Harware Classifier

Table of Contents

Our Project

Idea of our model is to reduce human effort of searching for things in a market place. We have taken a Hardware Store into consideration and implemented our model on “Mechanical Tools Dataset” available on the internet. If a customer comes into the store with the photo of some tool which he/she wants then manager searches for that tool in the store, to save time and effort our model can be implemented here.

Flow of the model

User gives an input image, our application takes image as input, pre-processes it, runs it through our neural network, then puts it in one of the eight classes.
For our model, we assume user will be manager of the store so flow goes like-
Manager scans the photo which will be passed as input into our model and it will give as output name of the tool and location where it is present in the store. If the tool is not present in the store our model will give as output - name and location of a tool which looks very similar to the one that customer wants.

Practical Applications

- If you have a bunch of images and you want to somehow classify them or run regressions, our model is a fast way to apply deep neural networks to solve the problem.
- If a customer comes to the store with a google image of some item then the shopkeeper will scan the image which will be passed as an input to our application and results of similar images with their locations(in the store) will be displayed.
- It will work like a localised search engine for the store.
- Successful Implementation of this model will reduce man power. When we go to some store and can’t find things we ask the staff working there who are appointed to help us locate items we need, with our model in action these helpers won’t be required.
- If a dataset of Grocery items is available then the same model can be used on a large scale like a

supermarket.

Dataset

The Mechanical Tools Dataset consists of 8 classes:

- Tool Box
- Hammer
- Pliers
- Rope
- Pebbel
- Wrench
- Screw Driver
  It consists of minimum 210 images from each class.

Models

Pre-trained Inception-v3 (GoogleNet)

Inception-v3 is a convolutional neural network architecture from the Inception family that makes several improvements including using Label Smoothing, Factorized 7 x 7 convolutions, and the use of an auxiliary classifer to propagate label information lower down the network (along with the use of batch normalization for layers in the sidehead).
It has multi layer perceptron (MLP) convolutional layers instead of linear convolutional layers that include a linear filter.
Instead of adopting the traditional fully connected layers for classification in CNN, InceptionV3 directly outputs the spatial average of the feature maps from the last mlpconv layer as the confidence of categories via a global average pooling layer, and then the resulting vector is fed into the softmax layer.

Architecture

The architecture of an Inception v3 network is progressively built, step-by-step, as shown below:

1. Factorized Convolutions
2. Smaller convolutions
3. Auxiliary classifier
4. Grid size reduction

All the above concepts are consolidated into the final architecture.

Our Fine-tuned Architecture

After importing headless network, the penultimate layer of Inception-v3 is:
mixed7 with shape: (None, 17, 17, 768)
We flatten this layer out then add dense(128) layer and another dense(8) layer on top of it.
We use pre-trained weights ‘imagenet’ and train only the dense layers(setting previous layers as untrainable) we added, on the dataset.

Inputs

Our model has 8 classes and was trained on 5710 images belonging to these classes, and then it was validated on 400 images (50 belonging to each class) over 40 epochs.

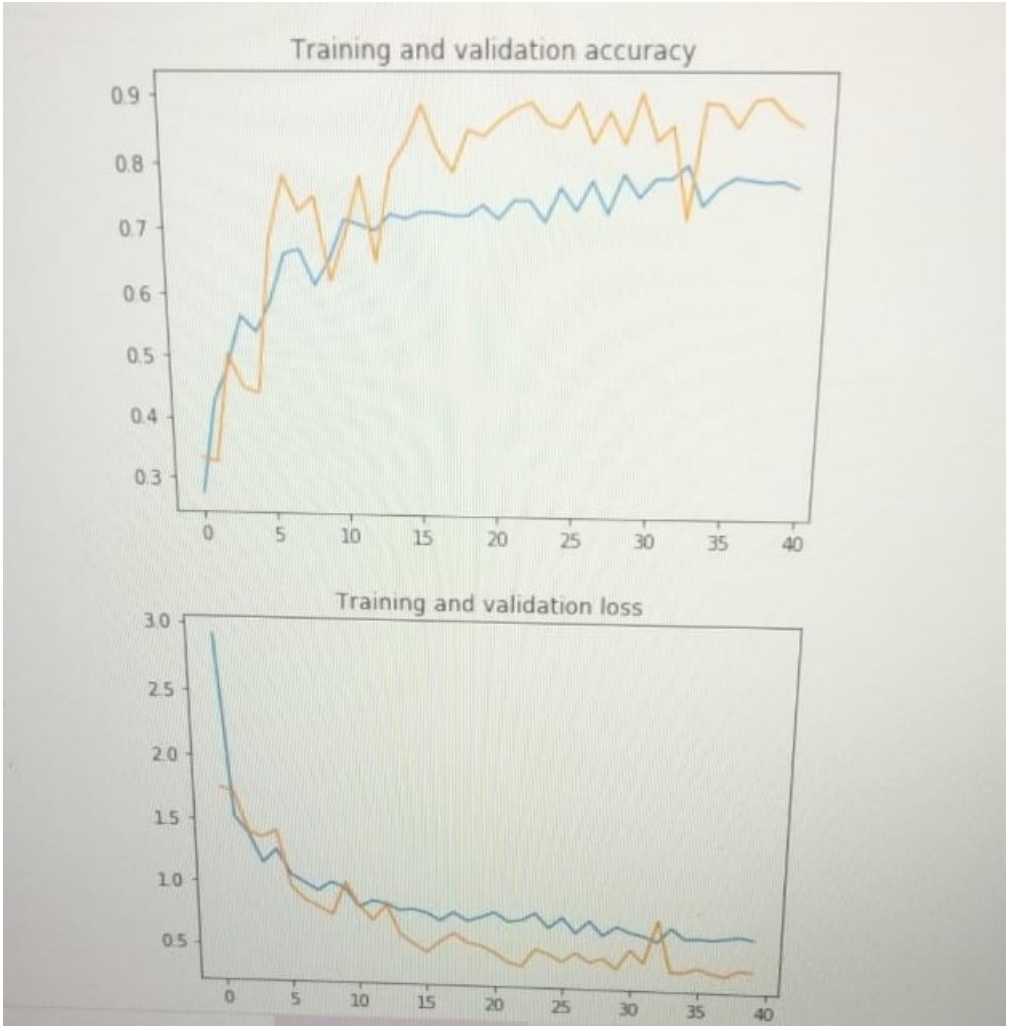
Pre-processing

Each image is resized into shape (300, 300) with all 3 channels present.
Then we augment our inputs as shown:

```
rescale = 1./255.,
rotation_range = 40,
width_shift_range = 0.2,
height_shift_range = 0.2,
zoom_range = 0.2,
horizontal_flip = True
```

Results

Here is the plot of “training and validation accuracy per epoch” and “training and validation loss per epoch”.



For testing our model we have a test dataset with images from all classes, when we passed the path of an image to our model it generates the following output- Name of the tool and its location in the store. Accuracy of our model is 87%.

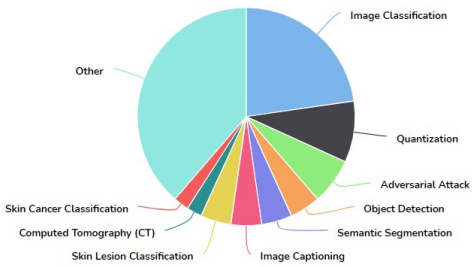
```
This is a Tool box
It is present at location: Shelf F
```

Advantages

- In comparison to VGGNet, Inception Networks (GoogLeNet/Inception v1) have proved to be more computationally efficient, both in terms of the number of parameters generated by the network and the economical cost incurred (memory and other resources).

Application

Tasks



TASK	PAPERS	SHARE
Image Classification	10	22.73%
Quantization	4	9.09%
Adversarial Attack	3	6.82%
Object Detection	2	4.55%
Semantic Segmentation	2	4.55%
Image Captioning	2	4.55%
Skin Lesion Classification	2	4.55%
Computed Tomography (CT)	1	2.27%
Skin Cancer Classification	1	2.27%

Limitations

- If any changes are to be made to an Inception Network, care needs to be taken to make sure that the computational advantages aren't lost. Thus, the adaptation of an Inception network for different use cases turns out to be a problem due to the uncertainty of the new network's efficiency.

Basic Convolutional Neural Network

Architecture

```
conv = Conv3x3(8)                  # 28x28x1 -> 26x26x8
pool = MaxPool2()                  # 26x26x8 -> 13x13x8
softmax = Softmax(13 * 13 * 8, 10) # 13x13x8 -> 10
```

1. Convolution - Conv layers consist of a set of filters, which you can think of as just 2d matrices of

numbers.We can use an input image and a filter to produce an output image by convolving the filter with the input image.

2. Pooling - Neighboring pixels in images tend to have similar values, so conv layers will typically also produce similar values for neighboring pixels in outputs. As a result, much of the information contained in a conv layer's output is redundant. Pooling layers solve this problem. They reduce the size of the input it's given by pooling values together in the input. The pooling is usually done by a simple operation like max, min, or average. We have used a Max Pooling layer with a pooling size of 2.
3. Softmax - Softmax layer, a fully-connected (dense) layer uses the Softmax function as its activation.The outputs of the Softmax transform are always in the range [0,1] and add up to 1. Hence, they form a probability distribution.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

We used a softmax layer with 8 nodes, one representing each class, as the final layer in our CNN. Each node in the layer will be connected to every input. After the softmax transformation is applied, the digit represented by the node with the highest probability will be the output of the CNN!

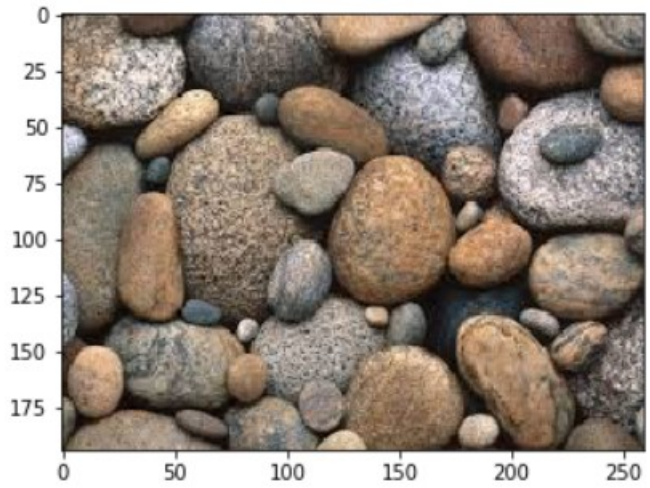
Inputs

Input is a numpy array consisting of 1600, 28x28 images, 200 from each of 8 classes.
From every class, 200 images were selected and converted to grayscale then resized to 28x28 array, making the final dataset a numpy array with shape (1600 , 28 , 28).

Results

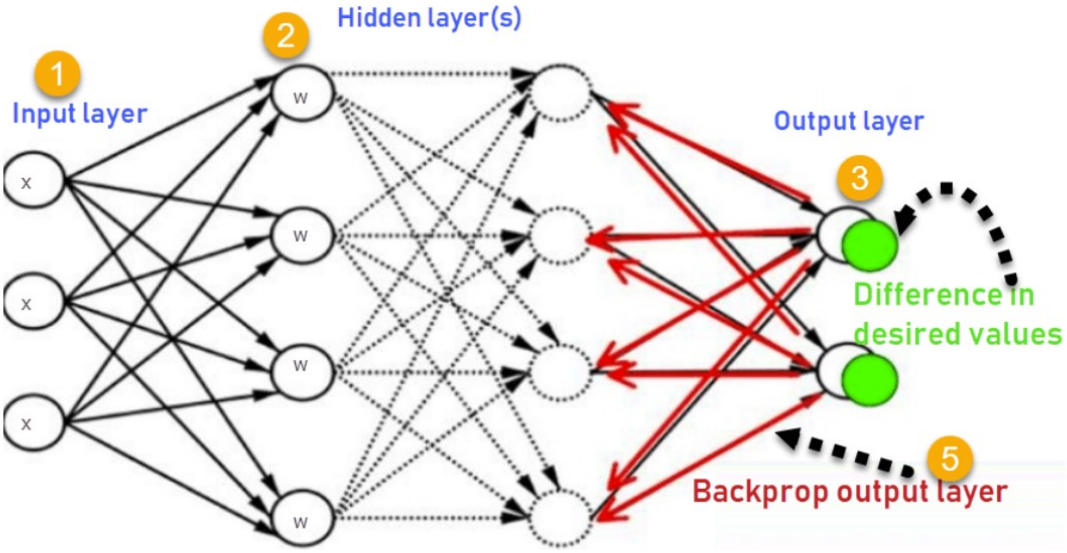
For testing our model we have a test dataset with images from all classes, when we passed the path of an image to our model it generates the following output- Name of the tool and its location in the store. Accuracy of our model is 49%.

```
This is a pebbel
It is present at location: Shelf B
```



Basic EBPA Model

Architecture



1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs
5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.
   Keep repeating the process until the desired output is achieved.

Inputs

Input is a numpy array consisting of 160, 10x10 images, 20 from each of 8 classes.
From every class, 20 images were selected and converted to grayscale then resized to 10x10 array and converted to a binary array of size 1x100. Number of epochs is 200.

Results

For testing our model we have a test dataset with images from all classes, when we passed the path of an image to our model it generates the following output- Name of the tool and its location in the store. Accuracy of our model is 11%.

```
This is a Wrench
It is present at location: Shelf E
```

Comparison

Model	Layers	Input Images	Epochs	Learning rate	Accuracy
Inception	51	5710 *(300x300x3)*	40	0.0001	87%
Basic CNN	3	1600 *(28x28x1)*	3	0.005	49%
Basic EBPA	3	160 *(10x10x1)*	200	0.5	11%

tags: `NFT` `Neural Networks` `Hardware` `Convolution` `CNN` `EBPA`