Venice boat classification project report.

Name: **Oluwatoyin Sanni**
Matricola: **1871835**
Department: **Artificial Intelligence and Robotics** November 22nd, 2018

This report is accompanied with two major files:

- A python script named *"classify_test_images"* which is the script used to extract images contained in the test folder into their corresponding classes using the *"ground_truth.txt"*. The newly classified test images was used to validate our model performance and this script was done in collaboration with another colleague, Ricardo.
- Two jupyter notebook which contains
  - VGG16model ran on google colab using the GPU hardware accelerator.
  - A jupyter notebook of 4 different architectures ran on my local computer which was also converted to a PDF.

All jupyter notebooks was solely written by me.

I. Definition

Project Overview

Image Classification problem, which is the task of assigning an input image one label from a fixed set of categories is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. As data acquisition advances and grows large, deep learning which is a subset of machine learning mimicking the brain neurons has proved to be one of the great tool in Computer vision.

A lot of work has emerged in this field from different researchers over the world. Examples include Prof Fei Fei Li outstanding project with ImageNet, the MNIST and the Zalando's article images of Fashion MNIST. It's no doubt Computer vision is an interesting field that's making breakthrough in different applications both in the academic and industry.

This project is focused on the detection and classification of Venice boats using Deep learning. Past related project used the following classification methods: K-Nearest Neighbor (KNN), a Decision Tree Learning Algorithm, and Random Forest (RF) from the WEKA environment. In this project, We will build a convolution neural network(CNN) that can detect Venice boats and identify its category while explaining the techniques used throughout the process.

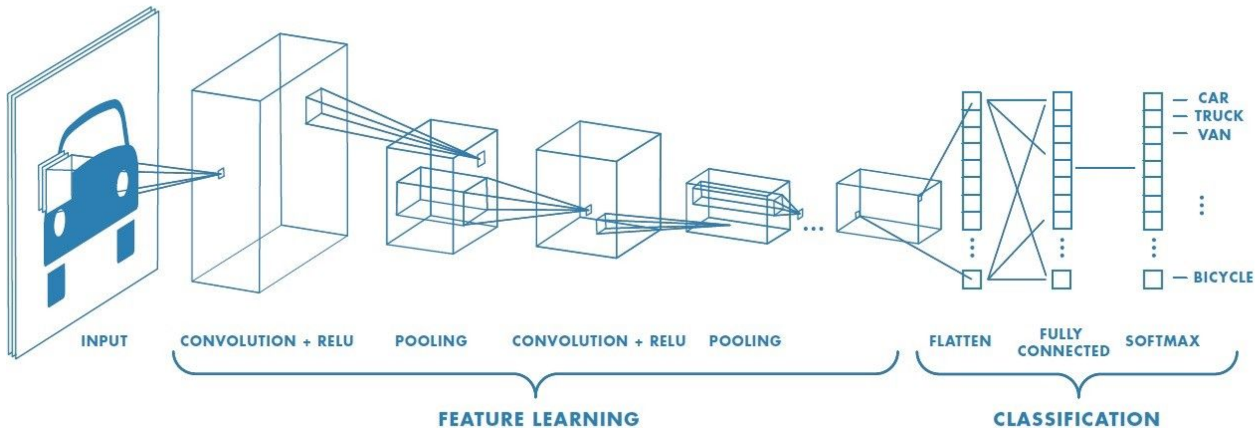


Figure 1: A CNN architecture in diagram

Problem statement

The dataset for this problem contains two folders; the classified training data of 24 categories(excluding water image folder) and 1969 unclassified images accompanied with a text file that contains each of the 1969 category. Each image is a **800*240 in 3 channels(RGB).**

According to the research paper, the images were acquired from the ARGOS system. The ARGOS is a satellite-based system which collects, processes and disseminates environmental data from fixed and mobile platforms worldwide. tracks all the boats within the views of the cameras. The system was used for automatically generating snapshots of the boats. In order to gather high resolution snapshots of the boats, they are taken when the boats traverse the middle line of the central camera of a survey cell (which corresponds to the minimum distance from the cell).

This project aims to propose a befitting neural network to classify the Venice boats snapshots. In this work, I employed different approaches, from a custom CNN to using the concept of Transfer learning. i.e the application of pre-trained model to classify the images.

Metrics

We shall consider Loss and Accuracy as our metrics. The lower the **loss,** the better a model (unless the model has over-fitted to the training data). The loss is calculated on **training** and **validation** and its interpretation is how well the model is doing for these two sets. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets.

In the case of neural networks, the loss is usually negative log-likelihood and residual sum of squares for classification and regression respectively. This naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks.

Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of misclassification is calculated.

II. Analysis

Data Extraction and Pre-processing

As said earlier, our downloaded original dataset contains the classified training data of 25 categories(including the water image folder) and 1969 unclassified images accompanied with a text file that contains each of the 1969 category.

After writing a script to classify the test data in order to use it for validation of our model, it was discovered not all boats names found in the classified test folder were found in the training set.

Classes that were **deleted from the training data** because they weren’t found in the test data after classification can be found below:

1. Cacciapesca
2. Caorlina
3. Lanciamaggioredi10mMarrone
4. Sanpierota
5. VigilidelFuoco
6. Water

After cleaning up the missing classes and unifying the number of classes in our training and test set, we were left with a total of 18 classes where are; *'Alilaguna', 'Ambulanza', 'Barchino', 'Gondola', 'Lanciafino10m', 'Lanciafino10mBianca', 'Lanciafino10mMarrone', 'Lanciamaggioredi10mBianca', 'Motobarca', 'Motopontonerettangolare', 'MotoscafoACTV', 'Mototopo', 'Patanella', 'Polizia', 'Raccoltarifiuti', 'Sandoloaremi', 'Topa', 'VaporettoACTV'*.

It was also discovered that each class had different number of images both in the training and test set, some had low numbers as **Lanciamaggioredi10mBianca : 9** in the training data and some had high numbers like **Mototopo : 878.** We could see this is a little bit similar to an imbalanced dataset but we will explore how well a CNN can deal with the images.

Fig. 2 further shows a view of the number of images in each of our class for the training and test(validation data).

Training Data	**Test Data**
Gondola : 24	Gondola : 3
Topa : 78	Topa : 29
Mototopo : 878	Mototopo : 284
Ambulanza : 85	Ambulanza : 22
Lanciafino10mMarrone : 355	Lanciafino10mMarrone : 125
Motopontonerettangolare : 18	Motopontonerettangolare : 3
Lanciafino10m : 22	Lanciafino10m : 7
VaporettoACTV : 949	VaporettoACTV : 325
Barchino : 112	Barchino : 51
MotoscafoACTV : 11	MotoscafoACTV : 1
Lanciamaggioredi10mBianca : 9	Lanciamaggioredi10mBianca : 6
Lanciafino10mBianca : 484	Lanciafino10mBianca : 217
Motobarca : 215	Motobarca : 59
Raccoltarifiuti : 94	Raccoltarifiuti : 19
Sandoloaremi : 13	Sandoloaremi : 3
Polizia : 71	Polizia : 15
Patanella : 279	Patanella : 74
Alilaguna : 113	Alilaguna : 19

Figure 2

I also applied cross validation to our data. The original dataset was used for training while the classified test data was used for validation. However, we could have splitted the given training data into two subsets which will consists of our training and validation data. The reason why this path wasn't explored was because lot of data is very essential to a good performance of our network. Simply put, the more the training data, the merrier and happy our architecture.

Data Exploratory and Visualization

In order to train a function that can be a good performance model, we explored different techniques with our data. A convolutional neural network that can robustly classify objects even if its placed in different orientations is said to have the property called **invariance**. More specifically, a CNN can be invariant to **translation, viewpoint, size** or **illumination** (Or a combination of the above).

This essentially is the premise of **data augmentation**. In the real world scenario, we may have a **dataset** of images taken in a **limited set of conditions**. But, our **target application** may exist in a **variety of conditions**, such as different orientation, location, scale, brightness etc. We account for these situations by training our neural network with additional **synthetically modified data**.

In order to make sure our model takes the above paragraph analogy into consideration, we applied Data augmentation to help increase the amount of **relevant data** in our dataset. We applied the following Augmentation Techniques to our data:

- Flip: Images can be horizontally and vertically flipped. In our case, we did a horizontal flip.
- Rotation: We performed random rotation on our images.
- Resize: For fast computation process, we resized our images from 800*240 to 256*256.

Figure 2 below shows what our data augmentation looks like.

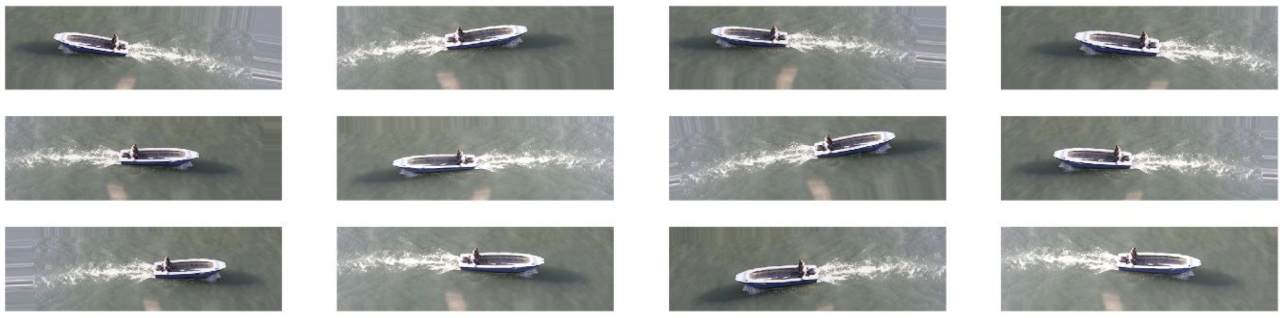


Figure 3

There are other augmentation techniques like Cropping, translation, Gaussian noise, GANS but they weren’t explored in this project.

Algorithms and Techniques

In order to have a robust overview of different network architectures, 4 network architecture layers were explored using the **Pytorch** library:

1. **Customized CNN Layer:** This architecture performed badly. We set this up with 3 convolution layers, with a kernel size of 3-by-3 and a padding value of 1 in order to preserve the border of our images to avoid the loss of important image pixels that could be valuable to our network. One max pooling layer was also added with a kernel size of 3-by-3 and one stride. Finally, we used 1024 number of output features for our first connected layer, followed by 18 classes of output features for our last layer which represents our 18 classes probability predictions of the Venice boat classes. LogSoftmax was used for our final activation layer, this was selected as it's proven to be the best for multi-classification purpose. Our 9 network layers graph is shown below.

   INPUT => CONV1 => RELU => CONV2 => RELU => POOL => CONV3 => SOFTMAXLOG => FC1 => FC2

   The best accuracy result we got from this architecture is **0.257528** which of course, is below an acceptable benchmark score.

2. **LeNet:** The LeNet architecture is an excellent "first architecture" for Convolutional Neural Networks (especially when trained on the MNIST dataset, an image dataset for handwritten digit recognition). LeNet is small and easy to understand — yet large enough to provide interesting results. The LeNet architecture consists of the following layers:

   INPUT => CONV1 => RELU => POOL => CONV2 => RELU => POOL => FC => RELU => FC

   We will see it is slightly different from our CNN architecture with few tweaks like reduction of our convolution layers and the max pooling layer after each convolution layer. Since, it's been proved to perform excellently on the MNIST data, we explored it for our problem set as well without any modification. Its architecture is shown in Figure 4.

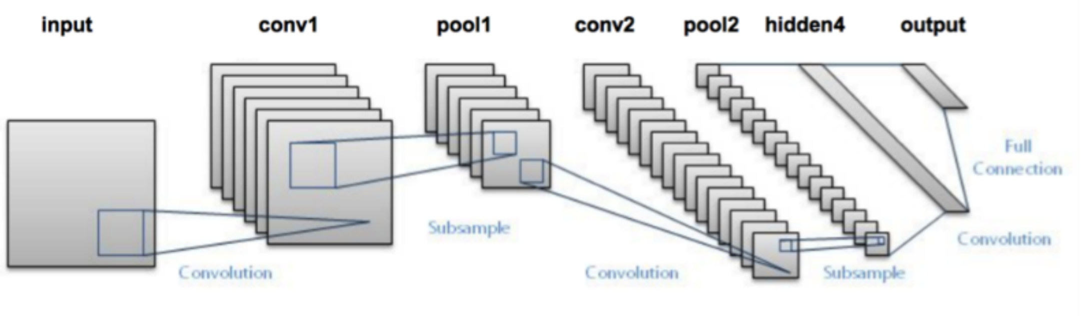


Figure 4

The best accuracy result we got from this architecture is **0.557845** which is just slightly above average.

3. **AlexNet**: According to wikipedia, AlexNet is the name of a convolutional neural network, designed by Alex Krizhevsky, and published with Ilya Sutskever and Krizhevsky's PhD advisor Geoffrey Hinton. It is a pretrained convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 8 layers deep and can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 227-by-227.

   It contains 5 convolutional layers and 3 fully connected layers. Relu is applied after very convolutional and fully connected layer. Dropout is applied before the first and the second fully connected year. The image size in the following architecutre chart should be 227 * 227 instead of 224 * 224, as it is pointed out by Andrei Karpathy in his famous CS231n Course. More interestingly, the input size is 224 * 224 with 2 padding in the pytorch torch vision. The output width and height should be (224–11+4)/4 + 1=55.25! The explanation here is pytorch Conv2d apply floor operator to the above result, and therefore the last one padding is ignored.

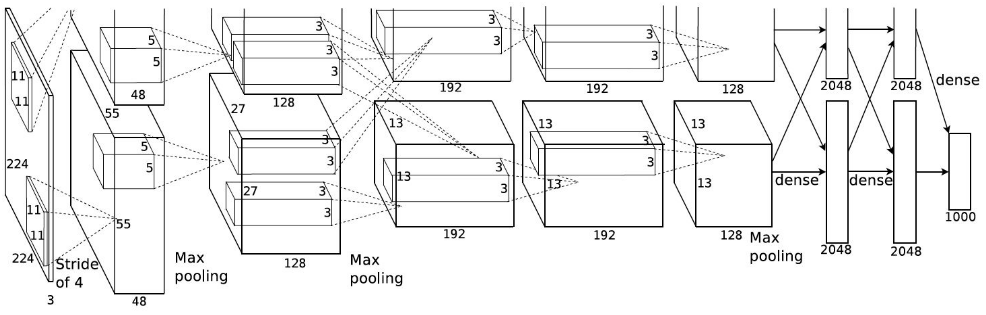


Figure 5

The only change we made to this network architecture was changed its last output layer from 1000 images to 18 which fits our own problem set of classifying venice boat images into different 18 categories.

The best accuracy result we got from this architecture is 0.267036 which is almost the same as our customized CNN network. We can say this performed badly as well.

4. **VGG16**: The last pre trained CNN we considered. It's a huge in number of layers than the previous two we discussed above. VGG16 (also called OxfordNet) is a convolutional neural network architecture named after the Visual Geometry Group from Oxford, who developed it. It was used to win the ILSVR (ImageNet) competition in 2014. To this day is it still considered to be an excellent vision model, although it has been somewhat outperformed by more recent advances such as Inception and ResNet. Just like AlexNet, we only changed the last FC layer from 1000 to 18 for our project. A broad explanation of its architecture can be found in figure. 6

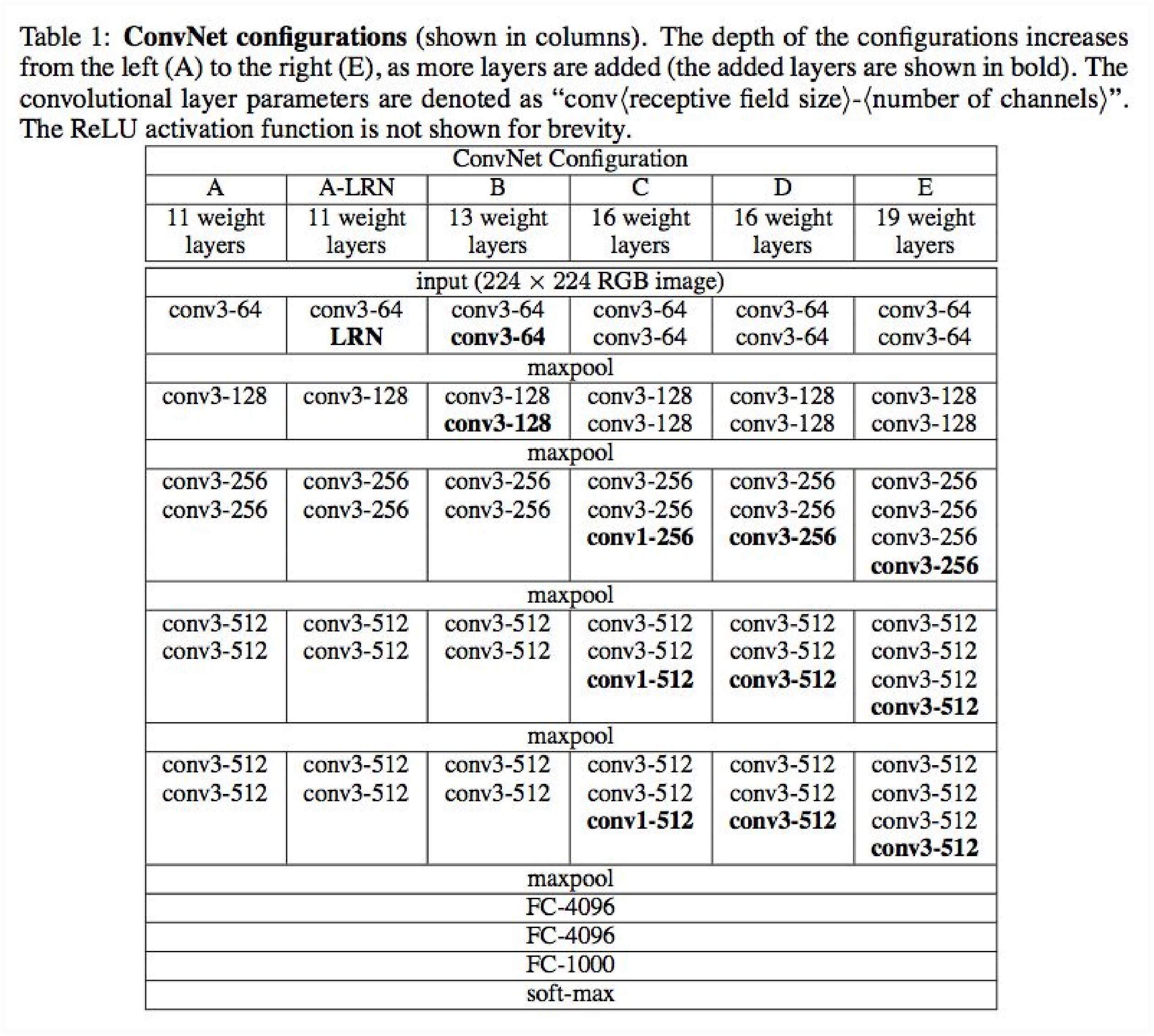


Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 6

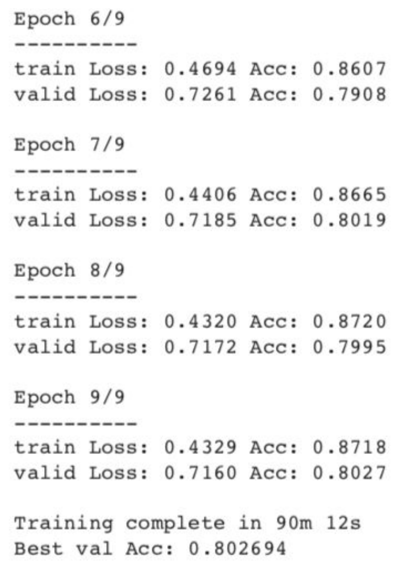
Due to its complexity of its layers, we can guess it gave the best accuracy value of **0.802694** which is far above average and close to a 1.0. Its layers were also froze to avoid pre-training for our classification. This is a great improvement to the previous architectures we've discussed in this project. More improvement could have been done by adjusting our learning rate and changing the loss function to different options.

However, it took about 90 minutes of computation and due to memory error on google colab, not much could be done within the timeframe.

III. Results

Model benchmark, evaluation and validation

To benchmark which architecture performed best, I selected VGG16 model based on the validation accuracy score of our model on the validation data. I also considered the fact that the validation and training loss didn't diverge too much as shown in figure.



```
Epoch 6/9
----------
train Loss: 0.4694 Acc: 0.8607
valid Loss: 0.7261 Acc: 0.7908

Epoch 7/9
----------
train Loss: 0.4406 Acc: 0.8665
valid Loss: 0.7185 Acc: 0.8019

Epoch 8/9
----------
train Loss: 0.4320 Acc: 0.8720
valid Loss: 0.7172 Acc: 0.7995

Epoch 9/9
----------
train Loss: 0.4329 Acc: 0.8718
valid Loss: 0.7160 Acc: 0.8027

Training complete in 90m 12s
Best val Acc: 0.802694
```

Figure 7

IV. Conclusion

The most interesting part of this project for me is exploring different pre-trained model which I had never done before. It was quite challenging especially with running out of memory on google colab notebook and also time consuming as well.

Reflection

The process for this project include the following:

1. Download and pre-process the data.
2. Selection of validation data
3. Selecting a performance metric to benchmark our network.
4. Asides our poor customized CNN network, research various pre-trained models to apply to our problem statement.

5. Application of different pretrained network to our images.

Improvement

I believe there are several ways of improvement:

1. **Hyperparameter tuning** and the application of **grid or random search** across the selected models in order to select the best parameter values will be an improvement to this project.
2. Exploring other metric evaluation models like
3. Exploring more network layers in the custom CNN models we wrote from scratch.

References

- https://en.wikipedia.org/wiki/Argos_system
- https://blog.algorithmia.com/convolutional-neural-nets-in-pytorch/
- https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced
- https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
- https://www.pyimagesearch.com/2016/08/01/lenet-convolutional-neural-network-in-python/
- https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637
- https://hackernoon.com/learning-keras-by-implementing-vgg16-from-scratch-d036733f2d5
- https://blog.keras.io/how-convolutional-neural-networks-see-the-world.html