CS-E4890 - Deep Learning
Group: FruitDealer
Project Report

1 Problem and data description

Definition 1.1: Data

Project uses the dataset found in a research "Fruit recognition from images using deep learning" by Horea Muresan and Mihai Oltean.

Data Properties

Property	value
Total number of images	71125
Training set size	53177 images (one fruit per image)
Test set size	17845 images (one fruit per image)
Multi-fruits set size	103 images (more than one fruit (or fruit class) per image)
Number of classes	103 (fruits)
Image size	100x100 pixels

Each image is read so that each pixel's RBG value is transformed via normalization to a mean and std of 0.5.

Definition 1.2: Problem

Goal of the project is use the data defined in definition 1.1 to solve a multiclass classification problem where given images are categorized to correct class by using Recurrent Convolutional Neural Network.
In addition to trying to solve this multiclass classification problem, project also compares this approach to Convolutional Neural Network used by Horea Muresan and Mihai Oltean in their research to see whether there is an improvement to be had by using a completely different Convulutional Neural Network.

Definition 1.3: Research Material

Two research papers were used:

(a) "Fruit recognition from images using deep learning" by Horea Muresan and Mihai Oltean.

(b) "Recurrent Convolutional Neural Network for Object Recognition" by Ming Liang and Xiaolin Hu.

2 Method

As said in definition 1.3 this project will focus on two methods: original CNN used by Horea Muresan and Mihai Oltean and RCNN described in a research "Recurrent Convolutional Neural Network for Object Recognition" by Ming Liang and Xiaolin Hu.

For both cases project will use Adam Optimization Algorithm with learning rare of 0.001 and Cross Entropy Loss function for training the models.

Definition 2.1: Convolutional Neural Network (CNN)

Horea Muresan's and Mihai Oltean's research uses multiple different variations of dimensions in their research, but this project focuses on the following CNN which gave their best results:

Layer type	Dimensions	Output
Convolutional	5x5x4	16
ReLU		
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 16	32
ReLU		
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 32	64
ReLU		
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 64	128
ReLU		
Max pooling	2 x 2 — Stride: 2	-
Fully connected	5 x 5 x 128	1024
ReLU		
Dropout(p=0.8)		
Fully connected	1024	256
ReLU		
Dropout(p=0.8)		
Softmax	256	60

So in total, 4 convulational layers, each followed by max pooling, which are then flollowed by two fully connected linear layers after which softmax is used to give 103 outputs (i.e. the number of classes). Linear Layers are also followed by a dropout with 0.8 probability.

Definition 2.2: Recurrent Convolutional Neural Network (RCNN)

RCNN shares many similar charasteristics of normal CNN, like the name applies it uses Convolutional layers. Unlike normal CNN, all of the convolutional layers share the same dimensions (usually defined/indicated with variable K, i.e. RCNN-92 would indicate that it uses convolutional layers with dimension of 92).

The main strucutre of RCNN is based on 5 different layers:

Layer	Layer type	Dimensions	Output
1	Convolutional	5 x 5 x 4	K
-	Max pooling	3 x 3 — Stride: 2	-
2	RCL	3 x 3 x K — Stride: 1	-
-	Dropout(p=0.8)		-
3	RCL	3 x 3 x K — Stride: 1	-
-	Max pooling	3 x 3 — Stride: 2	-
-	Dropout(p=0.8)		-
4	RCL	3 x 3 x K — Stride: 1	-
-	Dropout(p=0.8)		-
5	RCL	3 x 3 x K — Stride: 1	-
-	Max pooling (global)	Size of previous layer	-
-	Softmax	-	150

So each (except last) Recurrent Convolutional Layer (RCL) is followed by dropout with probability of 0.8. Max pooling is done after the first convolutional layer and at the half point of RCLs. Global max pooling is done after all RCLs which is then followed by softmax that gives an output with size of number of classes.
So what is a RCL? It's a series of convulational layers with same dimensions that are repeated T times. In this project we focus on T=3 which means that each RCL consists total of 4 convulational layers. T=3 also provided the best results in Ming Liang's and Xiaolin Hu's research. Structure of RCL is described in the table below.

RCL

T	Layer type	Input	Notes
0	Convolutional	3 x 3 x K Stride=1	No weight sharing
-	ReLU		
-	BatchNorm	K-features	
1	Convolutional	3 x 3 x K Stride=1 Padding=1	weight sharing T1-T3
-	ReLU		
-	T0 + T1		Sum of two layers
-	BatchNorm	K-features	
2	Convolutional	3 x 3 x K Stride=1 Padding=1	weight sharing T1-T3
-	ReLU		
-	T0 + T2		Sum of two layers
-	BatchNorm	K-features	
3	Convolutional	3 x 3 x K Stride=1 Padding=1	weight sharing T1-T3
-	ReLU		
-	T0 + T3		Sum of two layers
-	BatchNorm	K-features	

So T=0 differs from other of the convolutional layers a little bit. It does not share the weights with the other layers, while T=1-3 use the same weights for each layer. In addition, in T=1-3 we also add T=0 layer to our current layer before we move to next one.

> For this project, K=32 is chosen since video card memory would not be enough to support much bigger dimensions.

Project could have chosen different parameters for these models and test out different variations, but unfortunately it did not fit to the scope of this project. Because of this, project focuses on the best models found in each research material and uses parameters that can be supported with 12GB of Video RAM.

3 Experiments and results

Since we want to get as comparable results as possible, project built both models described in section 2 (which can be looked into in section 5). In both cases project used Adam Optimization Algorithm with learning rate of 0.001 and Cross Entropy Loss for training the models.

For CNN, training configured to batch size of 60 which was run 75 epochs. As for RCNN, batch size of 150 was chosen and it was run only 10 epochs. Training and Validation sets were done using the original data split defined in definition 1.1.

This training setup resulted following accuracy curves visualized in a plot:

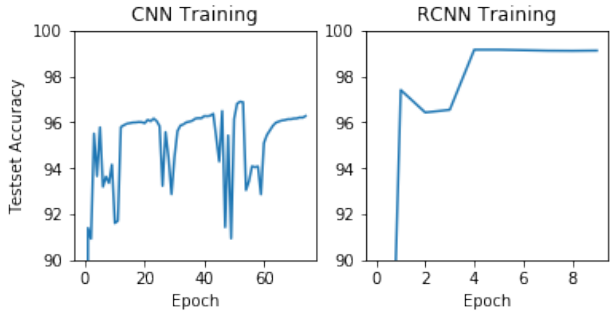

Figure 1: Training results of CNN and RCNN

From which we can extract best and final results for each training and compare it to the research:

	H. Muresan's and M. Oltean's Research	CNN Pytorch (best)	CNN Pytorch (final)	RCNN Pytorch (best)	RCNN Pytorch (final)
Validation Set Accuracy	95.88%	96.90%	96.27%	99.16%	99.12%

From these results we can see quite significant improvement in test set where we can see over 2% improvement over the Pytorch reimplementation of H. Muresan's and M. Oltean's Research and the research itself. Pytorch implementation seems to do slightly better than indicated in the research paper, but arguably not significant.

However, while CNN was able to take full power of the GPU, current RCNN Pytorch implementation was able to take advantage about 17% of the Nvidia 2080 TI, leading to believe that some preprocessing steps could bring huge improvements to the training performance of the model. However due to the scope of this project these improvements were left out. For RCNN, easy fixes to batch size and RCNN dimension sizes only affected how much of memory the model will eat up. Though, even this observation would require much more thorough investigation which is out of scope for this project.

4 Conclusions

RCNN-32 was able to outperform tradition CNN with quite significant margin, with quite small layer dimensionality, so there might be further improvements to be had by using even bigger dimensions or tweaking other parameters of the network.

Coding the RCNN model also proved to be little more challenging than I expected. Even though the dimensions of the network are quite simple, the Pytorch implementation of RCL is not as clear cut as it would be with either Tensorflow or Keras. This is especially true when you have to use weight sharing for RCL's layers or figuring out padding and strides for some of the layers (where as in Tensorflow or Keras you could use keyword "same"). But as it is with deeper (i.e. more layers in total) networks, it's more challenging to check that each part is implemented correctly.

Code could also be improved by saving the best x models during the execution and picking up the the best one (i.e. best validation/test accuracy) after all iterations have finished. Now it only saves the last one.

5 Code

Code for RCNN and CNN implementation are added to the end of this report. For easier reading, you can also look them up in Github.

Description	**URL**
Repository containing all the files and environment used for this project	Repository
pretrained RCNN Model	RCNN Model
pretrained RCNN Model	CNN Model
RCNN Code	RCNN Model
CNN Code	CNN Model