Clab-2 Report

ENGN6528

Han Zhang

U6541559

29/04/2022

Task 1 Harris Corner Detector. (5 marks)

```
######################################################################
# Task: Compute the Harris Cornerness
######################################################################

def get_Harris_Cornerness(kernel_size,Ix2,Iy2,Ixy,k):
    ''' Compute the Harris Cornerness using the determinnet formula, and eignvalues
    '''
    kernel = np.ones((kernel_size,kernel_size))# weight vector of 2-d array with all ones for computing the sum later
    lambda_1 = conv2(Ix2, kernel) # Compute the sum of the IX2 in the neighbourhood area (localized 1st eigenvalue)
    lambda_2 = conv2(Iy2, kernel) # Compute the sum of the Iy2 in the neighbourhood area (localized 2nd eigenvalue)
    squared_sum_Ixy =  np.square(conv2(Ixy, kernel)) # anti-diagnal product for calculation of determinent later

    det = lambda_1 * lambda_2 - squared_sum_Ixy # determinent of matrix M
    trace = lambda_1 + lambda_2 # trace of matrix M
    R = det - (k * trace**2) # Harris Cornerness
    return R

######################################################################
# Task: Perform non-maximum suppression and
#       thresholding, return the N corner points
#       as an Nx2 matrix of x and y coordinates
######################################################################
def non_maximum_suppression(R):
    ''' Given the R value of the location in the picture and obtain the location of picture which is 8-way local maxima
        and also passes the trheshold
    '''
    corner_list = []
    # get the maximum R value and compute the thresh propotion of the maximum R,get real threshold the R value need to reach
    threshold = R.flatten().max()*thresh
    row,col = R.shape
    for i in range(1, row-1):
        for j in range(1, col-1):
            # Select corners that are 8-way local maxima
            if ( (R[i,j] >=R[i-1,j]) & (R[i,j] >= R[i,j+1]) &  (R[i,j] >= R[i+1,j]) &(R[i,j] >= R[i+1,j+1]) & (R[i,j] >= R[i-1,j-1]) &
                (R[i,j] >= R[i-1,j+1]) & (R[i,j] >= R[i,j-1]) &(R[i,j] >= R[i-1,j-1])):
                    # Double check the corner has passed the threshold
                    if R[i,j] >= threshold:
                        corner_list.append([i, j]) # record the selceted corner
    return corner_list
```

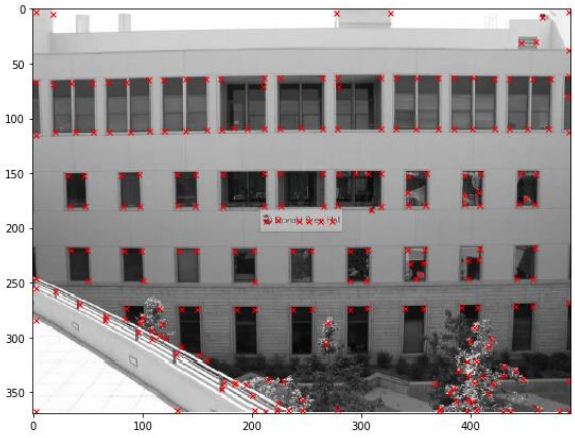
Fig 1. Screenshot of the code of missing part.

```
# Set up a gaussian kernel with shape of no less than 1*1, the shape is also dependent on the sigma size
g = fspecial((max(1, np.floor(3 * sigma) * 2 + 1), max(1, np.floor(3 * sigma) * 2 + 1)), sigma)
```

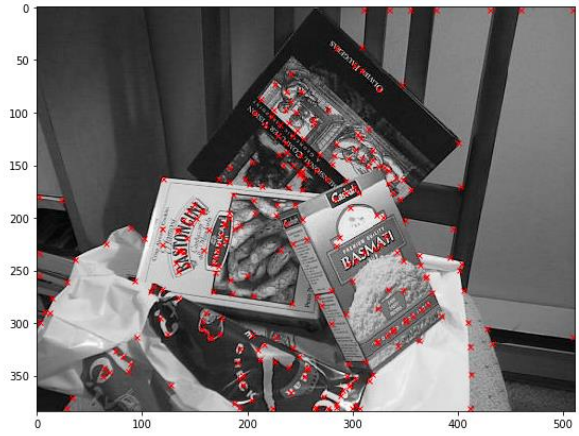
Fig 2. Comment on block 5

The comment of the solution (missing part) is shown in the previous part of screenshot and is also available in the script.

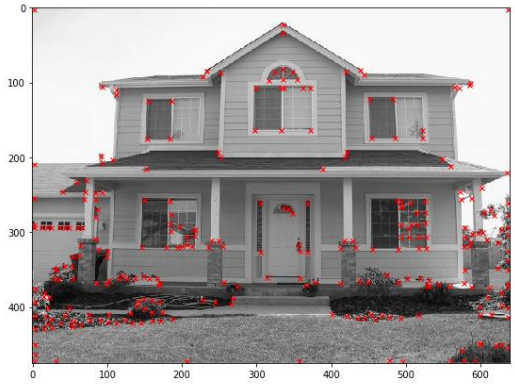
Computational result of Harris corner detection of ‘Harris_1.jpg’.



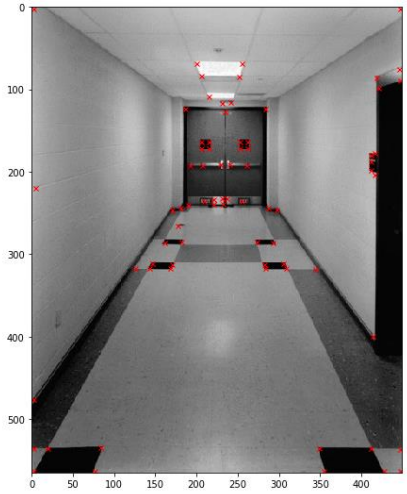
Computational result of Harris corner detection of ‘Harris_2.jpg’



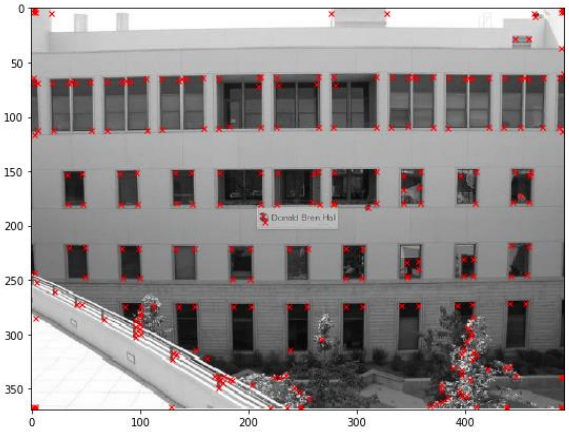
Computational result of Harris corner detection of ‘Harris_3.jpg’.



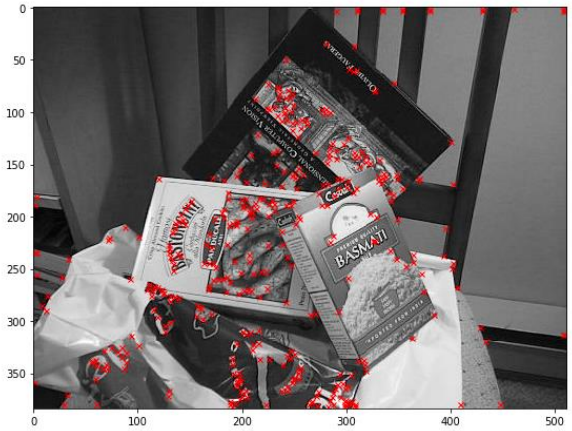
Computational result of Harris corner detection of ‘Harris_4.jpg’



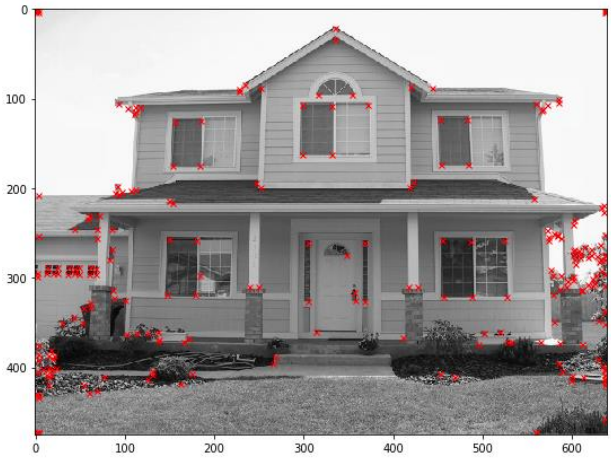
Built-in result of Harris corner detection of ‘Harris_1.jpg’.



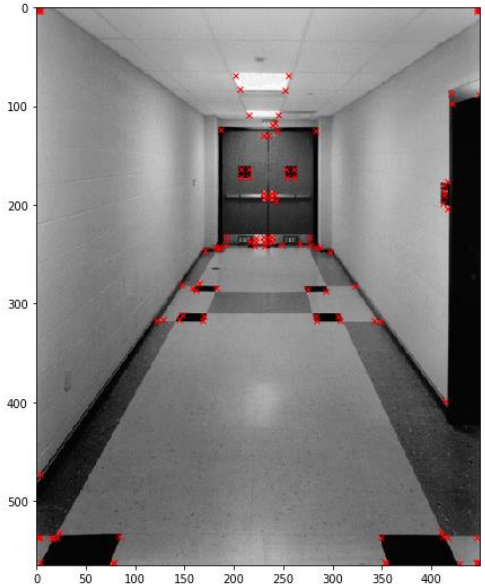
Built-in result of Harris corner detection of ‘Harris_2.jpg’



Built-in result of Harris corner detection of 'Harris_3.jpg'.



Built-in result of Harris corner detection of 'Harris_4.jpg'.



Compared with built-in result, my result might identify a bit less corners. For example. In the enlarged pictures of the picture 2 shown below. There are a bit more nested corners counted in the logo of the clothes. While my result only counts one instance of the corner in the neighborhood area. If several corners nested in very closed local area, all of the others will be omitted. I believe it is related to the window (kernel) size when calculating the two eigenvalues of the M matrix. In my calculation, I set the kernel to 5, which corresponding a 5 * 5 localize window to calculate the eigenvalue, trace, and also determinant. It must be a factor that affect the strongness of harris cornerness in a local area.

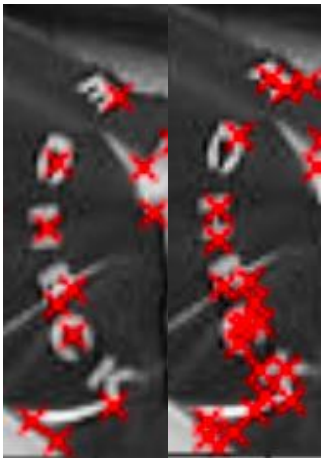


Fig 3 enlarged pic 2 computational result (left), built-in result (right)

Also, the value of empirical constant k will affect the identification. K ranges from 0.01 to 0.1. The bigger k is, the smaller the R value. I chose k equals to 0.05. It can be further tuned to match the standard result.

Task 2 - Deep Learning Classification (10 Marks)

normalize the data to the range between (-1, 1).

```
# Normalization
transform = transforms.Compose([
    transforms.ToTensor(),  # Tensor normalization  (0,255)~(0,1)
    transforms.Normalize(mean = 0.5, std = 0.5)])  #  normalization  (0,1)~(-1,1)
```

```
normal_train_img =  np.array([transform(train_img[i]).tolist()[0] for i in range(0,len(train_img))])
normal_test_img =  np.array([transform(test_img[i]).tolist()[0] for i in range(0,len(test_img))])
normal_val_img =  np.array([transform(val_img[i]).tolist()[0] for i in range(0,len(val_img))])
```

randomly flip the image left and right.

```
# Random choose  pictures to do the flipping, the index is random
number_of_flips = 1000
random_index = np.random.randint(len(train_img), size=number_of_flips) #random index
```

```
for index in random_index:
    # flip left to right, right to left
    normal_train_img[index] =  cv2.flip(normal_train_img[index], 1)
```

zero-pad 4 pixels on each side of the input image

```
pad_train_img = []
for i in range(0,len(normal_train_img)):
    #pad each of the trainning image with size of 4 zeroes
    pad_train_img.append(np.pad(normal_train_img[i], pad_width=4))
```

randomly crop 28x28 as input

```
cropped_train_img = []
for i in range(0,len(normal_train_img)):
    # crop the 32*32 image to 28*28 randomly
    initx, inity = np.random.randint(4),np.random.randint(4) #random index of 0,1,2,3
    cropped_train_img.append(pad_train_img[i][initx:initx+28, inity:inity+28]) #insure the cropped size is 28
```

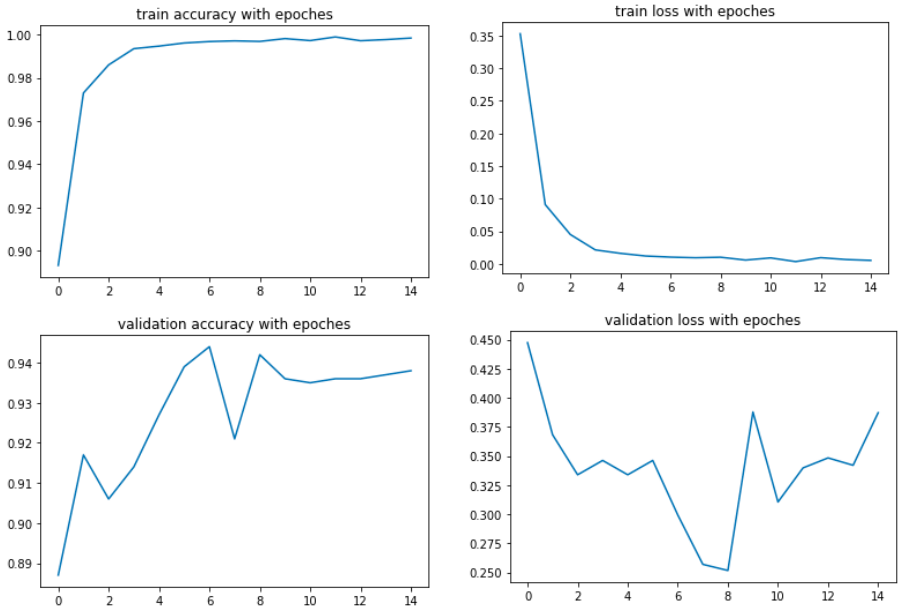
Build a CNN with the following architecture:

```
model = models.Sequential()
# 5×5 Convolutional Layer with 32 filters, stride 1 and padding 2 and then  ReLU Activation Layer.
# padding of size 2 is equivelant to 'same'  (28+2*2-5+1==28)
model.add(layers.Conv2D(32, (5, 5), strides = 1, padding ='same', activation='relu', input_shape=(28, 28,1)))

#2×2 Max Pooling Layer with a stride of 2.
model.add(layers.MaxPooling2D((2, 2),strides = 2))

# 3×3 Convolutional Layer with 64 filters, stride 1 and padding 1 then  ReLU Activation Layer.
# padding of size 1 is equivelant to 'same'  (14+1*2-3+1==14)
model.add(layers.Conv2D(64, (3, 3), strides = 1, padding ='same', activation='relu'))

#2×2 Max Pooling Layer with a stride of 2.
model.add(layers.MaxPooling2D((2, 2),strides = 2))

#flatten layers for constructure of fully-connected layer later
model.add(layers.Flatten())

# Fully-connected layer with 1024 output units. ReLU Activation Layer.
model.add(layers.Dense(1024, activation='relu'))

#Fully-connected layer with 10 output units
model.add(layers.Dense(10,activation="softmax"))
```

Set up cross-entropy loss, Adam optimizer, with 1e-3 learning rate and betas=(0.9, 0.999)

```
adam = tf.keras.optimizers.Adam(lr = 1e-3, beta_1=0.9, beta_2=0.999)
model.compile(
  adam,
  loss='categorical_crossentropy',
  metrics=['accuracy']
)
```

Train your model. Draw the following plots:



In practice, I used batch size of 128 in training, which can fasten the training speed and thus much less iterations are required. The model can reach relatively good performance in less than 15 epochs. I have also tried training without setting mini batch. The performance of loss and accuracy is similar but requiring much more iterations and time.

Train a good model.

To obtain a better result than the model using original setting, it is crucial to identify the main problem of the previous model. It is obvious to observe from the previous 4 plots, the empirical loss (training loss) is shrinking from epoch 0 to 8, and almost keeps unchanged from epoch 8. The leaning becomes hard. Similarly, the validation accuracy is increasing in the initial training period but also oscillated from epoch 8. It means the model is a bit over-training and the performance of the eventual model is even worse than the performance of the intermediate model (at epoch 6-8). The main problem of the model is overfitting.

To reduce overfitting, there are some measures.

- Early stopping
  Stop the training as long as reach the satisfiable performance. In practice, I trained 10 epochs first and decided whether to continue from the previous model based on the performance. The performance curves were plotted with the combination of several periods.

- Regularization
  Use L1 or L2 norm embedded in the layers of the model.

- Dropout
  Add dropout layer between layers.

- Use different sets of parameters and tune the best model
  Try combination of batch size, learning rate, optimizer, decay rate, etc. And find the best one.

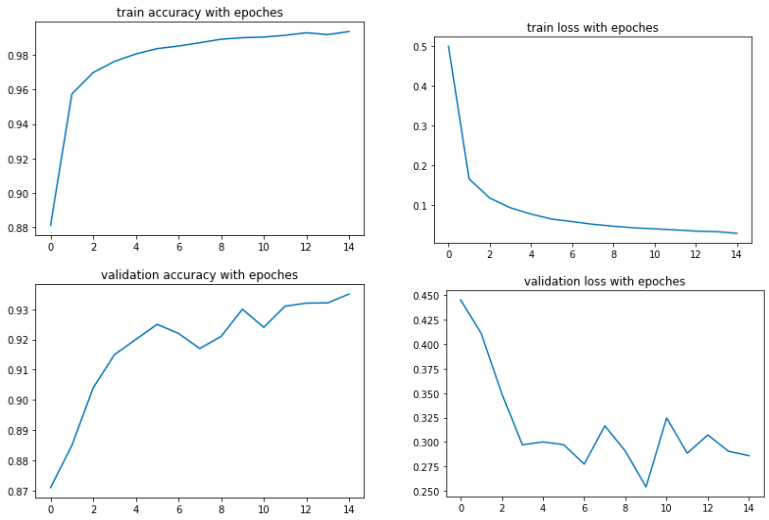
- Batch normalization

  Add batch normalization layer between layers.

Meanwhile, to enhance generalization, more convolution layers are added. The batch size was still set to 128 and thus the learning process was still very fast. Much less computation time and iteration steps are required. The best combination of setting is ‘bacth_size =128’, ‘lr = 1e-3’,’regulairzer =L2’, ‘batch normalization = false’, ‘dropout=0.1’,‘optimizer = adam’, and ‘convolution layers added’ = True.

The training loss has already closed to 0 but the validation loss is still unstable. The training accuracy has already closed to 100% but the maximum validation accuracy is around 94%.

Therefore, the overfitting problem is reduced bit but still exists. The good thing is that the derivative of training loss seems to be not 0, motivating the model being learnable. While in the original setting, the derivative of training loss vanishes quickly.



When visually comparing the validation accuracy, it is hard to tell any improvement of the modified model. However, if all the performance is listed, it is easily to see the improvement on testing set. Although the training accuracy and validation accuracy become worse, the testing accuracy has a large improvement. That is, the original model overfits to the training sample and give false indication of the good generalization but the modified one is much more reasonable to indicate the true performance. The tuned model is much better in generalization and thus has a good testing score.

	Training accuracy	Testing accuracy	Validation accuracy
Original model	0.9984	0.862	0.938
Modified model	0.9935	0.903	0.935

```
predicted=model1.predict(np.expand_dims(normal_test_img, axis=3))
result  =np.argmax(predicted,axis=1)
num = 0
for i in range(len(test_label)):
    if test_label[i]==result[i]:
        num+=1
print('testing accuracy for modified model ',num/len(test_label))
```

```
testing accuracy for modified model  0.903
```

```
predicted=model.predict(np.expand_dims(normal_test_img, axis=3))
result  =np.argmax(predicted,axis=1)
num = 0
for i in range(len(test_label)):
    if test_label[i]==result[i]:
        num+=1
print('testing accuracy for original model',num/len(test_label))
```

```
testing accuracy for original model 0.8622
```

Compare and discuss.

Compared with the benchmark result, my model has a relative lower performance. The lowest benchmark is no less than 92% and with several models even high up to 98%. I can see that the models that have high performance up to 95% are actually the ensemble of several models. It is acceptable that the ensembled model have higher performance than vanilla CNN. Compared with some simple models such as Nearest neighbor, KNN, SVM, my result is worse.

Despite the model itself, I believe one important factor is the size of given training samples. The training sample size for my model is 59000 while they have 70,000 samples. The sufficient training sample can play a key role in reducing the overfitting thus motivating a good model.

Compared to the simple CNN model, which is most similar to mine. Its score is 94% which is slightly better than mine. The difference is that the images are not processed using data augmentation techniques. It is possible that involving random noise to a set of training samples that is not very large will lead to worse performance.

Compared to ResNet18 model, it is much more superior than mine with over 98% accuracy. The reason is that the Resnet topology has a much better generalization ability than basic CNN model.

In the Resnet, the input from previous layer is added directly to the output of the other layer, and the network is actually leaning the residual. The residual learning can deal with the problem of vanishing gradient when there are many layers in model. In the later stage of the training, the learning will not be stuck, it can still keep improving and finally generate a better model. While my model is hard to learn any more in the end, and thus has a worse performance than ResNet18 model.