Capstone Project
Prepared by: Tina Lam
Date: July 18, 2021

Problem Statement & Project

According to a recent study, there are 160.7 million people in the working age population (15 – 64 years old) with blindness or moderate to severe vision impairment. Vision impairment not only affects the quality of life, but it could also be a barrier to finding and keeping jobs. The research estimates the annual, global productivity loss is $410.7 billion USD. This is an area that deserves attention and can benefit from technological advancement. The goal of this project is to prototype an app that can identify the object in an image in a specific language to allow visually impaired to identify their surroundings.

How is this related to Data Science?

This project can be achieved by conolutional neural network (CNN) to enable computers to learn what different objects look like and subsequently recognizing these objects. Currently, there are apps in the market, such as, Seeing AI by Microsoft and Lookout by Google, that allow visually impaired users to use their phone camera to identify surroundings and audibly describe those objects. They utilize machine learning that enables the computer to derive meaningful information from digital images, videos and other visual inputs and take actions based on that information.

Dataset & Data Structure

The dataset used is called the CIFAR 100, from Alex Krizhevsky's website. CIFAR100 is a subset of another dataset containing millions of tiny images collected by MIT and NYU from web search engines. Alex later created CIFAR by selecting and grouping images from that huge dataset and assigning reliable superclass and class labels to each.

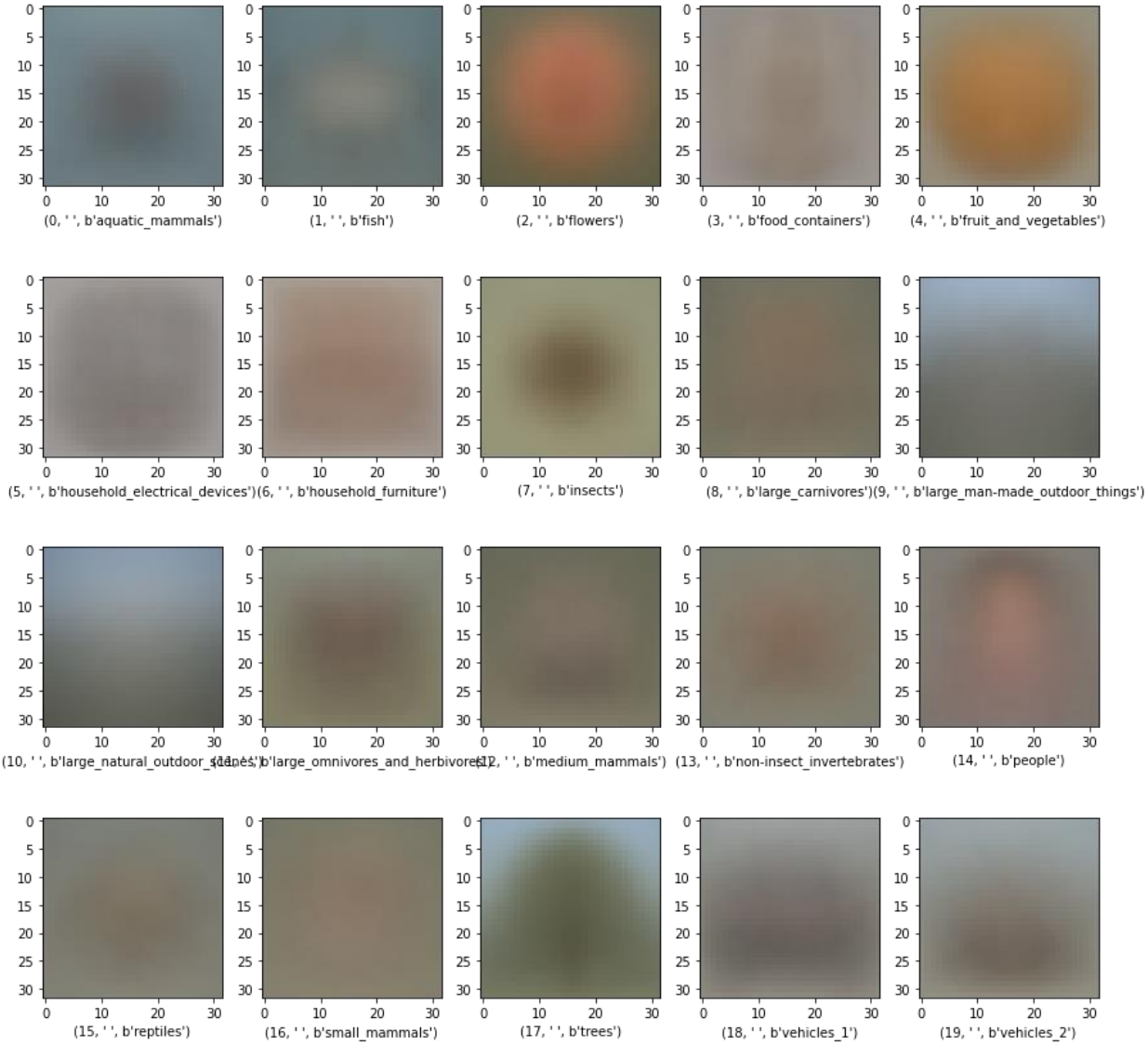
CIFAR 100 contains 60,000, 32x32 colour images. While my goal is to allow visually impaired individuals to identify their surroundings, I chose this dataset because images in the dataset are natural, not drawings, and will allow my app to identify real-life objects. It contains 20 superclasses, and each superclass contains 5 classes making a total of 100 classes. Each class contains 600 different images.

The dataset comes as a pre-split train set of 50,000 images and test set of 10,000 images. The data comes in a dictionary format with 5 keys, containing the labels and image data. The image data comes in a flattened structure of (60,000 x 3072). The first 1,024 entries represent the red channel values, the next 1,024 are the green values and the final 1,024 are blue. I reshaped this data into a 4D structure (60,000 x 32 x 32 x 3) for the CNN models.

Preprocessing and EDA Performed

1. Reviewing for duplicate images and any null value. There were no null values but there are 14 duplicate images found in different superclasses. While that is not a lot relative to the size of the dataset, the duplicate images were not removed.
2. Visualizing images from each superclass and class to get a better understanding of the dataset.
3. Transforming images into grayscale to see whether this will yield different results in my classification model if I feed in colour versus grayscale images.
4. Reviewing the average image of each superclass and class to get a sense knowing how similar or different are the images within each superclass or class and where my classification model might potentially get confused. For example, Figure 1 is the average image of each of the 20 superclasses. The most distinct image is possibly the trees. I will probably expect my model to score a higher accuracy in identifying trees.

Figure 1 Average Images of Superclasses



Modelling & Results

Approach

The goal is to train a CNN model that can predict the augmented image by the 100 classes. To make it easier in optimizing parameters of the CNN models, I first ran 8 CNN models on the non-augmented image by 20 superclasses, then trained the 2 best models with augmented superclasses. Once that worked out, I applied the best model to train the augmented images by classes.

Step 1: Non-Augmented Superclasses

A logistic regression was first done to find the baseline accuracy (26%). Then, 8 different CNN models were tried, where 4 models had only 3 convolutional layers, 2 improvised from a research paper and 2 transfer learning models.

Results (see Figure 2):

1. *Models with 3 layers* – These models had different number of filters in each layer and dropout rates. Highest accuracy (57%) on test set was achieved on the Model 4 which has more filters and higher dropout rate.
2. *Research paper* – In an attempt to improve accuracy, I found a research paper where the model is relatively replicable as it is not very deep. Note that the research paper uses the model to predict classes, not superclass. The results from these models were not better than my previous models.
3. *Transfer Learning* – Dataset was also feed into ResNet50V2 and VGG16 to see how it performs on deep models. ResNet50V2 yielded an accuracy like the Logistic Regression (26%), and VGG16 yielded a higher score of 47%. The poor accuracy is likely due to ResNet50 and VGG16 were trained on ImageNet which has a higher resolution. The default input size of ResNet and VGG16 is 224 x 224, while images in this dataset is 32 x 32. While ResNet50 is a much deeper model than VGG16, ResNet50 might have also 'over-analyzed' these low-resolution images, yielding a lower accuracy score than VGG16.

Instead of having more convolution layers, I found having higher number of filters per layer accompanied with higher dropout rates tend to yield higher accuracy scores. This is perhaps related to the low-resolution of this dataset. *Model 2 and 4* were selected for training augmented images.

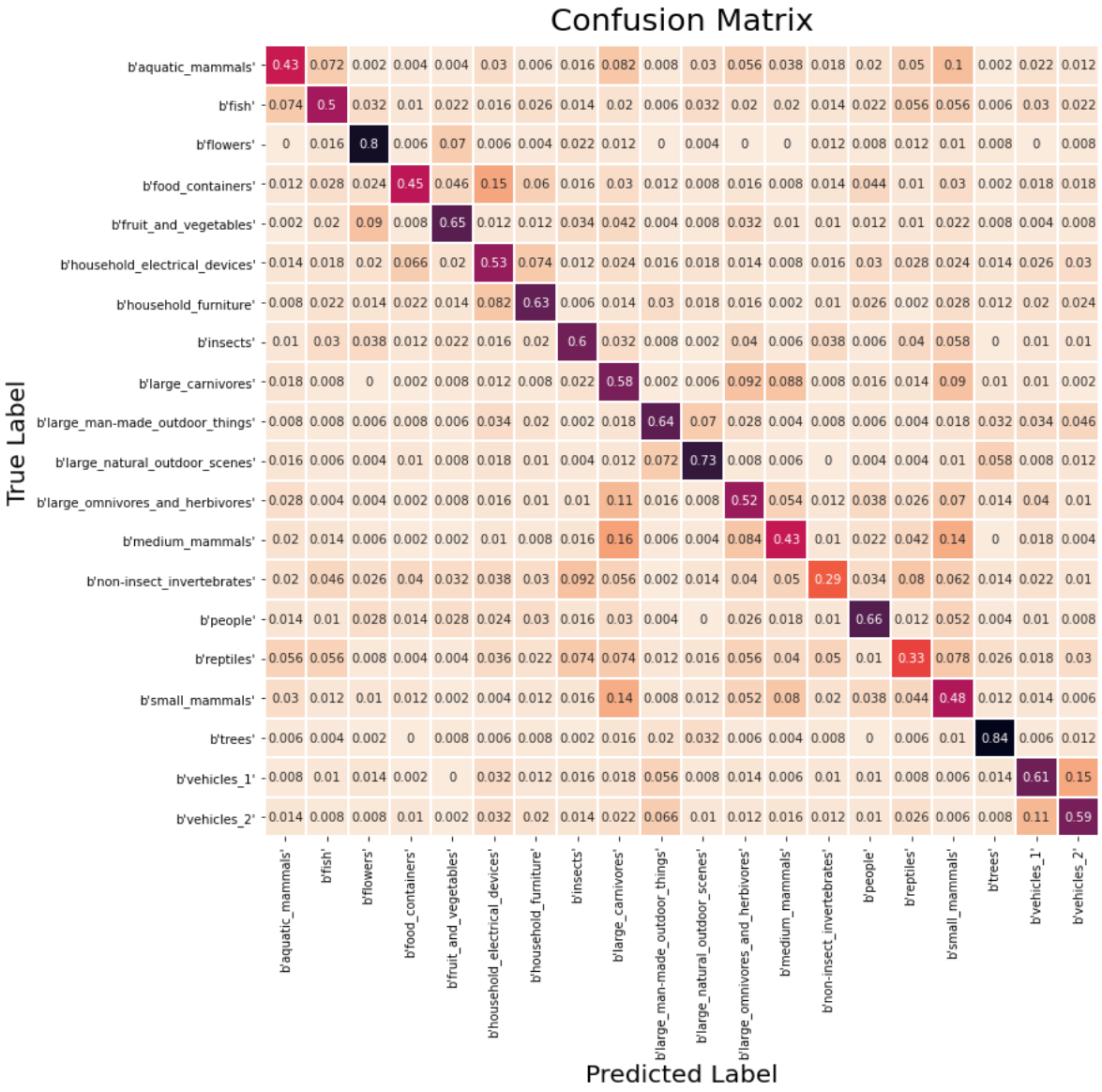
Figure 2 Results on Superclass

			Non-Augmented Colour Images			Grayscale images		Augmented, coloured	
Model	Description	Parameters	Test Loss	Test Accuracy		Test Loss	Test Accuracy	Test Loss	Test Accuracy
LR	Logistic Regression			26%					
1	self-constructed CNN	3 layers, 32/64/256 filters, 0.2 dropout	1.50	54%					
2	self-constructed CNN	3 layers, 32/64/256 filters, increased dropout	1.46	55%		1.96	41%	1.58	8%
3	self-constructed CNN	3 layers, 128/256/512 filters, 0.2 dropout	1.56	54%					
4	self-constructed CNN	3 layers, 128/256/512 filters, increased dropout	1.43	57%	BEST	1.92	44%	1.58	3%
5	research paper	10 layers, 192-300 filters each	1.89	52%					
6	research paper	9 layers, 32-512 filters each	1.78	55%					
7	ResNet	3 dense layers, 2048/1024/512	3.94	26%					
8	VGG16	3 dense layers, 512/256/128	2.47	47%					

Through confusion matrix (Figure 3), we see that many superclasses were predicted with an accuracy above 57%. 'Flowers', 'Tree' and 'Large Outdoor Natural Scene' are ones with highest accuracy, whereas 'Non-

Insects Invertebrates' and 'Reptiles' were lowest amongst all. There were confusions between 'Small Mammals', 'Medium Mammals' and 'Large Carnivores', as well as 'Aquatic Mammals' and 'Fish'.

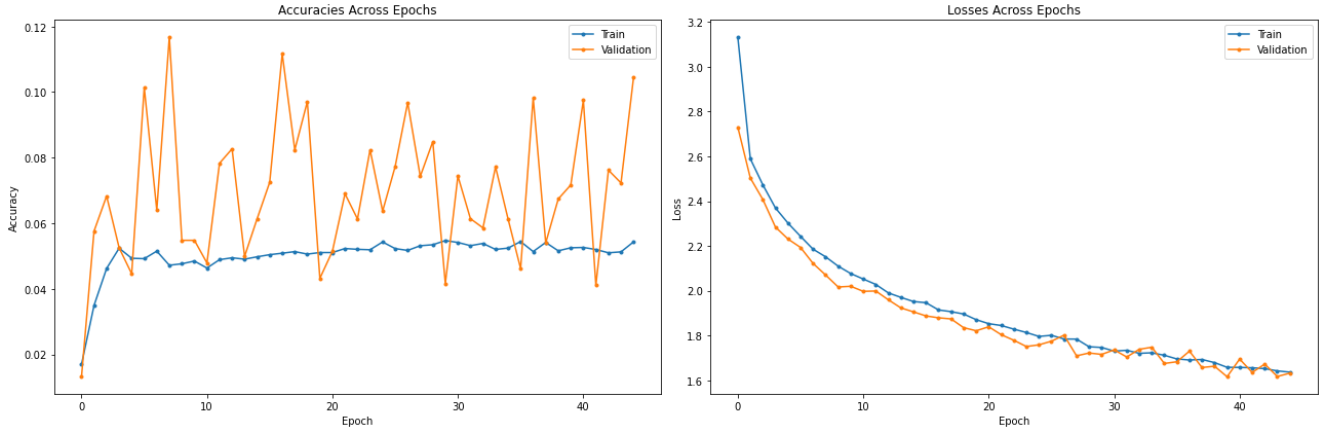
Figure 3 Confusion Matrix on Model 4 with non-augmented images (test accuracy 57%)



Step 2: Augmented Superclass

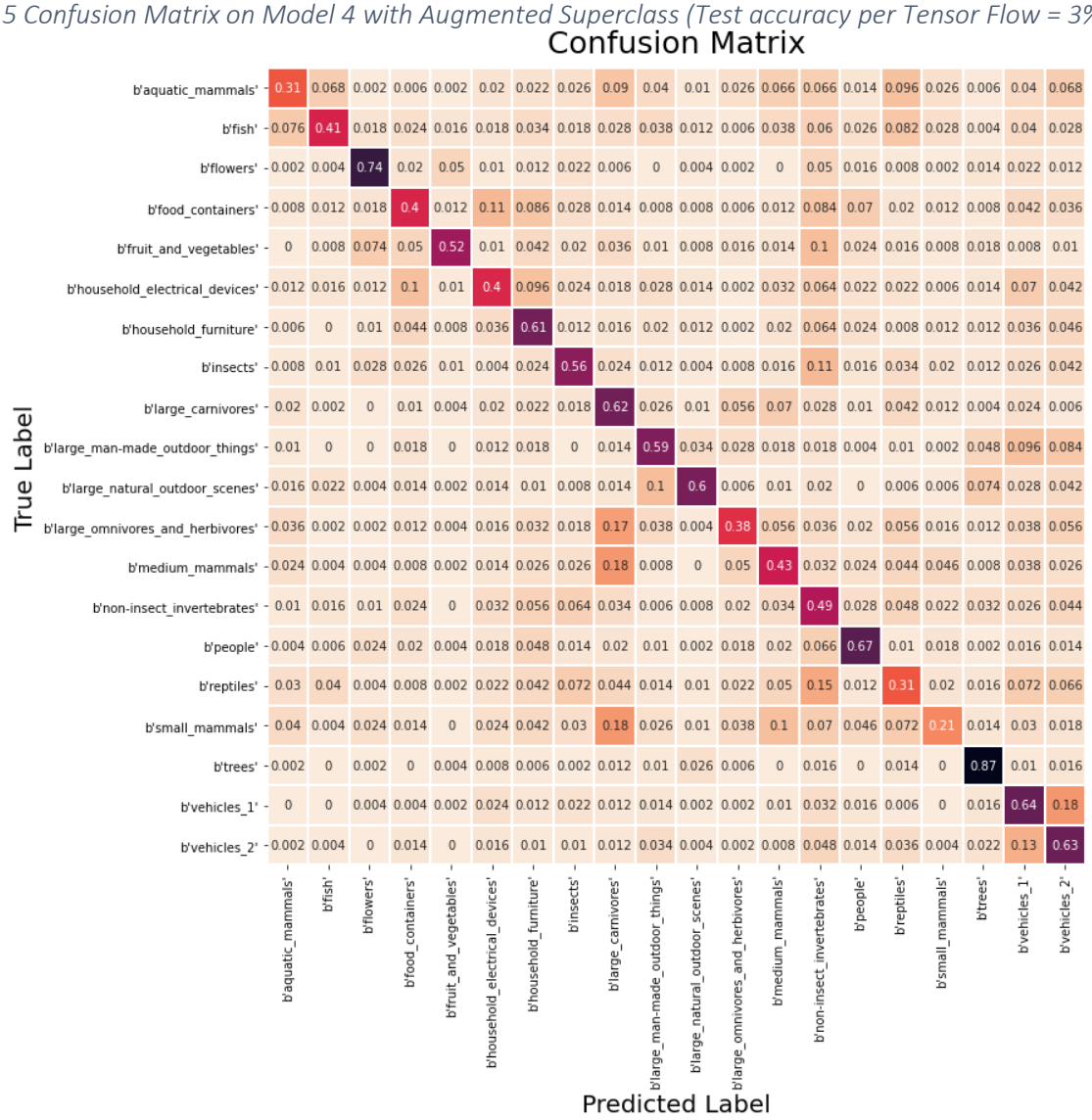
Model 2 and 4 yield an accuracy of 3-7% on the test set on augmented superclasses! The behaviour of the models was strange with the loss decreased in each epoch, but accuracy did not improve.

Figure 4 Accuracy/Loss Plot on Model 4 with Augmented Superclass (Test accuracy per TensorFlow= 3%)



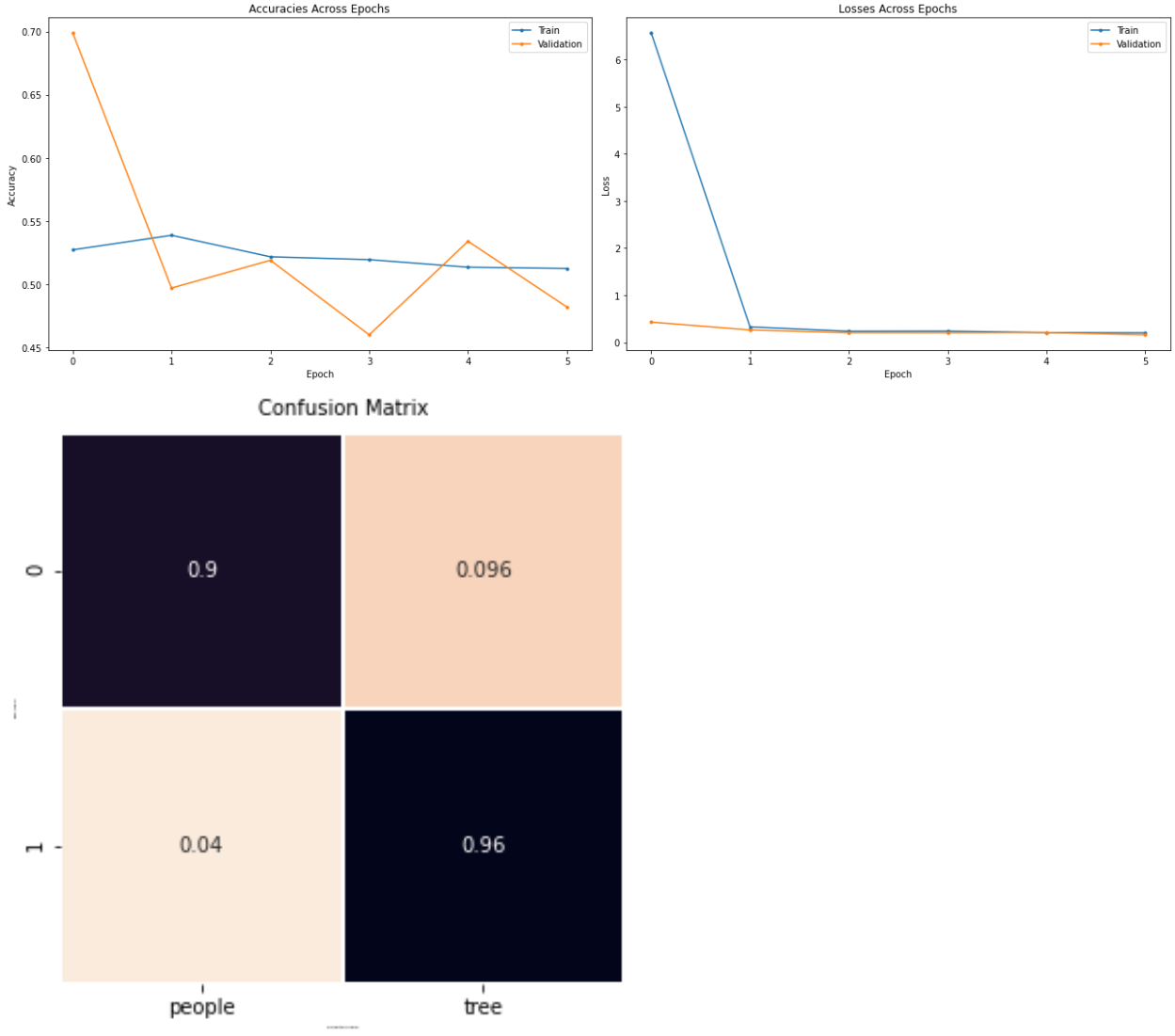
However, the confusion matrix shows good accuracy for each superclass (Figure 5).

Figure 5 Confusion Matrix on Model 4 with Augmented Superclass (Test accuracy per Tensor Flow = 3%)



To troubleshoot, I trained *Model 4* again with only 2 superclasses. Although I used the 'ImageDataGenerator' to split the test/validation set, I removed all augmentation for this trial. The model behaves as before (Figure 6), however, the confusion matrix showed an accuracy above 90% in predicting the correct superclass! The 'ImageDataGenerator' seems to have caused this. Despite the strangely low accuracy score indicated by Tensor Flow, the model is still useful as it can predict the images correctly.

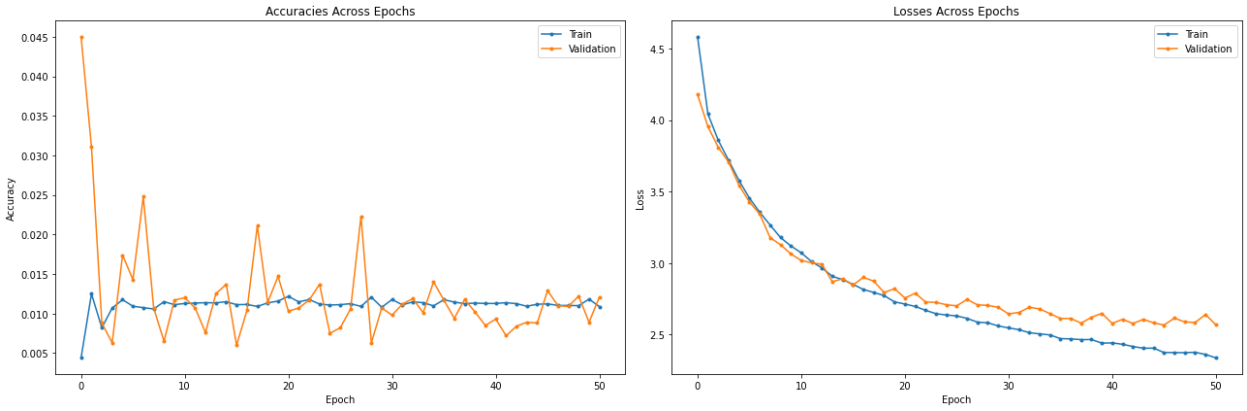
Figure 6 Accuracy/Loss Plot and Confusion Matrix on 2 Superclasses only (Non-Augmented)



Step 3: Augmented Classes

Model 4 was selected to train the 100 classes. Using the `ImageDataGenerator`, images were augmented, and the model once again behaved similarly as before (Figure 7).

Figure 7 Accuracy/Loss Plot from Model 4 with Augmented 100 Classes

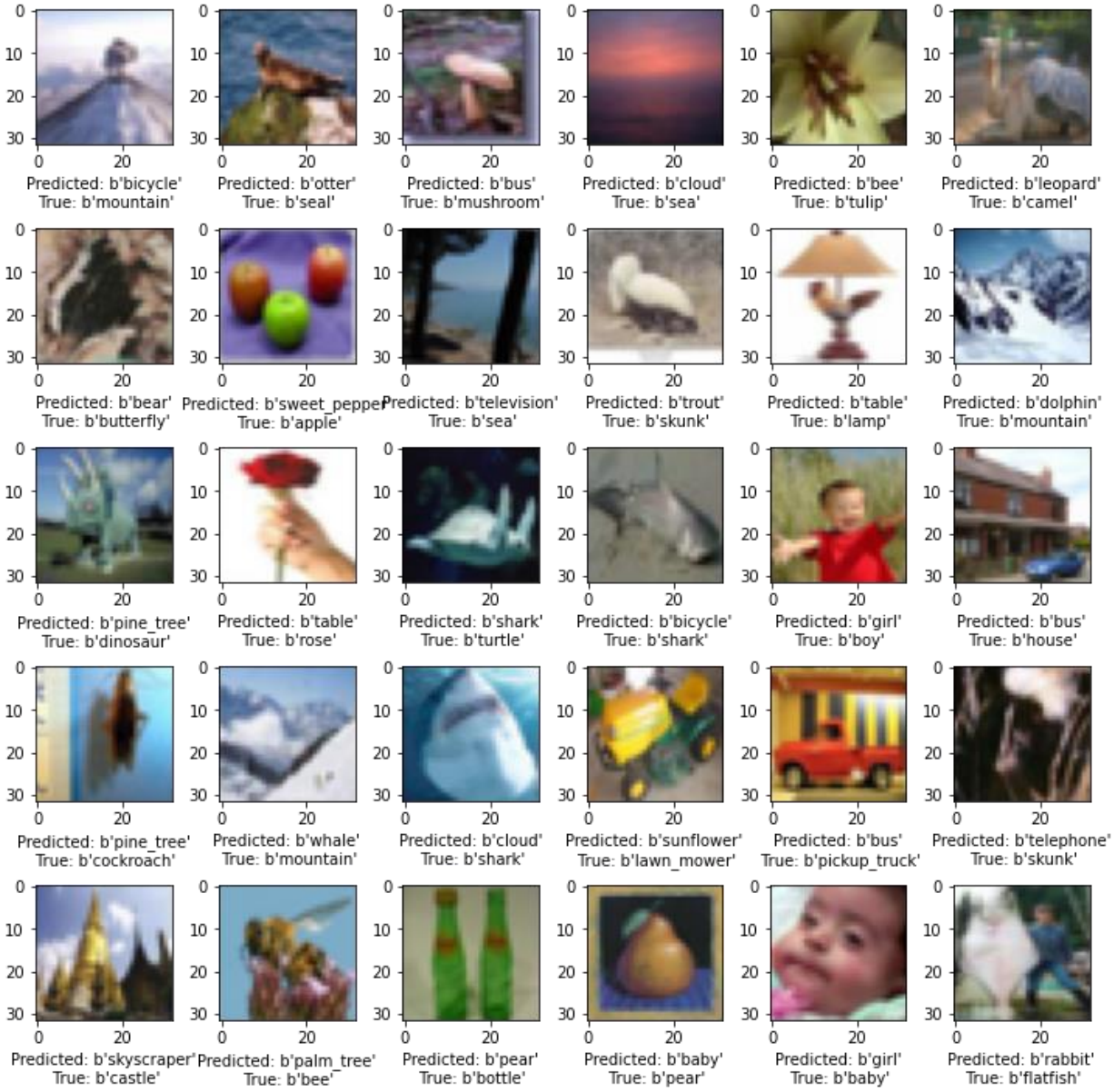


Per the Classification Report (Figure 8), the accuracy score on the test set is 39% with over 30 classes having an accuracy above 39%. (*Note that a logistic regression was done on the 100 classes, and yielded accuracy of 17% on test set.*) To understand the inaccuracy, some of the incorrect predictions were visualized (Figure 9).

Figure 8 Classification Report on Model 4 with Augmented Images by Class
Top 40 Classes only, sorted by f1-score

class	precision	recall	f1-score	support
b'sunflower'	0.764706	0.7800	0.772277	100.0000
b'plain'	0.717172	0.7100	0.713568	100.0000
b'road'	0.693069	0.7000	0.696517	100.0000
b'skyscraper'	0.702128	0.6600	0.680412	100.0000
b'skunk'	0.595420	0.7800	0.675325	100.0000
b'apple'	0.593220	0.7000	0.642202	100.0000
b'castle'	0.700000	0.5600	0.622222	100.0000
b'lawn_mower'	0.714286	0.5500	0.621469	100.0000
b'orange'	0.607843	0.6200	0.613861	100.0000
b'cockroach'	0.563025	0.6700	0.611872	100.0000
b'palm_tree'	0.525547	0.7200	0.607595	100.0000
b'chair'	0.480769	0.7500	0.585938	100.0000
b'oak_tree'	0.496454	0.7000	0.580913	100.0000
b'cloud'	0.492537	0.6600	0.564103	100.0000
b'wardrobe'	0.445783	0.7400	0.556391	100.0000
b'rocket'	0.517241	0.6000	0.555556	100.0000
b'sea'	0.532110	0.5800	0.555024	100.0000
b'chimpanzee'	0.486957	0.5600	0.520930	100.0000
b'orchid'	0.616438	0.4500	0.520231	100.0000
b'cup'	0.602740	0.4400	0.508671	100.0000
b'motorcycle'	0.394444	0.7100	0.507143	100.0000
b'poppy'	0.528090	0.4700	0.497354	100.0000
b'trout'	0.417808	0.6100	0.495935	100.0000
b'keyboard'	0.385542	0.6400	0.481203	100.0000
b'plate'	0.517647	0.4400	0.475676	100.0000
b'pickup_truck'	0.479592	0.4700	0.474747	100.0000
b'maple_tree'	0.489362	0.4600	0.474227	100.0000
b'aquarium_fish'	0.427419	0.5300	0.473214	100.0000
b'whale'	0.419355	0.5200	0.464286	100.0000
b'telephone'	0.403226	0.5000	0.446429	100.0000
b'bottle'	0.493827	0.4000	0.441989	100.0000
b'hamster'	0.452632	0.4300	0.441026	100.0000
b'dinosaur'	0.451613	0.4200	0.435233	100.0000
b'worm'	0.420561	0.4500	0.434783	100.0000
b'tank'	0.407080	0.4600	0.431925	100.0000
b'bee'	0.437500	0.4200	0.428571	100.0000
b'beetle'	0.500000	0.3600	0.418605	100.0000
b'pear'	0.514706	0.3500	0.416667	100.0000
b'rose'	0.507246	0.3500	0.414201	100.0000
b'bridge'	0.377193	0.4300	0.401869	100.0000
b'spider'	0.396040	0.4000	0.398010	100.0000
accuracy	0.393600	0.3936	0.393600	0.3936
b'television'	0.386139	0.3900	0.388060	100.0000
b'tractor'	0.395833	0.3800	0.387755	100.0000

Figure 9 Images Predicted Incorrectly by Model 4



In conclusion, the dataset has low-resolutions image and do not benefit from deep models, and this was discovered during the training of non-augmented superclasses. The approach taken have probably saved time on optimizing the CNN models with the 100 classes directly. Although the accuracy on the 100 class was only 40%, the level of accuracy makes sense once I looked at Figure 9. Some images are quite confusing, especially when it comes to `boy`, `girl` and `baby`. Even with human eyes, I might not have guessed it correctly.

Application & Next Steps

I saved and load my best CNN model for Superclass into PyCharm and I wrote codes to recognize objects through my webcam using OpenCV and translate and audibly describe the object in a specified language using GoogleTranslate and Google Text-to-Speech. I intended to classify by class but the app became laggy when predicting 100 classes, thus I used the model for superclass for now.

There are still many areas to improve, for example, improving the translation codes as the GoogleTranslate API often locks me out after multiple changes in prediction and creating an input field to allow users to easily changed the language. Moreover, the app does not perform well when the background is noisy or when there are multiple things in the background. The overall usability of the app will need more work.