Object Recognition

Machine Learning

Paul Lorigan	**Michal Hryciuk**
17213517	**13343826**

MSc in Computing (Data Analytics)
CA684 – Machine Learning
April 2018

Table of Contents

Table of Figures

Project Description

The project we chose to undertake was image classification using a Convolutional Neural Network (CNN). This project involved running a large dataset of images, labelled according to its contents, through a neural network to teach it to distinguish which classification to which the image belongs.

We started off with a simple classifier that classified two categories of images, and then expanded our classifier out to include multiple categories of images. The original classifier was built primarily using tutorials found online, modified slightly with unique code and architecture, whereas the later classifier was built using primarily unique code, but with an overall structure similar to the original framework. We built the system using an architecture file which created the overall infrastructure of the CNN, including code to detect and parse images, create multiple layers within the network. We further went on to build the system in a somewhat modular approach, with specific sections in the code for the different overall functions. Another file was created to handle the training of the system and the prediction capabilities, and a final file was created to bring the system together in order to carry out some classification technique.

Challenges and Difficulties

Building the neural net from the ground up presented a few issues as it meant we couldn't check the build until the build was complete. We adopted a baby steps approach to the task, building a simple CNN, before moving onto more complex versions so as to iteratively attack and overcome the challenge. Thus, if at any time, we ventured down a bad path, we could simply roll the system back to an earlier version.

We incorporated various different datasets into our network, first building a basic bimodal system to classify based on two classes, before building up to a more multimodal system later in the project.

Other Work on the topics

CV Tricks provides a good tutorial explaining how to create a CNN and apply it to a 2 different types of images. This provides a template for image classification that can then be expanded to classify larger datasets with multiple image classes.

http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classification/

Another good source for understanding CNNs and developing the building blocks for their use is available at hackernoon. This website provides a good understanding of some of the capabilities and implementations of CNNs.

https://hackernoon.com/deep-learning-cnns-in-tensorflow-with-gpus-cba6efe0acc2

General Solution Used

The solution we found that worked best was to use a three layered CNN that then flattened our images and fed them to two fully connected layers, the second of which simply contained the classes to which our machine would classify them.

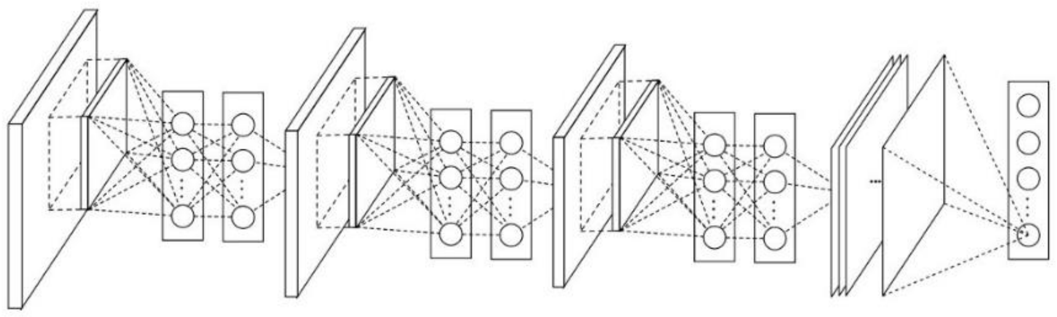


Figure 1: Network Architecture

Data Preparation

For the basic system, we decided on the standard dataset of cat/dog images (taken from https://github.com/sankit1/cv-tricks.com/tree/master/Tensorflow-tutorials/tutorial-2-image-classifier). This provides 500 images of cats and 500 images of dogs for training data, along with 200 images of each for test data. All images are clearly labelled and ordered sequentially, and there are no duplicate images in the dataset. The dataset was broken into 32 x 32 x 3 (w * h * d) batches, and 64 x 64 in the final convolutional layer.

Data used in this project is CIFAR 10 dataset, it consists of 10 classes like vehicles and animals. Overall it has 60,000 images of size 32 by 32 pixel and each class has equal amount of examples, authors divide their dataset using 5:1 ratio for training and validation subsets. Images arrive in encoded files that require to be unpacked, which yields a constant string of RGB pixel values, these are encoded in the following order, an image consists of 3,076 bytes, first 1,028 bytes correspond to red channel values for every pixel, second green values and finally blue values. Images in this dataset are small, thus make it perfect for evaluating image-related models, small images take less time to process. It also has a great amount of variation and class examples don’t overlap so data is already pre-processed and designed to produce best results.

Project Methods

We chose to use the CNN approach to object recognition due to the range of tasks that could be completed using it, from basic image classification to more advanced topics such as facial recognition, CNNs are good multipurpose frameworks for achieving a wide range of classification tasks.

Our basic CNN was implemented based on several online tutorials related to image classification through the use of a CNN. The model consisted of:

- **Architecture**
  - *Methods and classes for image processing*
    - Load training set
      - Declare variable arrays
      - Set up system paths for each class in dataset
      - For each image in dataset, pre-process using OpenCV
      - Add pre-processed images to variable arrays
    - Declare DataSet class object
      - Initialise images, labels, etc.
      - Define next batch processing to run following each Epoch
    - Read training set
      - Declare DataSets class Object
      - Build Validation and Training instances
  - *CNN setup*
    - Create weights
    - Create biases
    - Create convolutional layer
      - Declare layer attributes
      - Use strides of 1, max pooling and RELU
    - Create Flatten Layer
    - Create Fully Connected Layer

- **Classifier**
    - *Call Architecture*
    - *Training*
        - Declare Classifier class object
            - Define Initialisation Variables
                - Define Load Classes
                    - Pass system path for all classes
                - Set variables, input data, and paths
            - Define Training
                - Start session
                - Apply Placeholder, labels and graph network parameters
                - Build layers using layer outputs
                - Reduce cost (mean), improve accuracy and optimise network
                - Execute epochs and save progress
            - Define Load Testing
                - Read and process test images
                - Adjust Weights, Biases, and prediction variables
    - *Prediction*
        - Define Prediction
            - Read image, resize and reshape
            - Plot on graph
            - Save state
        - Print Result
- **Executer**
    - *Call Classifier*
        - Execute program

Description of your project's program languages, framework, libraries.

For creation of our machine learning model we have used Python scripting language along with NumPy scientific calculations library, used for manipulating arrays, and TensorFlow framework, a machine learning framework powered by Google for implementation of AI tasks. It provides a broad range of models, loss, cost and error functions, it also consists of functions that allow for easy creation of neural network layers. TensorFlow is a data-flow architecture, it allows developers to define a flow graph that an array of data (tensors in this framework) is manipulated or used for calculations.

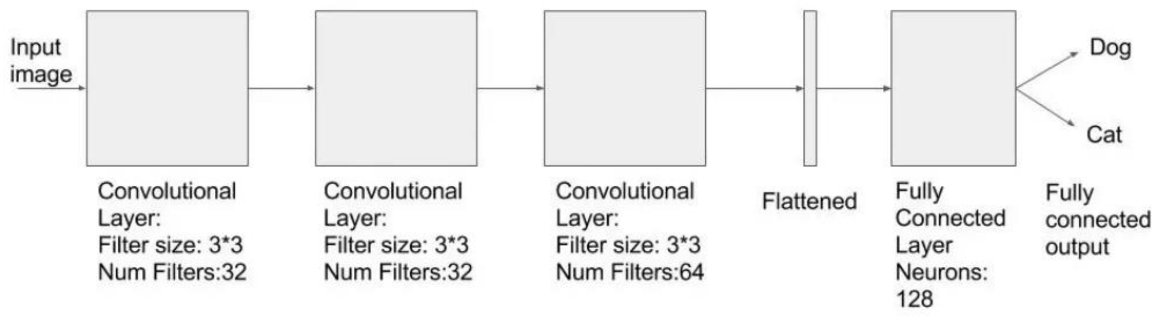
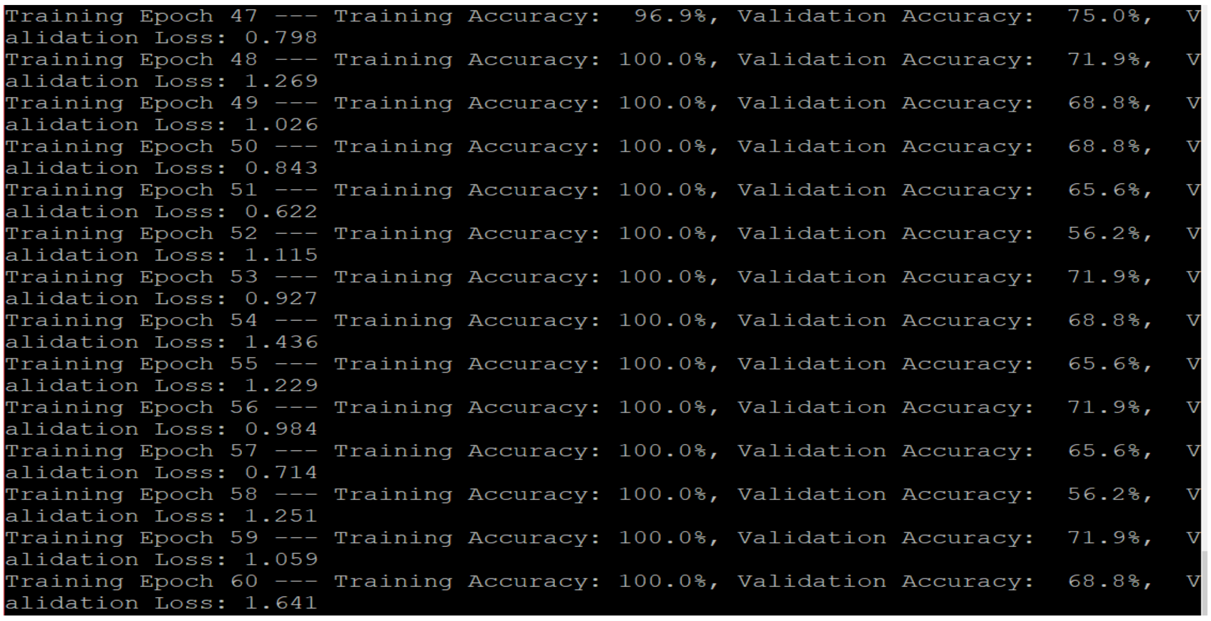

Input image
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:32
Convolutional Layer: Filter size: 3*3 Num Filters:64
Flattened
Fully Connected Layer Neurons: 128
Dog
Cat
Fully connected output

Figure 2: Cat/Dog Classifier Model Framework

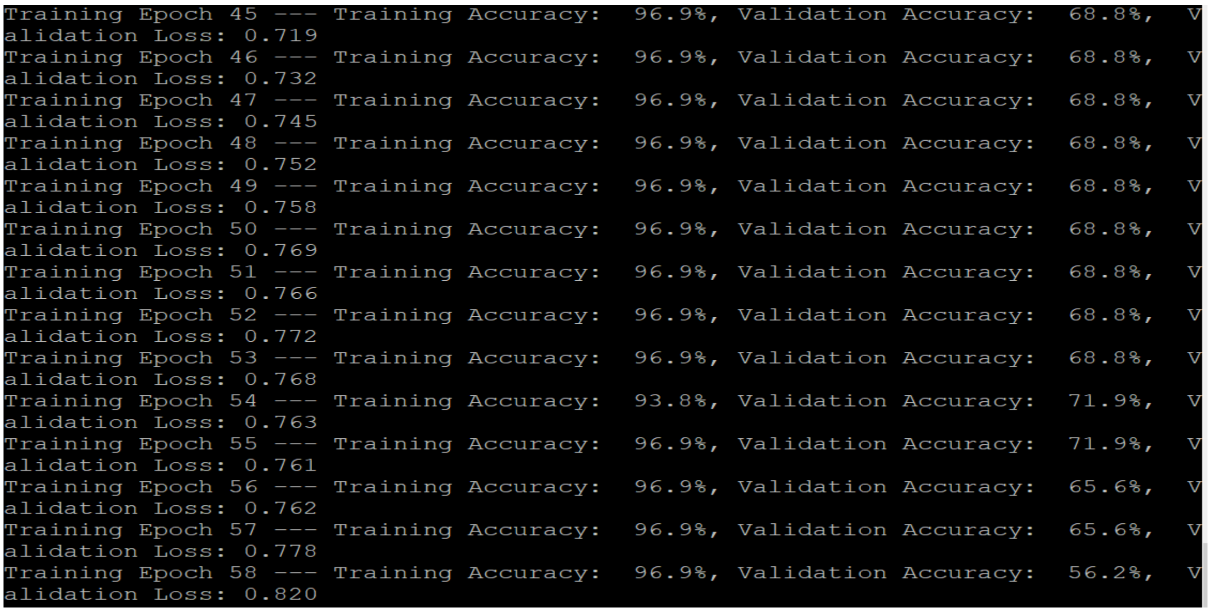
Experiments

We ran the training model between 1000 - 3000 rotations on batch sizes of 16 and 32 in order to compare the overall results. The batch sizes of 32 seemed to give a more uniform increase in training accuracy, though validation accuracy still fluctuated around the center. Testing the classifier using the test data, however, provided a more uniform increase in validation accuracy, with Validation accuracy climbing to 68.8%, with a testing accuracy of 96.9% and validation loss of about 0.7.



```
Training Epoch 47 --- Training Accuracy:  96.9%, Validation Accuracy:  75.0%,  V
alidation Loss: 0.798
Training Epoch 48 --- Training Accuracy: 100.0%, Validation Accuracy:  71.9%,  V
alidation Loss: 1.269
Training Epoch 49 --- Training Accuracy: 100.0%, Validation Accuracy:  68.8%,  V
alidation Loss: 1.026
Training Epoch 50 --- Training Accuracy: 100.0%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.843
Training Epoch 51 --- Training Accuracy: 100.0%, Validation Accuracy:  65.6%,  V
alidation Loss: 0.622
Training Epoch 52 --- Training Accuracy: 100.0%, Validation Accuracy:  56.2%,  V
alidation Loss: 1.115
Training Epoch 53 --- Training Accuracy: 100.0%, Validation Accuracy:  71.9%,  V
alidation Loss: 0.927
Training Epoch 54 --- Training Accuracy: 100.0%, Validation Accuracy:  68.8%,  V
alidation Loss: 1.436
Training Epoch 55 --- Training Accuracy: 100.0%, Validation Accuracy:  65.6%,  V
alidation Loss: 1.229
Training Epoch 56 --- Training Accuracy: 100.0%, Validation Accuracy:  71.9%,  V
alidation Loss: 0.984
Training Epoch 57 --- Training Accuracy: 100.0%, Validation Accuracy:  65.6%,  V
alidation Loss: 0.714
Training Epoch 58 --- Training Accuracy: 100.0%, Validation Accuracy:  56.2%,  V
alidation Loss: 1.251
Training Epoch 59 --- Training Accuracy: 100.0%, Validation Accuracy:  71.9%,  V
alidation Loss: 1.059
Training Epoch 60 --- Training Accuracy: 100.0%, Validation Accuracy:  68.8%,  V
alidation Loss: 1.641
```

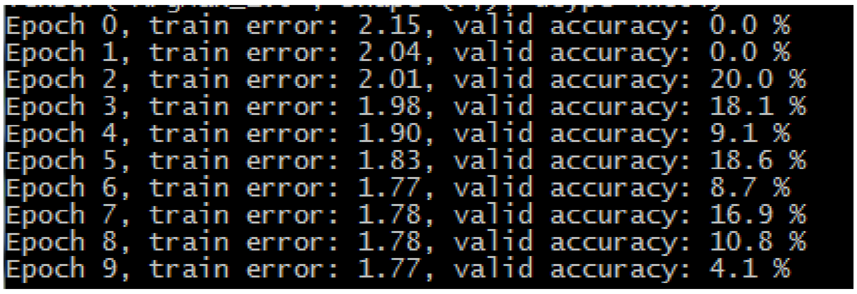
Figure 3: Training Data Accuracy



```
Training Epoch 45 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.719
Training Epoch 46 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.732
Training Epoch 47 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.745
Training Epoch 48 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.752
Training Epoch 49 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.758
Training Epoch 50 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.769
Training Epoch 51 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.766
Training Epoch 52 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.772
Training Epoch 53 --- Training Accuracy:  96.9%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.768
Training Epoch 54 --- Training Accuracy:  93.8%, Validation Accuracy:  71.9%,  V
alidation Loss: 0.763
Training Epoch 55 --- Training Accuracy:  96.9%, Validation Accuracy:  71.9%,  V
alidation Loss: 0.761
Training Epoch 56 --- Training Accuracy:  96.9%, Validation Accuracy:  65.6%,  V
alidation Loss: 0.762
Training Epoch 57 --- Training Accuracy:  96.9%, Validation Accuracy:  65.6%,  V
alidation Loss: 0.778
Training Epoch 58 --- Training Accuracy:  96.9%, Validation Accuracy:  56.2%,  V
alidation Loss: 0.820
```

Figure 4: Testing Data Accuracy

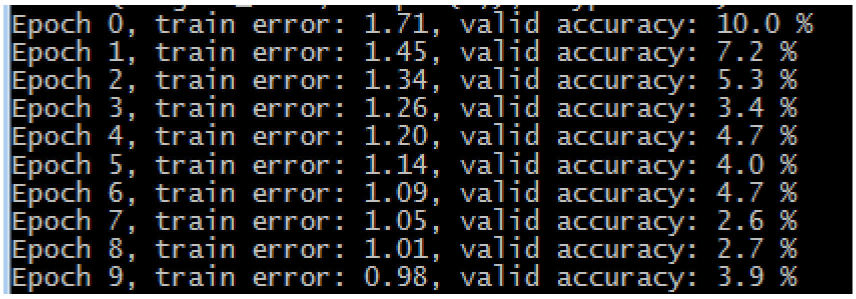
While developing our image classifier to test our solution we have performed experiments on various parameters. We tweaked number of iterations per epoch, number of epochs, value of learning rate, and size of each batch. During experiments our model was tested on relatively small number of iterations and epochs, while learning rate was changing, the batch size was also small. Values used were, 100 iterations, 10 epochs, 0.0001 as learning rate and 10 for batches. This yielded very small test accuracy, which we blamed on overfitting due to small values for these parameters.



```
Epoch 0, train error: 2.15, valid accuracy: 0.0 %
Epoch 1, train error: 2.04, valid accuracy: 0.0 %
Epoch 2, train error: 2.01, valid accuracy: 20.0 %
Epoch 3, train error: 1.98, valid accuracy: 18.1 %
Epoch 4, train error: 1.90, valid accuracy: 9.1 %
Epoch 5, train error: 1.83, valid accuracy: 18.6 %
Epoch 6, train error: 1.77, valid accuracy: 8.7 %
Epoch 7, train error: 1.78, valid accuracy: 16.9 %
Epoch 8, train error: 1.78, valid accuracy: 10.8 %
Epoch 9, train error: 1.77, valid accuracy: 4.1 %
```

Figure 5: Sample of dataset

Trying this model with epoch, iterations and batch size parameters which included the whole dataset in the training phase also proved to give decreasing accuracy, and also an extremely lengthy training process.



```
Epoch 0, train error: 1.71, valid accuracy: 10.0 %
Epoch 1, train error: 1.45, valid accuracy: 7.2 %
Epoch 2, train error: 1.34, valid accuracy: 5.3 %
Epoch 3, train error: 1.26, valid accuracy: 3.4 %
Epoch 4, train error: 1.20, valid accuracy: 4.7 %
Epoch 5, train error: 1.14, valid accuracy: 4.0 %
Epoch 6, train error: 1.09, valid accuracy: 4.7 %
Epoch 7, train error: 1.05, valid accuracy: 2.6 %
Epoch 8, train error: 1.01, valid accuracy: 2.7 %
Epoch 9, train error: 0.98, valid accuracy: 3.9 %
```

Figure 6: Whole Dataset

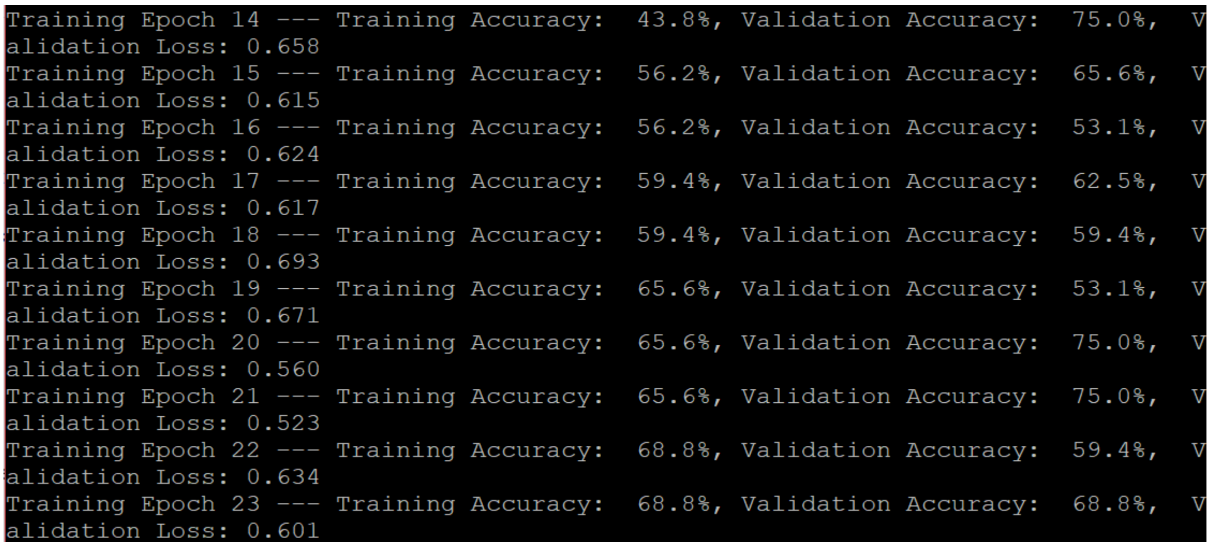
Increasing or decreasing learning rate did not make any difference. This suggests that an alternative optimizer function should have been used in our model. We did not have time to test this theory.

We used a stationary PC to test our application, processor Intel I7-2600K (4 Cores, 8 Threads), Windows 7 Home Premium. Approximate time of running for lower set of values was about 5 minutes, while running it on full dataset took around 20 minutes. There was definitely room for improvement in this area, as we did not utilize objects provided by TensorFlow that provide efficiency, neither have we attempted to incorporate a GPU into processing of our program.

Results Analysis and Conclusion

We measured how good and accurate our classification model is using accuracy metrics. This simply calculates the percentage of correct predictions to give an estimate of algorithms performance. As seen in output of our program below, we track accuracy of training set during each epoch and also check accuracy on a test dataset after an epoch ends.

Our initial results using the basic system was promising, as we were quickly able to get it to work, though we've had some problems with the validation accuracy, which is very likely a result of overfitting of the data. As can be seen below, while the training accuracy steadily improves, the validation accuracy remains fluctuating around the centre. The good performance of our initial basic system gave a lot better results that our further enhanced program.



```
Training Epoch 14 --- Training Accuracy:  43.8%, Validation Accuracy:  75.0%,  V
alidation Loss: 0.658
Training Epoch 15 --- Training Accuracy:  56.2%, Validation Accuracy:  65.6%,  V
alidation Loss: 0.615
Training Epoch 16 --- Training Accuracy:  56.2%, Validation Accuracy:  53.1%,  V
alidation Loss: 0.624
Training Epoch 17 --- Training Accuracy:  59.4%, Validation Accuracy:  62.5%,  V
alidation Loss: 0.617
Training Epoch 18 --- Training Accuracy:  59.4%, Validation Accuracy:  59.4%,  V
alidation Loss: 0.693
Training Epoch 19 --- Training Accuracy:  65.6%, Validation Accuracy:  53.1%,  V
alidation Loss: 0.671
Training Epoch 20 --- Training Accuracy:  65.6%, Validation Accuracy:  75.0%,  V
alidation Loss: 0.560
Training Epoch 21 --- Training Accuracy:  65.6%, Validation Accuracy:  75.0%,  V
alidation Loss: 0.523
Training Epoch 22 --- Training Accuracy:  68.8%, Validation Accuracy:  59.4%,  V
alidation Loss: 0.634
Training Epoch 23 --- Training Accuracy:  68.8%, Validation Accuracy:  68.8%,  V
alidation Loss: 0.601
```

Figure 7: Initial System

We chose to continue using the CNN framework for object recognition as it was best suited to the task, and in conjunction with Tensorflow, it provided a good existing library to streamline the process.

In summary, our initial multi-purpose API project idea turned out to be very challenging so we reverted back to our first and simpler idea of creating an image classifier. While future developments may allow us to expand on what have created, such as by using a facial recognition classification algorithm, our object classification model is currently well designed, constructed and fit for purpose.

This project has been a great introduction into neural networks and their inner workings, convolutional neural networks show promise and good results in area of image processing, as they learn patterns like edge, color and shape patterns. Research performed in this area for this project made machine learning more understandable and manageable, using TensorFlow framework also proved to be useful in understanding the concepts of AI optimization. This project will serve as an example to problems that participants of this group picked as their practicum ideas.

The complete files to this assignment can be located here, along with references to material used:

https://github.com/Lorigap/Machine-Learning-Project