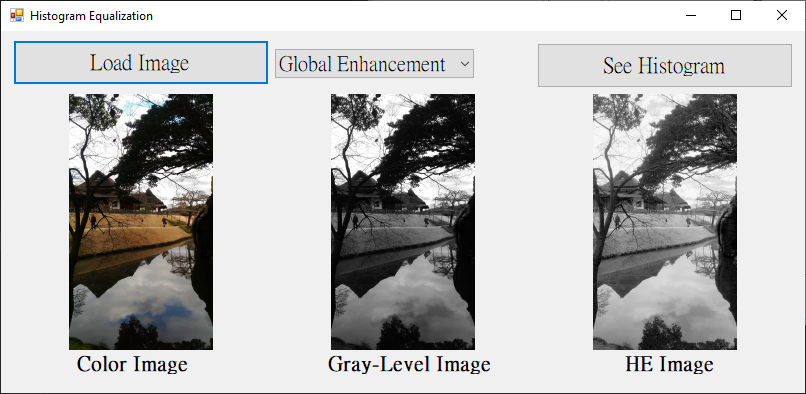
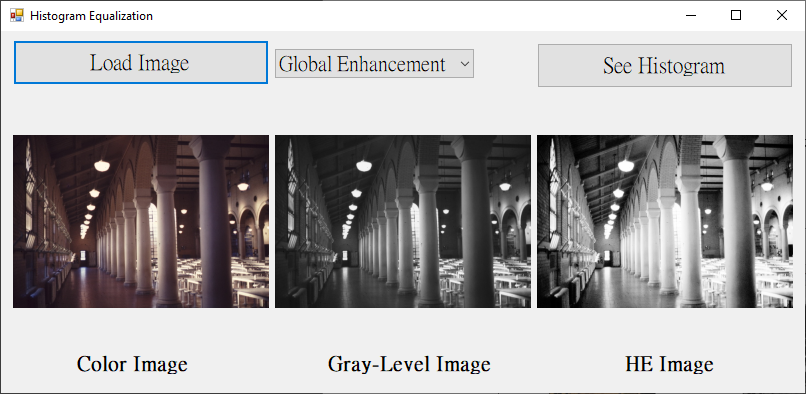
四資工三甲 C108151131 賴冠綸

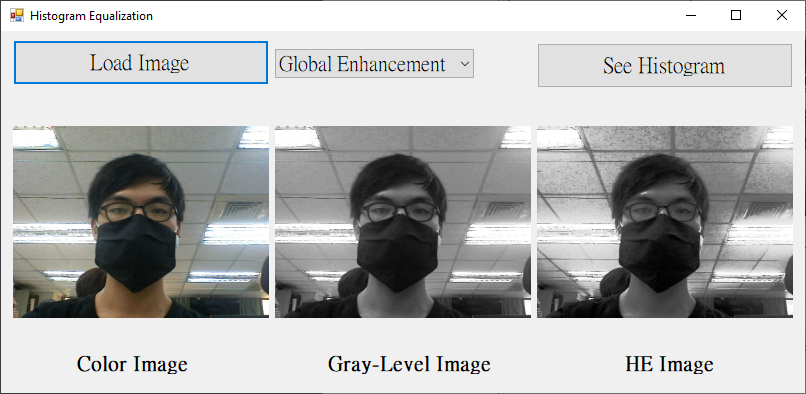
1. 程式執行畫面：
   * 1. 整張處理（Global Enhancement）（**上方下拉選單需選擇Global**）
        1. 載入測試圖片1



* + - 1. 載入測試圖片2



* + - 1. 載入自己的照片



* + 1. 局部處理（Local Enhancement）（**上方下拉選單需選擇Local**）

有numbericUpDown會顯示出來，讓我們可以自由選擇局部範圍n\*n（奇數）

* + - 1. 網路上找的測試圖

7\*7



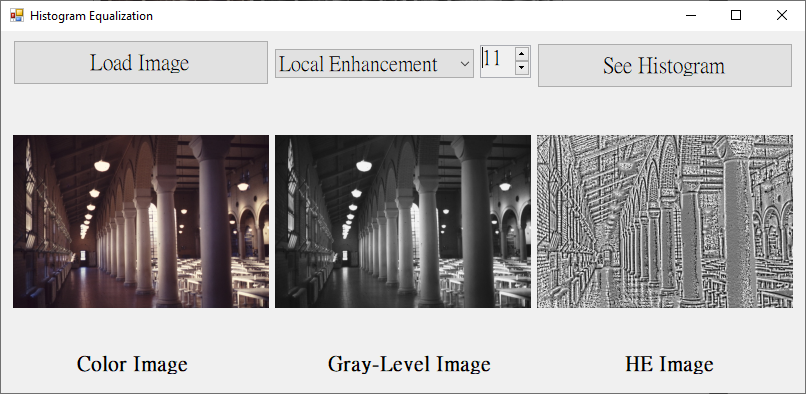
15\*15



49\*49

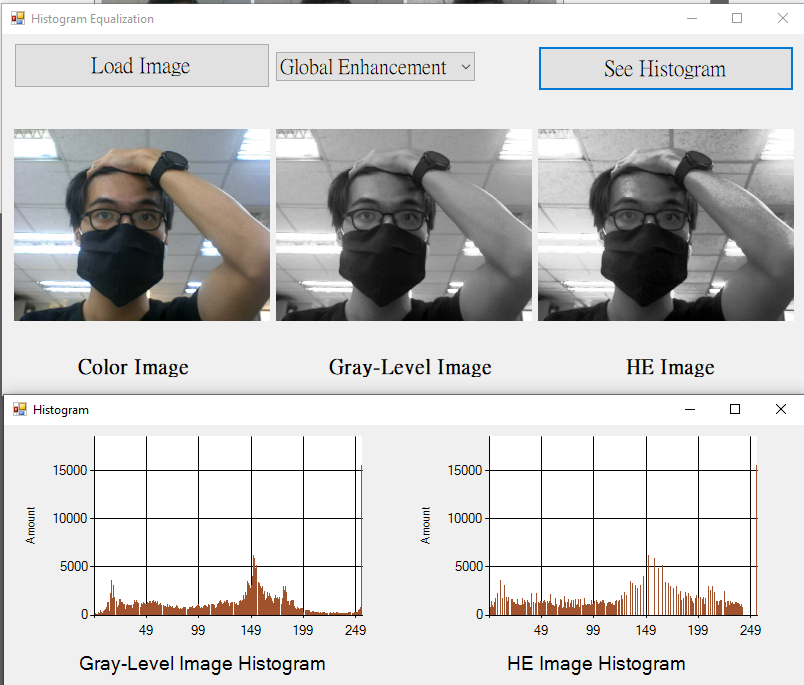


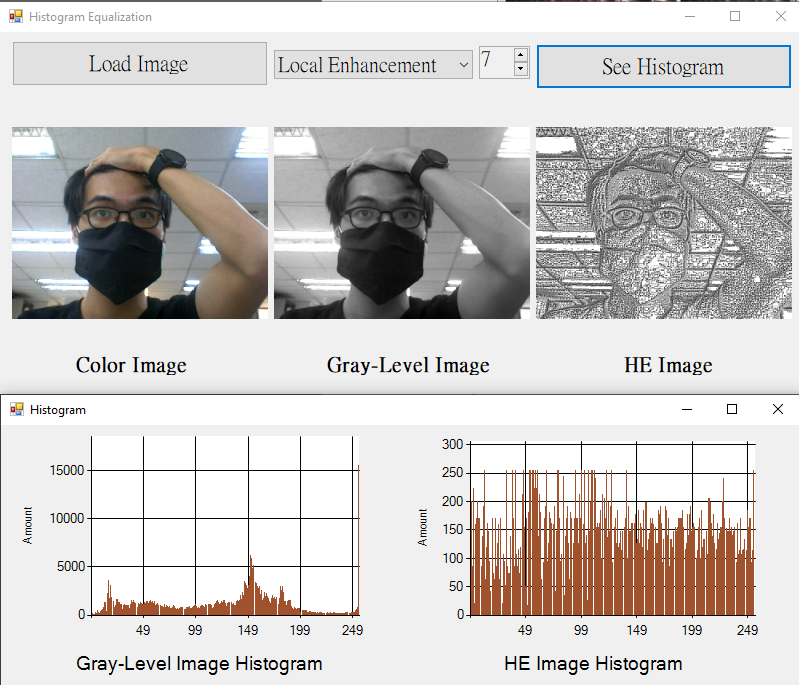
* + - 1. 測試圖片2（僅展示11\*11）



* + 1. 將直方圖分布視覺化（右上方Button）

下方為自己的照片之「整張處理」及「局部處理」的直方圖分布。



****

1. 程式碼（僅附主要程式碼）
   * 1. 整張處理

|  |
| --- |
| private: Bitmap^ HistogramEqualized\_Global(Bitmap^ gray) {  Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat); // HE image  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadWrite, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int grayByteskip = grayBD->Stride - grayBD->Width \* 3;  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  int r[256] = { 0 }; // count each level(0 ~ 255)'s amount  int s[256] = { 0 }; // equalization  int pixelCount = 0;  for (int i = 0; i < 256; i++) r[i] = s[i] = 0;  for (int y = 0; y < grayBD->Height; y++) {  for (int x = 0; x < grayBD->Width; x++) {  r[grayPtr[0]]++;  pixelCount++;  grayPtr += 3;  }  // throw unknown information away  grayPtr += grayByteskip;  }  int accumulation = 0; // accumulation before current level  for (int i = 0; i < 256; i++) // normalization {  /\*\* using to draw histogram \*\*/  if (r[i] > histogramMax) histogramMax = r[i];  GrayLevelCount[i] = r[i];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  accumulation += r[i];  s[i] = (accumulation \* 255) / pixelCount;  r[i] = 0;  }    grayPtr = (Byte\*)((void\*)grayBD->Scan0);  for (int y = 0; y < grayBD->Height; y++) {  for (int x = 0; x < grayBD->Width; x++) {  resultPtr[0] = resultPtr[1] = resultPtr[2] = s[grayPtr[0]];  r[resultPtr[0]]++;  grayPtr += 3;  resultPtr += 3;  }  grayPtr += grayByteskip;  resultPtr += resultByteskip;  }  /\*\* using to draw histogram \*\*/  for (int i = 0; i < 256; i++) HELevelCount[i] = r[i];  histogramMax\_forLocal = histogramMax;  result->UnlockBits(resultBD);  gray->UnlockBits(grayBD);  return result;  } |

* + 1. 局部處理

|  |
| --- |
| private: Bitmap^ HistogramEqualized\_Local(Bitmap^ gray) {  // nn\*nn range in local enhancement  int nn = (int)LocalRange\_numericUpDown->Value;  int n = (nn - 1) / 2;// the distance of the center point to border    Bitmap^ result = gcnew Bitmap(gray->Width, gray->Height, gray->PixelFormat); // HE image  BitmapData^ grayBD = gray->LockBits(Rectangle(0, 0, gray->Width, gray->Height)  , ImageLockMode::ReadWrite, gray->PixelFormat);  BitmapData^ resultBD = result->LockBits(Rectangle(0, 0, result->Width, result->Height)  , ImageLockMode::ReadOnly, result->PixelFormat);  /\* Image maybe have unknown information \*/  int grayByteskip = grayBD->Stride - grayBD->Width \* 3;  int resultByteskip = resultBD->Stride - resultBD->Width \* 3;  Byte\* grayPtr = (Byte\*)((void\*)grayBD->Scan0);  Byte\* resultPtr = (Byte\*)((void\*)resultBD->Scan0);  for (int y = 0; y < grayBD->Height; y++) {  for (int x = 0; x < grayBD->Width; x++) {  int r[256] = { 0 }; // count each level(0 ~ 255)'s amount  int s[256] = { 0 }; // equalization  int pixelCount = 0;  for (int i = 0; i < 256; i++) r[i] = s[i] = 0;  // run n\*n range  for (int range\_y = n \* -1; range\_y < n; range\_y++) {  for (int range\_x = n \* -1; range\_x < n; range\_x++) {  if (x + range\_x >= 0 && x + range\_x < grayBD->Width  && y + range\_y >= 0 && y + range\_y < grayBD->Height)  { // avoid out of range  Byte\* ptr = grayPtr + range\_x \* 3 + range\_y \* grayBD->Stride;  r[ptr[0]]++;  pixelCount++;  }  }  }  int accumulation = 0; // accumulation before current level  for (int i = 0; i < 256; i++) // normalization {  accumulation += r[i];  s[i] = (accumulation \* 255) / pixelCount;  }  /\*\* using to draw histogram \*\*/  if (s[grayPtr[0]] > histogramMax\_forLocal) histogramMax\_forLocal = s[grayPtr[0]];  if (GrayLevelCount[grayPtr[0]] > histogramMax)  histogramMax = GrayLevelCount[grayPtr[0]];  GrayLevelCount[grayPtr[0]]++;  HELevelCount[grayPtr[0]] = s[grayPtr[0]];  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // only change the center point's value  resultPtr[0] = resultPtr[1] = resultPtr[2] = s[grayPtr[0]];  grayPtr += 3;  resultPtr += 3;  }  grayPtr += grayByteskip;  resultPtr += resultByteskip;  }  result->UnlockBits(resultBD);  gray->UnlockBits(grayBD);  return result;  } |