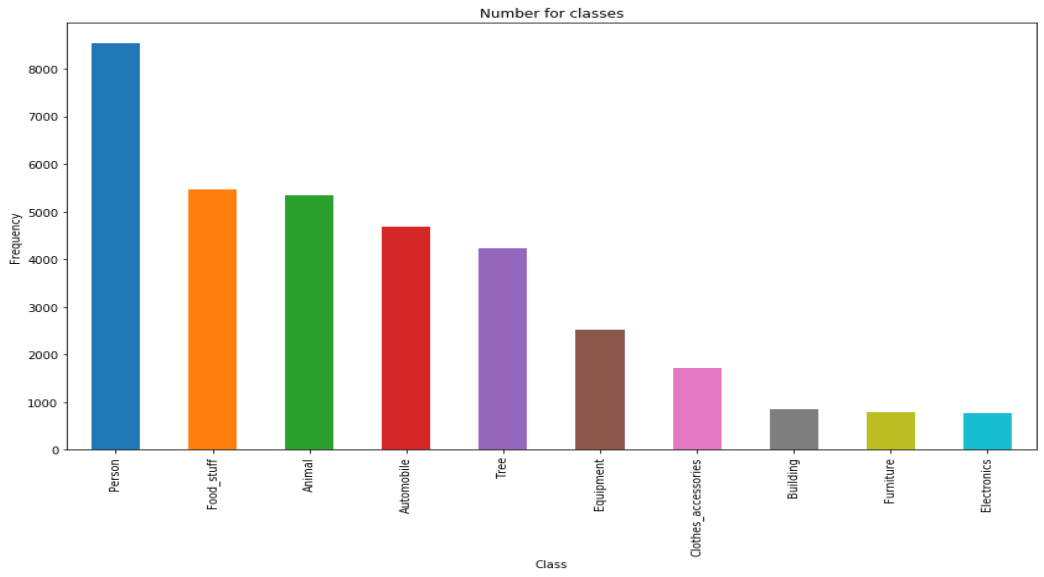
Google AI Open Image Classification
Project: Report

Group: 8
Hninn, Priyanka, Richa

1.1 Introduction

In this project, we have chosen Google AI Open Images dataset. This dataset is available on Google Open Image Dataset V4 website. It is built for the purpose of object detection and segmentation. It consists of natural images that reflect everyday scene and provides contextual information. There are total 516 classes with total 9 million images. The original size of image is 1024x760. The type of objects in these datasets vary in shape, size and color across the same class. Since the raw dataset is very large, we only used a portion of it for our analysis. The aim of this project is to classify the different images in our dataset. The below figure shows the total 10 classes which we are using in this project.



The first step of the learning process is to create an usable set of images for training and testing from the raw image files. The first thing that we did on the dataset was to resize all the images in one size that is (60x60) and give 3 channels for training and testing purposes. The original size of image has a very high resolution of 1024x760 and also had some gray

images in our dataset, we only considered coloured images. After this preprocessing, we ended up with 34,471 images and since the data is huge, training the data in our own computer is not feasible, therefore we are working on GCP and AWS.

Framework

For our project, we have used tensorflow as the framework. It is an open source API developed by Google mainly for machine learning and deep learning. The advantages of using tensorflow is that it has many built-in functions. Our main motivation for this project is to learn how to set up, train and build CNN's in tensorflow and other frameworks. We also used keras as our framework to compare our results. We have used RELU and softmax activation functions with different frameworks.

Convolution Network:

We used the convolutional neural network for our project because the main idea of the project is to be able to classify the images. We think that it makes sense to use Deep convolutional neural networks. CNN has been used to great effect in applications such as image classification, object detection and other applications. A common choice for the structure of our CNN consists of layers of convolutional and max pooling layers which are then combined in a deep layer. It has size of the filter, the kernel size, strides, padding and activation function. For max pooling layer, the parameters are kernel size and strides. The functions applied in this project are RELU and softmax. In this case, we are using a max pooling layer which is a common choice for convolutional neural networks. We have also used other commonly used layers that are flatten and dropout layers. Dropout layers are used to prevent overfitting.

CNN has two major advantages:

1. Feature engineering / preprocessing – turn our images into something that the algorithm can interpret more efficiently
2. Classification – train the algorithm to map our images to the given classes, and understand the underlying relationship

The next step is to choose the loss function. For classification problems, one of the most popular choices for loss function is softmax cross entropy function and we have used the softmax cross entropy function in our project. The parameters available for the loss function includes weights, biases. By minimizing the loss function during the training process, the error of the network is minimized.

Data Problem:

We have following issues with the data when we started the project:

i. We have many images that has no labels.
ii. The format of the image IDs does not match with labels image ID
iii. The dataset has both gray images and color images
iv. Images are not in the same shape

2.1 Data Augmentation

Labeling the Class:

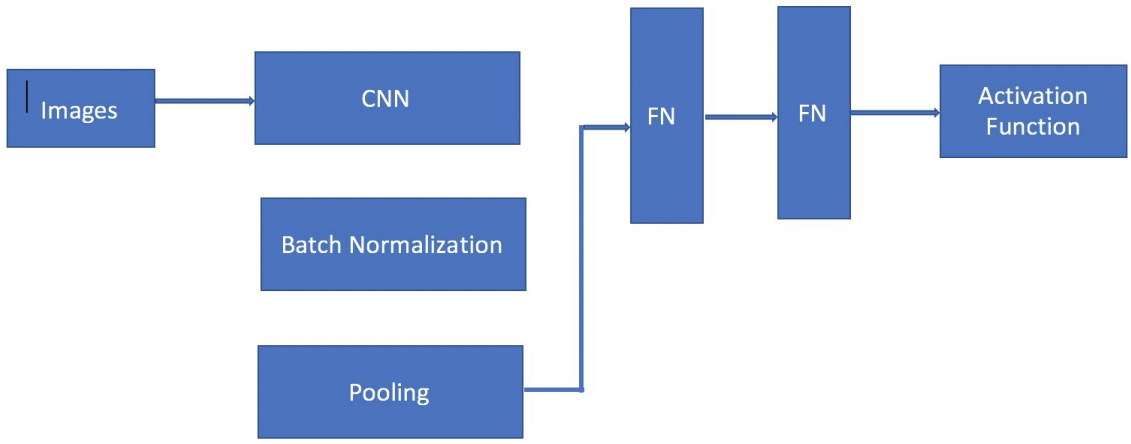
There are a lot of classes that share common features that might be helpful for our model, but such hierarchical structure also rises new problems such as model gets confused by same classes. Since there was no way for us to use the whole 9 million images dataset and 516 classes to pursue this kind of network, the results are inconclusive. The labels were brought down to 10 classes were labeled manually by us. we are only taking 35,000 images as our input data. We tried to put the subset of all classes into one class for particular labels and bring down the 516 classes to 10 classes. Initially, we have different image sizes, but we resized them to 60x60. We removed the gray images and only considered only coloured images with 3 channels.

One hot encoding:

Since our Labels are in categorical form and we have 10 labels so we changed our labels to binary form by using one hot encoder before we split the data into train and test in keras utils function.

4.1 Batch Normalization:

In pursuit of improving the accuracy of CNN models with larger and deeper network, vanishing gradient is one of the critical challenges which prevents the network from converging fast. To overcome this problem, recent CNN models have adopted batch normalization. Since we have very wide range of images and shapes, inputs are highly distributed. Studies have shown that CNN with batch normalization is very good at classification task and also showed performance improvements in Coco dataset detection.
We used the batch normalization in our network so that it does not make weight and biases much shift in networks and learns faster.



To evaluate our model we used using K-fold cross validation to improve our model performance. K-fold cross validation is often used with five or ten folds but we did with 2 folds because we big computation power so we have considered 2 folds to check our model performance. And also one of the advantage of k-fold cross validation is that it doesn't make model overfit. The below figure shows the architecture for CNN model for K-fold cross validation.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 56, 56, 32)	2432
batch_normalization_1 (Batch	(None, 56, 56, 32)	128
max_pooling2d_1 (MaxPooling2	(None, 28, 28, 32)	0
conv2d_2 (Conv2D)	(None, 24, 24, 64)	51264
batch_normalization_2 (Batch	(None, 24, 24, 64)	256
max_pooling2d_2 (MaxPooling2	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 9, 9, 64)	65600
batch_normalization_3 (Batch	(None, 9, 9, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_1 (Dense)	(None, 256)	262400
batch_normalization_4 (Batch	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 10)	2570

Total params: 385,930
Trainable params: 385,098
Non-trainable params: 832

3.1 Network Parameters:

i. Tensorflow using simple CNN

The input data on our networks is given in batches. In order for the network to use the data for training, it has to be reshaped into tensor format. In our case we have initially 1024x760 image size and some images are grayscale as well. There is no one right answer to how to choose the networks parameters. A common method when finding good parameters is to simply try different values and compare the results. The learning rate for our network is 1e-4. The parameters used in the model that we have evaluated are listed in following tables. We have tried the below network without implementing batch normalization.

Table1: First convolutional layer

Input data shape	[34471, 60, 60, 3]	[34471, 60, 60, 3]	[34471, 60, 60, 3]
Number of filters	32	32	32
Kernel size	5x5	5x5	5x5
Padding	None	None	None
Activation	Softmax	softmax	softmax
Optimizer	Adam	Adam	Adam
Batch size	50	50	50
epochs	2	200	500

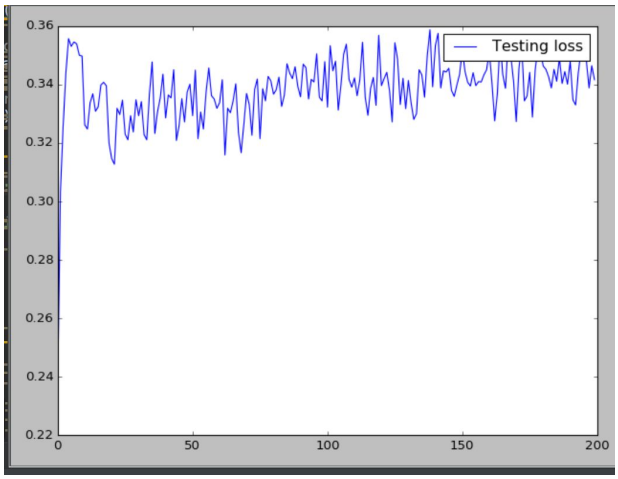
Table2: Second convolutional layer

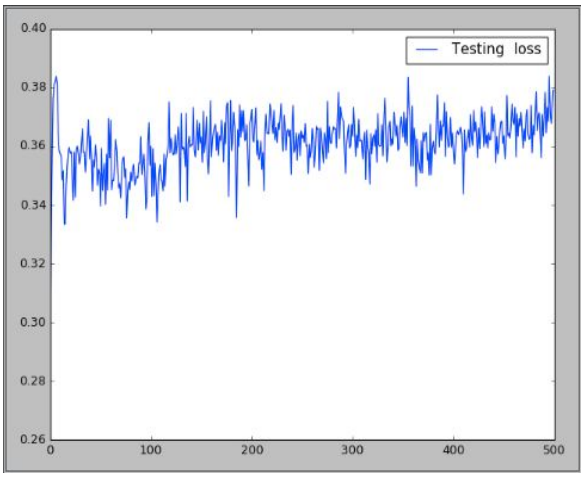
Input data shape	[34471, 60, 60, 3]	[34471, 60, 60, 3]	[34471, 60, 60, 3]
Number of filters	64	64	64
Kernel size	5x5	4x4	4x4
Padding	None	None	None
Activation	softmax	Softmax	Softmax
Optimizer	Adam	Adam	Adam
Batch size	50	50	50
epochs	200	2	500

Table3: Third convolutional layer

Input data shape	[34471, 60, 60, 3]	[34471, 60, 60, 3]	[34471, 60, 60, 3]
Number of filters	128	128	128
Kernel size	3x3	3x3	5x5
Padding	None	None	None

Activation	softmax	softmax	softmax
Optimizer	Adam	Adam	Adam
Batch size	50	50	50
epochs	200	2	500





Accuracy: Accuracy from the above network without batch normalization is 35% with 200 epochs, 2 epochs is 31.69% and with 500 epochs our accuracy is 37%. Loss functions provide more than just a static representation of how our model is performing – it shows how our algorithms fit data in the first place.

The following network is implemented with different optimizer that is Adagrad and without batch normalization. This networks gives us very less accuracy. Accuracy from the below network using tensorflow is 20.25%.

Table1: Convolutional layers with Adagrad optimizers

Parameters	First layer	Second layer	Third Layer
Input data shape	[34471, 60, 60, 3]	[34471, 60, 60, 3]	[34471, 60, 60, 3]
Number of filters	32	64	128

Kernel size	5	4	5
Padding	None	None	None
Activation	Softmax	Softmax	Softmax
Optimizer	Adagrad	Adagrad	Adagrad
epochs	2	2	2

We have implement the below network with batch normalization:

The following table defines the parameter for the model implemented in Keras:

Parameters	First layer	Second layer	Third Layer
Input data shape	[34471, 60, 60, 3]	[34471, 60, 60, 3]	[34471, 60, 60, 3]
Number of filters	32	64	64
Kernel size	5x5	5x5	5x5
Padding	None	None	None
Activation	Relu	Relu	Relu
Optimizer	Adam	Adam	Adam
Batch size	32	32	32
epochs	20		

The model implemented with batch normalization gives us 50% accuracy. In every convolution layers we have used batch normalization.

6.1 Model Accuracy and Results:

The training data was split into train/test set with the help of python functions. 0.80 and 0.30 was the train and test split simultaneously. When trained on this data without batch normalization the resulting of model reached as high as 37% accuracy in tensorflow taking 500 epochs with Adam optimizer and 4 layers. We have also trained the CNN models with

batch normalization. We have trained with 20 epochs. The result from one of the CNN model with batch normalization on test set is 50% with 4 layers. We tried with different parameters and best kernel size was 5x5 with 2 fully connected layers.

We found that implementing batch normalization improves our model from 35% accuracy to 50% accuracy.

We only have results from small dataset because, unfortunately we were not able to get results on the whole data set due to various problems. In large dataset, we tried to use with 120,000 images, but we were experiencing memory errors. Even though we use GPU because it runs slow, but generating the data itself was also too slow.

7.1 Conclusion

The accuracy from our small dataset is low and we think it could be because of the lack of data. It could also be because of an unfit model for this data. Probably, the biggest challenge we faced in this project is the size of the data. It severely limited what we could actually produce and massively slowed down the training and testing. This approach would have been interesting if we have more data for our 10 numbers of classes. Instead, we have each model with less training data, it would also be very interesting to see the comparison of one model trained on the full dataset and another model on the smaller data. Distribution of classes was very unequal in the dataset, some classes had 10 products in one category and some of them had thousands, which is not very good, so for better training on late stages we might have to use some data augmentation.

Overall, it was a very interesting challenge with biggest challenge being the amount of data and time needed to make something with this data. We get the idea of how we can efficiently train a model for such large amount of data.

References

https://storage.googleapis.com/openimages/web/index.html

https://blog.algorithmia.com/convolutional-neural-nets-in-pytorch/

https://github.com/amir-jafari/Deep-Learning