Gaurav Kumar (u2151556)
Coursework Report

Task-1 :

Approach - All the cards are component labelled using ***sklearn.measure.label*** on the binarised training images with the threshold. Cards are being extracted using the bounding box slicing on the component labelled images. Extracted cards are then manually labelled with their Ids, and saved in the results directory in the following format - "*/results_dir/train/Id/image.jpg*"

Challenges & Solutions - first challenge was to find an appropriate threshold for binarising the training images. Histogram distribution of the training images is studied to come up with a common threshold, i.e. **248** which can be seen from the below distribution in (fig-1-a).
This was not enough, because many training images have more than one card in some of their labelled components(fig-1-b). Cards in those components were separated by further reducing the threshold value to **205**. But, thresholding the entire image on this value resulted in losing some cards while component labelling(fig-1.c). So we applied a recursive method to further threshold only those components, leveraging the average number of pixels of a card in training images(*varies between 250000 to 350000*).

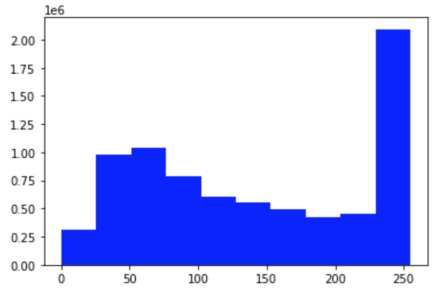

fig-1-a

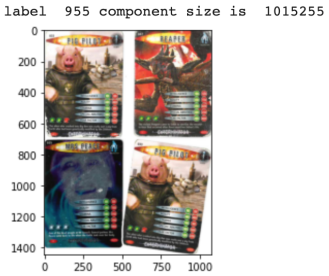

fig-1-b

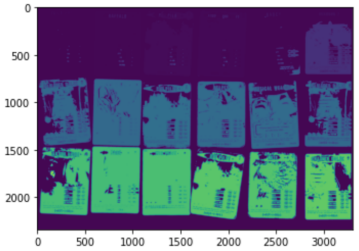

fig-1-c

Another challenge was the runtime of the code, it was taking 25-30 mins to extract all the cards from all the training images because it was iterating over all the noisy components. To reduce it, we first dropped all the noisy labels based on their number of pixels using the **Counter** method on the flattened component image, from the unique list of labels. The extraction of cards becomes 10 times faster.

Task-2 :

Approach - All the cards are extracted from the "train-001" image using the same approach in *Task-1*. SIFT features with FLANN based matcher (with *KDtree algorithm)* is implemented to find nearest neighbours. This approach resulted in 100% accuracy for all the cards which are known in train-001.

Challenges & Solutions - the only challenge was to find a way to score the match between any two images using the above approach.

Lowe's keypoint logic is used to deduce a score in order to find k-nearest cards from the training card set(extracted in task-1) for a given card in train-001.
Where **score** is defined to be the ratio of total Lowe's key points in an image divided by total keypoints. And the k-closest cards are selected iteratively based on the score between each training image and a given card. A threshold is defined for the scores(***greater than 0.2***) to segregate known cards separately post studying the scores of the k-nearest cards matches for all the cards, and finally these known cards are saved as validation dataset in the same format as train images("*/results_dir/val/ Id/image.jpg*"). Fig-2 shows the top 3-NN(nearest neighbours) of a known and an unknown card with their scores, where the known card has a score value greater than 0.2. Although the matches for unknown cards are showing some sort of similarity, for example the yellow-black stripe bar is common in all the matches in Fig-2, and since these keypoints will be a very less fraction of the total keypoints, that is why their similarity score is low which is visible in the fig-2.

fig-2

Task-3 :

Approach -

1. *Data preparation* -
    a. **Data extraction from Test images** - All the cards are component labelled using the same approach as *task-1&2*, but with an efficient binarization technique to make the components more precise and accurate(fig-3-b). Given the background is green, we have applied a conditional masking between channels of a *BGR* image by studying the distribution of the channels separately, rather than thresholding the entire pixel of an image. Condition is defined as, the intensity of the green channel should be less than value **50**(can be seen from fig-3-a) or red & blue channel's intensities should be greater than the green channel's intensity at any pixel.

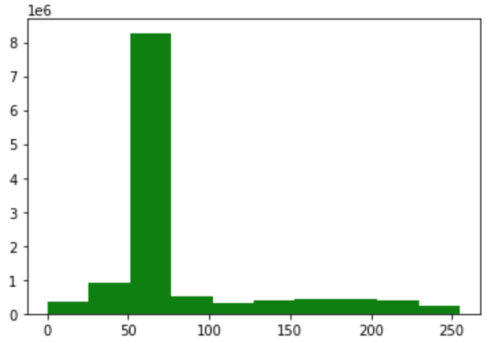

fig-3-a

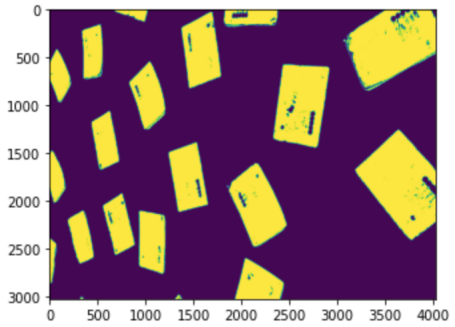

fig-3-b

Then **perspective transformation** is applied on all the cards to wrangle the images into the required shape. Minimum area rectangle (*cv2.minarearect*) is employed to find source and destination corners of a component. Finding the minimum area rectangle was not accurate on the actual image, so we applied it on the contours of the binarised component labelled images.

All the extracted test cards are then labelled manually, with the unknown ones as Id - “000”, and stored in the same format as training images “*/results_dir/test/Id/image.jpg*”

b. **Train-test-val split** - It is clear that the above created train-val-test data cannot be used for modelling because training cards are significantly well defined and oriented with respect to test/validation datasets, so we come up with a better way by shuffling and redistributing the whole data, with following condition.
    1) All the Ids containing just one image are kept to training dataset
    2) And all the other Ids are splitted in 80-10-10 ratio for train-val-test respectively, giving priority to the validation dataset in cases where 80-10-10 split is not feasible for the count of images in an Id.
    3) The validation and test are further mixed to shuffle and resplit into a 50-50 ratio.
c. **Train augmentation** - As after the above splitting approach, the train dataset is observed to have high class imbalance, where many Ids just have one image in it. So we replicated the images within each Id to keep the count uniform across all Ids.
d. Finally the re-splitted datasets are converted to dataloader format for model training and evaluation

2. *Model development and results* - Transfer learning is used to train the model because of the small data size and rich features of the images. We have experimented with four different pretrained models, keeping learning rate = 0.01 and SGD optimizer for all the experiments. All the models are trained for 30 epochs only, due to fast convergence. From the experiment, with different architectures, RESNET50 came out to be the best one as shown in “table-1-a”. Further, to fine tune the model parameters and reduce overfitting

we have trained the model with various data augmentation techniques and added variability of unfreezing the weights of the last two layers of the RESNET50 network. From the results of fine tuning, as shown in table-1-b, the model without data augmentation and with unfrozen last two layers stood to show the best performance. Refer to the fig-4-a and fig-4-b for the accuracy and loss vs epoch charts of the best fit model.

	Accuracy		
	training	validation	test
VGG16	99.78	83.33	80.00
Resnet18	99.85	88.89	88.57
Resnet34	99.85	80.56	85.71
Resnet50	99.85	94.44	85.71

table-1-a

Resnet50 with Data Augmentation		Accuracy		
		training	validation	test
rotation	Freeze	99.33	91.67	88.57
	Unfreeze	99.26	91.67	85.71
Perspective & rotation	Freeze	96.30	72.22	77.14
	Unfreeze	96.23	77.78	74.29
no augmentation	Freeze	99.85	94.44	85.71
	Unfreeze	99.85	94.44	88.57

table-1-b

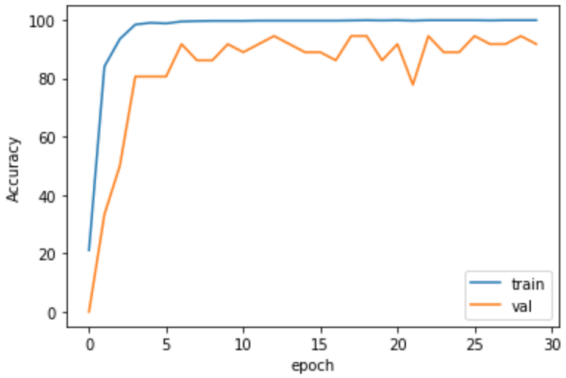

fig-4-a

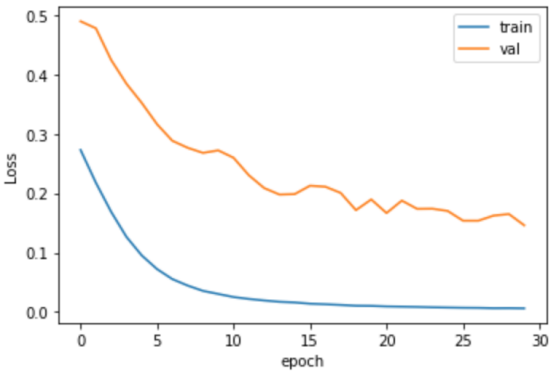

fig-4-b

Task-4 :

Approach - All the cards are extracted using the same approach as in section “Data extraction from Test images”. The best fit trained model from task-3 is used to label all the cards in selected frames. The selected frames are used to manually evaluate the model’s performance. The overall accuracy manually calculated on the selected frames is around 60% excluding the unknown cards. Model seems to perform better for the cards which are extracted from the test images (~80%) due to similar transformational variation, compared to the cards which are only extracted from train-xxx images. Below are some labelled frames from vid-002.