Caltech101

Object Classification

Leo Gan, Hanming Wang, Tilak Agarwal, Hunter Chun

Dataset description

101 categories of objects.

Each image is colored, which are stored as 3 color channels(RGB).

Each category has some items in the format of jpg, range from the least has 31 items, and the most has 800 items.

The dataset has 8677 images in total.

The average dimensions of all images is 300x200

Objective

The objective of this project is to implement a simple object classification model using two different methods, namely, **neural network** and **locality sensitive hashing**.

In the following section, we will introduce how we implement these two methods and how well they performed.

Neural Network Approach

Convolutional Neural Network(CNN)

- Using multiple convolutionary 2d layers
- Typical structure:
    - Input -> Conv. -> Conv. -> Maxpool -> Conv. -> Conv. -> MaxPool -> Flatten-> Dense
- Each convolutional layer takes samples of the image and to form a channel map by certain dimension(width x height)
- For color image, there are three kernel channels, which the sum of the three(red, green, blue) are calculated
- Maxpool layer reduces the dimensions of the Convolutional layer by taking the max value of every block of pool_size x pool_size within the image. This saves lots of computational power
- Flatten layer converts the pooled feature maps to a single column which passed to the fully connected layer
- Dense layer: the last layer which takes the fully connected layer as input for the neural network, outputs the result of the prediction.

Data processing and formatting

After importing the dataset, resize each image to 300x200 which is the average size of all image.

Then for each image, store the data of each image in an array with shape (total_num_images,200,300,3) as the X variable

For the category of the image, store it in another array with dimension (total_num_images,1) as the Y variable

With all the images are processed into arrays, the values are normalized by dividing by 255, which is the max value of a single color chanel, this puts all value between 0 and 1

Now models can be trained using the dataset.

For the training and validation of the models, the dataset is divided using train_test_split from sklearn.

Activation function

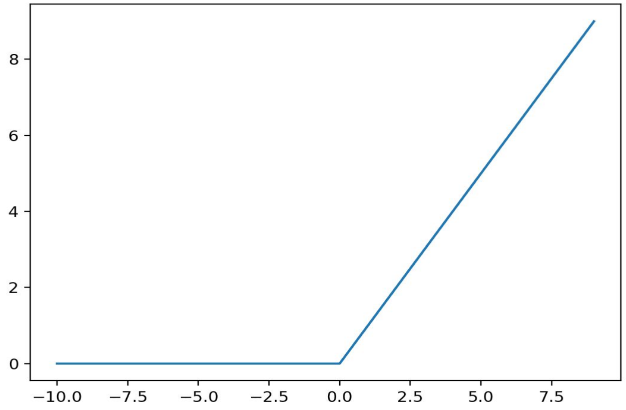
The convolutional and dense layer(as hidden layer) use the rectified linear unit(Relu) as their activation function.

Relu is the function that f(x) = max(0,x), where the negative value just becomes 0, and positive number is the number itself

Advantages vs. other activation functions:

- Efficiency
- No vanishing gradients

When dense layer is the output layer, it uses softmax to find the max value in the result array which predicts the category that is closest to the input



List of models

Model 1 single layer:

convolutional layer-> max-pooling->flatten->dense(output)

Model 2:

2x convolutional layers ->max-pooling->flatten->dense(output)

Model 3 :

2x convolutional layers ->max-pooling->flatten->dense->dropout->dense(output)

Model 4:

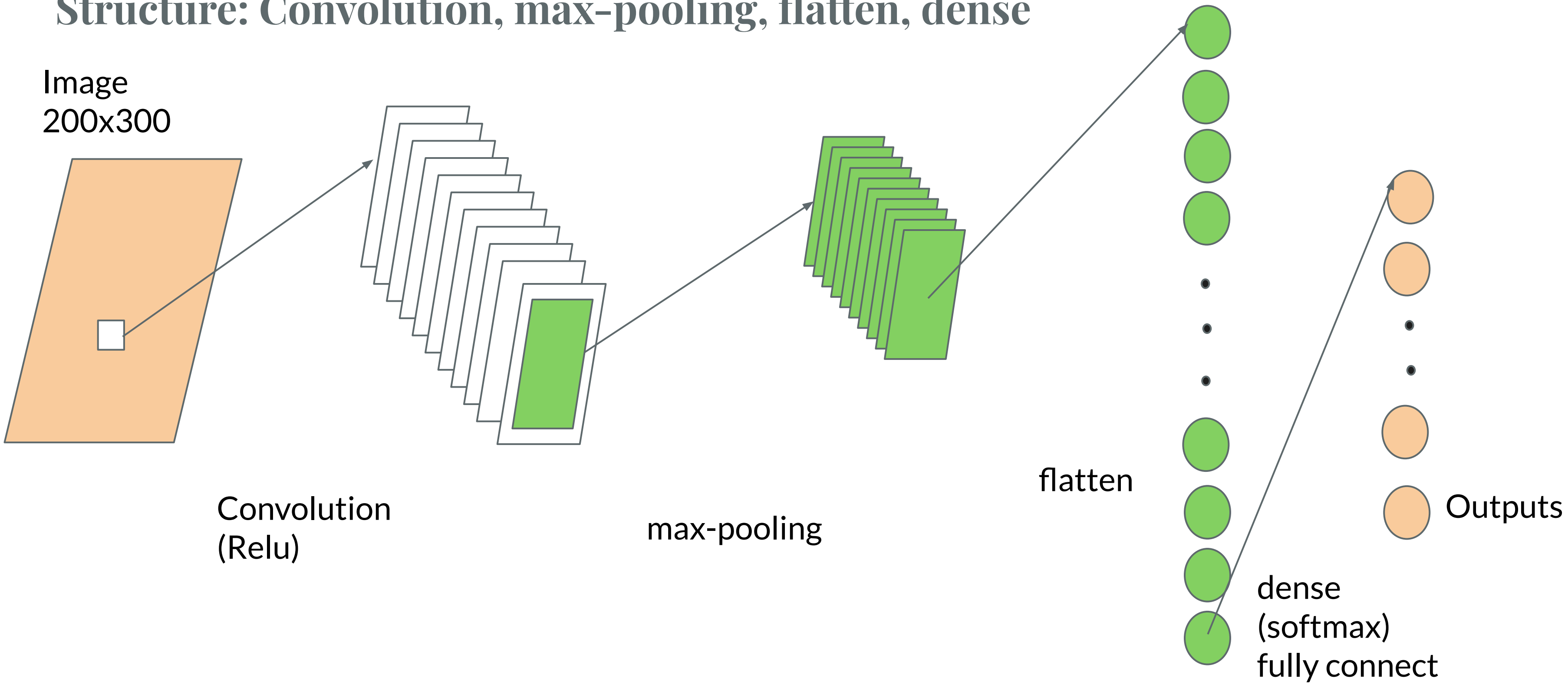
2x convolutional layers->max-pooling->
2x convolutional layers->max-pooling->flatten->dense(output)

Model 5:

2x convolutional layers->max-pooling->
2x convolutional layers->max-pooling->flatten->dense->dropout->dense(output)

First attempt

Structure: Convolution, max-pooling, flatten, dense



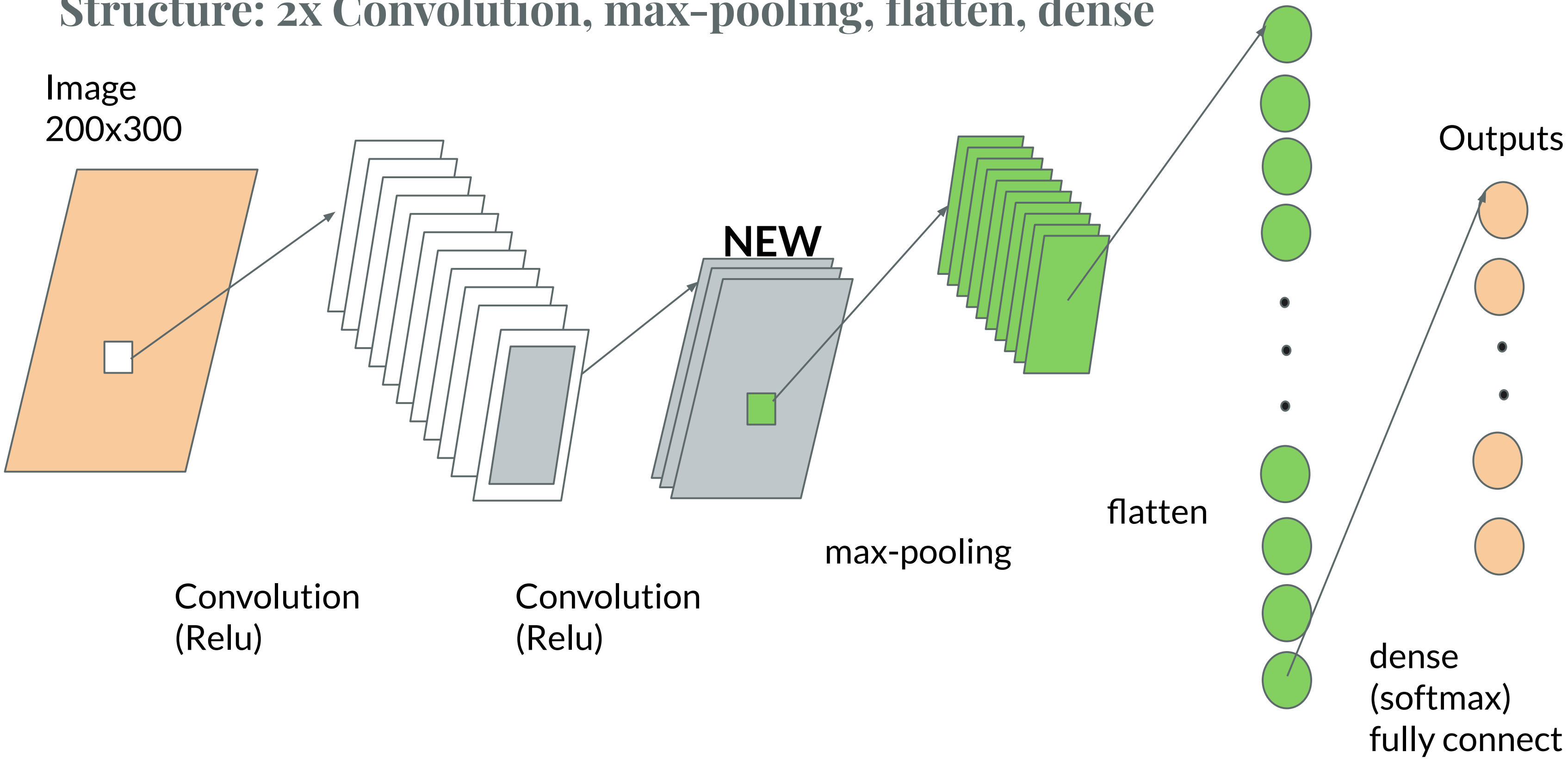
Result: first attempt

Epoch	Time took	loss	accuracy	val_loss	val_accuracy
1	183s	18.4729	0.1337	2.9286	0.4493
2	161s	1.0203	0.8039	2.9436	0.5074
3	165s	0.1877	0.9767	3.3579	0.5166
4	175s	0.0548	0.9946	3.7252	0.4940
5	166s	0.0318	0.9973	3.4671	0.5235

- Training accuracy and loss improves significantly in first three epochs(higher accuracy and lower loss)
- Validation accuracy improves slightly, and is around 0.5
- Overfitting

Adding one more Convolutional layer

Structure: 2x Convolution, max-pooling, flatten, dense



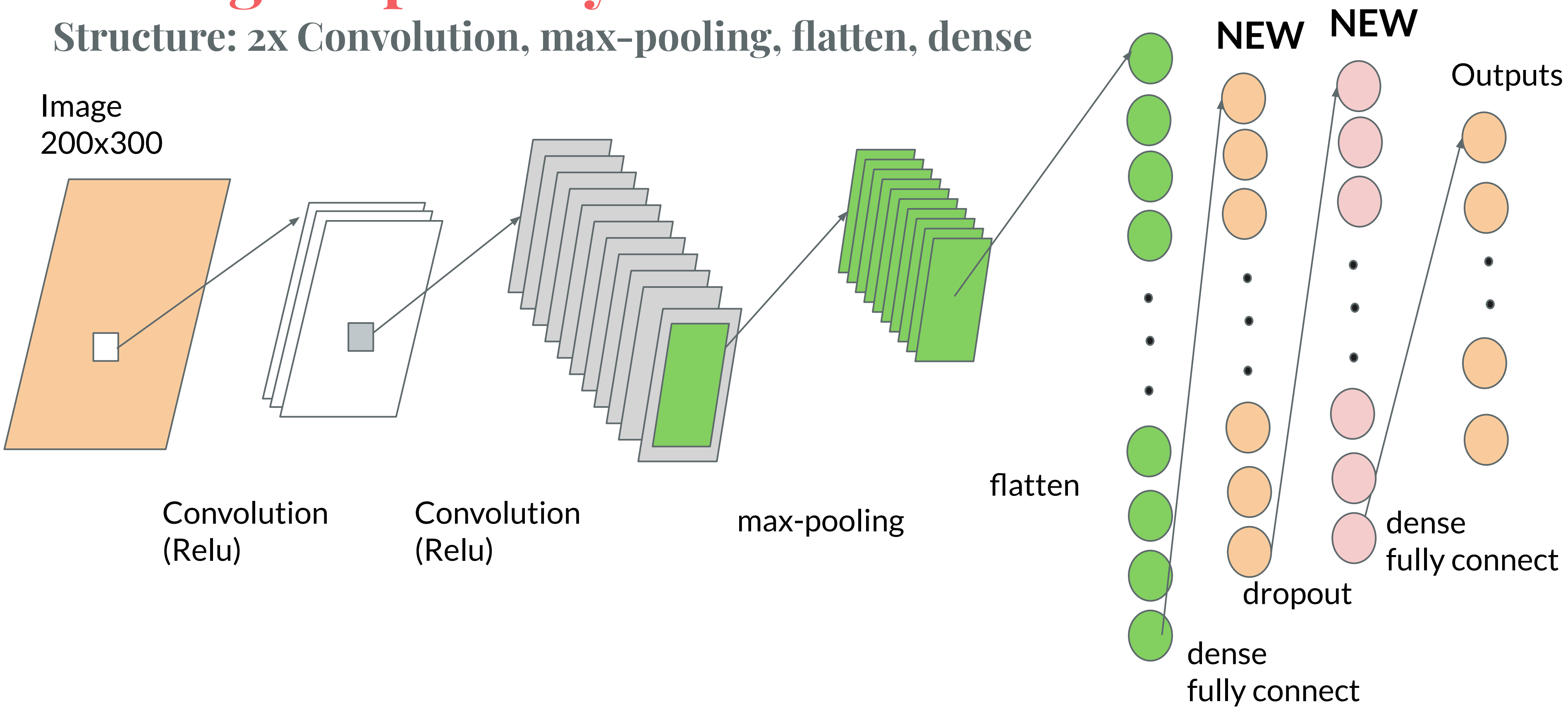
Result: 2 convolutional layers

Epoch	Time took	loss	accuracy	val_loss	val_accuracy
1	612	4.4677	0.2570	2.3611	0.5041
2	587s	0.5460	0.8911	2.7804	0.5447
3	593s	0.0539	0.9911	2.9356	0.5530
4	605s	0.0170	0.9982	3.2555	0.5350
5	599s	0.0135	0.9987	3.1584	0.5484

- Training accuracy close to 1
- Both loss and accuracy of validation improves
- It takes three times longer to train

Adding dropout layer

Structure: 2x Convolution, max-pooling, flatten, dense

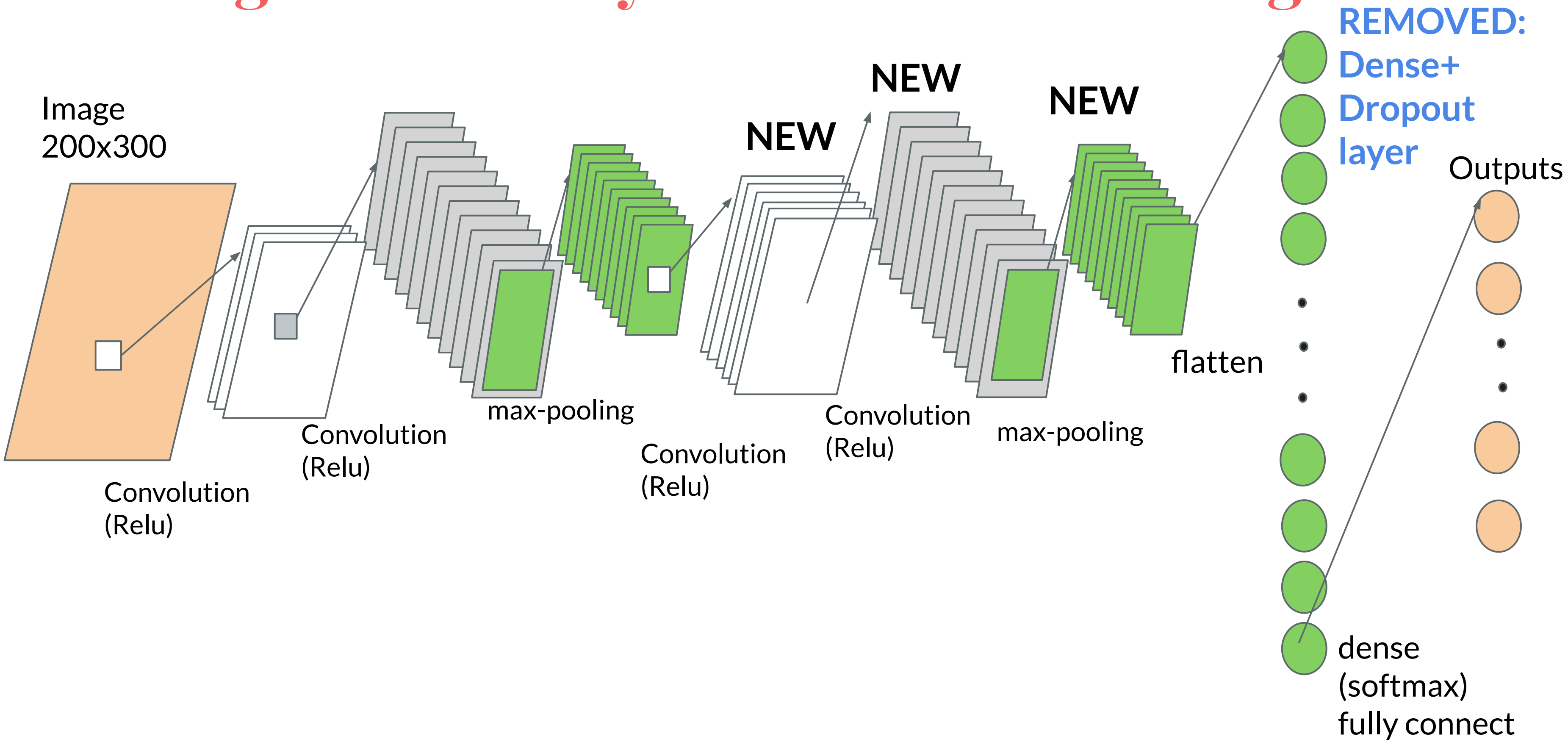


Result: two convolutional layers with dropout

Epoch	Time took	loss	accuracy	val_loss	val_accuracy
1	1013	10.5499	0.0904	3.3695	0.3424
2	984	3.1911	0.3291	2.8513	0.3880
3	965	2.4881	0.4219	2.6099	0.4539
...					
15	899	0.2887	0.9147	3.0062	0.5037
16	923	0.2923	0.9176	3.1858	0.4986

- With a dropout layer, 16 epochs before the final model
- In the final model, validation loss is higher and validation accuracy is lower
- Model is not improved from previous one

Adding one more cycle in feature learning

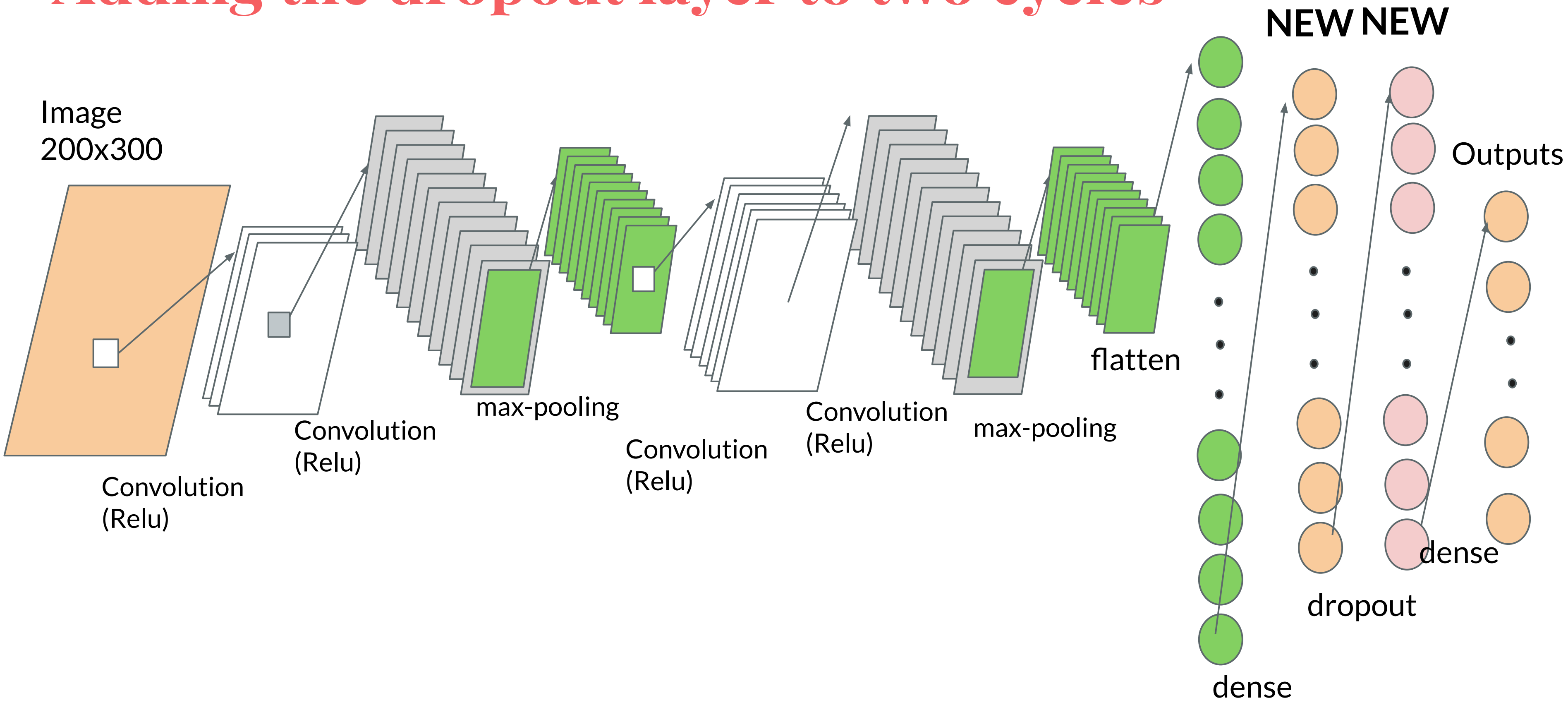


Result: two cycles no dropout

Epoch	Time took	loss	accuracy	val_loss	val_accuracy
1	1212s	4.3987	0.2360	2.3943	0.4903
2	1204s	1.3791	0.6816	2.3450	0.5244
3	1120s	0.2621	0.9368	3.4764	0.5309
4	1145s	0.0524	0.9890	3.9383	0.5415
5	1177s	0.0166	0.9972	4.2347	0.5450

- Validation loss and accuracy are similar to the model with one cycle.
- Higher validation loss than one cycle, there is more overfitting
- Similar time to train compared to previous on

Adding the dropout layer to two cycles



Result: two cycles with dropout

Epoch	Time took	loss	accuracy	val_loss	val_accuracy
1	1781s	4.2345	0.2570	3.2379	0.3194
2	1742s	3.5747	0.3159	2.9391	0.3677
3	1766s	3.2726	0.3820	2.5421	0.4433
4	1751s	2.8775	0.4705	2.4949	0.4618
5	1755s	2.3542	0.5991	2.1903	0.5106
...					
16	1784s	0.0478	0.9885	2.8247	0.5530
17	1810s	0.0333	0.9919	2.8196	0.5576

Comparison of all CNN models

Model	Loss	Accuracy	Val_loss	Val_accuracy	Average category accuracy
One conv. layer	0.0318	0.9973	3.4671	0.5235	0.8283
Two conv. layers	0.0135	0.9987	3.1584	0.5484	0.8362
Two layers with dropout	0.2923	0.9176	3.1858	0.4986	0.8164
Two cycles of two conv. layers	0.0166	0.9972	4.2347	0.5450	0.8347
Two cycles with dropout	0.0333	0.9919	2.8196	0.5576	0.8472

Introduction to Locality Sensitive Hashing: Random Projection Method

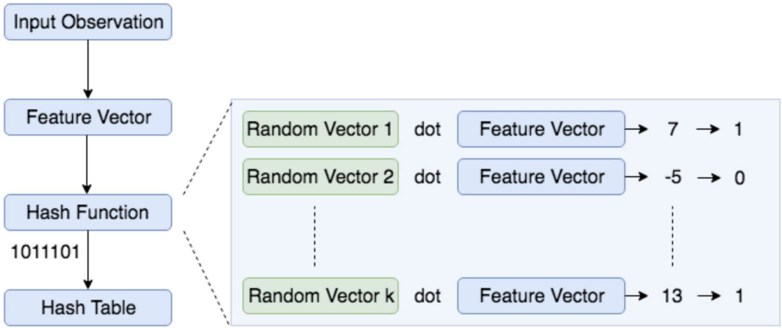
Consider a image dataset matrix \`D\` with \`n\` vectors of size \`d\`. This database \`D\` can be projected onto a lower dimensional space with \`n\` vectors of size \`k\` using a random projection matrix.

$$\begin{bmatrix} \quad Projected(P) \quad \end{bmatrix}_{k\times n} = \begin{bmatrix} \quad Random(R) \quad \end{bmatrix}_{k\times d} \begin{bmatrix} \quad Original(D) \quad \end{bmatrix}_{d\times n}$$

Ideas behind this method

We construct a table of all possible bins where each bin is made up of similar items. Each bin can be represented by a bitwise hash value so that two images with same bitwise hash values are more likely to be similar than those with different hashes.

Steps to generate a bitwise hash table



1. Create `k` random vectors of length `d` each, where `k` is the size of bitwise hash value and `d` is the dimension of the feature vector (in our case, this is the dimension of the image).

2. For each random vector, compute the dot product of the random vector and the image. If the result of the dot product is positive, assign the bit value as 1, else 0.

3. Concatenate all the bit values computed for `k` dot products.

4. Repeat the above two steps for all images to compute hash values for all images.

5. Group images with same hash values together to create a LSH table.

Multiple tables

In addition, because of the randomness, it is not likely that all similar items are grouped correctly. To overcome this limitation, a common practice is to create multiple hash tables and consider an image `a` to be similar to image `b`, if they are in same bin in at least one of the tables. It is also worth noting that multiple tables generalize the high dimensional space better and amortize the contribution of bad random vectors.

In practice, the number of hash tables and size of the hash value `k` are tuned to adjust the trade-off between recall and precision.

Example output

```
lsh — -zsh — 80×24
(CS542-Team-13) wanghanming@Mac lsh % python lsh_util.py
Number of hash tables: 3
Size of hash values: 50
The image ID that we want to search is query image 20
The type of the query image is gerenuk
The image IDs that are similar to the query image are:
['20', '7397', '7395']
The type of the nearest image is gerenuk
Time spent: 4.120605945587158
(CS542-Team-13) wanghanming@Mac lsh %
```

Resources

https://santhoshhari.github.io/Locality-Sensitive-Hashing/

https://necromuralist.github.io/neural_networks/posts/image-to-vector/

https://docs.python.org/3/tutorial/venv.html

http://www.vision.caltech.edu/Image_Datasets/Caltech101/

https://stackoverflow.com/questions/48121916/numpy-resize-rescale-image

https://github.com/bhavul/Caltech-101-Object-Classification