Deep Learning Training on Carousell data

1. **Exploratory data analysis:**

First of all, to understand the data source, I guess it's coming from user uploaded content, their listing with corresponding uploaded image, user's choice of class and price. For user uploaded content, there is generally noise expected.
First step of data analysis, move all images to their respective class folders, and check through images and price for any misclassification or wrong labeling.
The following shows case 1: misclassifications, for these samples of laptop, their price is also quite low, suggesting they really are not meant to be laptops.

96ee8532	4e3df02c		666e6dbe
			
	8925eff1		
			

The following shows case 2: sometimes for one listing, they have several images related, and some could be related but not directly reflect the label class. This might be confusing to the model if there are not enough samples for each indirect case.

1.2GHz quad-core Intel Core i7 Intel Iris Plus Graphics 16GB 3733 MHz LPDDR4X memory 256GB SSD Storage Force Touch Trackpad Two Thunderbolt 3 Ports Touch ID Retina Display with True Tone KEYBOARD COUNTRY KIT-ITP			

Correction of the labels, by either assigning them to the right class if the class exists, or just deleting the sample from the dataset, will be quite useful. And will be part of the future jobs to be done since it will be quite time consuming.

As research shows, if data is clean, the model only needs less data to train, while for noisy data, we might need a lot more data to get the same result from training the same model. This might also explain why any model is easy to overfit when no noise is removed.

2. Modelling approach:

Images for this task are not complicated, according to pixel size distribution they are mostly around 320x320, and total training data size is around 2000+. Therefore, a CNN based neural network image classification model would be good. I did some research online for benchmarks of similar level image classification tasks, since too complex models might have a risk of overfitting, while simpler models might cause it hard to learn. And after my research I experimented with different models, InceptionV3, ResNet, as well as a simple convolution models(lenet).

As a result, ResNet perform best for evaluation metrics, with overall evaluation f1 score around 60%/(without any label adjustment). So the full training and finetuning is done based on ResNet.

3. Training:

I split the overall training dataset to 0.8 for training and 0,2 for evaluation. Since different classes have a different number of samples, I decide to split the them in a way that both training and evaluation have the same distribution of class samples. After splitting is done, I train the model and optimize model weights using training datasets, and then evaluate model performance on evaluation datasets. To avoid overfitting, I tune hyper-parameters by observing the metrics of both training and validation dataset.
First set iterations to large number, 50000 in my case.

Checking tensorboard for loss movement over time, as following:

After 30000 iterations, training loss starts to look unstable, loss reaches 0.01 and evaluation f1 score does not improve anymore. Thus I decide to end training at iteration 30000.

During hyper-parameters tuning, I experiment with different optimization method with the same learning rate. Observing training and validation accuracy result, I decide to use Adam. Then I also tried with different learning rate to determine the suitable value to be a stepwise decay learning rate with 1e-4 for the first iterations and 5e-5 for the next iterations.

Performance evaluation

The best score for Precision, recall, and f1 score for each class:

	precision	recall	f1-score	support
Laptops & Notebooks	0.68	0.65	0.67	43
Desktops	0.61	0.53	0.57	36
Cables & Adaptors	0.56	0.60	0.58	25
Chargers	0.78	0.50	0.61	14
Mouse & Mousepads	0.67	0.65	0.66	34
Monitor Screens	0.61	0.54	0.58	35
Computer Keyboard	0.71	0.81	0.76	43
Hard Disks & Thumbdrives	0.40	0.46	0.43	13
Networking Parts & Accessories	0.58	0.62	0.60	42
Webcams	0.60	0.69	0.64	13
Laptop Bags & Sleeves	0.68	0.68	0.68	38
Printers, Scanners & Copiers	0.72	0.77	0.74	30
accuracy			0.64	366
macro avg	0.63	0.63	0.63	366
weighted avg	0.64	0.64	0.64	366

The corresponding confusion matrix(class sequence same as above):

```
[[28  2  0  0  0  3  4  0  0  1  5  0]
 [ 2 19  1  0  2  7  2  0  2  1  0  0]
 [ 1  0 15  1  2  0  1  0  3  2  0  0]
 [ 0  0  4  7  2  0  0  0  1  0  0  0]
 [ 0  1  1  1 22  0  1  1  3  0  4  0]
 [ 4  6  0  0  0 19  1  1  2  1  0  1]
 [ 0  0  0  0  1  1 35  0  2  0  2  2]
 [ 1  0  0  0  2  0  0  6  0  0  0  4]
 [ 0  3  5  0  0  1  1  3 26  0  1  2]
 [ 1  0  0  0  0  0  0  3  0  9  0  0]
 [ 4  0  1  0  1  0  3  0  3  0 26  0]
 [ 0  0  0  0  1  0  1  1  3  1  0 23]]
```

Future work:

Since this assignment has a limiting time constraint, this is a basic illustration and showcase of initial steps for what can be done. In future, we definitely need to clean the current data by removing wrong labeling noises, and at the same time collect more data if it’s possible. More data augmentation techniques and hyper-parameters tuning can be an option to further improve model performance.