Capstone Project

SIDDHARTH SHORYA

Machine Learning Engineer Nanodegree

~April 6th, 2016

1 Introduction:

Image recognition has gained a lot of interest more recently which is driven by the demand for more sophisticated algorithms and advances in processing capacity of the computation devices. Convolutional neural networks sound like a weird combination of biology and math with a little CS sprinkled in, but these networks have been some of the most influential innovations in the field of computer vision. The CNN have had a drastic impact on Image recognition.

Image recognition algorithms can be classified into two major categories: Pre-defined image descriptor based and descriptor learning based. Algorithms in pre-defined category uses image processing techniques to generate important observations (or more technically features) and use these observations to detect/recognize an object. On the other hand, descriptor learning based algorithms generate these important observations (or features) while building (or more technically training) the algorithm. They use a large dataset of images and their categories to find these features.

In this project I attempt to apply convolution Neural networks for animal recognition from images from wildlife. Thus, I will train a neural network to predict the animal for the ease and accurate classification.

2 Dataset:

I had two types of files CSV and Images. The train data consisted of 13,000 images and the test data of 6,000 images of 30 different species of animals. The image ID and the corresponding animal name are stored in .csv format, while the image files are sorted into separate train and test image folders. Data in the.csv file is in the following format:

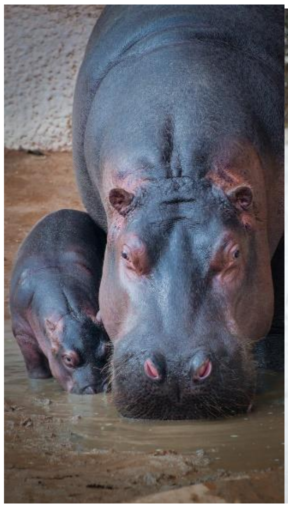
Variable	**Description**
Image_id	*Image name*
Animal	*Name of the Animal*

Following are the 30 different species of animals in the dataset:

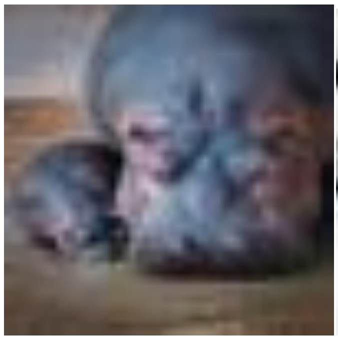
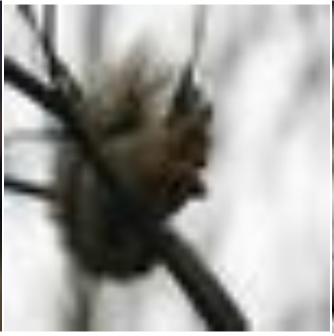
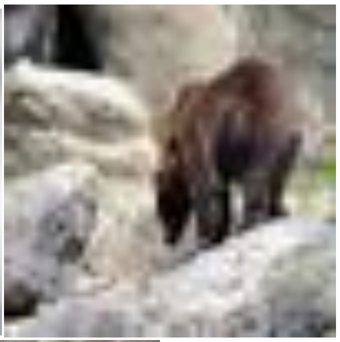
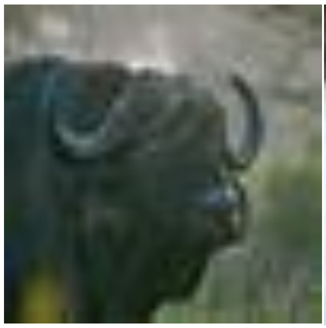
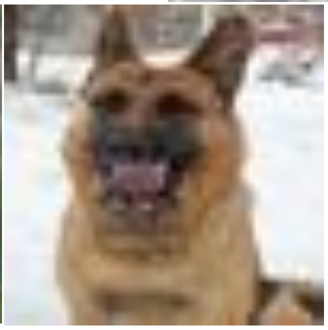
- antelope
- bat
- beaver
- bobcat
- buffalo
- chihuahua
- chimpanzee
- collie
- dalmatian
- german + shepherd
- grizzly + bear
- hippopotamus
- horse
- killer + whale
- mole
- moose
- mouse
- otter
- ox
- Persian + cat
- raccoon
- rat
- rhinoceros
- seal
- Siamese + cat
- spider + monkey
- squirrel
- walrus
- weasel
- wolf

The input to CNN was resized to be the images of resolution 224 x 224 (will convert high resolution to lower resolution). This came to be the best resolution to gain information according to me.

<u>IMAGES INITIALLY WERE OF THIS RESOLUTION (proportional to this I have show smaller in order to fit page.)</u>

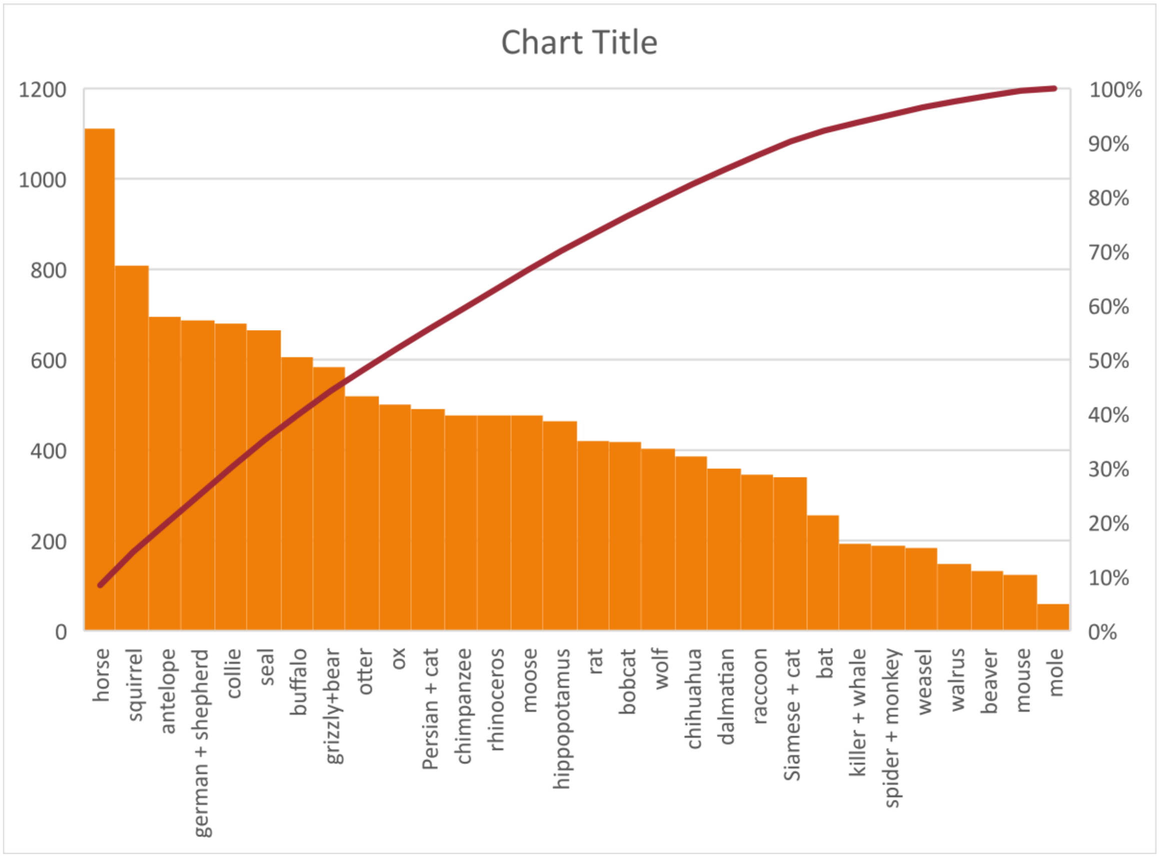


I resized them to 224*224

2.1 Data exploration:

We have counted the number of occurrences of each label in the training set (13000K set).



2.2 Metrics:

The evaluation metric for this project will be the transcription accuracy on the test images in the dataset. Accuracy will be defined as correctly predicting the full house number in the image. No partial credit will be given for identifying part of the number. The model must correctly predict the number of digits present as well as the identity of each of those digits.

3. Problem Statement:

The objective of this project is to train a deep learning convolutional neural network to predict the class of animal that is in an image.

Wildlife images captured in a field represent a challenging task while classifying animals since they appear with a different pose, cluttered background, different light and climate conditions, different viewpoints, and occlusions. Additionally, animals of different classes look similar. Also, processing such a large volume of images and videos captured from camera traps manually is extremely expensive, time-consuming and also monotonous. All these challenges necessitate an efficient algorithm for classification.

Thus, I made a model of the same.

4. Benchmark:

While researching this project, I came across approaches for image segmentation. Image segmentation is one of the mostly used methods to classify the pixels of an image correctly in a decision-oriented application. It divides an image into a number of discrete regions such that the pixels have high similarity in each region and high contrast between regions . This is what I have done with CNN but most of the approaches I came across were using K-means clustering algorithm.

Undoubtably among many techniques K-means is one the most efficient algorithm to use and thus I use it as a benchmark. My model gives the accuracy of 50-60%. That is great.

5. Data Preprocessing:

Preprocessing of data is carried out before model is built and training process is executed. Following are the steps carried out during preprocessing.

1. Initially the images are divided into training and validation sets.

2. The images are resized to a square images i.e. 224 x 224 pixels.

3. All three channels were used during training process as these are color images.

4. The images are normalized by dividing every pixel in every image by 255.

6. Algorithm and Techniques:

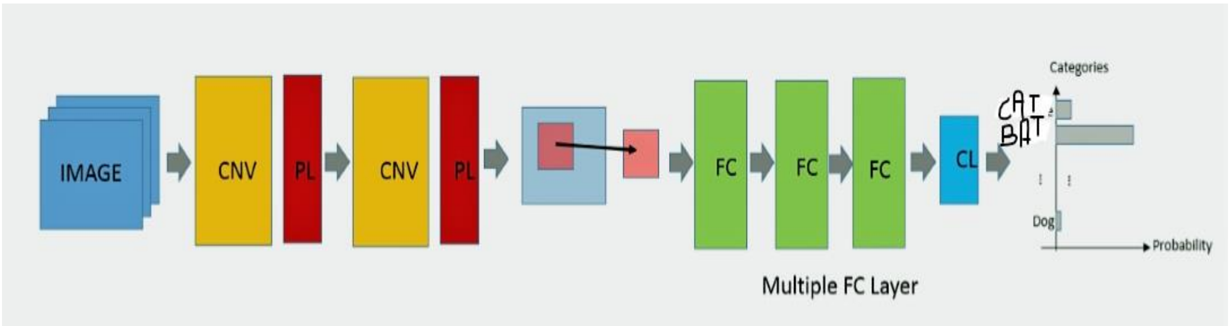
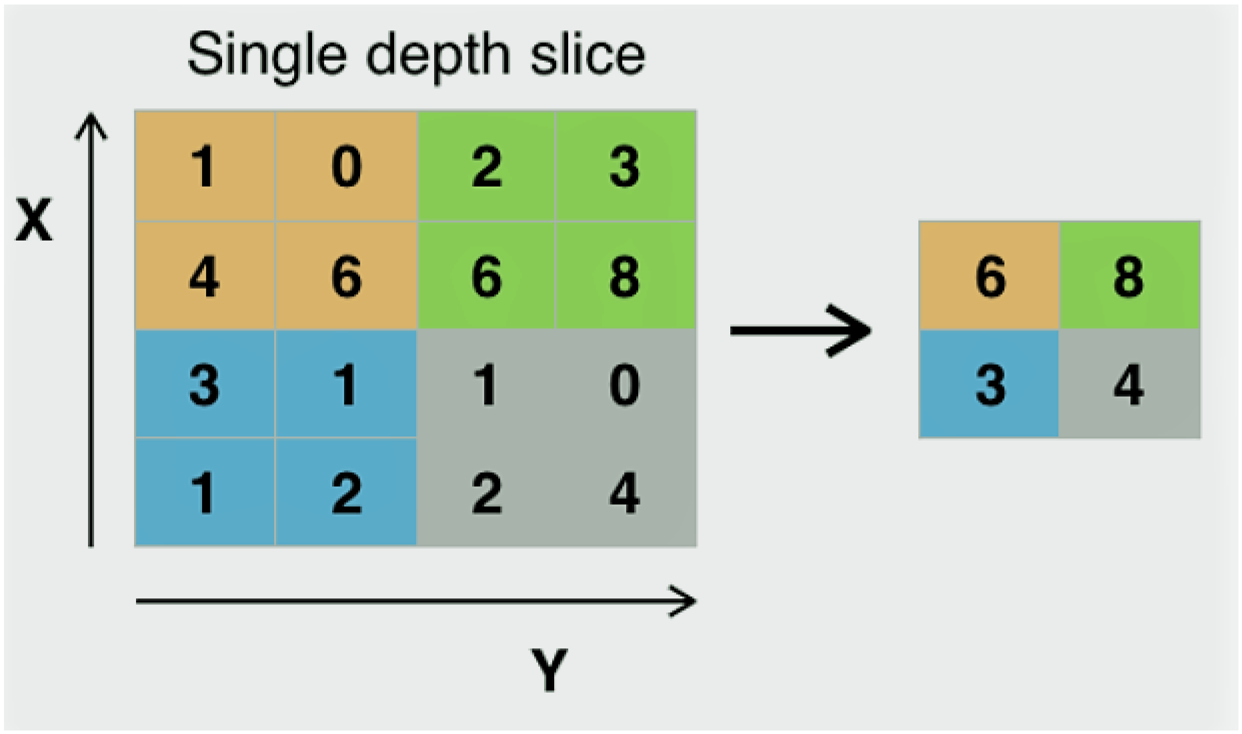


Figure 8 Flow Diagram of Localization

Convolutional layer (CNV): The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters, which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2- dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Pooling layer (PL): Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that the exact location of a feature is less important than its rough location relative to other features. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to also control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The pooling operation provides another form of translation invariance. The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged. In addition to max pooling, the pooling units can use other functions, such as average pooling or L2-norm pooling.



Fully connected layer (FC): Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Classification Layer (CL): The classification layer specifies how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there. SoftMax loss is used for predicting a single class of K mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting K independent probability values in [0,1]

Convolutional neural network architecture is the current state of the art for image classification and detection. Convolutional networks have a number of different architectural decisions and parameters that must be selected and tuned. Keras the python library has been used to create the model. Since I was familiar with TensorFlow first I did study all about keras and wrote the following for the model layers. Clear from this what layers order I used.

```
from sklearn.model_selection import cross_val_score
from keras.models import Sequential, load_model, Model
from keras.layers import Input, BatchNormalization
from keras.layers import Dense, LSTM, GlobalAveragePooling1D, GlobalAveragePooling2D,Dropout
from keras.layers import Activation, Flatten, Dropout, BatchNormalization
from keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D

def conv_classifier(a):
    model_input = Input(shape=(a, a,3))
    # Define a model architecture
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(model_input)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D(pool_size=(2, 2))(x)
    x = Dropout(0.25)(x)

    x = Flatten()(x)
    x = Dense(512, activation='relu')(x)
    x = Dropout(0.25)(x)

    y1 = Dense(30, activation='softmax')(x)

    model = Model(inputs=model_input, outputs= y1)

    # Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Following are the steps carried out for training a CNN:

1. The entire dataset i.e. train, validation and test dataset is pre-processed (Refer “Data Preprocessing” section).

2. A simple CNN is created to classify images. The CNN consists of 4 convolutional layers with 4 max pooling layers in between.

3. Filters were increased from 64 to 512 in each of the convolutional layers and drop out is added

4. Flattening layer along with two fully connected layers were added at the end of the CNN

5. Number of nodes in the last fully connected layer were setup as 10 which is number of categories in the dataset, along with using SoftMax activation function. This layer is being used as classification layer.

6.” Relu” activation function was used for all other layers along with Xavier initialization.

7. The model is compiled with ‘rmprop’ as the optimizer and ‘categorical_crossentropy’ as the loss function.

8. The model is trained for 30. During the training process the model parameters were saved when there is an improvement of loss for validation dataset.

9. Finally predictions were made for the test dataset and were submitted in suitable format.

7. Refinement:

To get the initial result simple CNN architecture was built and evaluated. This resulted in a decent loss. After this to further improve the loss, transfer learning was applied to VGG16 along with investigating 2 types of architectures for fully connected layer. Model Architecture1 showed good results and was improved further by using the below techniques

• Drop out layer was added to account for overfitting.

• Training was carried out with 30 epochs to further improve the loss metric, VGG16 along with Model Architecture1 was selected and fine-tuning was applied. SGD optimizer was used with very slow learning rate of 0.01.

These things made a drastic change in the scores of the model.

8. Conclusion:

We have trained simple CNN algorithm of size 224x224 using mage set and applied it to larger image sizes for localization. To further improve the score, we need to consider using bigger size when re-sizing the image. Currently the algorithm was trained by using 224x224 re-sized images. An image size of 448x448 might be relevant as original image size is 1360x780 as an example.