**Code Customization**

**Accuracy graphs –**

acc = history.history['acc']

val\_acc = history.history['val\_acc']

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend()

plt.show()

**Confusion matrix, without normalization**

plot\_confusion\_matrix(confusion\_matrix, ['circle', 'rectangle', 'square', 'triangle'])

**Mesh generator**

grey\_img = Image.open('/content/DSC04701.JPG').convert('L')

max\_size=(200,200)

max\_height=10

min\_height=0

#height=0 for minPix

#height=maxHeight for maxPIx

grey\_img.thumbnail(max\_size)

imageNp = np.array(grey\_img)

maxPix=imageNp.max()

minPix=imageNp.min()

print(imageNp)

(ncols,nrows)=grey\_img.size

vertices=np.zeros((nrows,ncols,3))

for x in range(0, ncols):

for y in range(0, nrows):

pixelIntensity = imageNp[y][x]

z = (pixelIntensity \* max\_height) / maxPix

#print(imageNp[y][x])

vertices[y][x]=(x, y, z)

faces=[]

for x in range(0, ncols - 1):

for y in range(0, nrows - 1):

# create face 1

vertice1 = vertices[y][x]

vertice2 = vertices[y+1][x]

vertice3 = vertices[y+1][x+1]

face1 = np.array([vertice1,vertice2,vertice3])

# create face 2

vertice1 = vertices[y][x]

vertice2 = vertices[y][x+1]

vertice3 = vertices[y+1][x+1]

face2 = np.array([vertice1,vertice2,vertice3])

faces.append(face1)

faces.append(face2)

print(f"number of faces: {len(faces)}")

facesNp = np.array(faces)

# Create the mesh

surface = mesh.Mesh(np.zeros(facesNp.shape[0], dtype=mesh.Mesh.dtype))

for i, f in enumerate(faces):

for j in range(3):

surface.vectors[i][j] = facesNp[i][j]

# Write the mesh to file "cube.stl"

surface.save('surfacenew.stl')

print(surface)