Motivation

Shopee is an e-commerce brand with a lowest-price guarantee program and is looking to improve product categorization and detect marketplace spam through image and text recognition. This project focuses on the image recognition side of product matching. The goal is to build a model that predicts and categorizes images into their designated label groups. To create image classification models to detect label grouping PyTorch is used to create and initiate the neural network.

Data

Data was sourced from a Kaggle competition:

Shopee Price Match Guarantee
https://www.kaggle.com/competitions/shopee-product-matching/overview/description

The data comprised of two image folders, train and test, and two csv files corresponding to each folder. The **train dataset** contains **32412 unique images**, with 5 columns: posting id, image, image phash, title, and label group. The test dataset contains 3 images with the same columns excluding label group, as it is the target prediction.

The number of unique entries per a column in the train data frame are:
posting_id: 34250
Image: 32412
image_phash: 28735
Title: 33117
label_group: 11014

Positing id is a unique identifier in the dataset. Image represents the image id/name associated to the image folder. Image phash is a perceptual hash of the image. Title is the product description for the posting. Label group is the ID code for all products that map to the same product.

Import Data
Initial work and data exploration were completed using Jupyter notebooks.
Majority of the project was using Google Colab. To import the data:

1. Store the dataset and images in google drive
2. Mount Google Drive to the Colab Project
3. Import the images with pathlib.
4. Import labels and assign labels with pandas.
5. Use PyTorch to transform and resize images and convert data into tensors

Exploratory Data Analysis (EDA)
Given the large number of images and unique label groups, only the train dataset was used and split into test, train, and validation sets during the 'Prepare Data for Modeling' section.
In the train dataset, image shape was consistent across images at (1024,1024,3) in RGB.
The EDA consisted of exploring each column in relationship to other columns and images.

Label Group
There are 11,014 unique label groups. The largest label group contained 51 images. 7 unique label groups had the maximum of 51 images. The majority of label groups, 63%, contained 2 images. When visualizing a max label group, it contained duplicate and similar images with various image automations and different images but with the same product. Example:

Identical Images
There are 1246 duplicated images with identical image path. The max number of identical images is 2. Posting id are always different in duplicated images, but some also have alternated titles, indicating possible editing involved. Image id and image phash are unique throughout identical image groups

Identical Image Phash
There were multiple groups with same image phash with the max group consisted of 26 images. Image id and title are different in identical image phash groups. When visualizing an identical image phash group, all images were indeed duplicates.

Identical Title
The max number of images with the same title is 9. When visualizing the max identical title group, it showed different images with the same title. Therefore, indicating that title is not a good measure to match images.

Data Cleaning
As the target prediction is label group, the size of 11,014 unique classes in label group is of concern as it will take too long for google colab pro to process all the images and classes in the neural network. This concern is compounded by the majority of label groups is comprised of 2 images. Therefore, in order to build a model for image classification on PyTorch the train image dataset was reduced to the top 21 label groups. The reduced dataset comprised of 899 images, of which 72 were duplicates and were not included as their image and paths were identical, thus leaving a total of 827 images to train, validate, and split.

Prepare Data for Modeling
With my data mounted to the colab notebook, the first step was to create a list of image paths to the image folder. Afterwards a data frame containing just the image name and label groups was created. This was done to map each label group (class) to an associated index . The map is used to generate predictions later with the associated label group.

The first function that was made was to initialize and generate a PyTorch **dataset** and to feed it into a data loader. Specifically this functions calls an image with its associated label group and converts to the image to PIL, which is needed for tensor data transformations. The image is saved as the input transformed as a tensor. Next the associated label group is saved as the output and converted into a tensor. Both the input and the output are sent to the device, which in this case is GPU using Colab Pro. Furthermore, in this section, it is possible to toggle image augmentation, which would randomly arguments an image with a 50% change in either a random grayscale, horizontal or vertical flip.

The Second function calls the dataset defined above and splits the data into train (80%), validation (10%), and test sets (10%). Next, the split datasets are loaded using PyTorch's Dataloader, with batch size made easily adjustable.

Model Fundamentals
3 models were used in this project to deploy the neural network.

1. Convolution Neural Network
    - 3 blocks of 2d convolution, ReLU function, and Max pool 2d layer
    - 1 flatten layer
    - 2 Linear dense layers
2. Efficient net b0 without weights
    - Efficient net is based on a CNN
    - This model adds more layers and complexity to the model that aim to increase prediction of labels
3. Transfer Learning: Efficient net b0 with pre-trained weights

Every model is compiled with:
- Criterion = cross entropy loss

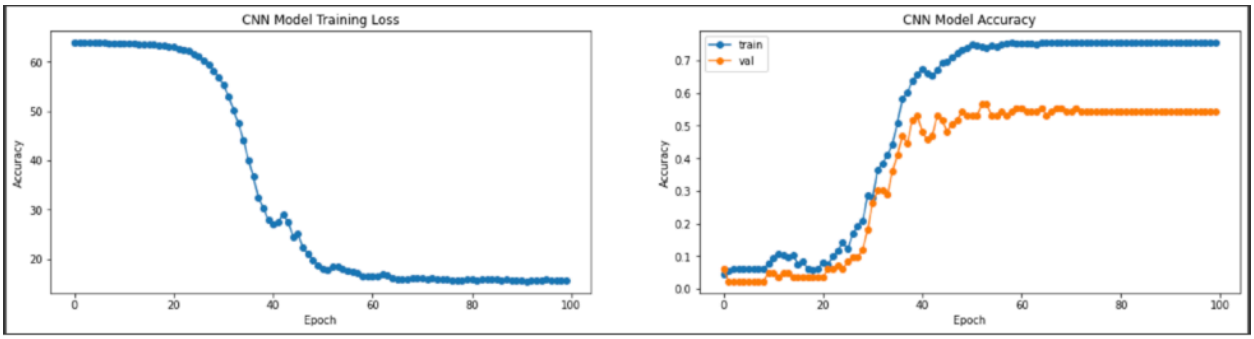
- Optimizer = stochastic gradient descent
- metrics = accuracy

Number of epochs varies based on the model. With each epoch, the model is minimizing loss and maximizing accuracy. More epochs take more processing time.

Model Comparison

After testing, each model is judged by its training and validation accuracy and ultimately the model's prediction accuracy in categorizing label groups using image recognition.

Personally built CNN model
Train accuracy = 80%, Validation accuracy = 60%, Final Prediction accuracy = 27%.



An additional layer block of Convolution did not improve the model.

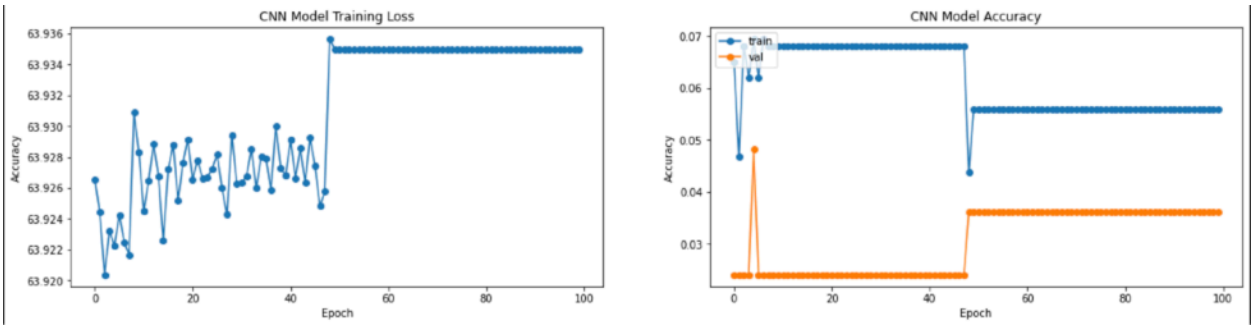
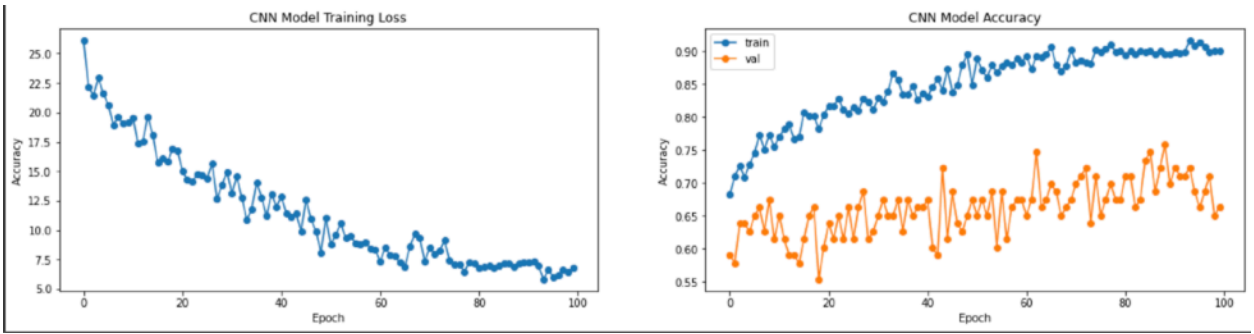
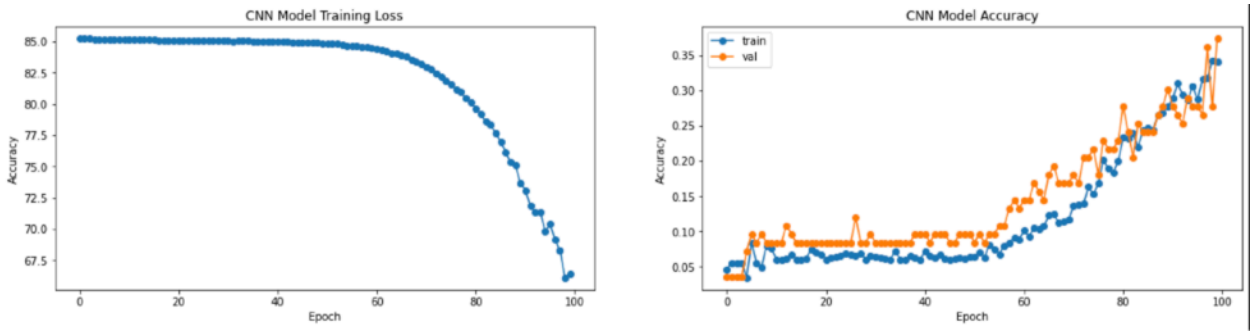


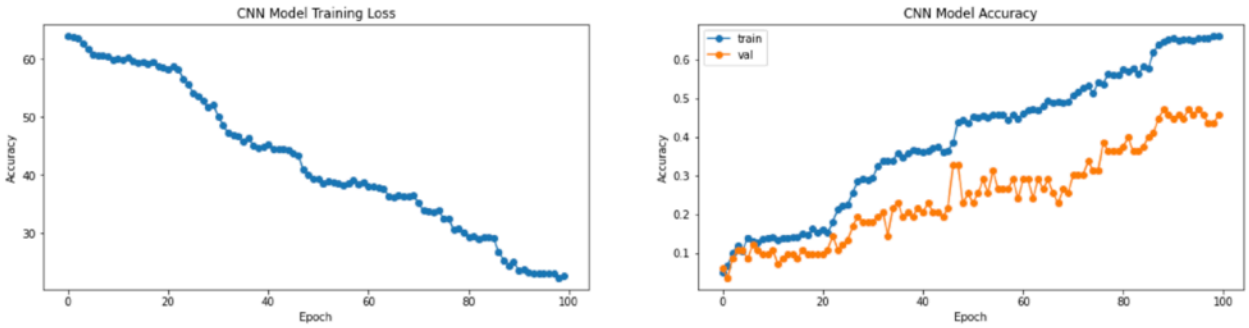
Image Augmentation
Increased train accuracy = 90%, but decreased validation accuracy = 65%, Final prediction accuracy = 30%. The increase in the difference between train and validation accuracy indicates overfitting.



Adding a weight decay = 0.9
This was done to decrease overfitting in the model, which was successful but drastically decreased the train and validation accuracy.

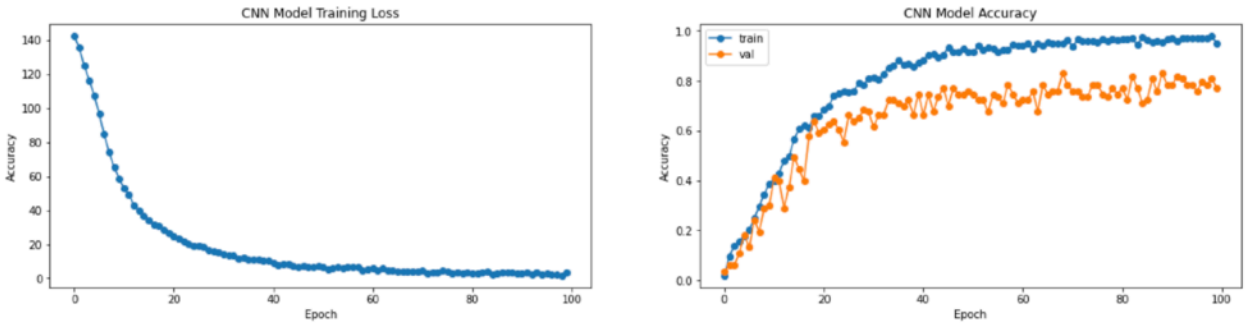


Decreasing the weight decay=0.001, improved train and validation accuracy.
However, loss with the current model has remained high at 30 and Final tests prediction remained low at below 30%.



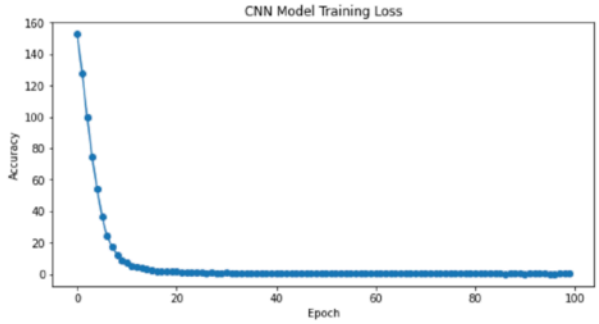
EfficientNet-0 Model

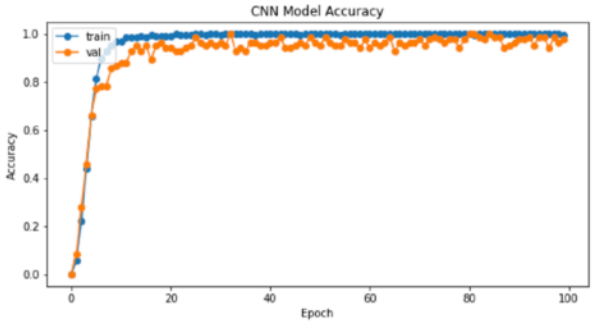
Loading EfficientNet-0 model improved training and validation accuracy, with a final test prediction = 31.71%



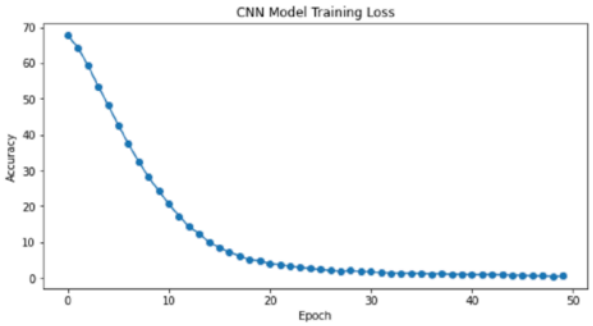
EfficientNet-0 Pre-trained weights

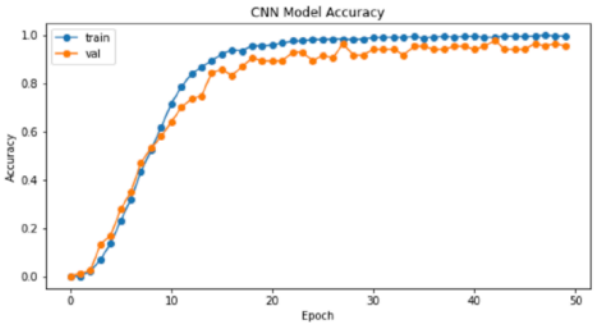
Loading the pre-trained weights was aimed to increasing the final testing prediction
Accuracy increased towards 100% for the train and validation.
Final testing prediction increased to 37.8%





After adjusting batch size and other parameters of learning rate and weight decay. The biggest factor in increasing the final learning rate with the pre-trained model was to increase the batch size to 75.





As shown above the model maintained a train and validation accuracy above 90% and reached a final prediction accuracy of 89.4%. This is to say that the model is able to predict the category 89% of the images to the proper label group.

Further improvements

The next steps to improve the model is to use more techniques of transfer learning, such as limiting or blocking some of the transformations made by efficient net. Another step is to try other models of transfer learning, such as ResNet. Lastly, a major improvement for this use case of image categorization is to use another package on top of PyTorch, such as Fast ai.