CPSC 425: Assignment 3
Name: Terence Chen
Student number: 42602136

Note: Along with this PDF, I also handed in Holefill.py and the original images used for question 6. I did not hand in polyselect.py since I didn't really change anything to it except when testing the original images (just changing file names for the image that is opened).

Q1) Nothing to hand in

Q2) Nothing to hand in

Q3) Nothing to hand in

Q4)
Code:

```
def ComputeSSD(TODOPatch, TODOMask, textureIm, patchL):
        patch_rows, patch_cols, patch_bands = np.shape(TODOPatch)
        tex_rows, tex_cols, tex_bands = np.shape(textureIm)
        ssd_rows = tex_rows - 2 * patchL
        ssd_cols = tex_cols - 2 * patchL
        SSD = np.zeros((ssd_rows,ssd_cols))
        for r in range(ssd_rows):
                for c in range(ssd_cols):
                        # Compute sum square difference between textureIm and TODOPatch
                        # for all pixels where TODOMask = 0, and store the result in SSD
                        #
                        # ADD YOUR CODE HERE

                        # Loop through each row and coloumn of the patch
                        for row in range(patch_rows):
                            for col in range(patch_cols):

                                # ignore all empty pixels that have a value of 1
                                if TODOMask[row][col] == 1:
                                    continue

                                # Use helpers to calculate the row and column positions
                                textures = textureIm[row + r][col + c]
                                patches = TODOPatch[row][col]

                                # Compute the sum squared difference for each color value and add it to SSD.
                                # We know that there are 3 different color value for each pixel since we have a dimension of 3
                                for colorValue in range(0,3):

                                    # Compute the sum squared difference and makes sure its a folating point
                                    SSD[r][c] += ((patches[colorValue] * 1.0 - textures[colorValue]) ** 2)

                        pass
                pass
        return SSD
```

Q5)
Code:

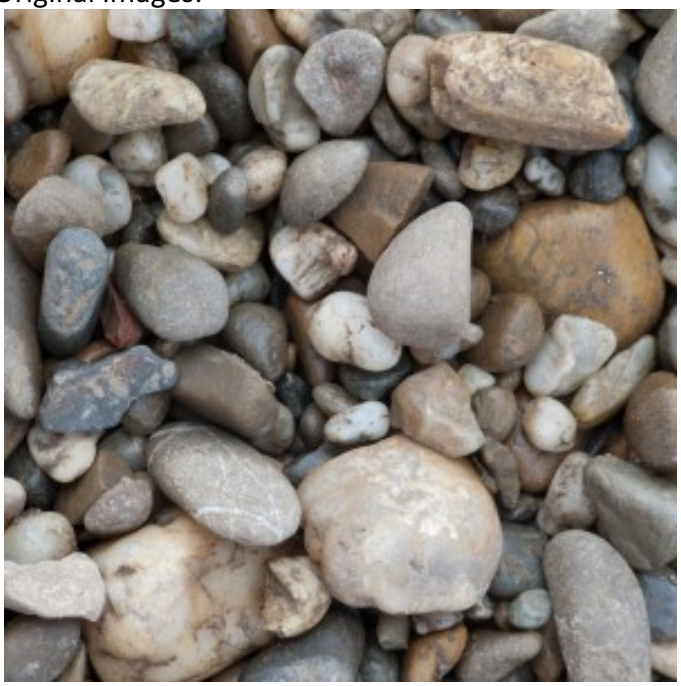
```
def ComputeSSD(TODOPatch, TODOMask, textureIm, patchL):
        patch_rows, patch_cols, patch_bands = np.shape(TODOPatch)
        tex_rows, tex_cols, tex_bands = np.shape(textureIm)
        ssd_rows = tex_rows - 2 * patchL
        ssd_cols = tex_cols - 2 * patchL
        SSD = np.zeros((ssd_rows,ssd_cols))
        for r in range(ssd_rows):
                for c in range(ssd_cols):
                        # Compute sum square difference between textureIm and TODOPatch
                        # for all pixels where TODOMask = 0, and store the result in SSD
                        #
                        # ADD YOUR CODE HERE

                        # Loop through each row and coloumn of the patch
                        for row in range(patch_rows):
                            for col in range(patch_cols):

                                # ignore all empty pixels that have a value of 1
                                if TODOMask[row][col] == 1:
                                    continue

                                # Use helpers to calculate the row and column positions
                                textures = textureIm[row + r][col + c]
                                patches = TODOPatch[row][col]

                                # Compute the sum squared difference for each color value and add it to SSD.
                                # We know that there are 3 different color value for each pixel since we have a dimension of 3
                                for colorValue in range(0,3):

                                    # Compute the sum squared difference and makes sure its a folating point
                                    SSD[r][c] += ((patches[colorValue] * 1.0 - textures[colorValue]) ** 2)

                        pass
                pass
        return SSD
```

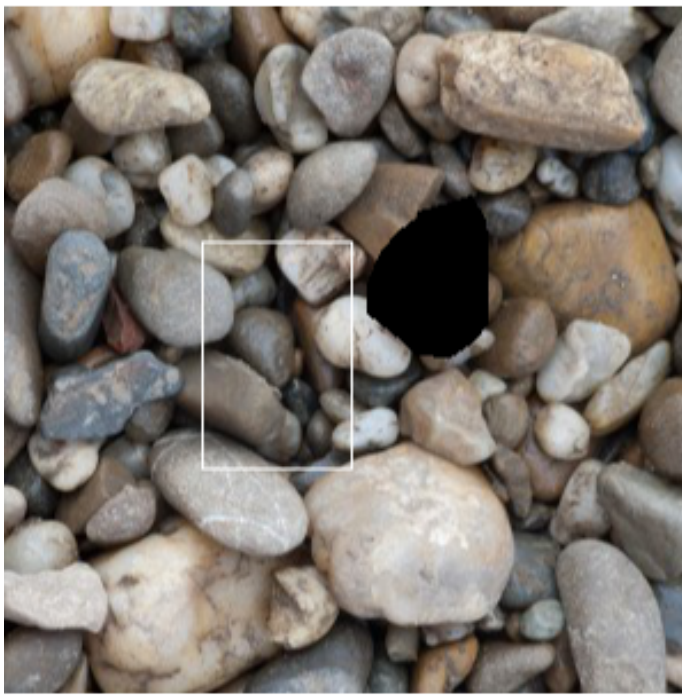
Donkey image after texture synthesis:

6)
Image of Sky: Performs well
Original images:

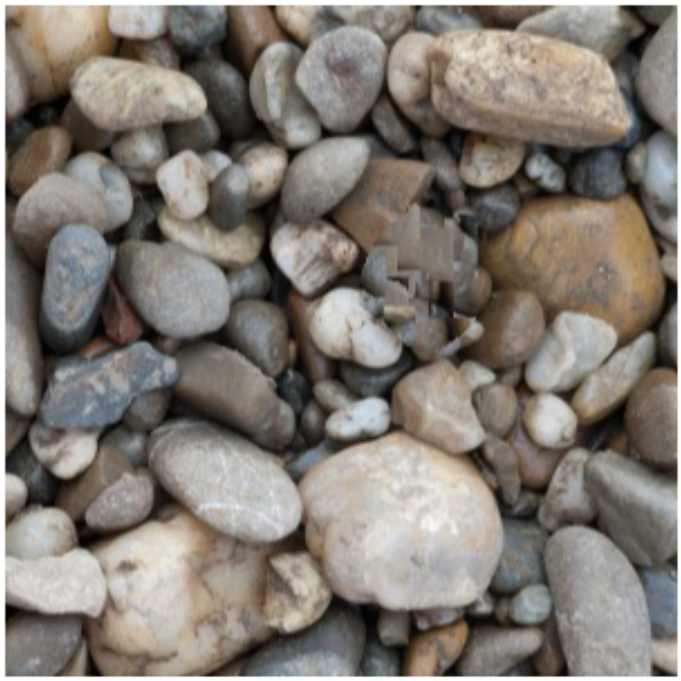
Result of algorithm:

Image of Rocks: Performs poorly
Original images:





Result of algorithm:



As shown, the algorithm performs poorly on the rock image compared to the sky image, this is most likely due to the fact that the algorithm could not find suitable matches to fill the hole. The texture/patterns of the rock makes it so that any misalignment in image becomes very noticeable. Therefore, the algorithm performs poorly on the rock images.

7)

randomPatchSD:

The first parameter randomPatchSD determines which texture patch is chosen to fill the hole (creates variation in texture patches used). A small value for randomPatchSD makes it so the patch used will have a higher chance of repeated textures. Meaning result will most likely look terrible since we will have randomness thus a greater chance that textures will be repeated during the filling. A high value of randomPatchSD makes it so we have a higher variation of textures when filling the hole. However, a high value can lead to bad matches.

PatchL:

The parameter patchL is a parameter that determines the size of the patches. When this parameter is too small, patches may be skipped or missed since the algorithm will be finding matches on a scale that is too small leading to inaccurate fills. A large value makes it so the patch will be noticeable since it will be matching on a larger scale. Meaning the patch may contain large instances of patterns/details.