Quiz **Name:**

CS276: Fall 2021 **Student Number:**

Instructor: Jingyi Yu

Instructions:
Please answer the questions below. Show all your work. This is a close-book test. NO discussion/collaboration is allowed.

Problem 1. CV (10 points)

(a) Name two different color representations.
RGB, HSV

(b) What do you do to sharpen an image?
Scale the intensities of an image by 2 and then subtract from the result a smoothed version of the original image. Essentially, apply an impulse filter of amplitude 2, separately apply an averaging flter and subtract this from the former.
Or,
Laplace kernel is also OK.

(c) What is the camera calibration problem? If the internal camera parameters are known, how many calibration parameters need to be determined to solve the 2-camera stereo vision problem.
5 or 6 is OK

Problem 2. CG (10 points)

(a) Given the camera position $p = [1.0, 0.5, -2.0]$, viewing direction $v = [1.0,\ 0.5,\ 1.0]$, and the up direction $u = [0,\ 1,\ 0]$, determine the view transformation matrix M which transforms 3D points onto camera coordinate system. 5

(b) Please briefly explain the difference between Ray Tracing and Rasterization. 5

```
nR = np.array([[-0.70710678,  0.06786548,  0.66666667,  1.        ],
               [ 0.        ,  0.06126151,  0.33333333,  0.5       ],
               [ 0.70710678, -0.99581188,  0.66666667, -2.        ],
               [0.,0.,0.,1.0]])
```

```
np.linalg.inv(nR)
```

```
array([[-1.34831592,  2.5648366 ,  0.06589764,  0.19769291],
       [-0.85252041,  3.41008169, -0.85252041, -2.55756126],
       [ 0.15668006,  2.37327976,  0.15668006, -1.02995982],
       [ 0.        ,  0.        ,  0.        ,  1.        ]])
```

Problem 3. Deep learning (10 points)

(a) You are designing a CNN classifier. For each layer, calculate the number of weights, number of biases and the size of the associated feature maps. The notation follows the convention:

- **CONV-K-N** denotes a convolutional layer with N filters, each them of size K×K, Padding and stride parameters are always 0 and 1 respectively.
- **POOL-K** indicates a K × K pooling layer with stride K and padding 0

Layer	Activation map dimensions
INPUT	128 × 128 × 3
CONV-9-32	120 × 120 × 32
POOL-2	60 × 60 × 32

(b) Why is it important to place non-linearities between the layers of neural networks?

Non-linearity introduces more degrees of freedom to the model. It lets it capture more complex representations which can be used towards the task at hand. A deep neural network without non-linearities is essentially a linear regression

(Make sense.)

Problem 4. Optimizer (10 points)

(c) The code below is meant to implement a single step of the training loop using the Adam optimizer, but some parts are missing. Finish the implementation of each line marked TODO. Recall the parameter update equations for Adam optimization:

$$V = \beta_1 V + (1 - \beta_1)\frac{\partial J}{\partial W}$$

$$S = \beta_2 S + (1 - \beta_2)\left(\frac{\partial J}{\partial W}\right)^2$$

$$V_{corr} = \frac{V}{1 - \beta_1^t}$$

$$S_{corr} = \frac{S}{1 - \beta_2^t}$$

$$W = W - \frac{\alpha}{\sqrt{S_{corr} + \epsilon}} V_{corr}$$

```
def optim_adam(weights_dict, gradients_dict, cache_dict, step):
    """
    v is VdW, s is SdW, v_corr is VcorrdW, s_corr is ScorrdW.
    """
    lr, beta1, beta2, eps = 1e-3, 0.9, 0.999, 1e-8
    for weight_name in weights_dict:
        w = weights_dict[weight_name]
        grad = gradients_dict[weight_name]
        v = cache_dict["v" + weight_name]
        s = cache_dict["s" + weight_name]

        # TODO: Exp weighted avg of grad
        v =

        # TODO: Exp weighted avg of grad^2
        s =

        # TODO: Bias correction. divide by (1 - beta1^step))
        v_corr =

        # TODO: Bias correction. divide by (1 - beta2^step))
        s_corr =

        # TODO: Update rule for Adam
        w =

        cache_dict["v" + weight_name] = v
        cache_dict["s" + weight_name] = s
        weights_dict[weight_name] = w
```

```
v = beta1 * v + (1 - beta1) * grad
s = beta2 * s + (1 - beta2) * (grad ** 2)
v_corr = v / (1 - (beta1 ** step))
s_corr = s / (1 - (beta2 ** step))
w = w - lr * v_corr / np.sqrt(s_corr + eps)
```