MACHINE LEARNING ENGINEER NANODEGREE

Fruit Images Recognition

Capstone Project

By: Yahia Elshahawy
September , 2018
yahia.elshahawy@gmail.com

1. Definition

Overview

Intro

- **We want to build a Classifier that could distinguish between different classes of Fruits given their images.**

Continue Overview

Domain from which the project is proposed #1

1

Like we said we want to build a classifier that could classify images of fruits.

such classifier will be based on a very important field in machine learning called deep learning.

2

Deep Learning:

1. Deep learning is a class of machine learning algorithms that use multiple layers that contain nonlinear processing units .
2. Each level learns to transform its input data into a slightly more abstract and composite representation.
3. Deep neural networks have managed to outperform other machine learning algorithms.
4. They also achieved the first superhuman pattern recognition in certain domains.
5. This is further reinforced by the fact that deep learning is considered as an important step towards obtaining Strong AI.
6. deep neural networks - specifically convolutional neural networks - have been proved to obtain great results in the field of image recognition.

Continue Overview

Domain from which the project is proposed #2

- **Convolutional neural networks**

1. Convolutional neural networks (CNN) are part of the deep learning models. Such a network can be composed of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers.
2. In a typical CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layer and finally one or more fully connected layer.
3. A characteristic that sets apart the CNN from a regular neural network is taking into account the structure of the images while processing them. Note that a regular neural network converts the input in a one dimensional array which makes the trained classifier less sensitive to positional changes.

Problem Statement

Problem that is to be solved

- **The problem**

  we want to build a good classifier that could classify a fruit to different classes up to 81 distinguishable classes given an image of that fruit.

- **The Solution**

  we will build a deep-learning convolutional network based classifier, since their great performance in image classification and computer vision .. the classifier will be trained by large and high quality data set to receive good results.

- **Solution should satisfy the following**

  1. Accurate results on given test set.
  2. Beating or approaching the scores of compared benchmark models.
  3. Achieving satisfying results on real-world data(messy test set)

Evaluation Metrics

Evaluation metric that can be used to quantify the performance of both the benchmark model and the solution model presented.

- ## Metric used:
  - accuracy metric: is the number of correct predictions made as a ratio of all predictions made.
  - This is the most common evaluation metric for classification problems.
  - it's used since our data in each class is nearly balanced.

- ## Loss function used:

  categorical loss function: because we have multiclassification problem(81 class).


2. Analysis

Data Exploration

Information such as how the dataset or input was obtained, and the characteristics of the dataset.

- **Obtaining the input set**
  1. The images were obtained by filming the fruits while they are rotated by a motor and then extracting frames.
  2. Behind the fruits, a white sheet of paper is placed as background.
  3. due to the variations in the lighting conditions, the background was not uniform.. so an algorithm(flood fill type) was written to extract fruit from background.
  4. images were resized to 100x100 pixels.
  5. labels are given in next slide, some fruits have multiple varieties such as apples each of them being considered as a separate object. but since no scientific/popular name was found for each apple so it's labeled with digits (e.g. apple red 1, apple red 2 etc).

- **Characteristics**

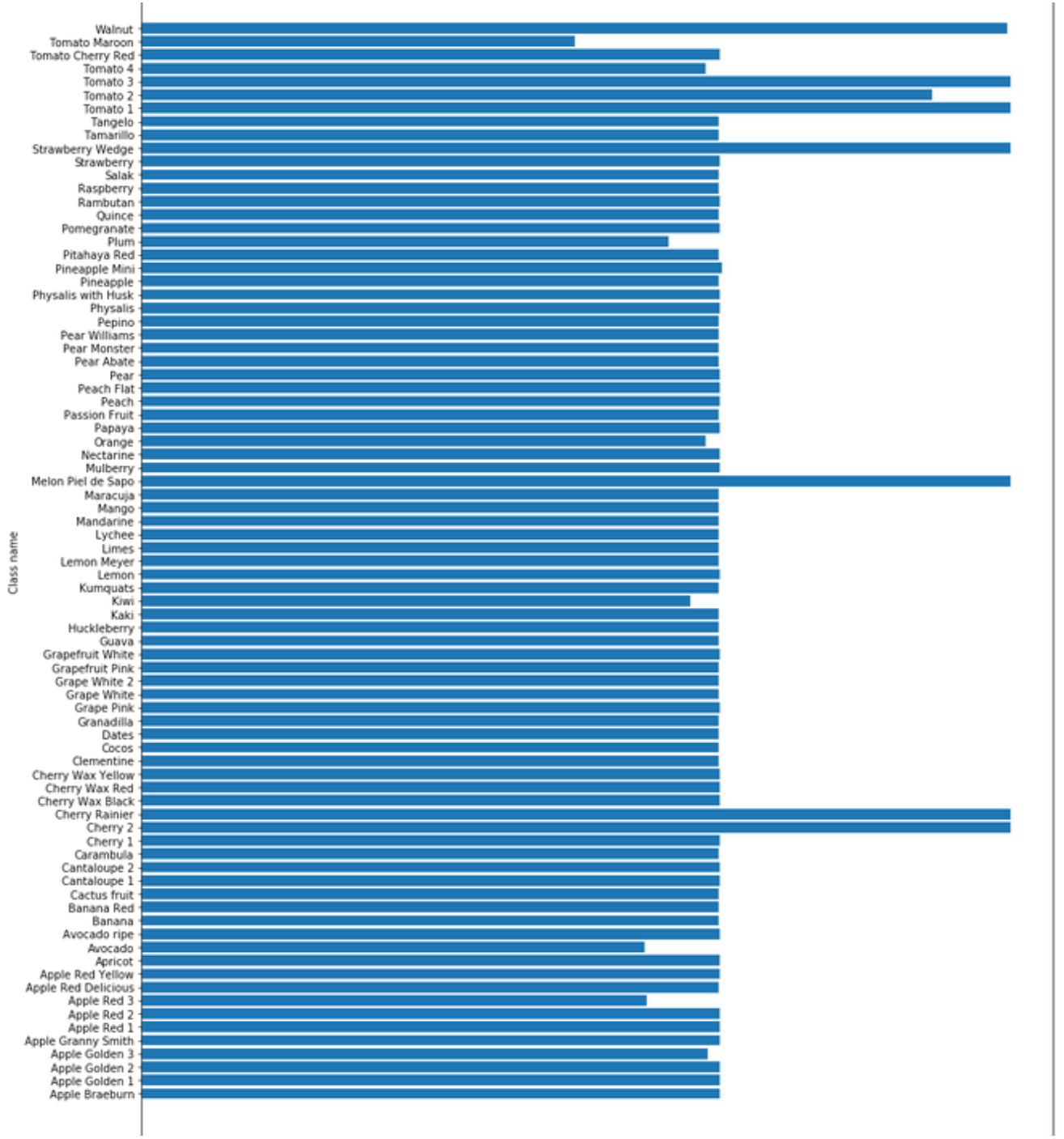
  the set contains 55244 images of 81 fruits and it is constantly updated with images of new fruits as soon as the authors have accesses to them.

Figure 1: Left-side: original image. Notice the background and the motor shaft. Right-side: the fruit after the background removal and after it was scaled down to 100x100 pixels.

Sample Data Grid Visualization

Data Distribution:

as we can see data is nearly balanced (each class has nearly equal set of images)



Detailed Distribution of Data

1

Label	Number of training images	Number of test images
Apple Braeburn	492	164
Apple Golden 1	492	164
Apple Golden 2	492	164
Apple Golden 3	481	161
Apple Granny Smith	492	164
Apple Red 1	492	164
Apple Red 2	492	164
Apple Red 3	429	144
Apple Red Delicious	490	166
Apple Red Yellow	492	164
Apricot	492	164
Avocado	427	143
Avocado ripe	491	166
Banana	490	166
Banana Red	490	166
Cactus fruit	490	166
Cantaloupe 1	492	164
Cantaloupe 2	492	164
Carambula	490	166
Cherry 1	492	164
Cherry 2	738	246
Cherry Rainier	738	246

2

Label	Number of training images	Number of test images
Cherry Wax Black	492	164
Cherry Wax Red	492	164
Cherry Wax Yellow	492	164
Clementine	490	166
Cocos	490	166
Dates	490	166
Granadilla	490	166
Grape Pink	492	164
Grape White	490	166
Grape White 2	490	166
Grapefruit Pink	490	166
Grapefruit White	492	164
Guava	490	166
Huckleberry	490	166
Kaki	490	166
Kiwi	466	156
Kumquats	490	166
Lemon	492	164
Lemon Meyer	490	166
Limes	490	166
Lychee	490	166
Mandarine	490	166
Mango	490	166
Maracuja	490	166
Melon Piel de Sapo	738	246
Mulberry	492	164
Nectarine	492	164
Orange	479	160
Papaya	492	164
Passion Fruit	490	166
Peach	492	164
Peach Flat	492	164
Pear	492	164
Pear Abate	490	166

3

Label	Number of training images	Number of test images
Pear Monster	490	166
Pear Williams	490	166
Pepino	490	166
Physalis	492	164
Physalis with Husk	492	164
Pineapple	490	166
Pineapple Mini	493	163
Pitahaya Red	490	166
Plum	447	151
Pomegranate	492	164
Quince	490	166
Rambutan	492	164
Raspberry	490	166
Salak	490	162
Strawberry	492	164
Strawberry Wedge	738	246
Tamarillo	490	166
Tangelo	490	166
Tomato 1	738	246
Tomato 2	672	225
Tomato 3	738	246
Tomato 4	479	160
Tomato Cherry Red	492	164
Tomato Maroon	367	127
Walnut	735	249

Algorithms and Techniques

Description of solution and Techniques used to solve the problem.

- **as previously mentioned building deep-learning based classifier is the main solution.**

1. since deep-learning algorithms outperformed some other supervised algorithms in classification , specifically in image classification problem like we have in hand.
2. I will not use MLPs since it's expected to get low accuracy, no surprise since in order to feed an image to MLP we first need to convert the image matrix to a vector with no special structure, it has no knowledge that these numbers were spatially arranged in a grid.. another issue is MLPs use a lot of parameters, even small size images can contain over 50 millions parameters, you can imagine that the computational complexity for even moderately sized images with large dataset could get out of control pretty fast..
3. I will use CNN's with typical structure .. that extracts patterns and lower dimensionality.
4. CNNs can detect patterns such as edges,shapes and particular characteristics such as colors gradient since they preserve spatial structure in your input data, by using windows-like (filters) to analyze images..
5. tuning hyperparameters, depending on how good accuracy could become.
6. I will add image augmentation to increase accuracy even more, image augmentation would depend on what data lacks.
7. then I will use pre-trained models that already trained over millions of images of different classes that would give us even higher accuracy.

Benchmark Model

Existing methods or known information in the domain and problem given, which can then be objectively compared to my solution.

1. **Compare the current stage of project with previous stages.**
   - transfer learning phase compared to usage of image augmentation compared to basic CNN implementation and applied tuning.

2. **Comparison to the paper results.**

3. **Comparisons to other implementations on Kaggle.**

[redacted]

Table 3: Results of training the neural network on fruits-360 dataset.

Scenario	Accuracy on training set	Accuracy on test set
Grayscale	99.53%	91.91%
RGB	99.51%	95.59%
HSV	99.32%	95.22%
HSV + Grayscale	98.72%	94.17%
HSV + Grayscale + hue/saturation change + flips	99.46%	96.41%

[redacted]

The images demonstrate accuracy gained in Paper.

3. Methodology

Data Preprocessing

1. **Data was clean (high quality images with no noise or ouliers)**
   - there was no need for cleaning data.
2. **Data is loaded in RGB color mode.**
3. **Size of each image is 100*100 pixels.**
4. **Image pixels is normalized to 0-1 range by dividing each pixel value by 255.**
5. **Further preprocessing is discussed in image augmentation implementation part.**

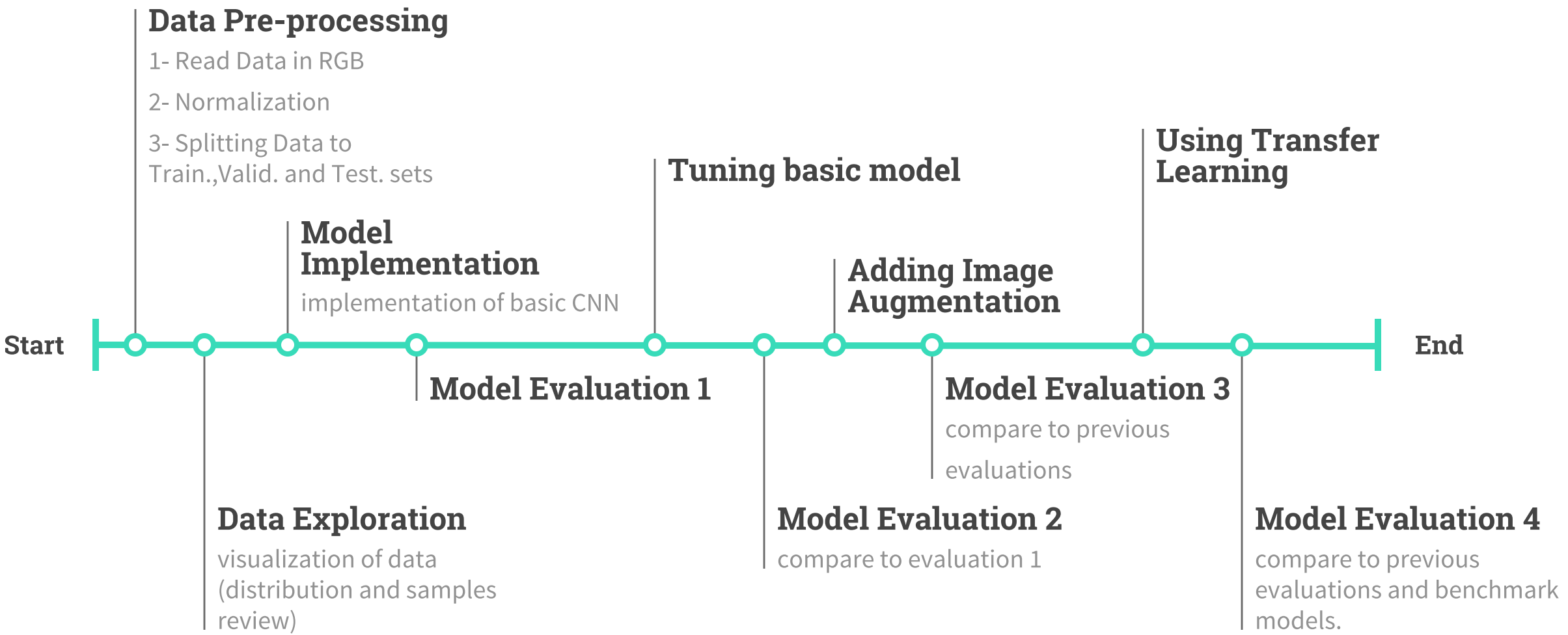
Dataset Splitting

- Data is divided (2/3 training, 1/3 test)
- the training set is split (75% training, 25% validation)

```
Found 31018 images belonging to 81 classes. (Training Set)
Found 10304 images belonging to 81 classes. (Validation Set)
Found 13877 images belonging to 81 classes. (Test Set)
```

Implementation

Implementation Timeline



Implementation

Data Pre-Processing

- **For Data preprocessing I used mainly ImageDataGenerator API from Keras as it's useful for the following.**

  1. fast for reading Data.

  2. easy splitting for data to train,test,valid.

  3. no need for hard coded labeling for data, as it contains funcion: flowfromdirectory() which label data depending on the subdirectory name they are in, and this fits exactly how our data is stored.

  4. preprocessing is so easy with it, as many preprocessing operations can be done on data with only passing one parameter to it's functions such as changing (image size in pixels, color mode, normalizing, zooming, shifting, rotating, brightness , ... etc)

  5. Rather than performing the operations on entire image dataset in memory, the API is designed to be iterated by the deep learning model fitting process, creating augmented image data just-in-time. This reduces memory overhead, but adds some additional time cost during model training.

Implementation

Implementation Notes

- **For all models implemented i used the following:**
  1. 50 epochs of training.
  2. early stopping with low patience (patience = 4) to lower the training time if there is no improvements; since the data is semi big the training takes time, and because i have low patience :'D .
  3. used Checkpointer to monitor the validation set loss and save the weights of the model that gave best (lowest) loss value, to avoid overfitting.

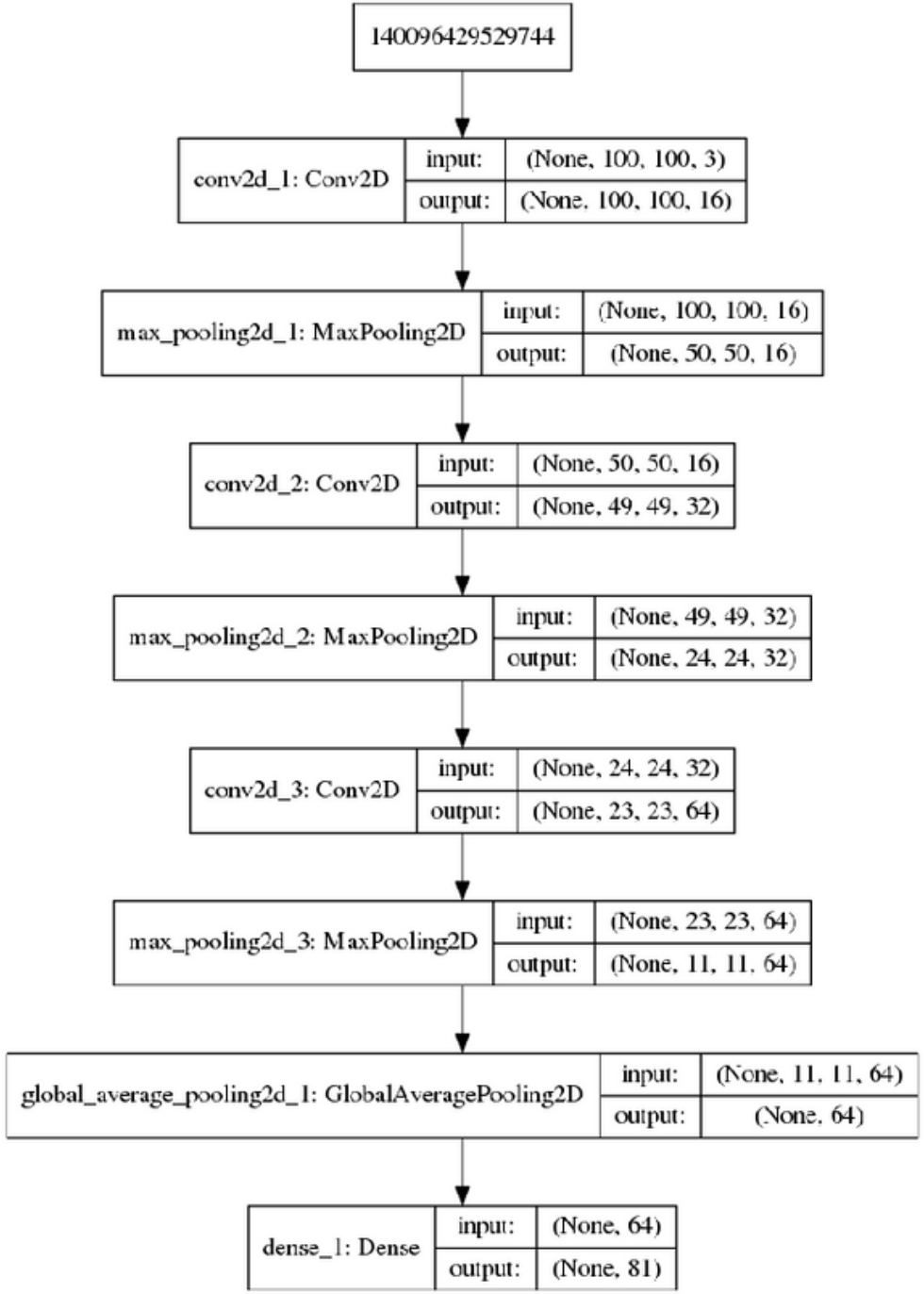
Implementation

Basic CNN Implementation

- **Typical CNN structure is used**
  - it's like a standard structure, as it consists of 3 hidden layers each to detect different kinds of features, upper one can detect edges, second can detect shapes, last is for more specific and deep details like colors gradient .. each one has a following max pooling for the decreasing of dimensionality.
  - followed by global average pooling (GAP) layer to minimize overfitting by reducing the total number of parameters in the model. Similar to max pooling layers, GAP layers are used to reduce the spatial dimensions of a three-dimensional tensor.
  - Ending with a dense layer with softmax activation function to get classification results.
- **Used RMSProp. optimizer.**

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 49, 49, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 23, 23, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 11, 11, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 81)                5265
=================================================================
Total params: 15,809
Trainable params: 15,809
```

Basic CNN Structure Visualization



Implementation

Basic CNN Evaluation

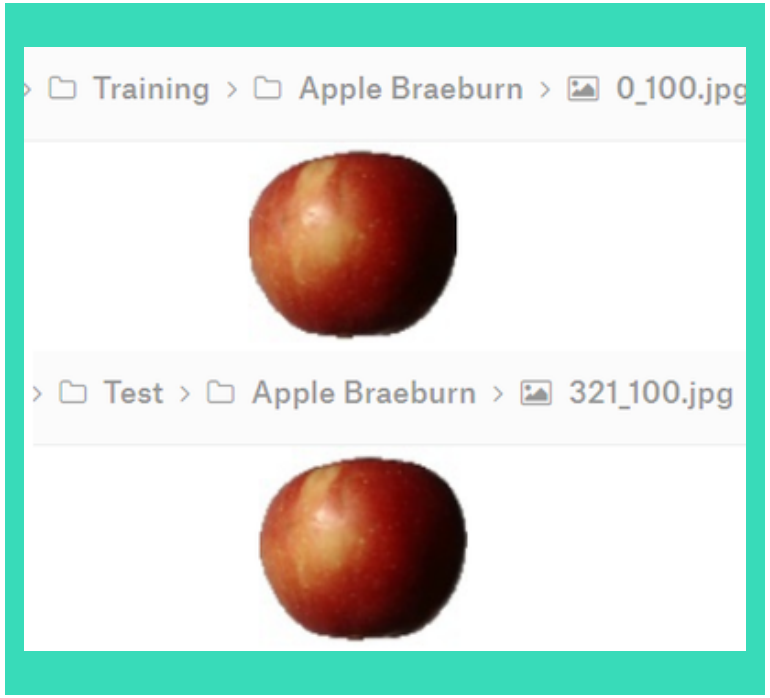
- **Basic CNN structure got 94% accuracy on test set.**
  - actually that wasn't surprising since we have a satisfying-sized dataset
  - quality of the dataset images also helped to get such result.
  - **the most IMPORTANT reason actually is that the test set isn't challenging. meaning, that it contains images with similar quality to the training set.**
  - the high score on such test set doesn't necessarily imply that our model is able to generalize when seeing real-life test set (messier data)
  - so, i will make another testing set to measure our model generalization against real-life test set.

- **The Real Test set**
  - it contains 50 images of 10 classes(fruits) that is most famous in my region, also avoided classes that are labeled with ambiguous name like apple 1,2.
  - the classes are (apple red yellow, Bannana, Kiwi, Lemon, Mango, Orange, Peach, Pear, Pomergranate, Strawberry)
  - each class contains 5 images, 2 of them are considered simple to detect because they contain single fruit, the other 3 are harder since they contain multiple ones or messy background.

- **Basic Model was able to successfully classify 4 images of 50 in that set, with accuracy = 8%**

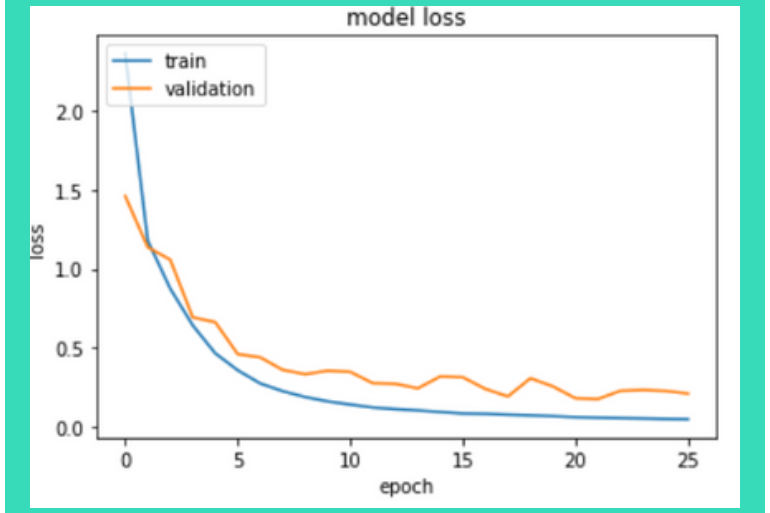
Implementation

Basic CNN Evaluation



Example of how similar Training set to the Test set (regular test set is not challenging)

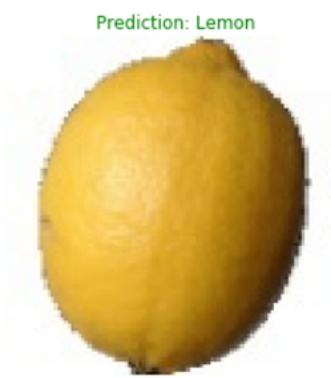
Basic CNN validation accuracy

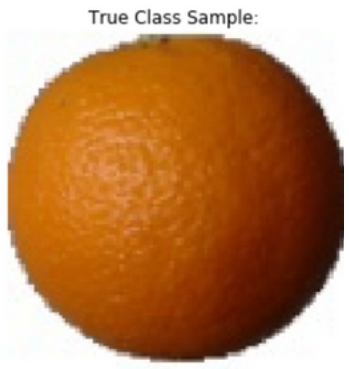


Basic CNN validation loss

Implementation

Basic CNN Evaluation with real test set



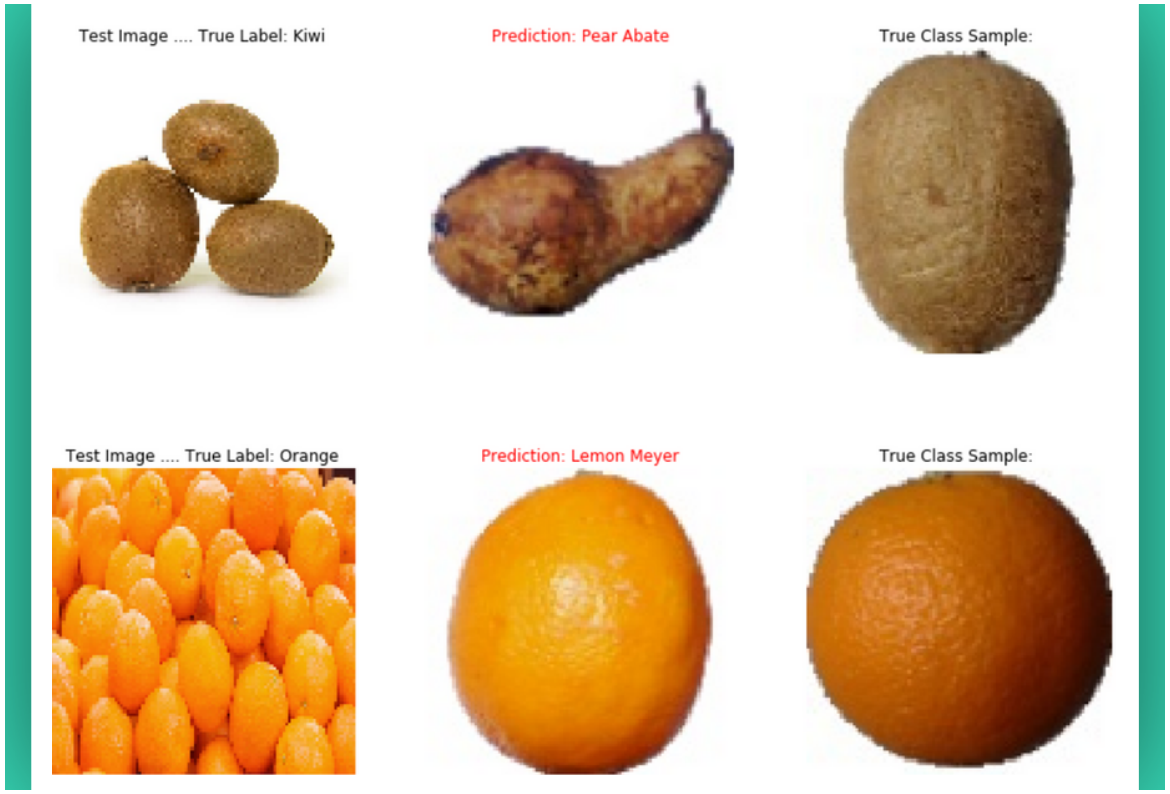


Right Predictions

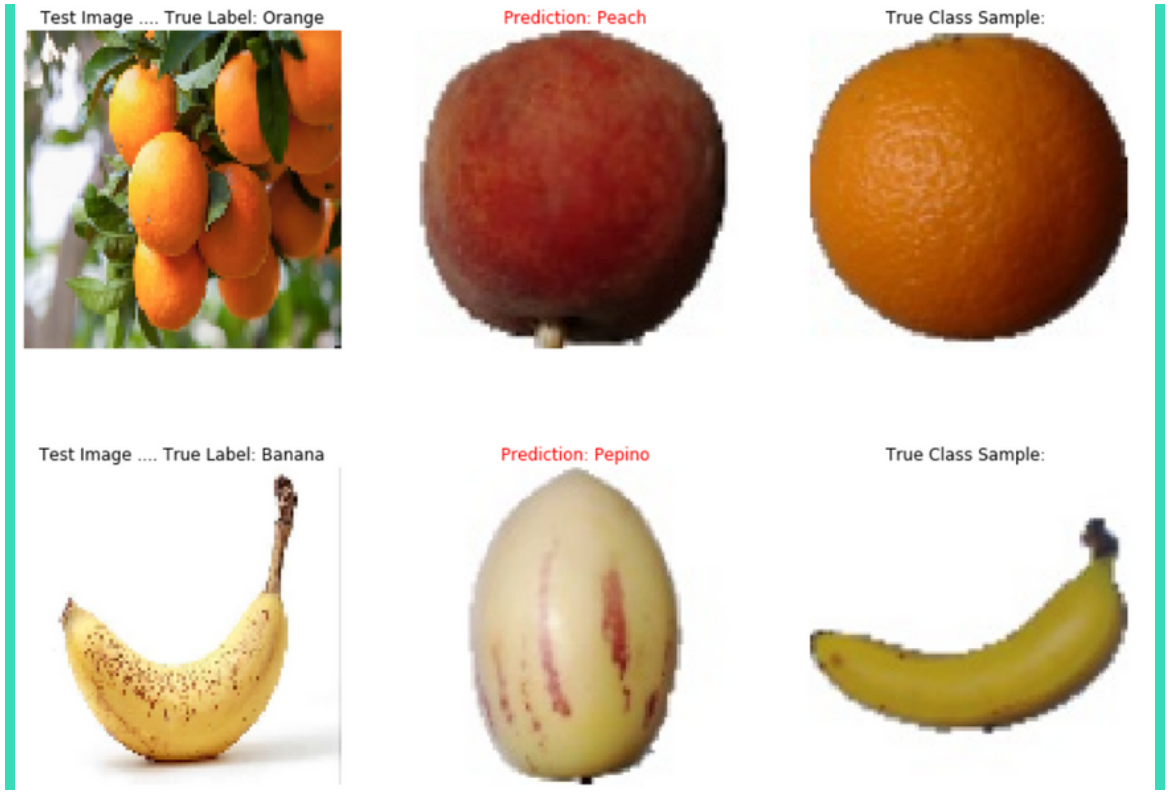
Right Predictions

Implementation

Basic CNN Evaluation with real test set



Wrong Predictions



Wrong Predictions

Refinement

CNN Tuning

- **The purpose of tuning now is getting higher score in both Test set and real-Test set.**
- **I chose to tune the learning rate.**
  - Instead of using a static learning rate i used 3 type of learning rates.
- **Decreasing learning rate (Details here)**
  - has the benefit of making large changes at the beginning of the training procedure when larger learning rate values are used, and decreasing the learning rate such that a smaller rate and therefore smaller training updates are made to weights later in the training procedure. This has the effect of quickly learning good weights early and fine tuning them later.
  - Two popular and easy to use learning rate schedules are as follows:
  - Decrease the learning rate gradually based on the epoch.
  - Decrease the learning rate using punctuated large drops at specific epochs.
- **Decreasing Learning rates methods**

  1- decaying learning rate :

  `LearningRate = LearningRate * 1/(1 + decay * epoch)`

  2- dropping learning rate:

  `LearningRate = InitialLearningRate * DropRate^floor(Epoch / EpochDrop)`
- **Adaptive Learning rate**
  - used Adam optimizer for that, because it's simple and achieves good results fast.
  - Adam is different to classical stochastic gradient descent.
  - Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change during training(unless using decreasing schedules).
  - but with Adam, a learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds.
  - More details on its benefits are discussed here

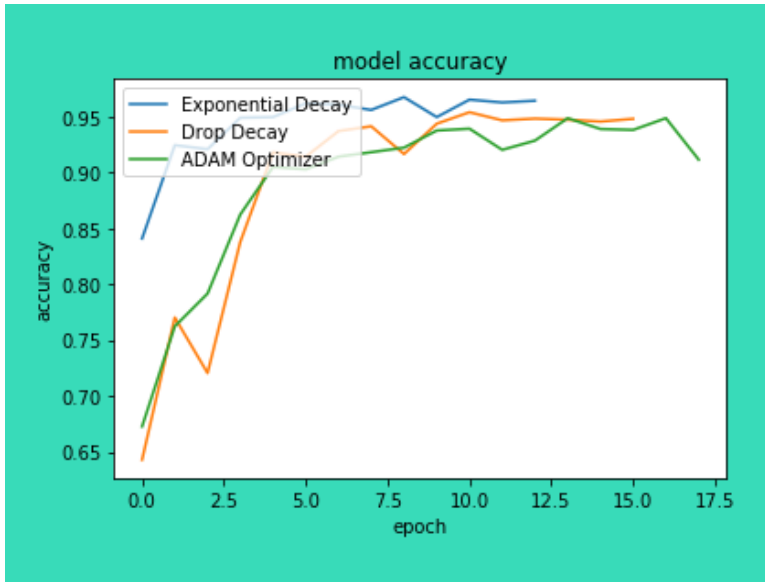
Refinement

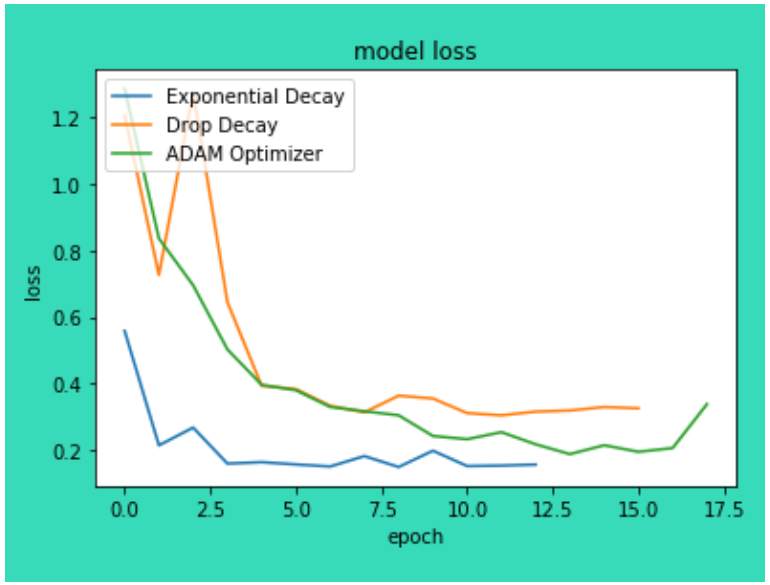
Tuned CNN Evaluation

```
del_exp_decay.evaluate_generator(test_generator, test_generator.
.13065101978553254, 0.9690855372198602]

del_drop_decay.evaluate_generator(test_generator, test_generator
.11713897851970824, 0.9621676154788499]

del_adam.evaluate_generator(test_generator, test_generator.sampl
.20386002428693534, 0.9517186711825323]
```





Test Accuracy Comparison

Accuracy Comparison

- as noticed Decreasing learning rate methods achieved higher accuracy earlier since they begin with high learning rate that decreases gradually

Loss Comparison

- also momentum helps to not get stuck in local minima as noticed in (drop decay plot)

Refinement

Tuned CNN Evaluation on Real Test Set

```
Model 1 predicted 4 images right,accuracy = 8.0%  (Decaying LR Model)
Model 2 predicted 2 images right,accuracy = 4.0%  (Dropping LR Model)
Model 3 predicted 5 images right,accuracy = 10.0% (Adam Model)
```

Real-Test Accuracy Comparison

- as noticed Adam despite having the lowest accuracy between the 3 on the regular Test set, but it has highest accuracy on real Test set
- actually it depends a lot on what characteristics that the model was able to learn that helped him in distinguishing specific cases.
- but it could be that Adam is less Memorizing for our regular test set and so it's considered more generalizing.
- however we didn't gain much using only such tuning, but we know for sure that the dataset here is the most part that actually needs tuning so next i'll use image augmentation, i'll use Adam optimizer in next implementations since it's suitable for most cases.

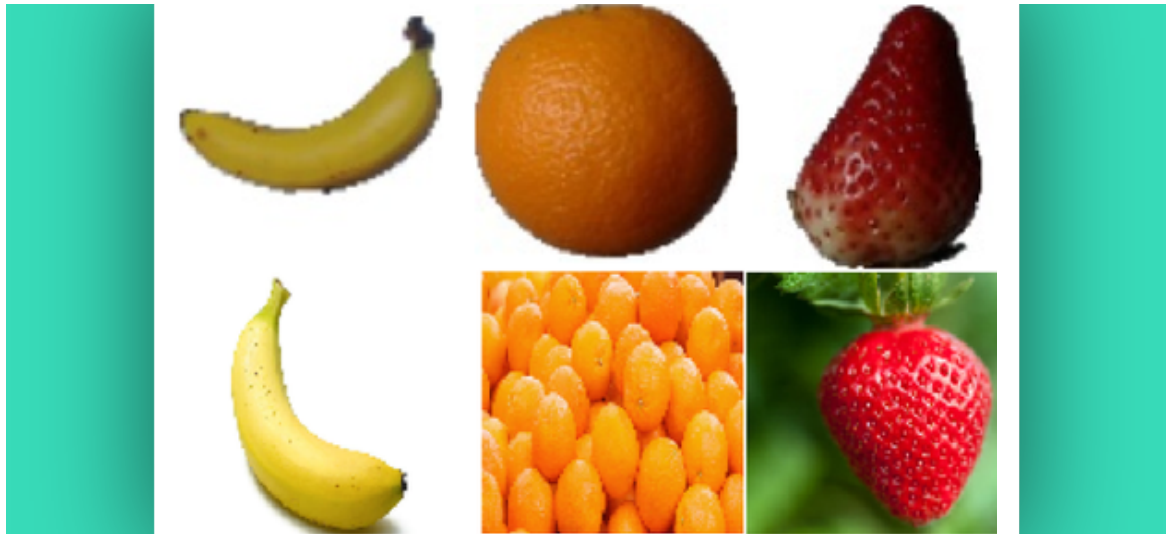
Refinement

Image Augmentation

- **Why Image Augmentation?**
  - the data has to have good diversity as the object of interest needs to be present in varying sizes, lighting conditions and poses if we desire that our network generalizes well during the testing (or deployment) phase.
  - To overcome this problem of limited diversity of data, we generate(manufacture) our own data with the existing data which we have. This methodology of generating our own data is known as data augmentation.
  - Our dataset here is good in size but is not diverse so image augmentation can be useful in our case.

Refinement

Image Augmentation



As we see, the top row contains samples of the data in our training set, and the bottom row contains real life data.. notice that our training data are a bit dim looking with low brightness..

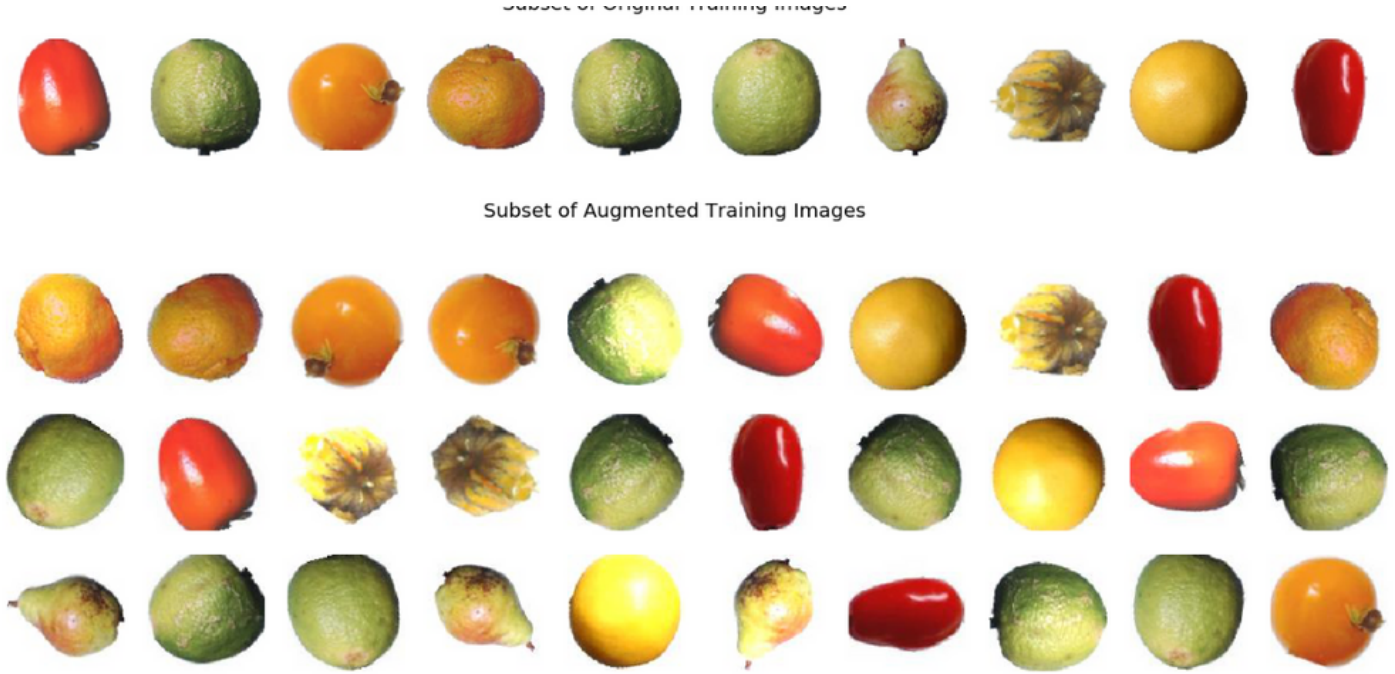
so i used Brightness augmentation on training images to train our model to recognize such cases.

These real life examples contain fruits that are rotated or have hidden portions.

So i used Rotation, flipping and shearing for the network to be able to recognize different poses .. also used shifting with low value to hide less than 20% of the image so that our model could detect such cases too.

Refinement

Image Augmentation

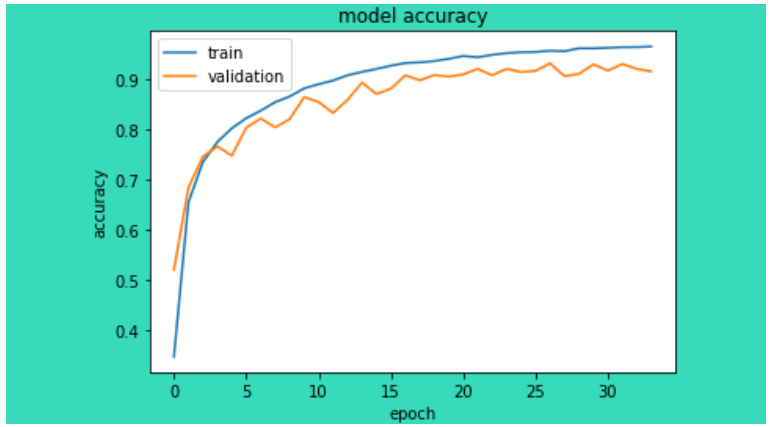


Visualization of Augmented Samples

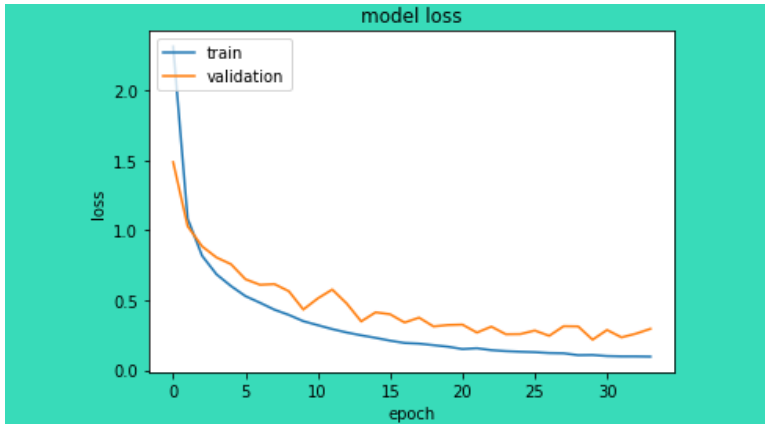
Note: I chose the filling method to the newly added pixels resulting from augmentations such as shearing or shifting to be with white pixels so images don't have weird looking or noise.

Refinement

Image Augmentation Evaluation



Accuracy plot of the model



Loss function Plot

- notice that with image augmentation we have more number of epochs needed to gain stable value.
- and that makes sense since we have bigger data than before so more learning time and steps are needed

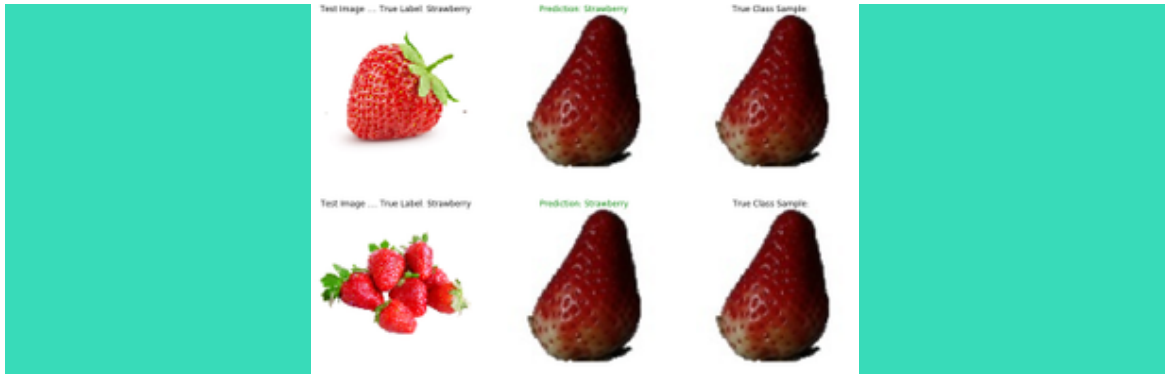
```
8]: model.evaluate_generator(test_generator, tes
8]: [0.20078831853925608, 0.9285148086762268]
```

Accuracy on Regular Test Set

- Lower accuracy! ..but why?!
- actually this also makes sense, adding brightness, shifting or shearing although it helps our model to generalize but it's also considered noise added to our images compared to the high quality images we have in validation set or the regular test set.

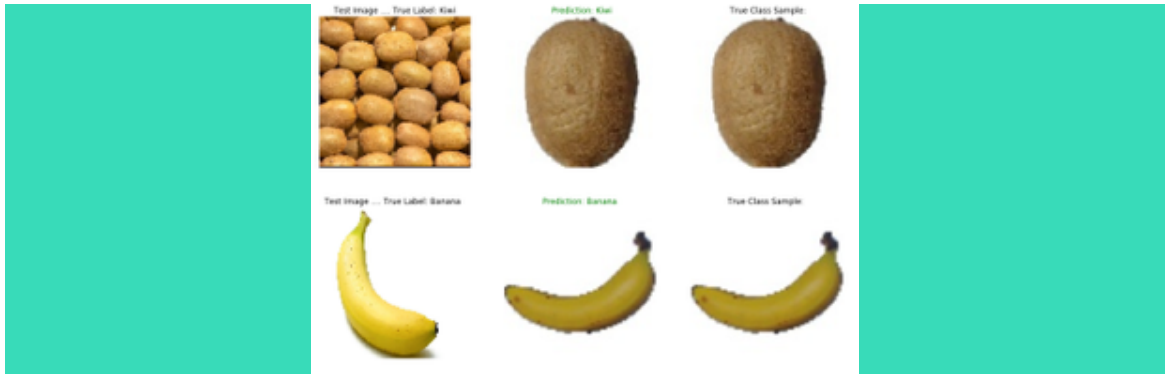
Implementation

Image Augmentation Evaluation with real test set



Right Predictions

Looks like we have manged to get right classifications of strawberry by adding the brightness to the data



Right Predictions

also some right prediction resulting from the mix of (brightness,shifting and rotating) added to our data

I got with image augmentation, 8 right predicted images with accuracy 16%, in some cases, I got 10 right predictions but no less than 6, it's an improvement but not much... maybe our model lacks depth, that's why it cannot detect more specific details... since training and tuning our model more would take some time, why not we use a model that already trained over millions of images and has deep structure.. so next I'll try transfer learning.

Refinement

Transfer Learning

- **Why Transfer Learning?**
  - Basically, we take a pre-trained model (the weights and parameters of a network that has been trained on a large dataset by somebody else) and “fine-tune” the model with our own dataset.
  - The idea is that this pre-trained model will either provide the initialized weights leading to a faster convergence or it will act as a fixed feature extractor for the task of interest.

- **These two major transfer learning scenarios look as follows:**
  - Finetuning the convnet: Instead of random initializaion, we initialize the network with a pretrained network, like the one that has been trained on a large dataset like imagenet 1000. Rest of the training looks as usual. In this scenario the entire network needs to be retrained on the dataset of our interest.
  - ConvNet as fixed feature extractor: Here, we will freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.
  - I won't try CovNet with Feature extraction in this project since it needs considerably large memory amount which is not enough on kaggle.
  - Instead i will try to use pre-trained model but with random initialization(Vanilla Model) to benefit only from the structure of that model.

Refinement

Transfer Learning Implementation

- **I used Xception as the pre-trained model.**
  - since it achieved good accuracy as we can see in keras comparison here.
  - it has good depth structure(126 layers) and parameters amount (~23 Millions)

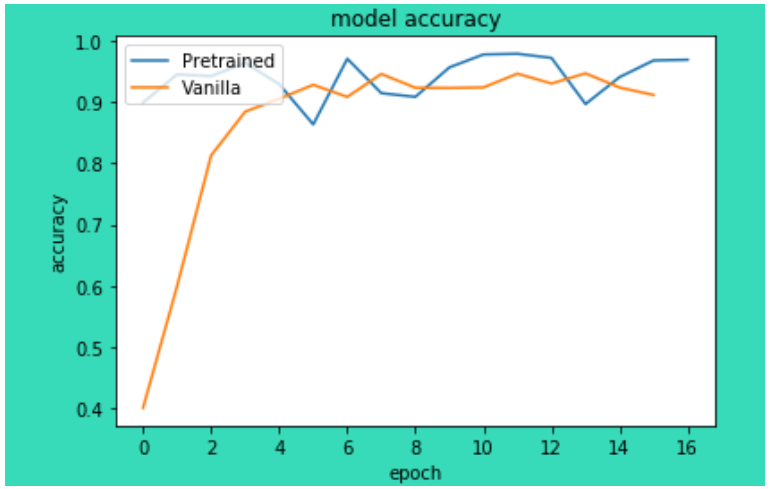
- **I compared two models**
  - the first is Xception model with loaded pre-trained weights.
  - the second is called vanilla model, with same structure of Xception model but randomized weights.
  - both having the top layer removed and i added one global average layer followed with dense softmax layer for final classification.
  - used Adam optimizer for both models.

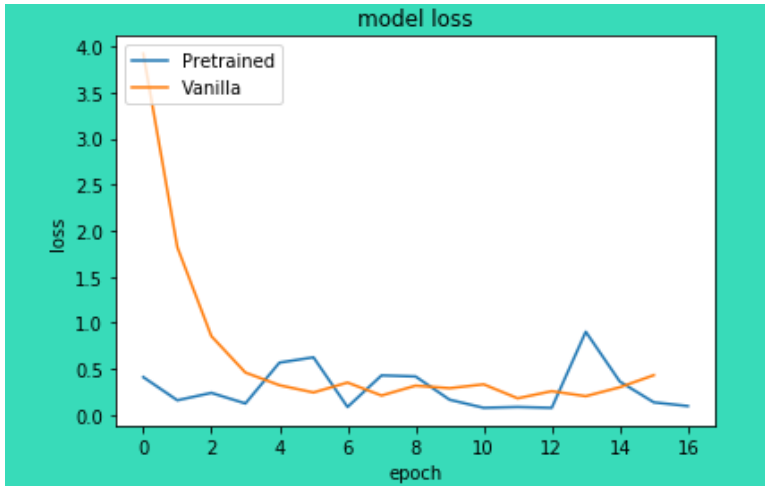
Refinement

Transfer Learning Evaluation

```
xception_transfer.evaluate_generator(test_gener
[0.10705800062605365, 0.968220797002234]

xception_transfer_vanilla.evaluate_generator(te
[0.08635824162688069, 0.9711032643943215]
```





Test Accuracy Comparison

- both have nearly equal score but at least we have more improvements over previous augmentation stage.

Accuracy Comparison

- as noticed pre-trained weights model is able to gain high accuracy much earlier since it doesn't need to change much in it's weights because it's already trained. on the other hand vanilla model needed some time to achieve stable good score.

Loss Comparison

- both models did good on validation set and regular test set even when using image augmentation. (~98% when not using image augmentation)

Refinement

Transfer Learning Evaluation on Real Test Set

```
Model 1 predicted 7 images right,accuracy = 14.0% (Pre-trained Model)
Model 2 predicted 7 images right,accuracy = 14.0%  (Vanilla Model)
```

Both didn't do any better in terms of generalization, but i guess that's the best i can do for now given such data, also tuning such models takes long time in training and waiting to see results of improvements... so i'm satisfied with results i got till now.

4. Results

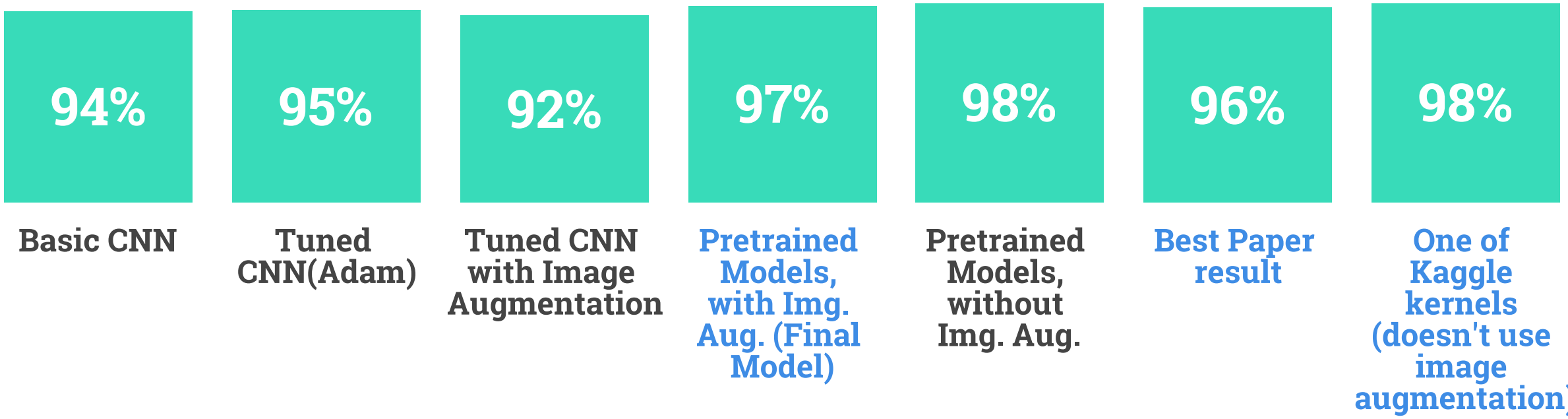
Model Evaluation and Validation

The final model's qualities and parameters

- **I will consider vanilla pre-trained model as my final model**
  - it has good accuracy on the regular test set. 97%
  - also not bad accuracy on the real test set. 16%
  - it can be trained to generalize if new data being added, since the other pre-trained model could over-fit easily if newly added data differ a lot from the imagenet 1000 dataset, as it's using pre-trained weights.
- **I used Xception as the pre-trained model with random weights and no top layer.**
- **Added Global Average layer with final dense layer with softmax activation for classification**
- **Used Adam optimizer, trained model over 50 epochs with early stopping with patience = 4, also used checkpointer to save best weights and load them later to avoid over fitting.**
- **Used Image Augmentation (shifting, shearing, rotating, brightness and flipping) to make our data diverse and our model more generalizing.**
- **Validation and Scores plotting is in previous section.**

Justification

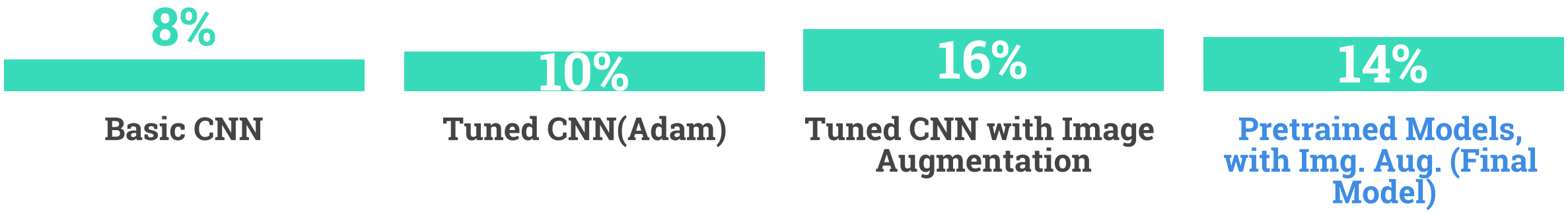
The final results are compared to the benchmark results. (results on the regular test set)



Our Final model was successful in beating most benchmark scores specificaly the scores from the Paper and Other Kaggle kernels, so our results is considered satisfying.

Justification

The final results are compared to the benchmark results. (results on the real test set)

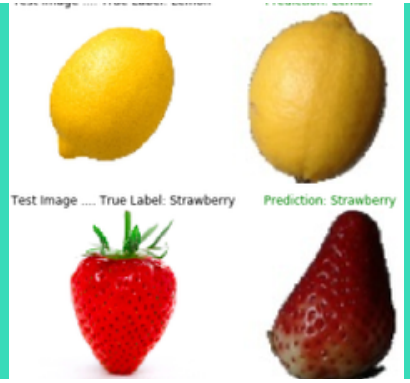


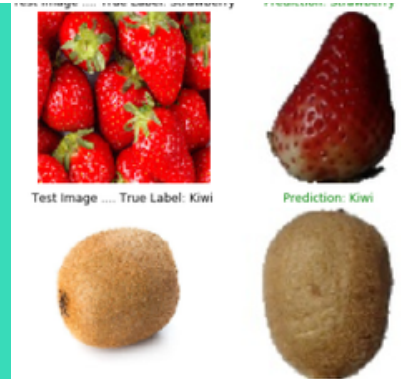
Also our final is considered a bit more generalizing than other basic structures.

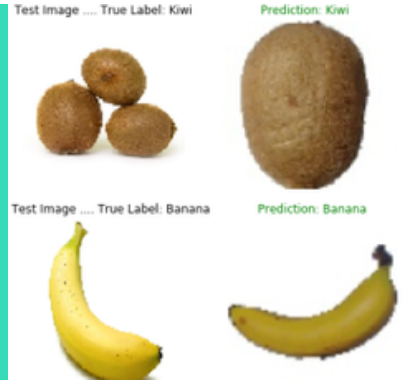
5. Conclusion

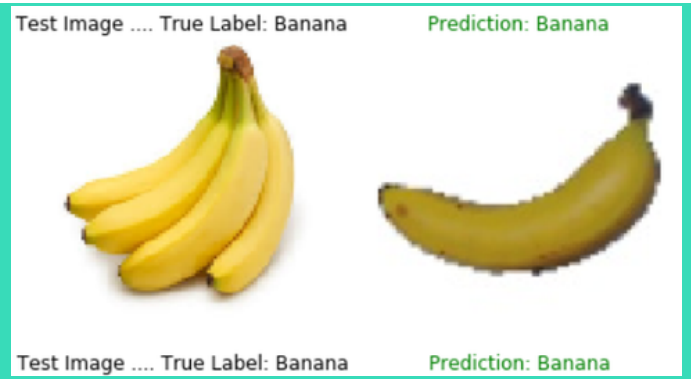
Free Form Visualization

Our Final was able to beat benchmark models in accuracy on the regular test set, but also is able to generalize and predict some real data life cases successfully









Reflection

summarization to the end-to-end problem solution and discussion on some project aspects

- **at the beginning of the project my objective was to get high accuracy on the regular test set and beat the benchmark models, actually that wasn't a hard part at all for the following reasons:**
  - the dataset has a fairly good size with high quality images.
  - the test set quality also good, and the fruits samples used in it is very similar to the training set.
  - that made the test set not challenging, a very basic CNN architecture can easily achieve more than 94% accuracy on such data set .

- **I thought with such large data set and high score results i could be able to test my model on real-life test set and get good results .. but that was harder than i imagined for the following reasons..**
  - our training set has only high quality images, with the same lighting conditions and backgrounds, this is actually bad for generalizing.
  - data in that training set is not diverse, the following slide will demonstrate that in more details.
  - some types of fruits such as Mango in the dataset was specific to only one type of it (the green ones), of course our model won't be able to detect other types of mango that has other collors.
  - taking pics of the same fruit over and over is the main reason that our data isn't diverse and our models isn't able to generalize since even single type of fruit could have different shape and slightly different colors.

Reflection

Dataset weaknesses

Training Sample

Real Life Sample

(Apple)

(Pink Grape)

as we can see, huge difference in color and shape between training sample and real-life sample.. since our dataset consists of that specific sample from different poses it will cause our model to memorize such shape and color, and won't be able to recognize any other shapes or colors.

Other bad characteristic of used data set, grapes and cherries are considered small sized fruits but in the dataset are as same size as large fruits like apples.. CNN models will have hard time distinguishing such types from each others.. also fruits like grapes always appear in groups so using that single sample for training isn't convenient

Reflection

- **also the data set has other weaknesses**
  - resizing images to small size like 100*100 can affect accuracy of classification badly, small size is detrimental when you have too similar objects (a red cherry looks very similar to a red apple in small images).
  - also a bad effect of using small size images is that some pre-trained models require bigger size for better performance.
  - labeling some fruits such as apples with apple1,apple2 .. etc , though it's valid but it's considered a vague labeling for data.
- **With That being said, my main objective was to build a model that is good in generalizing, and that was challenging.**
- **so i used image augmentation to make our data set more diverse and alleviate the bad characteristics of the Dataset**
- **sadly, the results of generalization wasn't satisfying, as some of the bad characteristics of the dataset aren't avoidable.. even with our best model we didn't gain too much generalization.**
- **also one of the main problems that limited my solution, was the long time used for training on such big dataset... tuning and waiting for results takes too much time specially in transfer learning; as one epoch takes approximately 6 minutes, so it was hard to make further tunings or using more models.**
- **however, it was interesting to work in this project and trying to solve such problem.**

Improvement

- **Mainly we can try to improve the dataset.**
  - adding more diverse data
  - avoid naming ambiguous fruit class names by simply gathering such subclasses into one class (apple1 , apple2 ,apple 3 unified into one class called apple)
  - using data with more good pixel size (256*256) for example, but this will require more computational power and training time :/ .
- **We could use other transfer learning models like VGG16,19 and InceptionV3.**
- **We could use the other type of transfer learning which is feature extraction model but on other environment like colab instead of kaggle.**
- **Other tweaks could be done to the structure of the final model as well to get higher accuracy.**

Resources

1. **Published research paper:**

   Horea Muresan, Mihai Oltean, Fruit recognition from images using deep learning, Acta Univ. Sapientiae, Informatica Vol. 10, Issue 1, pp. 26-42, 2018.

2. **Dataset can be downloaded from:**

   Fruits 360 Dataset on GitHub.

   Fruits 360 Dataset on Kaggle.

3. **A Study on Image Processing Methods for Fruit Classification**

   Link

4. **Evaluation Metric used**

   metrics-evaluate-machine-learning-algorithms

5. **Sklearn Docs , Keras Docs**

6. **Keras pretrained models weights to use on kaggle for download here**

7. **Real Test Set can be download from:**

   realtestdata on Kaggle

8. **Image Augmentation Resources**

   here

9. **Changing Learning Rate**

   here and here

10. **Adam Optimizer**

    here

11. **Useful Kaggle Kernels**

    here and here