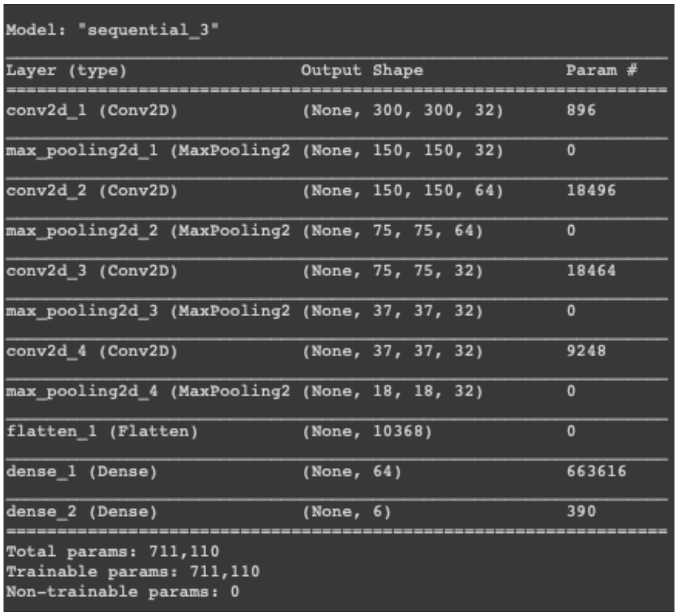
Mais 202- Project Deliverable 3

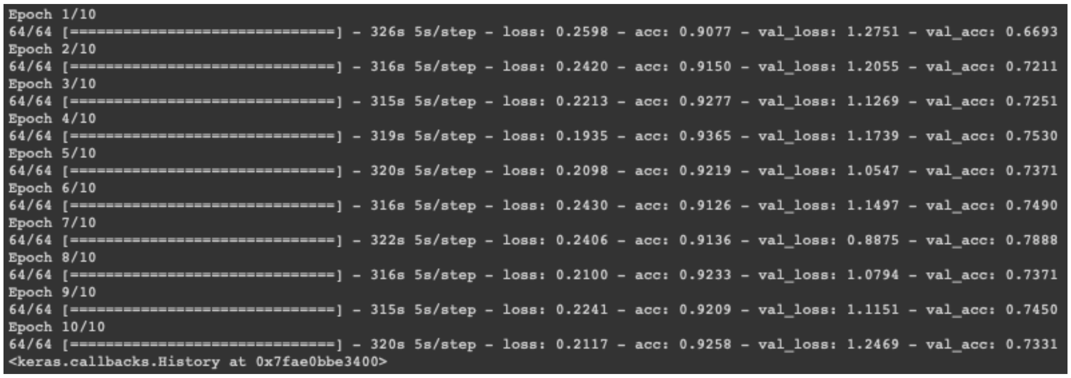
1. Final Training Results

In this project, various models were trained. Keras was used to trained the model with 1 CNN layer to 4 CNN layers. However, It was underfitting, and the validation accuracy was not high enough.

- Model with 4 cnn layers using Keras



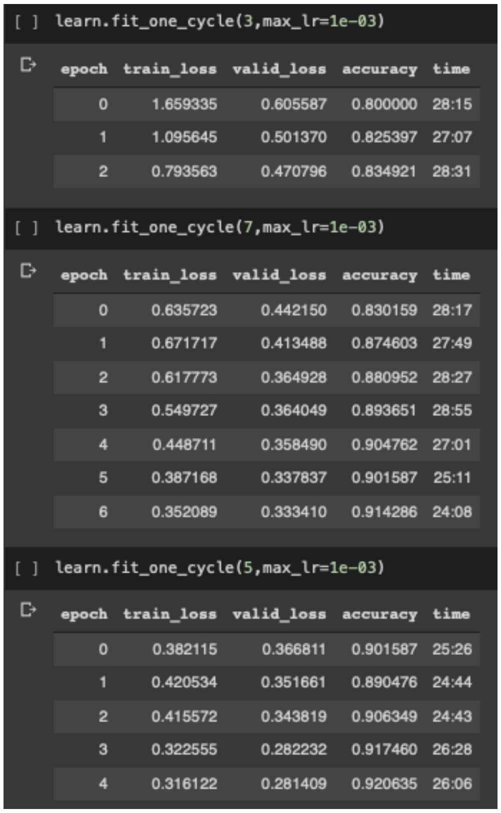
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 300, 300, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 150, 150, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 150, 150, 64)      18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 75, 75, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 75, 75, 32)        18464
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 37, 37, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 37, 37, 32)        9248
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 10368)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                663616
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 390
=================================================================
Total params: 711,110
Trainable params: 711,110
Non-trainable params: 0
```



```
Epoch 1/10
64/64 [==============================] - 326s 5s/step - loss: 0.2598 - acc: 0.9077 - val_loss: 1.2751 - val_acc: 0.6693
Epoch 2/10
64/64 [==============================] - 316s 5s/step - loss: 0.2420 - acc: 0.9150 - val_loss: 1.2055 - val_acc: 0.7211
Epoch 3/10
64/64 [==============================] - 315s 5s/step - loss: 0.2213 - acc: 0.9277 - val_loss: 1.1269 - val_acc: 0.7251
Epoch 4/10
64/64 [==============================] - 319s 5s/step - loss: 0.1935 - acc: 0.9365 - val_loss: 1.1739 - val_acc: 0.7530
Epoch 5/10
64/64 [==============================] - 320s 5s/step - loss: 0.2098 - acc: 0.9219 - val_loss: 1.0547 - val_acc: 0.7371
Epoch 6/10
64/64 [==============================] - 316s 5s/step - loss: 0.2430 - acc: 0.9126 - val_loss: 1.1497 - val_acc: 0.7490
Epoch 7/10
64/64 [==============================] - 322s 5s/step - loss: 0.2406 - acc: 0.9136 - val_loss: 0.8875 - val_acc: 0.7888
Epoch 8/10
64/64 [==============================] - 316s 5s/step - loss: 0.2100 - acc: 0.9233 - val_loss: 1.0794 - val_acc: 0.7371
Epoch 9/10
64/64 [==============================] - 315s 5s/step - loss: 0.2241 - acc: 0.9209 - val_loss: 1.1151 - val_acc: 0.7450
Epoch 10/10
64/64 [==============================] - 320s 5s/step - loss: 0.2117 - acc: 0.9258 - val_loss: 1.2469 - val_acc: 0.7331
<keras.callbacks.History at 0x7fae0bbe3400>
```

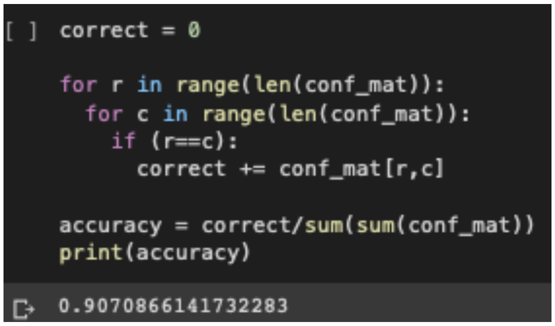
- Low validation accuracy compared to training accuracy.

Therefore, pre-trained model, resnet34, was with fastai pytorch. With 15 epochs, I managed to get training accuracy around 92% and test accuracy around 91%.



```
[ ] learn.fit_one_cycle(3,max_lr=1e-03)

    epoch  train_loss  valid_loss  accuracy  time
        0    1.659335    0.605587  0.800000  28:15
        1    1.095645    0.501370  0.825397  27:07
        2    0.793563    0.470796  0.834921  28:31

[ ] learn.fit_one_cycle(7,max_lr=1e-03)

    epoch  train_loss  valid_loss  accuracy  time
        0    0.635723    0.442150  0.830159  28:17
        1    0.671717    0.413488  0.874603  27:49
        2    0.617773    0.364928  0.880952  28:27
        3    0.549727    0.364049  0.893651  28:55
        4    0.448711    0.358490  0.904762  27:01
        5    0.387168    0.337837  0.901587  25:11
        6    0.352089    0.333410  0.914286  24:08

[ ] learn.fit_one_cycle(5,max_lr=1e-03)

    epoch  train_loss  valid_loss  accuracy  time
        0    0.382115    0.366811  0.901587  25:26
        1    0.420534    0.351661  0.890476  24:44
        2    0.415572    0.343819  0.906349  24:43
        3    0.322555    0.282232  0.917460  26:28
        4    0.316122    0.281409  0.920635  26:06
```

- High training accuracy and lower validation loss

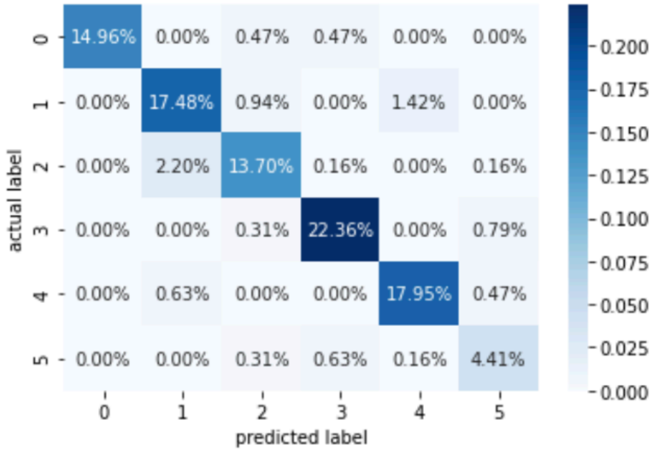


```
[ ] correct = 0

    for r in range(len(conf_mat)):
      for c in range(len(conf_mat)):
        if (r==c):
          correct += conf_mat[r,c]

    accuracy = correct/sum(sum(conf_mat))
    print(accuracy)

    0.9070866141732283
```

- Test accuracy

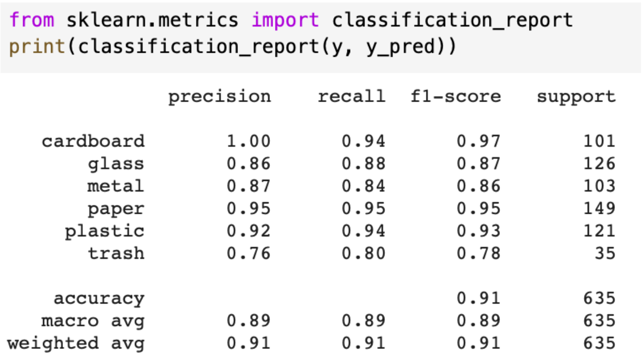
I have not changed much from previous model. I tried different models, and figured pre-trained model is the most powerful one, and with more training, I could get accuracy over 90%.

- Confusion matrix



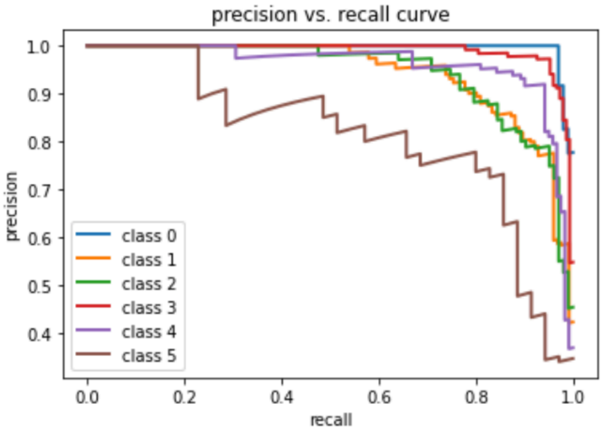
actual label \ predicted label	0	1	2	3	4	5
0	14.96%	0.00%	0.47%	0.47%	0.00%	0.00%
1	0.00%	17.48%	0.94%	0.00%	1.42%	0.00%
2	0.00%	2.20%	13.70%	0.16%	0.00%	0.16%
3	0.00%	0.00%	0.31%	22.36%	0.00%	0.79%
4	0.00%	0.63%	0.00%	0.00%	17.95%	0.47%
5	0.00%	0.00%	0.31%	0.63%	0.16%	4.41%

- Precision-recall



```
from sklearn.metrics import classification_report
print(classification_report(y, y_pred))
```

```
              precision    recall  f1-score   support

   cardboard       1.00      0.94      0.97       101
       glass       0.86      0.88      0.87       126
       metal       0.87      0.84      0.86       103
       paper       0.95      0.95      0.95       149
     plastic       0.92      0.94      0.93       121
       trash       0.76      0.80      0.78        35

    accuracy                           0.91       635
   macro avg       0.89      0.89      0.89       635
weighted avg       0.91      0.91      0.91       635
```

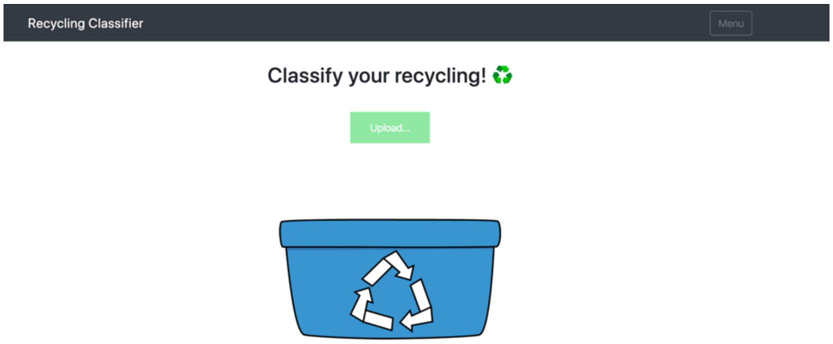


Classes go in the order of cardboard(0), glass(1), metal(2), paper(3), plastic(4), trash(0) in the precision-recall curve.

2. Final Demonstration Proposal

- Application

The application will be in the form of web-app and Flask will be used for the framework, and it will be deployed on Heroku or Render since they are two easy and free ways to deploy web application.
I have a little experience with flask, but to deploy web application, I will need to watch tutorials or look at online documents since there is no prior experience.
The application could be using real-time web-cam to predict the object with the model or make user upload image and then classify it. The latter choice would be easier to implement. However, the first option would be more practical in real world situation where a machine could classify objects quickly.



Classify your recycling! ♻

- Example for web-app