Practical Assignment Part 2

Shao Hsuan Hung (1723219)
s.hung@student.tue.nl

January 22, 2023

1 Gabor analysis

(1.)The command meshgrid is very useful for image processing. It creates two-dimensional coordinates, just like a command such as [5:.01:5] creates a one-dimensional coordinate system. Give the command [X,Y]=meshgrid([-5:0.1:5]); and plot the matrices X and Y using imagesc.

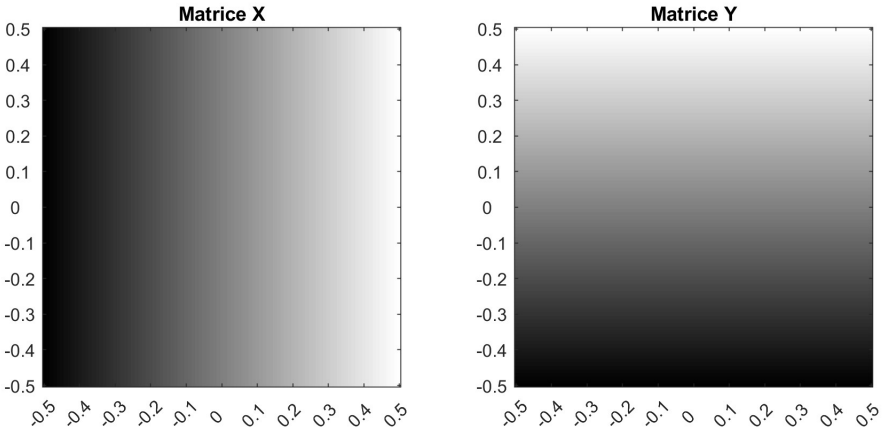


Figure 1: Matrics X and Y using imagesec

(2.)Create a 2D Gabor function using these X and Y, simply by using the definition in the formula defined in slide 37 of Module 05A. Try out different values for the parameters until you get a function which looks like the one in figures in slide 36 Module 05A.

```
[cosine,sine]= gabor_kernel(X,Y,pi/4,1,0.5,0.5);
function [kernel_map]= gabor_freq_domain(u,v,theta,gamma,eta,f0)
    u_rotate = u*cos(theta)+v*sin(theta);
    v_rotate = -u*sin(theta)+v*cos(theta);
    kernel_map = exp(-(pi/f0)^2.*(gamma^2.*(u_rotate-f0).^2 + eta^2.*(v_rotate.^2)));
end
```

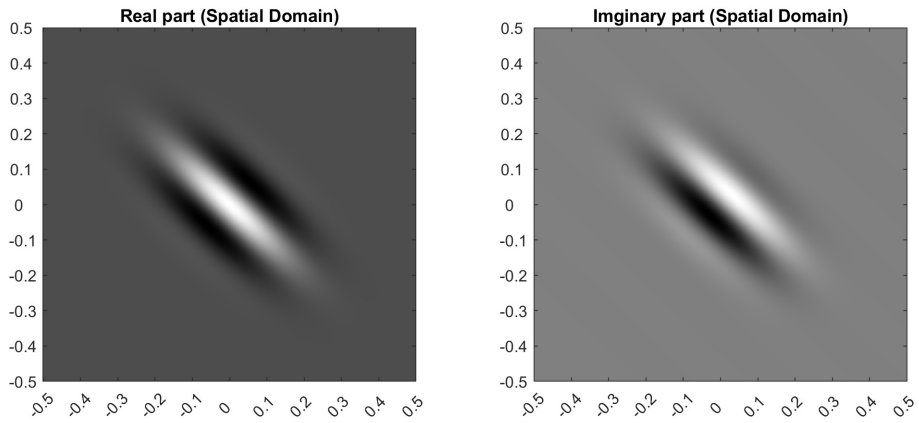


Figure 2: Gabor filter with parameters $(\theta, \gamma, \eta, f_0) = (\frac{\pi}{4}, 1, 0.5, 0.5)$

(3) Change the roles of X and Y to get a Gabor filter in a different orientation.

Change the role of x and y would make the wave propagation of the Gabor filter is perpendicular to the original one. The figure 3 shows the result, that has same parameters with the question(2).

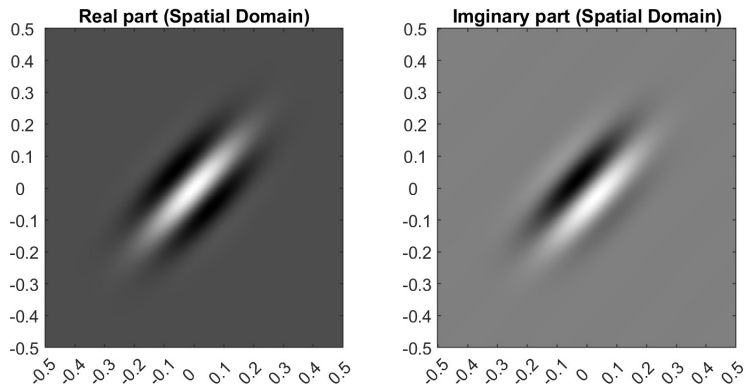


Figure 3: Change role of x and y, Gabor filter with parameters $(\theta, \gamma, \eta, f_0) = (\frac{\pi}{4}, 1, 0.5, 0.5)$

(5) Using the equation used in part (b), construct a 2D Gabor filter bank, which has 4 oriented filters at 3 scales. Design the filter bank such that the half power profiles of the filters touch each other in the spectral domain. Plot the filters in the spatial domain and their corresponding half power profiles in the spectral domain. Using the equations in the slides, you should be able to make the orientations within a single scale touch each other exactly, but you will need to play with the frequency a bit to have the different scales touch each other as well.

The figure 4 and 5 are the result of the Gabor filter bank with 4 orientation, 3 scales. Since we need 4 orientation, so the η parameter is equal to $\frac{2k}{\pi^2}\sqrt{-\ln\frac{1}{\sqrt{2}}}$, where k is equal to the number of the orientation: 4.

And the γ is calculated by solving the intersection of the two filters with same value of γ, η and θ but different frequency. Therefore, the γ is equal to $\frac{3}{\pi}\sqrt{\ln(\sqrt{2})}$. The f_0 is equal to the $\frac{x}{2^{n-1}}$, where x is the arbitrary number, n is 1,2,3 correspond to the inner circle to outer circle on the figure. the θ is rotate 0, 45, 90, 135 degree.

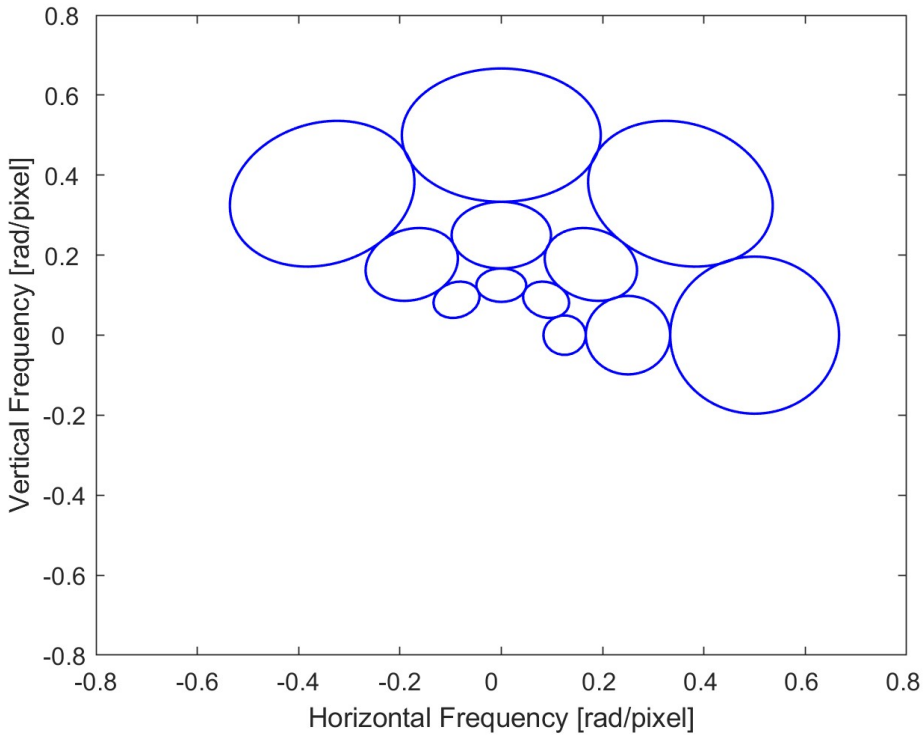
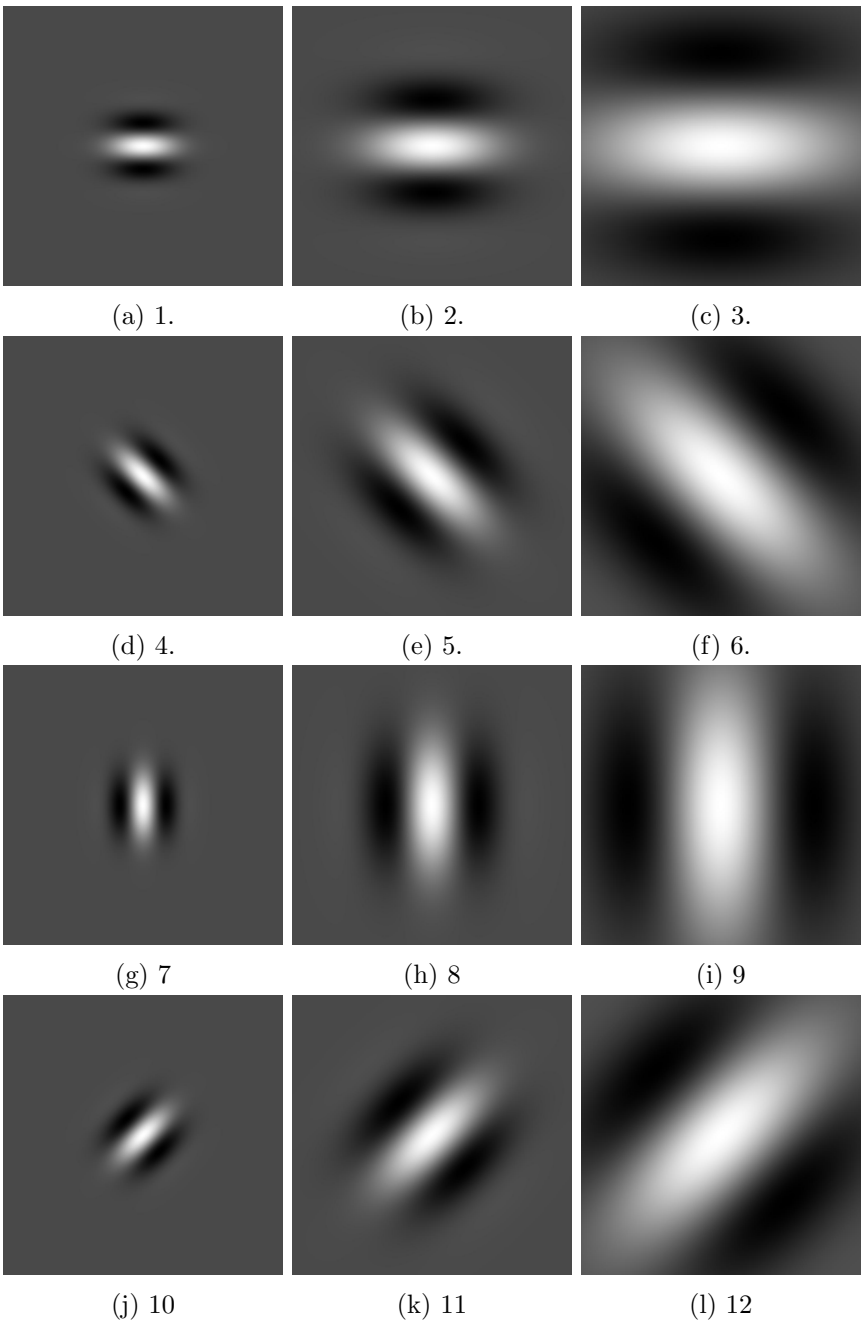


Figure 4: Gabor filter bank in frequency domain ($f_0 = 0.5$)



(a) 1. (b) 2. (c) 3.

(d) 4. (e) 5. (f) 6.

(g) 7 (h) 8 (i) 9

(j) 10 (k) 11 (l) 12

Figure 5: Gabor filter bank in spatial domain ($f_0 = 0.5$)

2 Principal components

(1.) Read in the image 'lena.pgm' and 'peppers.pgm'. Calculate and display the amplitude and phase spectrum of each image. For better visualization use the logarithmic scaling.

The amplitude and phase spectrum of the image are shown in the figure 6 and 7.

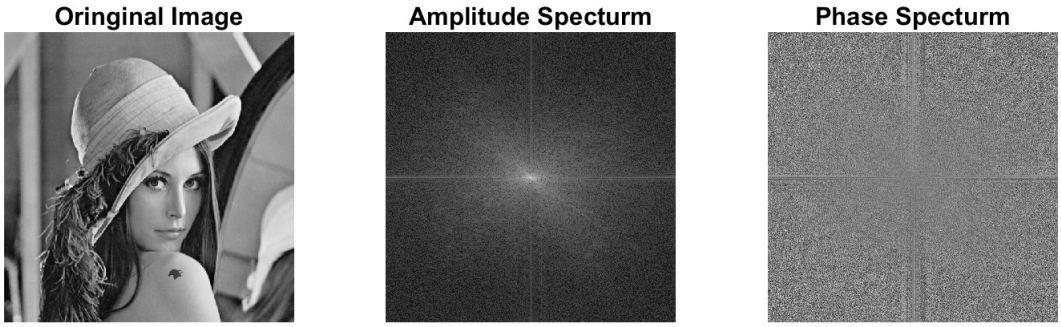


Figure 6: Amplitude and spectrum of the lena.pgm

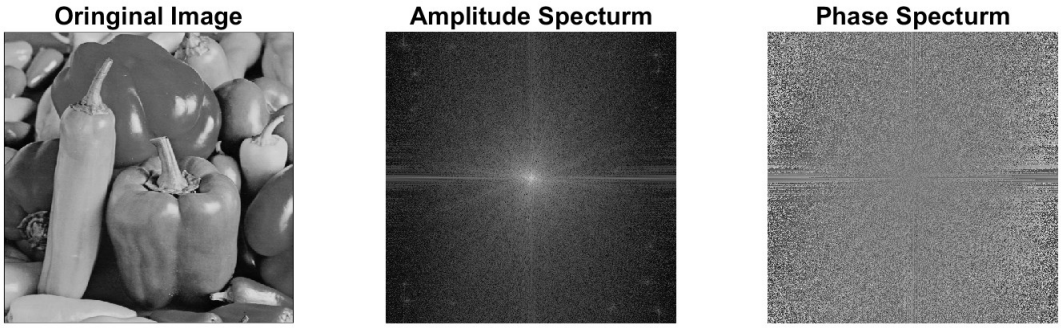


Figure 7: Amplitude and spectrum of the pepper.pgm

(2.) Create two new images by combining the phase spectrum of one image with the amplitude spectrum of the other image (and vice versa), and transforming back using the inverse DFT. Discuss briefly the resulting images. Which information are preserved by phase and which by amplitude of the spectrum?

As shown in the figure 8 and 9, combining magnitude of lena.pgm with phase of pepper.pgm show the contour of pepper.pgm. Combining magnitude of pepper.pgm with phase of lena.pgm show the contour of the lena.pgm. Becasue most of the spatial information in the image that human can perceive is stored in phase. The magnitude part corresponds to the amplitudes of the basis images in the Fourier representation.

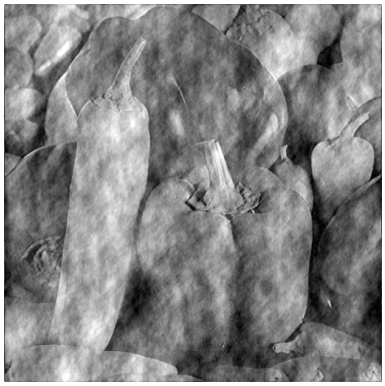
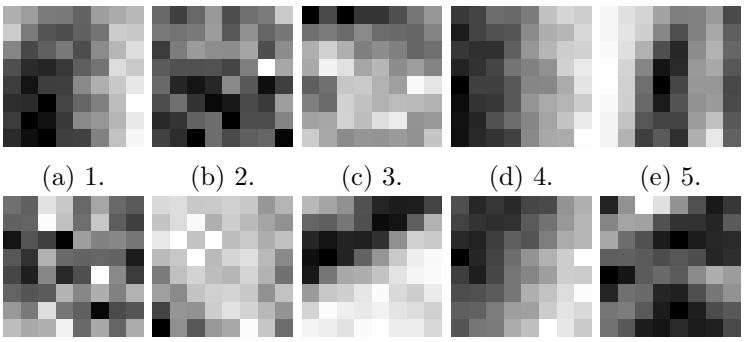


Figure 8: Magnitude: Lena.pgm, phase:pepper.pgm

Figure 9: Magnitude: pepper.pgm, phase: Lena.pgm

(c.) Read in the image 'lena.pgm' and cut out 1000 patches of size 8×8 pixels from random positions in the image. DC center and contrast normalize each patch. Display 10 of the resulting grayscale patches.

The resulting patches with DC centering and contrast normalization is shown in the figure 10.



(a) 1. (b) 2. (c) 3. (d) 4. (e) 5.

(f) 6. (g) 7 (h) 8 (i) 9 (j) 10

Figure 10: 10 resulting grayscale patches

(d.) Learn the 64 PCA basis images and show them as grayscale images, sorted according to the corresponding eigenvalues.

The 64 PCA basis images of the 1000 patches images is shown in the figure11. For first, calculate the mean of the pixel in the same position in the 8*8 block. Then calculate the covariance matrix of the 64 pixels, getting 64*64 matrix. Then find and sort the eigen vector by the eigen value.

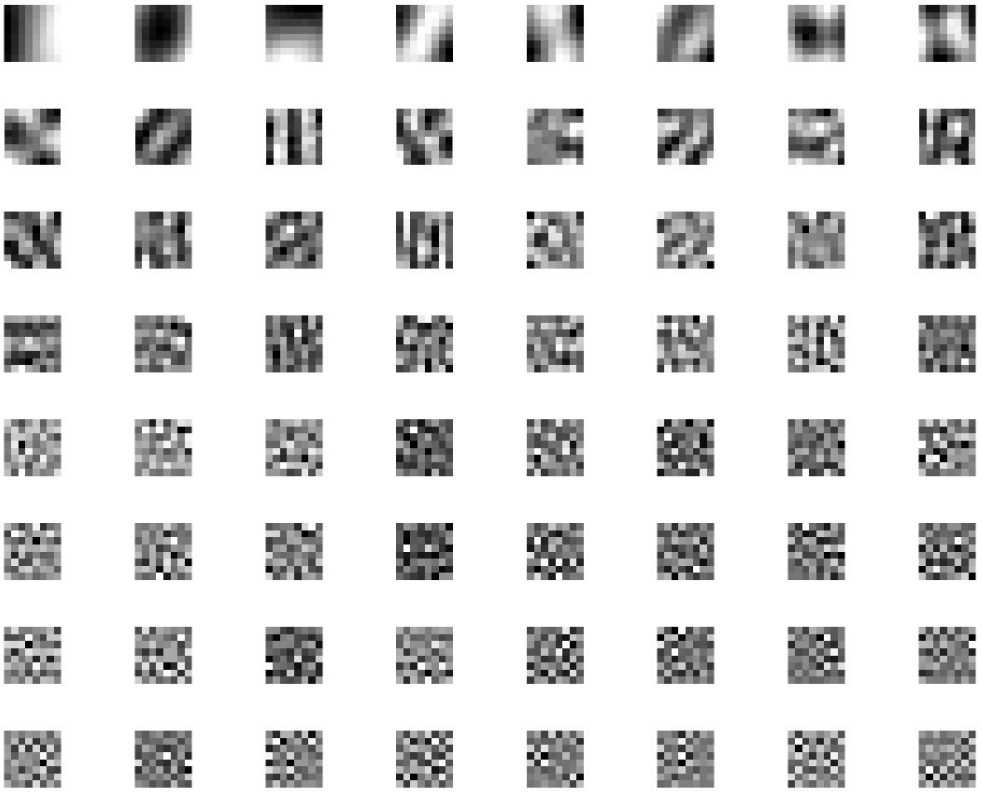


Figure 11: 64 PCA basis images, sorting according to the descending eigen value

(e.) Compute the PCA whitening matrix of the patches and show the whitened 'lena'.

As shown in the figure 12, the whiten lean.pgm is computing by whitening matrix based on the 1000 patches. As mentions in the previous question (2.d), the eigen vector and eigen value of the covariance matrix is computed. Then the whitening matrix of the patches is: $W = V\Lambda^{\frac{-1}{2}}V^T$, where the V matrix is eigen vector of the 1000 patches, Λ is diagonal eigen value. So the whiten image would be $X_{whitening} = W \cdot X$, where X is image, $X_{whitening}$ is whiten image.

Figure 12: The whiten lena.pgm

3 Hidden Markov models

(a.) Convert every image to a sequence of 28 numbers, such that the j-th number is the sum of intensities of all pixels in the j-th row of the image.

I convert the test dataset into the scheme as shown in the figure 13, meaning that each state is the sum of the pixel value in each column.

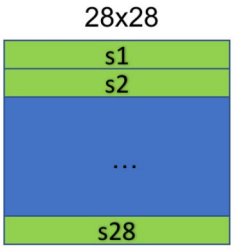


Figure 13: Add up all pixel in the column

(b.) Build an HMM for each of the 10 digit classes. Each HMM will consist of 28 states. Each state will correspond to a row of the image. The observation density for that state has to be appropriate for images of the class that particular HMM represents. Use a Gaussian to represent each density, but you can use other types of distributions if you prefer. However, the actual parameters of each distribution (e.g., the mean and variance of each Gaussian) have to be learned from training data. Use the first 5,000 images of test digits as training data to learn those distributions.

Assume that the each observation density is Gaussian distribution, so the parameters μ and σ can be learned from 5000 trainging data. Since we let each HMM consist of 28 states, the structure of each model is constructed as shown in the figure 14. Each model is consist of 28 latent variables and 28 observations (rows in images). The the transition matrix between the latent variables $p(z_n|z_{n-1})$ are equal to one since when human write down the digits number, the transition between the nearby row is almost 100 present happen for the train data is large. The emission from one latent variable to one observation is assumed to be the Gaussian distribution, so there have 2 parameters (μ and σ) for each path from latent variable to observation. So for

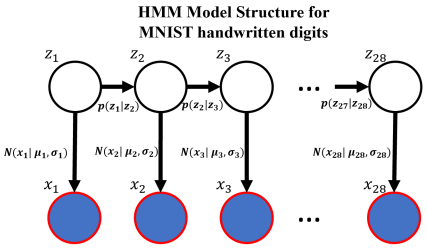


Figure 14: Model structure of the HMM model for MNIST handwritten digits

one class of digits we have one HMM model that consist of $2 \times 28 = 56$ parameters (560 parameters for 10 HMM model). The model is designed to calculate the probability of latent and observed variables:

$$p(X, Z) = p(Z_1) \prod_{n=2}^{28} p(Z_n \mid Z_{n-1}) \prod_{n=1}^{28} p(X_n \mid Z_n) \tag{1}$$

As mention before, since the transition between the latent variable is 100 percent, the probability of the z_n given z_{n-1} are all 1, so the emission probabilities are 1 and marginal distribution of the Z1 are 1. Then the equation would become:

$$p(X, Z) = \prod_{n=1}^{28} p(X_n \mid Z_n) = \prod_{n=1}^{28} p(X_n \mid \mu_n, \sigma_n) \tag{2}$$

Where the X_n is the each row input in testing data, μ_n and σ_n are the mean and standard deviation of the corresponding node n. To sum up, the overall trained HMM model is class 0 to 9 HMM model, every model have 28 mean and 28 variance that learn from the first 5000 training images.

(c.) Use your 10 HMMs to classify the second 5,000 thousand images (images 5,001-10,000).

When doing the classification on the testing data, the X_n would be the sum of the value of the row n, it would output one value of likelihood. For one image that doing the inference on 10 HMM models, it would result 10 likelihoods, choosing the maximum likelihood as the model output of that image. The testing results of the images 5,001 to 10,000 are shown in the question (d).

(d.) Report both the classification error rate, and the confusion matrix (a 10 × 10 matrix whose (i,j) entry says how many test images of class i were classified as class j).

The confusion matrix and the error rate of the classification are shown in the figure 15, the average accuracy is 66.8%. For class 1, 4, 6 and 7, these classes have above 80% of the accuracy. However, the class 5 have very bad performance, the model always predict the digits number 5 as 3 since they have similar row pixel distribution on half circle feature in the lower part of '5' and '3'.

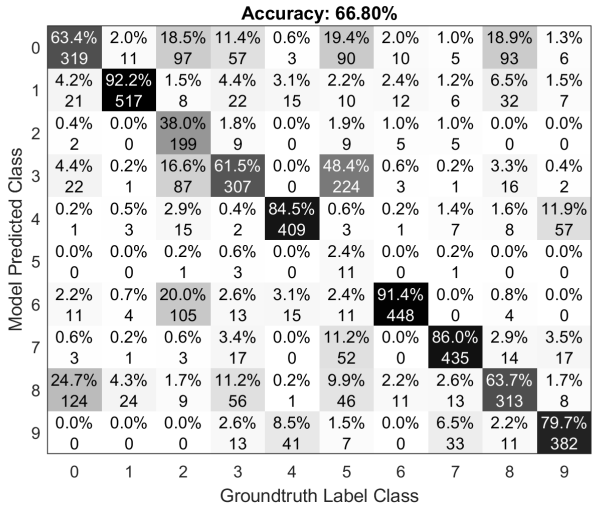


Figure 15: Confusion matrix

4 YOLO object detector

(a.) Run pretrained YOLO on the coco2 image and visualize the boxes that achieve an IoU of at least 0.1, 0.3, 0.5, 0.7 and 0.9. Which IoU do you consider most reasonable for real-world applications for an object to be considered "correctly detected"? Similarly, which confidence of the detector (the threshold parameter) would you consider "correctly detected"? In which real-world scenarios would you use a higher or lower IoU criterion?

As shown in figure 17, when increasing the IOU threshold (fix the confident score threshold = 0.5), the less bounding box that the YOLO model detected. Compare to the figure 16, the IOU threshold can filter out the wrong prediction (such as the zebra-appearance car) and the correct but not imprecise localized predictions. I think the most reasonable for real-world application for an objected is to set the IOU no less than 0.5. Otherwise, there will detected some other object with partially similar features. But it really depends on the application, for example, if the detection task need a really precise localization model, the IOU threshold should be really high. If the application want to detect as much as object and not really care about the false positive, the IOU threshold can be relative low. For the confident score, except the car object (have about 0.7 confident score in the coco2.jpg), I think confident score that above 0.5 is good enough to detect the zebra correctly. In this case, when the threshold score = 0.4, the model detect the most zebra. (have higher recall). But again, it really depends on the application, for example, if the application task about medical image analysis, the score threshold can be relative low, to catch out any possible disease.

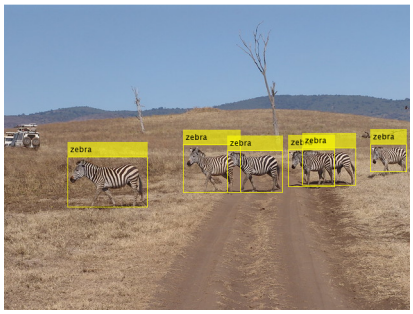


Figure 16: Ground truth of the coco2.jpg

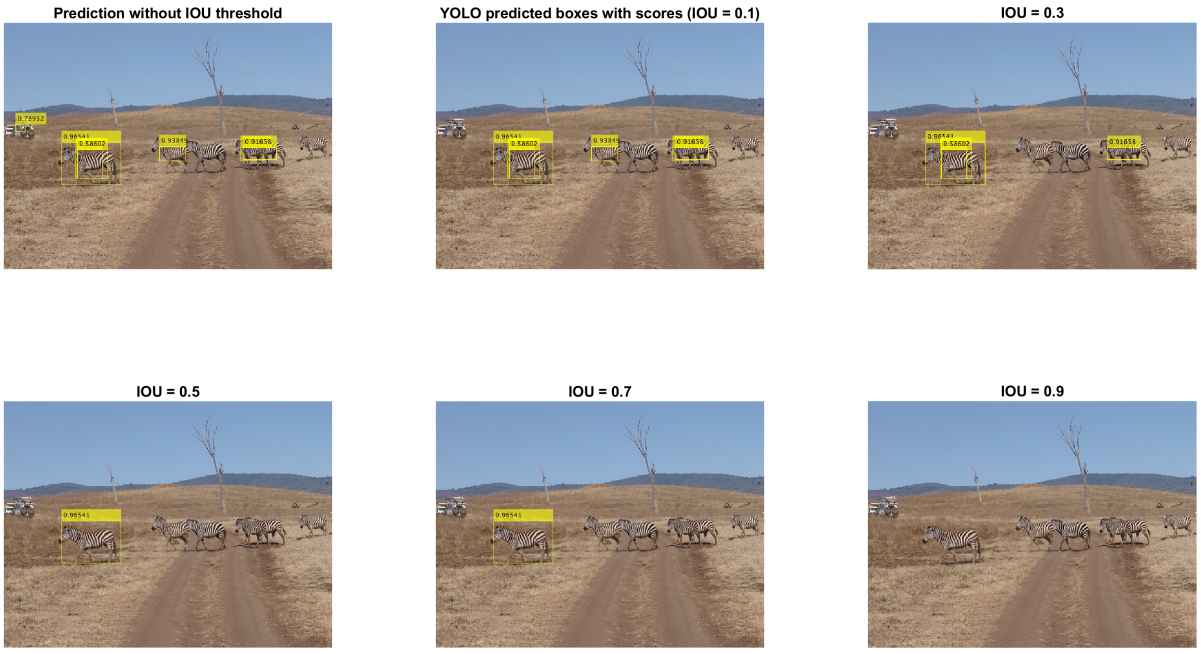


Figure 17: YOLO predicted with different IOU score (from 0.1 to 0.9) and fix confident score threshold = 0.5

(b.) Everyone knows the camera adds 10 pounds, even if you're an animal. Create a function that automatically changes the output of the network so that all bounding boxes of animals are 20% wider, but keep the box size of other classes such as cars the same. Do not change the box height. Visualize the three images with these detections. Hint: Check the categorical predictedLabels output to see which classes exist.

As shwon in the figure 18, 19, and 20, all the animal's bounding boxes is 20% wider, bounding boxes of other class still same in size.

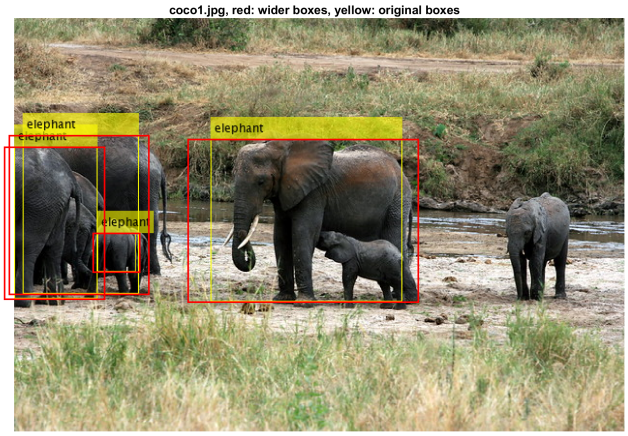


Figure 18: coco1.jpg with wider bounding boxes

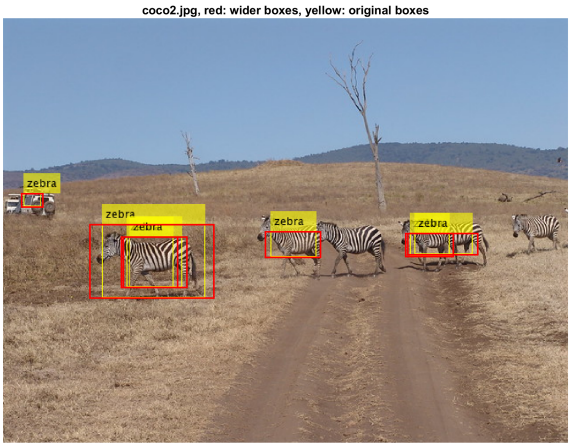


Figure 19: coco2.jpg with wider bounding boxes

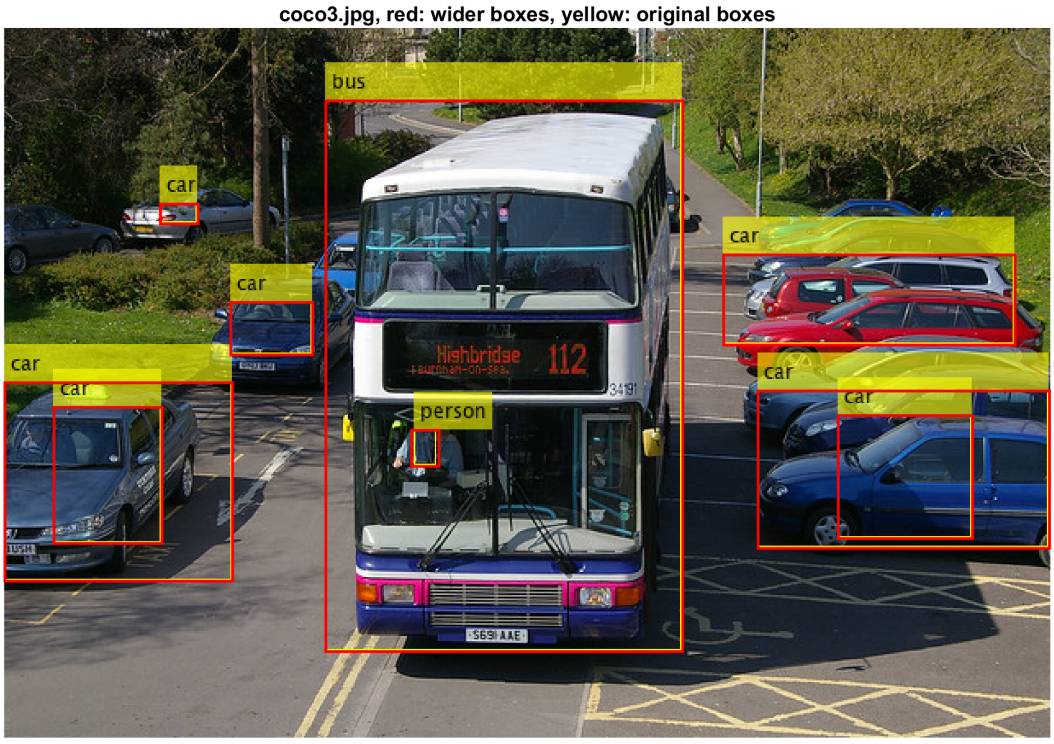


Figure 20: coco3.jpg with wider bounding boxes

(c.) Implement a custom variant of Non-Maximum Suppression yourself. In contrast to real NMS which keeps the best box, compute the box that fits all overlapping (with any amount of overlap, no matter how small) detection boxes of the same class with confidence score above some value T. Show the resulting detection boxes on the three example images before and after your custom NMS with T=0.2

The custom variant of Non-Maximum Suppression algorithm is implemented in the file:"HW2_part4.m". The result images before and after the non-maximum suppresion are shown in the figreu 21, 22, and 23.

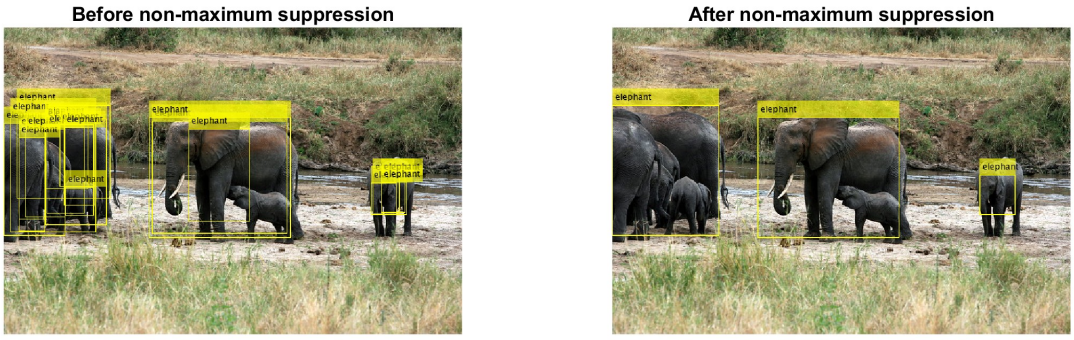


Figure 21: CoCo1.jpg resulting detection boxes before and after NMS with T=0.2

Figure 22: CoCo2.jpg resulting detection boxes before and after NMS with T=0.2

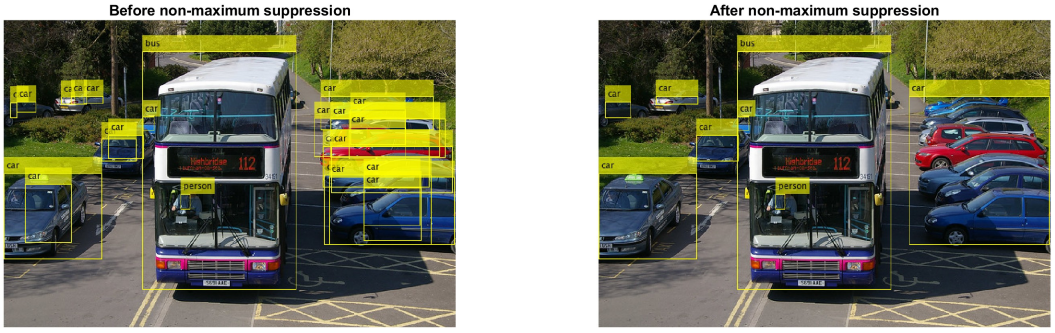


Figure 23: CoCo3.jpg resulting detection boxes before and after NMS with T=0.2