. It was chosen as it is straightforward to implement, it is computationally efficient and is well suited for large datasets (Brownlee, 2017).

b. Loss Function

The loss function selected was ‘categorical cross entropy’. This loss function is intended for multi-class classification and computes the categorical cross-entropy loss. Categorical cross entropy losscan be used in any kind of classification problem. The labels or targets must be an n-dimensional vector in which all entries are 0, except the entry corresponding to the class, which is 1 (Thapar, 2019). This can be achieved through a variety of methods. In this model, the argument ‘class mode’ is set to ‘categorical’ in the keras image data generator methodology which then hot encodes the labels.

c. Metrics

As per the Keras documentation, the metrics for a classification problem is always set to ‘accuracy’.

**4. Loading the Images for Use in the Model**

Image Data Generator from keras was used to preprocess the images and rescale them between 0 and 1 to get them ready for the model. The Image Data Generator creates batches of tensor image data with real-time data augmentation and loops the data over in batches (Keras, n.d). There are 2 steps to preparing the images. An instance of ImageDataGenerator is created and then a methodology is chosen to augment the images.

a. Create the Image Data Generator Instance

An instance is created for both the train and test set. The function ImageDataGenerator has a list of arguments that can be set. For the test set, the defaults were used and only rescaling was specified.

For the train set, the following arguments were used:

|  |  |
| --- | --- |
| **Argument and its Setting** | **Description** |
| rescale=1./255 | *# This is to rescale the data between 0 and 1* |
| width\_shift\_range=.2 | *# Range selected for random horizontal shifts* |
| height\_shift\_range=.2 | *# Range selected for random vertical shifts* |
| zoom\_range = 0.1 | # *Range for random zoom* |
| horizontal\_flip = True | *# Randomly flips inputs horizontally* |

b. Augment the images

The method used to augment the images was the keras ‘flow\_from\_directory’ method.

For the train\_set, the following arguments were used:

|  |  |
| --- | --- |
| **Argument and its Setting** | **Description** |
| 'fruits-360/Training' | *# Directory where the images are stored* |
| target\_size= (100, 100) | *# image size (100 x 100 pixels)* |
| batch\_size=32 | *# Used default batch\_size of 32* |
| class\_mode='categorical' | *# Multiple classes. Results in 2D hot encoded labels* |

For the test\_set:

|  |  |
| --- | --- |
| **Argument and its Setting** | **Description** |
| 'fruits-360/Test' | *# Directory where the Test images are stored* |
| target\_size= (100, 100) |  |
| batch\_size=1 | *# Batch size changed to 1* |
| class\_mode='categorical' |  |
| shuffle = False | *# Shuffle set to False* |

The batch size was left at the default of 32 for the train set but changed to 1 for the test set to ensure the images were processed exactly once (Vijayabhaskar, 2018). Shuffle was set to false in the test set to yield the images in order so the predicted outputs could be matched with their unique ids/filenames (keras, n.d).

**5. Fitting the model:**

The last step in the process is to train the model using the fit\_generator function.

The arguments were defined as following in the function:

|  |  |
| --- | --- |
| **Argument and its Setting** | **Description** |
| Generator = train\_set | *# Dataset to be trained* |
| steps\_per\_epoch=1775 | *# Number of training images / batch size* |
| epochs=100 | *# Randomly picked* |
| validation\_data=test\_set | *# Dataset to evaluate the loss/metrics at end of each epoch* |
| verbose = 1 | *# Shows progress bar* |
| callbacks= [es,plot\_losses, mc] | *# Callbacks include early stopping conditions (es), saving the best model (mc) and plotting the losses and accuracy during the running of the model(plot\_losses)* |
| validation\_steps=19053 | *# Batch size is 1, therefore steps = number of images* |
| use\_multiprocessing = True | *# specifies to use process-based threading* |
| workers = 0 | *# Refers to the maximum number of processes to spin up when using process-based threading. 0 means it will execute the generator on the main thread.* |

**6. Early Stopping and Best Model**

In the callbacks argument above, three functions were called: EarlyStopping, ModelCheckpoint and PlotLosses Keras.

a) Early Stopping

EarlyStopping is used to prevent overfitting. EarlyStopping is imported from keras.callbacks and monitors the specified metric (in this case, test loss). When the model achieves a minimum test loss, the model will wait 20 epochs (patience was set to 20 but can be set to any number) to determine if this loss is truly the minimum. If it is, the model will stop. If it isn’t, the model will continue running for another 20 epochs to see if this new minimum is the actual minimum loss. The patience argument is to ensure the model does not quit prematurely when a minimum loss has been reached.

b) ModelCheckpoint

The ModelCheckpoint function is called from the keras.callbacks library. This function will save the model with the highest test accuracy (or another metric if specified) at the end of each epoch. This allows the model to be reloaded at a later time without having to retrain the model.

c) PlotLossesKeras

To view the metrics as the epochs are stepped through, the function PlotLossesKeras can be used. To call the PlotLossesKeras function, livelossplot must first be imported.

**7. Evaluating the Model**

The model can be evaluated through the evaluate\_generator function. This function returns both the loss and the accuracy of the model. It was run for both the test and train set. The number of steps specified is the number of images in each dataset (test\_set = 19053 and train\_set = 56781). The model was run numerous times but the last trained model ran for 12 hours and completed 38 epochs before early stopping ended the model training. The results are below:

|  |  |  |
| --- | --- | --- |
|  | **Accuracy (%)** | **Loss** |
| Train Set: | 97.8 | 0.071 |
| Test Set: | 96.3 | 0.137 |

**8. Model Predictions**

The following libraries are required to make predictions: NumPy and Pandas.

The predict\_generator function allows the model to predict the classes of fruits from a test set. Before running the predict generator, the generator should first be reset by rerunning the ImageDataGenerator instances and flow\_from\_directory cells or by using ‘test\_generator.reset()’ (test\_set.reset() in this model) (Vijayabhaskar, 2018). If it is not reset, the predicted accuracy will not match the test model accuracy (Vijayabhaskar, 2018).

The generator of images is a required argument for predict\_generator. In this case, it is ‘test\_set’ and the number of steps is set to the number of test images (19053). The predict\_generator returns a NumPy array of predictions. This means that each test image will be assigned a probability of it belonging to each of the 111 classes of fruits. All the probabilities will sum to 1. Calling the NumPy argmax function will return the maximum probability for each test image resulting in one prediction for each image. These predictions are numerical and therefore need to be mapped to the actual fruit names. This can be done by creating a dictionary of the test fruit labels where the key is the numerical index and the value is the fruit name (ex. 0: Apple Braeburn, 1: Apple Crimson Snow, etc.)

Once this dictionary is created, the numerical predictions can be mapped to this dictionary to determine the corresponding predicted fruit label. The predicted and actual labels were then placed into a pandas data frame so that they could be visualized in tabular form and saved for later use if required.

**9. Visualizing the Predictions**

To visualize the actual images with their labeled predictions, the images were first imported. The required libraries for this step were: glob, os, and cv2. The glob function is used to look for a list of files in the file system with names matching a certain pattern. In this case, glob was looking for all the images in the ‘Test’ folder. Each image has a filename that includes which class of fruit it belonged to(image\_index\_100.jpg). As the filenames were read, the fruit name was split off and added to a list entitled “test\_label”.

Making use of the os and glob modules, the jpg files could be found in the Test Folder within the file directory and then each image was read and resized using the cv2 module. Since the cv2 module follows BGR order, there is a step in the function to convert the image to RGB order to use with the MatPlot library (OpenCV, 2015). As each image is read in, the image is added to a list entitled ‘test\_fruit\_img’. Both this list and the test\_label list are then converted into a NumPy array. To view the images, the NumPy function ‘squeeze’ must first be applied to change the images from a 3D array into a 2D array (SciPy.org, 2019). The images are in the format (100, 100, 3) where 3 refers to a color image. By squeezing the image, this dimension is removed and the image can then be plotted with the MatPlot library (SCIPy.org, 2019). Now the images with their actual and predicted labels can be visualized.

**10. Multiple Fruit Images**

The final part of this Jupyter notebook is to determine how well the model predicts the correct labels when multiple fruits were presented in the same image. A test set of pictures depicting multiple fruits was run though the model. Since the target labels are all single fruit labels, the model was asked to provide one fruit prediction for multiple fruits in the image. However, the inaccuracy was quite high (100%). The model is not designed to predict multiple fruits in an image and performs poorly. YOLO (You Only Look Once), and SSD (Single Shot Multibox Detector) are algorithms that can be implemented to help identify multi-classes within an image but are beyond the scope of this project (Meertens, 2018).

**References:**

Brownlee, J. (2017). *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. Retrieved July 12, 2019 from:<https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>

Keras Documentation (n.d). *Getting started with the Keras Sequential model*Retrieved July 12, 2019 from: <http://faroit.com/keras-docs/1.0.5/getting-started/sequential-model-guide/>

Machine Learning (2019). *Multi-Class Neural Networks: Softmax*. Retrieved on July 12, 2019 from: <https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax>

Meertens, R. (2018). [*How to Recognise multiple objects in the same image*](https://stackoverflow.com/questions/42364513/how-to-recognise-multiple-objects-in-the-same-image). Retrieved July 14, 2019 from: <https://stackoverflow.com/questions/42364513/how-to-recognise-multiple-objects-in-the-same-image>

Ng, A (2019). *Convolutional Neural Networks*. www.coursera.org. Retrieved July 14, 2019 from: <https://www.coursera.org/lecture/convolutional-neural-networks/pooling-layers-hELHk>

OpenCV (2015).*Getting Started with Images*Retrieved July 14, 2019 from: <https://docs.opencv.org/3.1.0/dc/d2e/tutorial_py_image_display.html>

Ricco, J. (2017)*. What is max pooling in convolutional neural networks?* www.quora.comRetrieved July 14, 2019 from: <https://www.quora.com/What-is-max-pooling-in-convolutional-neural-networks>

SciPy.org (2019). *numpy.squeeze***.** Retrieved July 14, 2019 from: <https://docs.scipy.org/doc/numpy/reference/generated/numpy.squeeze.html>

Thapar, D. (2019). *What is the difference between categorical\_crossentropy and sparse\_categorical cross entropy when we do multiclass classification using convolution neural networks?* www.quora.com Retrieved July 12, 2019 from:<https://www.quora.com/What-is-the-difference-between-categorical_crossentropy-and-sparse_categorical-cross-entropy-when-we-do-multiclass-classification-using-convolution-neural-networks>

Vijayabhaskar, J. (2018) *Tutorial on using Keras flow\_from\_directory and generators.* www.medium.com. Retrieved July 12, 2019 from: <https://medium.com/@vijayabhaskar96/tutorial-image-classification-with-keras-flow-from-directory-and-generators-95f75ebe5720>

Yang, J. (2017). *ReLU and Softmax Activation Functions*. Retrieved on July 12, 2019 from: <https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions>

Further Reading:

Kingma, D., Ba, J. (2015). *Adam: A Method for Stochastic Optimization*. Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego. <https://arxiv.org/abs/1412.6980>