



K-means Clustering based Inpainting Tool

전기전자공학부 201810909 이재현

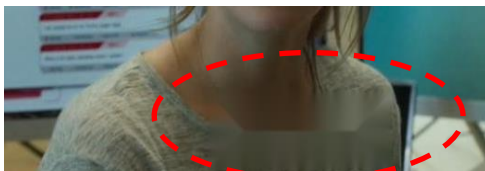
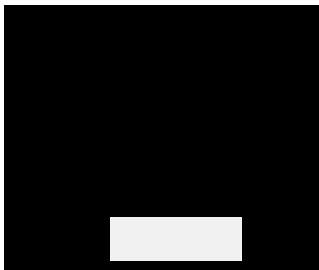
12. 09. 금

Where we are(Lec11.)

Input



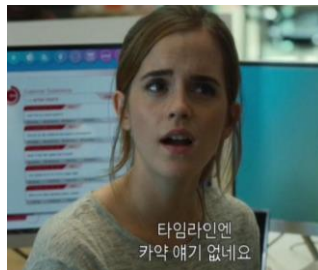
Inpaint Mask



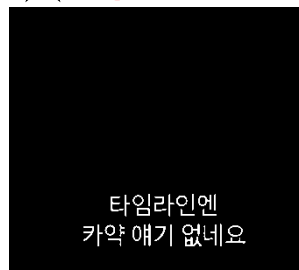
부자연스러움!

Where we want to be

Input

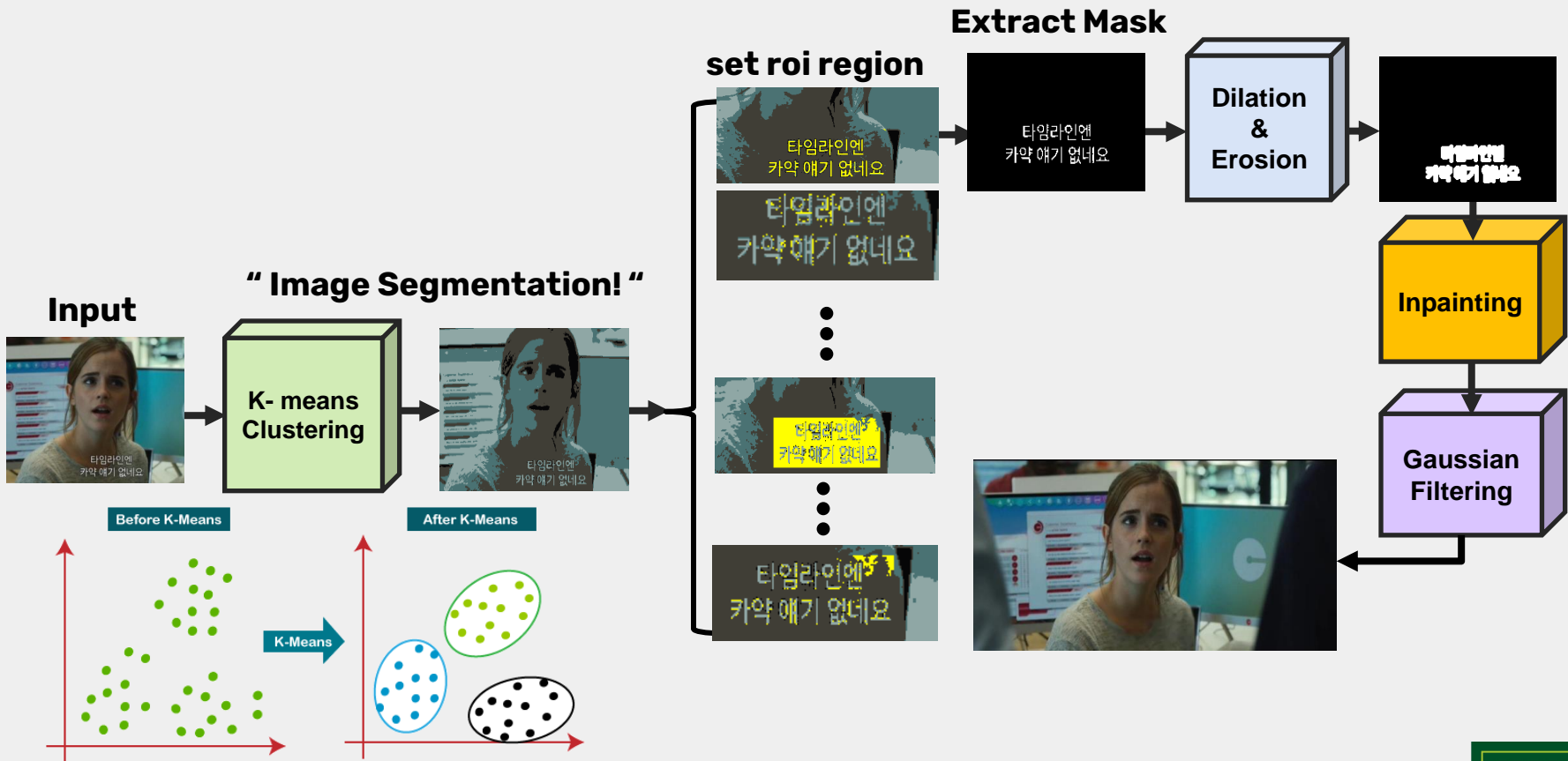


★ My Mask

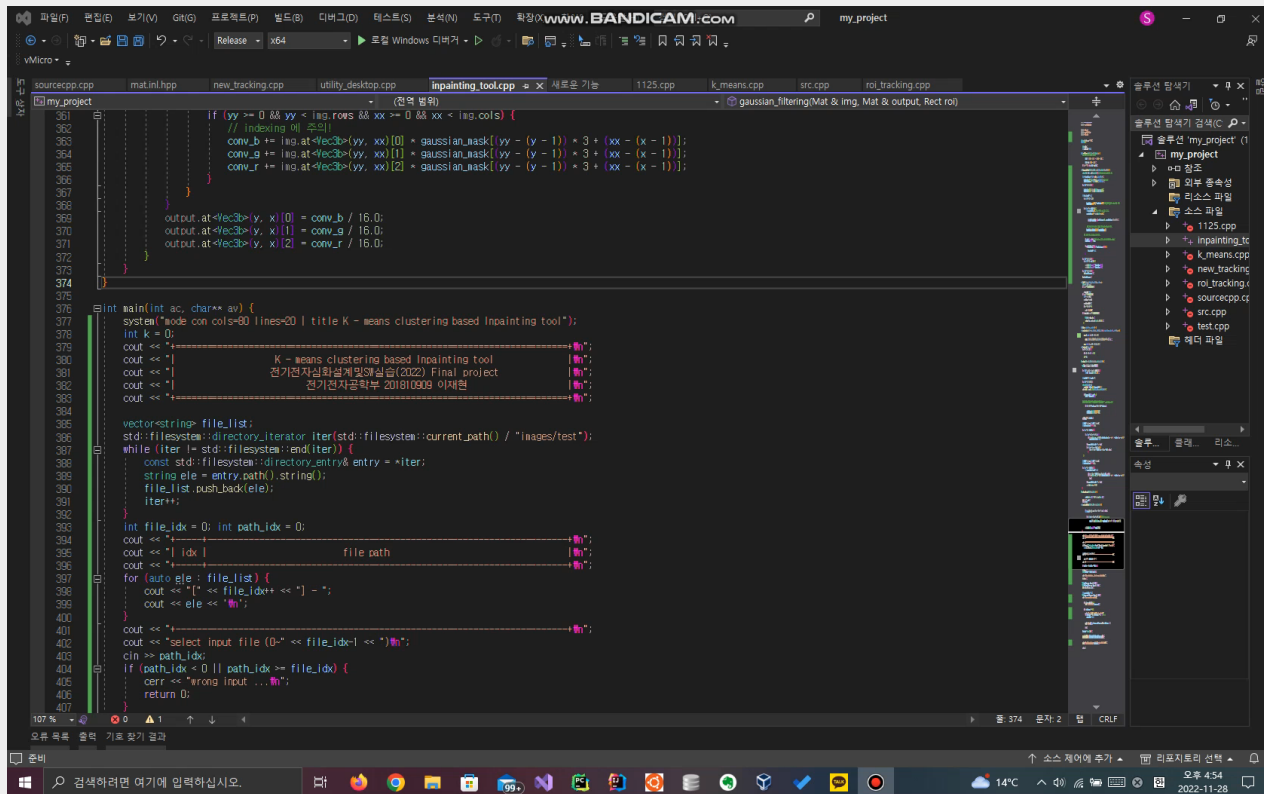


훨씬 자연스러움!

Program Architecture...

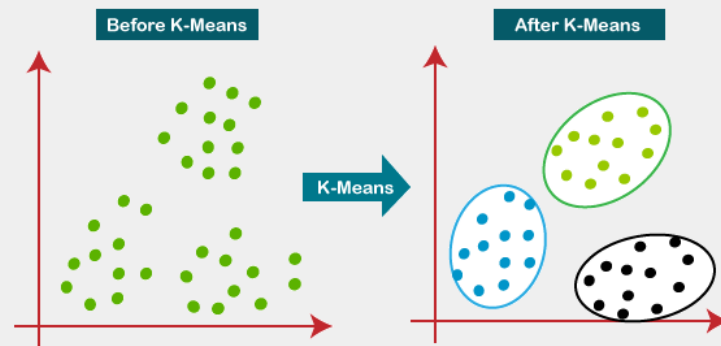


Demo



```
sourcecpp.cpp  mat.inl.hpp  new_tracking.cpp  utility_desktop.cpp  inpainting_tool.cpp  새로운 기능  1125.cpp  k_means.cpp  src.cpp  roi_tracking.cpp
my_project
(전역 범위)
gaussian_filtering(Mat &img, Mat &output, Rect roi)
361 // indexing에 주의!
362 if (yy >= 0 && yy < img.rows && xx >= 0 && xx < img.cols) {
363     conv_b += lag.at<Vec3b>(yy, xx)[0] * gaussian_mask[yy - (y - 1)] * 3 + (xx - (x - 1));
364     conv_g += lag.at<Vec3b>(yy, xx)[1] * gaussian_mask[yy - (y - 1)] * 3 + (xx - (x - 1));
365     conv_r += lag.at<Vec3b>(yy, xx)[2] * gaussian_mask[yy - (y - 1)] * 3 + (xx - (x - 1));
366 }
367
368
369 output.at<Vec3b>(y, x)[0] = conv_b / 16.0;
370 output.at<Vec3b>(y, x)[1] = conv_g / 16.0;
371 output.at<Vec3b>(y, x)[2] = conv_r / 16.0;
372 }
373
374
375
376 int main(int ac, char** av) {
377     system("mode con cols=80 lines=20 | title K - means clustering based inpainting tool");
378     int k = 0;
379     cout << "
380     K - means clustering based inpainting tool
381     전기전자심화설계및실습(2022) Final project
382     전기전자공학부 2018(09)9 이재원
383     " << endl;
384
385     vector<string> file_list;
386     std::filesystem::directory_iterator iter(std::filesystem::current_path() / "images/test");
387     while (iter != std::filesystem::end(iter)) {
388         const std::filesystem::directory_entry& entry = *iter;
389         string ele = entry.path().string();
390         file_list.push_back(ele);
391         iter++;
392     }
393
394     int file_idx = 0; int path_idx = 0;
395     cout << "
396     |
397     | idx |
398     |
399     | file path
400     |
401     |
402     |
403     |
404     |
405     |
406     |
407     |
408     |
409     |
410     |
411     |
412     |
413     |
414     |
415     |
416     |
417     |
418     |
419     |
420     |
421     |
422     |
423     |
424     |
425     |
426     |
427     |
428     |
429     |
430     |
431     |
432     |
433     |
434     |
435     |
436     |
437     |
438     |
439     |
440     |
441     |
442     |
443     |
444     |
445     |
446     |
447     |
448     |
449     |
450     |
451     |
452     |
453     |
454     |
455     |
456     |
457     |
458     |
459     |
460     |
461     |
462     |
463     |
464     |
465     |
466     |
467     |
468     |
469     |
470     |
471     |
472     |
473     |
474     |
475     |
476     |
477     |
478     |
479     |
480     |
481     |
482     |
483     |
484     |
485     |
486     |
487     |
488     |
489     |
490     |
491     |
492     |
493     |
494     |
495     |
496     |
497     |
498     |
499     |
500     |
501     |
502     |
503     |
504     |
505     |
506     |
507     |
508     |
509     |
510     |
511     |
512     |
513     |
514     |
515     |
516     |
517     |
518     |
519     |
520     |
521     |
522     |
523     |
524     |
525     |
526     |
527     |
528     |
529     |
530     |
531     |
532     |
533     |
534     |
535     |
536     |
537     |
538     |
539     |
540     |
541     |
542     |
543     |
544     |
545     |
546     |
547     |
548     |
549     |
550     |
551     |
552     |
553     |
554     |
555     |
556     |
557     |
558     |
559     |
560     |
561     |
562     |
563     |
564     |
565     |
566     |
567     |
568     |
569     |
570     |
571     |
572     |
573     |
574     |
575     |
576     |
577     |
578     |
579     |
580     |
581     |
582     |
583     |
584     |
585     |
586     |
587     |
588     |
589     |
590     |
591     |
592     |
593     |
594     |
595     |
596     |
597     |
598     |
599     |
600     |
601     |
602     |
603     |
604     |
605     |
606     |
607     |
608     |
609     |
610     |
611     |
612     |
613     |
614     |
615     |
616     |
617     |
618     |
619     |
620     |
621     |
622     |
623     |
624     |
625     |
626     |
627     |
628     |
629     |
630     |
631     |
632     |
633     |
634     |
635     |
636     |
637     |
638     |
639     |
640     |
641     |
642     |
643     |
644     |
645     |
646     |
647     |
648     |
649     |
650     |
651     |
652     |
653     |
654     |
655     |
656     |
657     |
658     |
659     |
660     |
661     |
662     |
663     |
664     |
665     |
666     |
667     |
668     |
669     |
670     |
671     |
672     |
673     |
674     |
675     |
676     |
677     |
678     |
679     |
680     |
681     |
682     |
683     |
684     |
685     |
686     |
687     |
688     |
689     |
690     |
691     |
692     |
693     |
694     |
695     |
696     |
697     |
698     |
699     |
700     |
701     |
702     |
703     |
704     |
705     |
706     |
707     |
708     |
709     |
710     |
711     |
712     |
713     |
714     |
715     |
716     |
717     |
718     |
719     |
720     |
721     |
722     |
723     |
724     |
725     |
726     |
727     |
728     |
729     |
730     |
731     |
732     |
733     |
734     |
735     |
736     |
737     |
738     |
739     |
740     |
741     |
742     |
743     |
744     |
745     |
746     |
747     |
748     |
749     |
750     |
751     |
752     |
753     |
754     |
755     |
756     |
757     |
758     |
759     |
760     |
761     |
762     |
763     |
764     |
765     |
766     |
767     |
768     |
769     |
770     |
771     |
772     |
773     |
774     |
775     |
776     |
777     |
778     |
779     |
780     |
781     |
782     |
783     |
784     |
785     |
786     |
787     |
788     |
789     |
790     |
791     |
792     |
793     |
794     |
795     |
796     |
797     |
798     |
799     |
800     |
801     |
802     |
803     |
804     |
805     |
806     |
807     |
808     |
809     |
810     |
811     |
812     |
813     |
814     |
815     |
816     |
817     |
818     |
819     |
820     |
821     |
822     |
823     |
824     |
825     |
826     |
827     |
828     |
829     |
830     |
831     |
832     |
833     |
834     |
835     |
836     |
837     |
838     |
839     |
840     |
841     |
842     |
843     |
844     |
845     |
846     |
847     |
848     |
849     |
850     |
851     |
852     |
853     |
854     |
855     |
856     |
857     |
858     |
859     |
860     |
861     |
862     |
863     |
864     |
865     |
866     |
867     |
868     |
869     |
870     |
871     |
872     |
873     |
874     |
875     |
876     |
877     |
878     |
879     |
880     |
881     |
882     |
883     |
884     |
885     |
886     |
887     |
888     |
889     |
890     |
891     |
892     |
893     |
894     |
895     |
896     |
897     |
898     |
899     |
900     |
901     |
902     |
903     |
904     |
905     |
906     |
907     |
908     |
909     |
910     |
911     |
912     |
913     |
914     |
915     |
916     |
917     |
918     |
919     |
920     |
921     |
922     |
923     |
924     |
925     |
926     |
927     |
928     |
929     |
930     |
931     |
932     |
933     |
934     |
935     |
936     |
937     |
938     |
939     |
940     |
941     |
942     |
943     |
944     |
945     |
946     |
947     |
948     |
949     |
950     |
951     |
952     |
953     |
954     |
955     |
956     |
957     |
958     |
959     |
960     |
961     |
962     |
963     |
964     |
965     |
966     |
967     |
968     |
969     |
970     |
971     |
972     |
973     |
974     |
975     |
976     |
977     |
978     |
979     |
980     |
981     |
982     |
983     |
984     |
985     |
986     |
987     |
988     |
989     |
990     |
991     |
992     |
993     |
994     |
995     |
996     |
997     |
998     |
999     |
1000    |
1001    |
1002    |
1003    |
1004    |
1005    |
1006    |
1007    |
1008    |
1009    |
1010    |
1011    |
1012    |
1013    |
1014    |
1015    |
1016    |
1017    |
1018    |
1019    |
1020    |
1021    |
1022    |
1023    |
1024    |
1025    |
1026    |
1027    |
1028    |
1029    |
1030    |
1031    |
1032    |
1033    |
1034    |
1035    |
1036    |
1037    |
1038    |
1039    |
1040    |
1041    |
1042    |
1043    |
1044    |
1045    |
1046    |
1047    |
1048    |
1049    |
1050    |
1051    |
1052    |
1053    |
1054    |
1055    |
1056    |
1057    |
1058    |
1059    |
1060    |
1061    |
1062    |
1063    |
1064    |
1065    |
1066    |
1067    |
1068    |
1069    |
1070    |
1071    |
1072    |
1073    |
1074    |
1075    |
1076    |
1077    |
1078    |
1079    |
1080    |
1081    |
1082    |
1083    |
1084    |
1085    |
1086    |
1087    |
1088    |
1089    |
1090    |
1091    |
1092    |
1093    |
1094    |
1095    |
1096    |
1097    |
1098    |
1099    |
1100    |
1101    |
1102    |
1103    |
1104    |
1105    |
1106    |
1107    |
1108    |
1109    |
1110    |
1111    |
1112    |
1113    |
1114    |
1115    |
1116    |
1117    |
1118    |
1119    |
1120    |
1121    |
1122    |
1123    |
1124    |
1125    |
1126    |
1127    |
1128    |
1129    |
1130    |
1131    |
1132    |
1133    |
1134    |
1135    |
1136    |
1137    |
1138    |
1139    |
1140    |
1141    |
1142    |
1143    |
1144    |
1145    |
1146    |
1147    |
1148    |
1149    |
1150    |
1151    |
1152    |
1153    |
1154    |
1155    |
1156    |
1157    |
1158    |
1159    |
1160    |
1161    |
1162    |
1163    |
1164    |
1165    |
1166    |
1167    |
1168    |
1169    |
1170    |
1171    |
1172    |
1173    |
1174    |
1175    |
1176    |
1177    |
1178    |
1179    |
1180    |
1181    |
1182    |
1183    |
1184    |
1185    |
1186    |
1187    |
1188    |
1189    |
1190    |
1191    |
1192    |
1193    |
1194    |
1195    |
1196    |
1197    |
1198    |
1199    |
1200    |
1201    |
1202    |
1203    |
1204    |
1205    |
1206    |
1207    |
1208    |
1209    |
1210    |
1211    |
1212    |
1213    |
1214    |
1215    |
1216    |
1217    |
1218    |
1219    |
1220    |
1221    |
1222    |
1223    |
1224    |
1225    |
1226    |
1227    |
1228    |
1229    |
1230    |
1231    |
1232    |
1233    |
1234    |
1235    |
1236    |
1237    |
1238    |
1239    |
1240    |
1241    |
1242    |
1243    |
1244    |
1245    |
1246    |
1247    |
1248    |
1249    |
1250    |
1251    |
1252    |
1253    |
1254    |
1255    |
1256    |
1257    |
1258    |
1259    |
1260    |
1261    |
1262    |
1263    |
1264    |
1265    |
1266    |
1267    |
1268    |
1269    |
1270    |
1271    |
1272    |
1273    |
1274    |
1275    |
1276    |
1277    |
1278    |
1279    |
1280    |
1281    |
1282    |
1283    |
1284    |
1285    |
1286    |
1287    |
1288    |
1289    |
1290    |
1291    |
1292    |
1293    |
1294    |
1295    |
1296    |
1297    |
1298    |
1299    |
1300    |
1301    |
1302    |
1303    |
1304    |
1305    |
1306    |
1307    |
1308    |
1309    |
1310    |
1311    |
1312    |
1313    |
1314    |
1315    |
1316    |
1317    |
1318    |
1319    |
1320    |
1321    |
1322    |
1323    |
1324    |
1325    |
1326    |
1327    |
1328    |
1329    |
1330    |
1331    |
1332    |
1333    |
1334    |
1335    |
1336    |
1337    |
1338    |
1339    |
1340    |
1341    |
1342    |
1343    |
1344    |
1345    |
1346    |
1347    |
1348    |
1349    |
1350    |
1351    |
1352    |
1353    |
1354    |
1355    |
1356    |
1357    |
1358    |
1359    |
1360    |
1361    |
1362    |
1363    |
1364    |
1365    |
1366    |
1367    |
1368    |
1369    |
1370    |
1371    |
1372    |
1373    |
1374    |
1375    |
1376    |
1377    |
1378    |
1379    |
1380    |
1381    |
1382    |
1383    |
1384    |
1385    |
1386    |
1387    |
1388    |
1389    |
1390    |
1391    |
1392    |
1393    |
1394    |
1395    |
1396    |
1397    |
1398    |
1399    |
1400    |
1401    |
1402    |
1403    |
1404    |
1405    |
1406    |
1407    |
1408    |
1409    |
1410    |
1411    |
1412    |
1413    |
1414    |
1415    |
1416    |
1417    |
1418    |
1419    |
1420    |
1421    |
1422    |
1423    |
1424    |
1425    |
1426    |
1427    |
1428    |
1429    |
1430    |
1431    |
1432    |
1433    |
1434    |
1435    |
1436    |
1437    |
1438    |
1439    |
1440    |
1441    |
1442    |
1443    |
1444    |
1445    |
1446    |
1447    |
1448    |
1449    |
1450    |
1451    |
1452    |
1453    |
1454    |
1455    |
1456    |
1457    |
1458    |
1459    |
1460    |
1461    |
1462    |
1463    |
1464    |
1465    |
1466    |
1467    |
1468    |
1469    |
1470    |
1471    |
1472    |
1473    |
1474    |
1475    |
1476    |
1477    |
1478    |
1479    |
1480    |
1481    |
1482    |
1483    |
1484    |
1485    |
1486    |
1487    |
1488    |
1489    |
1490    |
1491    |
1492    |
1493    |
1494    |
1495    |
1496    |
1497    |
1498    |
1499    |
1500    |
1501    |
1502    |
1503    |
1504    |
1505    |
1506    |
1507    |
1508    |
1509    |
1510    |
1511    |
1512    |
1513    |
1514    |
1515    |
1516    |
1517    |
1518    |
1519    |
1520    |
1521    |
1522    |
1523    |
1524    |
1525    |
1526    |
1527    |
1528    |
1529    |
1530    |
1531    |
1532    |
1533    |
1534    |
1535    |
1536    |
1537    |
1538    |
1539    |
1540    |
1541    |
1542    |
1543    |
1544    |
1545    |
1546    |
1547    |
1548    |
1549    |
1550    |
1551    |
1552    |
1553    |
1554    |
1555    |
1556    |
1557    |
1558    |
1559    |
1560    |
1561    |
1562    |
1563    |
1564    |
1565    |
1566    |
1567    |
1568    |
1569    |
1570    |
1571    |
1572    |
1573    |
1574    |
1575    |
1576    |
1577    |
1578    |
1579    |
1580    |
1581    |
1582    |
1583    |
1584    |
1585    |
1586    |
1587    |
1588    |
1589    |
1590    |
1591    |
1592    |
1593    |
1594    |
1595    |
1596    |
1597    |
1598    |
1599    |
1600    |
1601    |
1602    |
1603    |
1604    |
1605    |
1606    |
1607    |
1608    |
1609    |
1610    |
1611    |
1612    |
1613    |
1614    |
1615    |
1616    |
1617    |
1618    |
1619    |
1620    |
1621    |
1622    |
1623    |
1624    |
1625    |
1626    |
1627    |
1628    |
1629    |
1630    |
1631    |
1632    |
1633    |
1634    |
1635    |
1636    |
1637    |
1638    |
1639    |
1640    |
1641    |
1642    |
1643    |
1644    |
1645    |
1646    |
1647    |
1648    |
1649    |
1650    |
1651    |
1652    |
1653    |
1654    |
1655    |
1656    |
1657    |
1658    |
1659    |
1660    |
1661    |
1662    |
1663    |
1664    |
1665    |
1666    |
1667    |
1668    |
1669    |
1670    |
1671    |
1672    |
1673    |
1674    |
1675    |
1676    |
1677    |
1678    |
1679    |
1680    |
1681    |
1682    |
1683    |
1684    |
1685    |
1686    |
1687    |
1688    |
1689    |
1690    |
1691    |
1692    |
1693    |
1694    |
1695    |
1696    |
1697    |
1698    |
1699    |
1700    |
1701    |
1702    |
1703    |
1704    |
1705    |
1706    |
1707    |
1708    |
1709    |
1710    |
1711    |
1712    |
1713    |
1714    |
1715    |
1716    |
1717    |
1718    |
1719    |
1720    |
1721    |
1722    |
1723    |
1724    |
1725    |
1726    |
1727    |
1728    |
1729    |
1730    |
1731    |
1732    |
1733    |
1734    |
1735    |
1736    |
1737    |
1738    |
1739    |
1740    |
1741    |
1742    |
1743    |
1744    |
1745    |
1746    |
1747    |
1748    |
1749    |
1750    |
1751    |
1752    |
1753    |
1754    |
1755    |
1756    |
1757    |
1758    |
1759    |
1760    |
1761    |
1762    |
1763    |
1764    |
1765    |
1766    |
1767    |
1768    |
1769    |
1770    |
1771    |
1772    |
1773    |
1774    |
1775    |
1776    |
1777    |
1778    |
1779    |
1780    |
1781    |
1782    |
1783    |
1784    |
1785    |
1786    |
1787    |
1788    |
1789    |
1790    |
1791    |
1792    |
1793    |
1794    |
1795    |
1796    |
1797    |
1798    |
1799    |
1800    |
1801    |
1802    |
1803    |
1804    |
1805    |
1806    |
1807    |
1808    |
1809    |
1810    |
1811    |
1812    |
1813    |
1814    |
1815    |
1816    |
1817    |
1818    |
1819    |
1820    |
1821    |
1822    |
1823    |
1824    |
1825    |
1826    |
1827    |
1828    |
1829    |
1830    |
1831    |
1832    |
1833    |
1834    |
1835    |
1836    |
1837    |
1838    |
1839    |
1840    |
1841    |
1842    |
1843    |
1844    |
1845    |
1846    |
1847    |
1848    |
1849    |
1850    |
1851    |
1852    |
1853    |
1854    |
1855    |
1856    |
1857    |
1858    |
1859    |
1860    |
1861    |
1862    |
1863    |
1864    |
1865    |
1866    |
1867    |
1868    |
1869    |
1870    |
1871    |
1872    |
1873    |
1874    |
1875    |
1876    |
1877    |
1878    |
1879    |
1880    |
1881    |
1882    |
1883    |
1884    |
1885    |
1886    |
1887    |
1888    |
1889    |
1890    |
1891    |
1892    |
1893    |
1894    |
1895    |
1896    |
1897    |
1898    |
1899    |
1900    |
1901    |
1902    |
1903    |
1904    |
1905    |
1906    |
1907    |
1908    |
1909    |
1910    |
1911    |
1912    |
1913    |
1914    |
1915    |
1916    |
1917    |
1918    |
1919    |
1920    |
1921    |
1922    |
1923    |
1924    |
1925    |
1926    |
1927    |
1928    |
1929    |
1930    |
1931    |
1932    |
1933    |
1934    |
1935    |
1936    |
1937    |
1938    |
1939    |
1940    |
1941    |
1942    |
1943    |
1944    |
1945    |
1946    |
1947    |
1948    |
1949    |
1950    |
1951    |
1952    |
1953    |
1954    |
1955    |
1956    |
1957    |
1958    |
1959    |
1960    |
1961    |
1962    |
1963    |
1964    |
1965    |
1966    |
1967    |
1968    |
1969    |
1970    |
1971    |
1972    |
1973    |
1974    |
1975    |
1976    |
1977    |
1978    |
1979    |
1980    |
1981    |
1982    |
1983    |
1984    |
1985    |
1986    |
1987    |
1988    |
1989    |
1990    |
1991    |
1992    |
1993    |
1994    |
1995    |
1996    |
1997    |
1998    |
1999    |
2000    |
2001    |
2002    |
2003    |
2004    |
2005    |
2006    |
2007    |
2008    |
2009    |
2010    |
2011    |
2012    |
2013    |
2014    |
2015    |
2016    |
2017    |
2018    |
2019    |
2020    |
2021    |
2022    |
2023    |
2024    |
2025    |
2026    |
2027    |
2028    |
2029    |
2030    |
2031    |
2032    |
2033    |
2034    |
2035    |
2036    |
2037    |
2038    |
2039    |
2040    |
2041    |
2042    |
2043    |
2044    |
2045    |
2046    |
2047    |
2048    |
2049    |
2050    |
2051    |
2052    |
2053    |
2054    |
2055    |
2056    |
2057    |
2058    |
2059    |
2060    |
2061    |
2062    |
2063    |
2064    |
2065    |
2066    |
2067    |
2068    |
2069    |
2070    |
2071    |
2072    |
2073    |
2074    |
2075    |
2076    |
2077    |
2078    |
2079    |
2080    |
2081    |
2082    |
2083    |
2084    |
2085    |
2086    |
2087    |
2088    |
2089    |
2090    |
2091    |
2092    |
2093    |
2094    |
2095    |
2096    |
2097    |
2098    |
2099    |
2100    |
2101    |
2102    |
2103    |
2104    |
2105    |
2106    |
2107    |
2108    |
2109    |
2110    |
2111    |
2112    |
2113    |
2114    |
2115    |
2116    |
2117    |
2118    |
2119    |
2120    |
2121    |
2122    |
2123    |
2124    |
2125    |
2126    |
2127    |
2128    |
2129    |
2130    |
2131    |
2132    |
2133    |
2134    |
2135    |
2136    |
2137
```

k-means Clustering



Label 이 없는 data 들을 K 개의 Cluster 로 묶는 알고리즘으로, 각 Cluster 의 평균값(mean)을 활용한다.

$$V = \sum_{i=1}^k \sum_{x_j \in S_i} |x_j - \mu_i|^2$$

EE Experiments (Computer Vision)
(Lecture Note 10)

Image Segmentation

2022. 11. 18

Prof. Wonjun Kim
School of Electrical and Electronics Engineering

K-means Clustering (1/4)

- Unsupervised learning
- We have only data (no labels!) → how to assign labels?
- Minimize the following cost function:

$$\min_{\{b_j\}} \sum_{i=1}^N \sum_{j=1}^K w_{ij} \|x_i - b_j\|^2, \sum_j w_{ij} = 1$$
 - x_i : the i -th feature vector (e.g., RGB color point)
 - b_j : the j -th center feature
 - N, K : number of data and centers, respectively
 - w_{ij} : the binary weight (belong to → 1, otherwise → 0)
- ⚠ This problem is hard to be solved due to the binary weight w_{ij}
 - We need to check all the combinations (jointly optimize)!
 - Fix one parameter and update (alternative optimization)

K-means Clustering (2/4)

- Algorithm summary: think about weak points!
- Algorithm flow (very simple!)
 - Randomly set the cluster K centers
 - All the pixels are compared with K clusters and assign labels

$$l_i = \arg \min_j \|x_i - b_j\| \quad (\text{labeling})$$
 - Compute the new center with new labels

$$\hat{b}_j = \frac{1}{\text{Card}(J)} \sum_{i \in J} x_i \quad (\text{compute the mean vector})$$
 - If the difference between centers of current and previous state, then stop (labeling finished), otherwise, go to the step 2) and repeat

$$\text{if } D < \epsilon \rightarrow \text{STOP, where } D = \sum_{j=1}^K \|b_j^t - b_j^{t-1}\|$$

K-means Clustering (3/4)

- Implementation using C programming
- The number of centers (i.e., K) is user-settable
 - ⚠ Feature vector: RGB color point (i.e., three-dimensional feature)

Decreasing the number of cluster centers

In Lecture note 10.

Implementation of K - means Clustering(0)

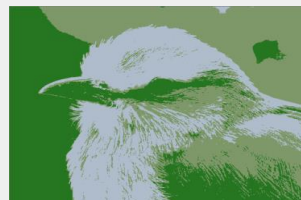
Input: 클러스터의 개수(k) 와 n 개의 데이터 object 를 포함하는 집합(Set)

Output: k 개의 클러스터

1. 데이터 오브젝트 집합 D에서 k 개의 데이터 오브젝트를 임의로 추출하고, 이 데이터 오브젝트들을 각 클러스터의 중심 (centroid)으로 설정한다. (초기값 설정)
2. 집합 D의 각 데이터 오브젝트들에 대해 k 개의 클러스터 중심 오브젝트와의 거리를 각각 구하고, 각 데이터 오브젝트가 어느 중심점 (centroid) 와 가장 유사도가 높은지 알아낸다. 그리고 그렇게 찾아낸 중심점으로 각 데이터 오브젝트들을 할당한다.
3. 클러스터의 중심점을 다시 계산한다. 즉, 2에서 재할당된 클러스터들을 기준으로 중심점을 다시 계산한다.
4. 각 데이터 오브젝트의 소속 클러스터가 바뀌지 않을 때까지 2, 3 과정을 반복한다.



k=2



k=3



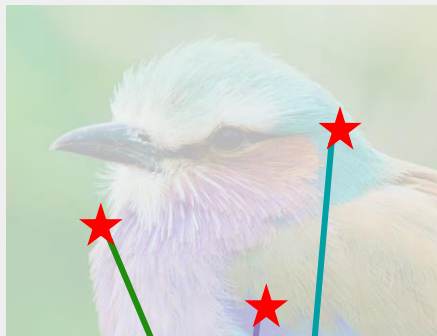
k=4



k=8

Implementation of K – means Clustering(1)

Random Choice!



초기 Centroid 값은 input
image 내의 random 한
좌표를 선택한 뒤, 해당
좌표의 B,G,R 값으로 설정함

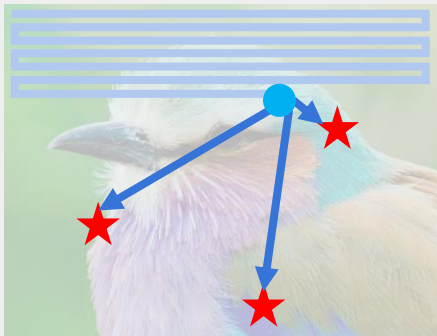
K	1	2	3
R	A	D	G
G	B	E	H
B	C	F	I

```
void K_means_clustering(Mat& input, Mat& result, int k, string Distance_Metric) {  
    int width = input.cols;  
    int height = input.rows;  
    vector<Point>* st = new std::vector<Point>[k]; // clustered pixel point 를 담을 배열  
    srand(time(NULL)); // for random initialization  
  
    float** k_centroid_arr = (float**)calloc(k, sizeof(float*));  
    float** k_centroid_arr_save = (float**)calloc(k, sizeof(float*));  
    for (int i = 0; i < k; i++) {  
        k_centroid_arr[i] = (float*)calloc(3, sizeof(float)); // bgr channel  
        k_centroid_arr_save[i] = (float*)calloc(3, sizeof(float));  
    }  
  
    // random initialization  
    for (int i = 0; i < k; i++) {  
        int pos_x = rand() % width;  
        int pos_y = rand() % height;  
  
        k_centroid_arr[i][0] = input.at<Vec3b>(pos_y, pos_x)[0];  
        k_centroid_arr[i][1] = input.at<Vec3b>(pos_y, pos_x)[1];  
        k_centroid_arr[i][2] = input.at<Vec3b>(pos_y, pos_x)[2];  
    }  
}
```

“ Centroid Initialization ! ”

Implementation of K - means Clustering(2)

pixel by pixel



“ Calculate Distance ! ”

K	1	2	3		?	K
R	A	D	G		x	R
G	B	E	H		y	G
B	C	F	I		z	B

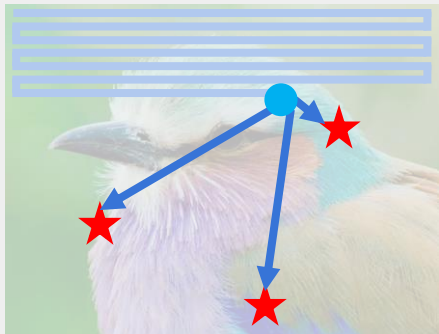
각각의 pixel 에 대하여 어느 centroid 와 가장 distance 가 작은지(유사한 지) 계산하여, 해당 cluster 로 분류함.

```
bool have_to_update = true;
while (have_to_update) {
    have_to_update = false;
    for (int i = 0; i < k; i++)
        st[i].clear();
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < 3; j++) {
            k_centroid_arr_save[i][j] = k_centroid_arr[i][j]; // 현재(업데이트 이전) center 정보를 저장함
            k_centroid_arr[i][j] = 0;
        }
    }

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int present_pixel_arr[3]; // (x,y) 에서의 r,g,b 값을 담을 배열
            present_pixel_arr[0] = input.at<Vec3b>(y, x)[0];
            present_pixel_arr[1] = input.at<Vec3b>(y, x)[1];
            present_pixel_arr[2] = input.at<Vec3b>(y, x)[2];

            float min_distance = INT_MAX;
            int cluster_idx = 0;
            for (int i = 0; i < k; i++) {
                float dist = get_distance(k_centroid_arr_save[i], present_pixel_arr, Distance_Metric);
                if (dist < min_distance) {
                    min_distance = dist;
                    cluster_idx = i;
                }
            }
            // 최소거리를 가지는 cluster 을 찾았으면 해당 cluster 에 좌표를 저장함.
            st[cluster_idx].push_back(Point(x, y));
            // 그리고 픽셀 rgb 값을 누적시킴.
            k_centroid_arr[cluster_idx][0] += present_pixel_arr[0];
            k_centroid_arr[cluster_idx][1] += present_pixel_arr[1];
            k_centroid_arr[cluster_idx][2] += present_pixel_arr[2];
        }
    }
}
// 위 단계를 지나면, clustering 이 한번 된 것.
```


Implementation of K – means Clustering(3)

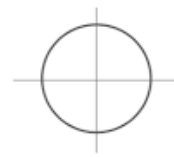


```
template <class T1, class T2>
float get_distance(T1* obj1, T2* obj2, String type) {
    //RGB pixel 에 대한 distance 를 계산함
    float dist = 0;
    for (int i = 0; i < 3; i++) {
        if (type == "L2") {
            dist += (obj1[i] - obj2[i]) * (obj1[i] - obj2[i]);
        }
        if (type == "L1") {
            dist += abs(obj1[i] - obj2[i]);
        }
    }
    if (type == "L2") return sqrt(dist);
    else return dist;
}
```

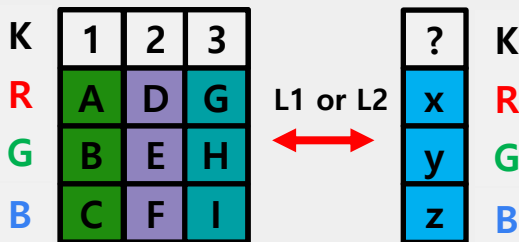
L1 (Manhattan) distance L2 (Euclidean) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$



distance 는 L1 – norm 혹은 L2 – norm 을 사용하여 구할 수 있도록 구현.
(수행시간 측면에서 L1 < L2)



k-means clustering 알고리즘 평균 수행 시간 (10회)		
	L1 - norm	L2 – norm
k = 3	3.904 sec	4.724 sec
k = 6	7.982 sec	11.059 sec

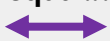
Implementation of K – means Clustering(4)



previous centroid

K	1	2	3
R	A	D	G
G	B	E	H
B	C	F	I

equal..?



present centroid

K	1	2	3
R	J	M	P
G	K	N	Q
B	L	O	R

update 이후 previous centroid 와 present centroid 가
같으면 수렴한 것으로 판단하여 clustering 종료.
previous centroid 와 present centroid 가 다르면
centroid 를 updating 하고 다시 한번 clustering

이후 clustering 된 point 들을 result image 와
mapping 하여 최종 이미지 생성

```
// 그러면 centroid 를 updating
for (int i = 0; i < k; i++) {
    int size = st[i].size(); // 각각의 cluster 로 분류된 point 개수를 셈.
    if (size == 0) continue; // 해당 cluster 로 분류된 point 가 하나도 없으면 continue;
    k_centroid_arr[i][0] /= size;
    k_centroid_arr[i][1] /= size;
    k_centroid_arr[i][2] /= size;
}

for (int i = 0; i < k; i++) {
    for (int j = 0; j < 3; j++) {
        if (k_centroid_arr[i][j] != k_centroid_arr_save[i][j]) {
            // 이전 centroid 의 값과 다르면, updating 을 계속함
            // 만약 같으면 optimized 되었다고 판단.
            have_to_update = true;
            break;
        }
    }
}

// Clustering Complete!
// Mapping
for (int i = 0; i < k; i++) {
    int size = st[i].size();
    for (int j = 0; j < size; j++) {
        int x = st[i][j].x;
        int y = st[i][j].y;
        result.at<Vec3b>(y, x)[0] = k_centroid_arr[i][0];
        result.at<Vec3b>(y, x)[1] = k_centroid_arr[i][1];
        result.at<Vec3b>(y, x)[2] = k_centroid_arr[i][2];
    }
}

for (int i = 0; i < k; i++) {
    free(k_centroid_arr[i]);
    free(k_centroid_arr_save[i]);
}

free(k_centroid_arr);
free(k_centroid_arr_save);

delete[] st;
```

Get Mask

```
Mat get_mask(Mat& origin, Mat& clustered_img, Rect roi, int k=0) {
    Mat hsv_img = clustered_img.clone();
    Mat roi_cluster = clustered_img.clone();
    cvtColor(clustered_img, hsv_img, CV_BGR2HSV);
    Mat mask(origin.rows, origin.cols, CV_8UC1);
    mask = Scalar::all(0);

    int frequency_table[256] = { 0 };
    for (int i = roi.y; i < roi.y + roi.height; i++) {
        for (int j = roi.x; j < roi.x + roi.width; j++) {
            int idx = (hsv_img.at<Vec3b>(i, j)[2]);
            frequency_table[idx]++;
        }
    }

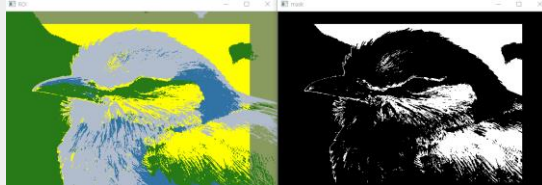
    int mode_label = 0; int max_fre = 0;

    int label_arr[256] = { 0 };
    for (int i = 0; i < 256; i++)
        label_arr[i] = i;

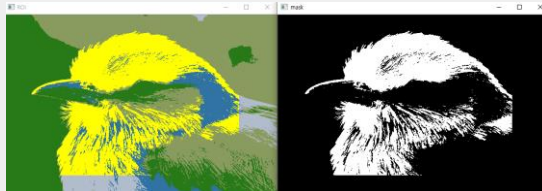
    vector<pair<int, int>> vr;
    for (int i = 0; i < 256; i++) {
        vr.push_back((label_arr[i], frequency_table[i]));
    }
    sort(vr.begin(), vr.end(), [](pair<int, int> a, pair<int, int> b) {
        return a.second > b.second;
    });

    for (int i = 0; i < 256; i++) {
        if (frequency_table[i] > max_fre) {
            max_fre = frequency_table[i];
            mode_label = i;
        }
    }
    mode_label = vr[k].first;
    // roi 내에서 가장 많은 cluster label 은 mode_label 이다.
    //cout << "mode _ label : " << mode_label << " vr[0].first = " << vr[0].first << "\n";
    for (int i = roi.y; i < roi.y + roi.height; i++) {
        for (int j = roi.x; j < roi.x + roi.width; j++) {
            if (i < 0 || i >= hsv_img.rows || j < 0 || j >= hsv_img.cols)
                continue;
            if (hsv_img.at<Vec3b>(i, j)[2] == mode_label) {
                mask.at<uchar>(i, j) = 255;
                roi_cluster.at<Vec3b>(i, j)[0] = 0;
                roi_cluster.at<Vec3b>(i, j)[1] = 255;
                roi_cluster.at<Vec3b>(i, j)[2] = 255;
            }
        }
    }

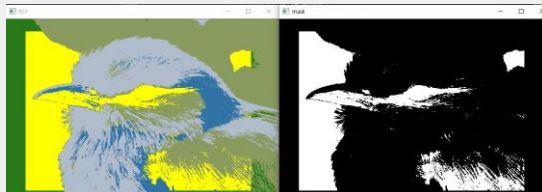
    cv::imshow("ROI", roi_cluster);
    return mask;
}
```



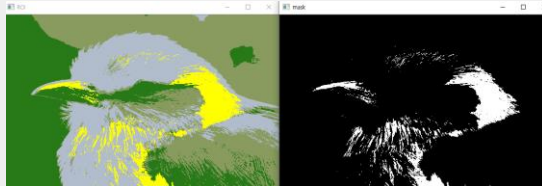
키보드 입력[0]



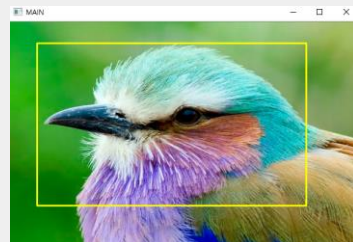
키보드 입력[1]



키보드 입력[2]



키보드 입력[3]

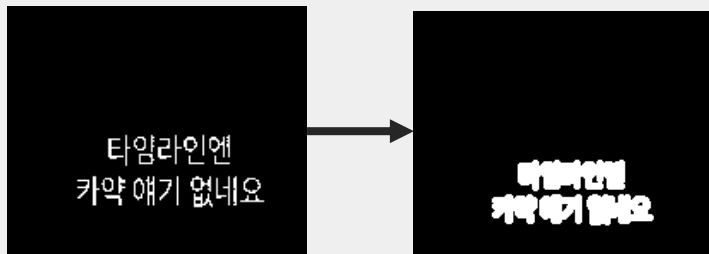


Clustered image 에서 cluster
별로 highlight 하여 roi 를
선택하도록 제안. User 가
선택한 cluster 에 대한
inpainting mask 를 생성

K	0	1	2	3
R	A	D	G	J
G	B	E	H	K
B	C	F	I	L

Dilation and Erosion (Mask post-processing)

Convolution 을 응용하여, Structuring Element 와 OR 연산을 통해 Dilation 을 수행하여 Mask 영역 내부의 hole 을 채우고 AND 연산을 이용하여 Erosion 을 수행해서 확대된 Mask 의 경계영역을 보정

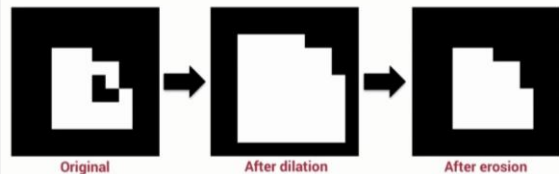


```
void dilation(Mat& input, Mat& result) {
    int height = input.rows; int width = input.cols;
    int size = 3;
    int structuring_element[9] = { 0,1,0,
                                    1,1,1,
                                    0,1,0 };

    int ele_size = 3;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            bool escape_double_loop_flag = false;
            bool fill_flag = false;
            for (int yy = y-1; yy <= y+1; yy++) {
                for (int xx = x-1; xx <= x+1; xx++) {
                    if (yy < 0 || yy >= height || xx < 0 || xx >= width) continue;
                    if (input.at<uchar>(yy, xx) == 255 && structuring_element[(yy - (y-1)) * ele_size + (xx - (x-1))] == 1) {
                        escape_double_loop_flag = true;
                        fill_flag = true;
                        break;
                    }
                }
            }
            if (escape_double_loop_flag == true) break;
            if (fill_flag == true) {
                for (int yy = y; yy < y+size; yy++) {
                    for (int xx = x; xx < x+size; xx++) {
                        if (yy < 0 || yy >= height || xx < 0 || xx >= width) continue;
                        result.at<uchar>(yy, xx) = 255;
                    }
                }
            }
        }
    }
}
```

Structuring Element = {0, 1, 0
1, 1, 1
0, 1, 0}

Dilation then erosion

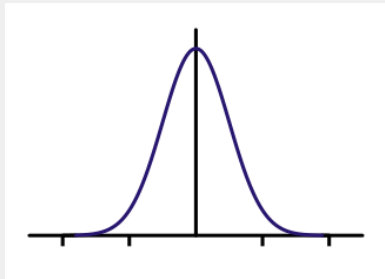


```
void erosion(Mat& input, Mat& result) {
    int height = input.rows; int width = input.cols;
    int size = 3;
    int structuring_element[9] = { 0,1,0,
                                    1,1,1,
                                    0,1,0 };

    int ele_size = 3;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            bool escape_double_loop_flag = true;
            bool fill_flag = true;
            for (int yy = y-1; yy <= y+1; yy++) {
                for (int xx = x-1; xx <= x+1; xx++) {
                    if (yy < 0 || yy >= height || xx < 0 || xx >= width) continue;
                    if (input.at<uchar>(yy, xx) == 255 && structuring_element[(yy - (y-1)) * ele_size + (xx - (x-1))] == 1) {
                        escape_double_loop_flag = false;
                        fill_flag = false;
                        break;
                    }
                }
            }
            if (escape_double_loop_flag == true) break;
            if (fill_flag == true) {
                result.at<uchar>(y, x) = 255;
            }
        }
    }
}
```

Inpainting and Gaussian Filtering

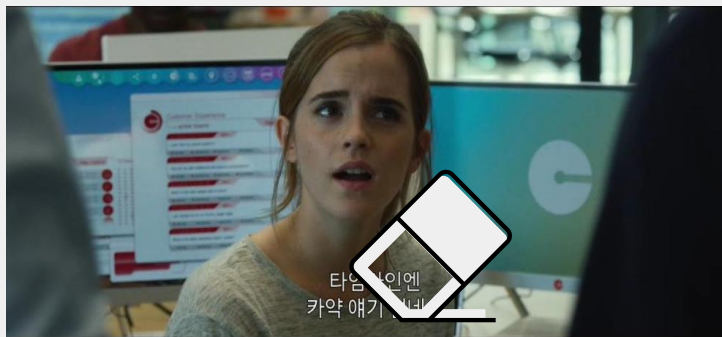
```
void gaussian_filtering(Mat& img, Mat& output, Rect roi) {  
    output = img.clone();  
    int gaussian_mask[9] = { 1,2,1,2,4,2,1,2,1 };  
    for (int y = roi.y-10; y < roi.y+roi.height+10; y++) {  
        for (int x = roi.x-10; x < roi.x+roi.width+10; x++) {  
            if (y < 0 || y >= img.rows || x < 0 || x >= img.cols) continue;  
            float conv_b = 0.0;  
            float conv_g = 0.0;  
            float conv_r = 0.0;  
  
            for (int yy = y - 1; yy <= y + 1; yy++) {  
                for (int xx = x - 1; xx <= x + 1; xx++) { // calc conv_x, conv_y  
                    if (yy >= 0 && yy < img.rows && xx >= 0 && xx < img.cols) {  
                        // indexing 에 주의!  
                        conv_b += img.at<Vec3b>(yy, xx)[0] * gaussian_mask[yy - (y - 1) * 3 + (xx - (x - 1))];  
                        conv_g += img.at<Vec3b>(yy, xx)[1] * gaussian_mask[yy - (y - 1) * 3 + (xx - (x - 1))];  
                        conv_r += img.at<Vec3b>(yy, xx)[2] * gaussian_mask[yy - (y - 1) * 3 + (xx - (x - 1))];  
                    }  
                }  
            }  
  
            output.at<Vec3b>(y, x)[0] = conv_b / 16.0;  
            output.at<Vec3b>(y, x)[1] = conv_g / 16.0;  
            output.at<Vec3b>(y, x)[2] = conv_r / 16.0;  
        }  
    }  
}
```



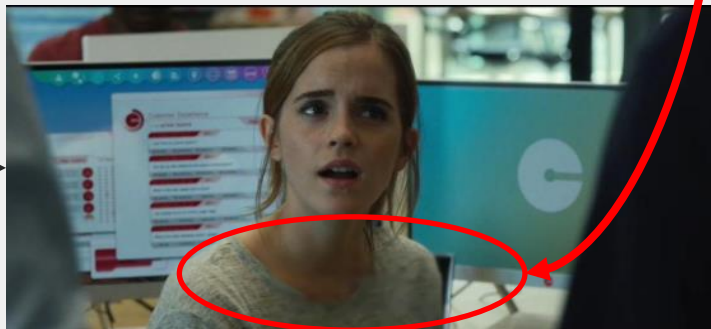
1	2	1
2	4	2
1	2	1

/16

roi 내에서 선택한 Cluster 에 대하여,
Inpainting 을 수행한 뒤, roi 영역에 Gaussian
filtering 을 적용하여 최종 이미지를 생성함.



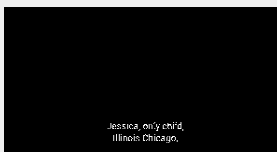
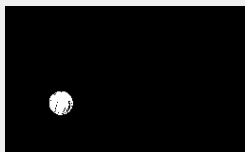
Inpainting!



Gaussian filtering! (using convolution)

전기전자심화설계 및 소프트웨어실습(2022)

K-means Clustering based Inpainting Tool



Thanks!

전기전자공학부 201810909 이재현



Do you have any questions?



전기전자심화설계 및 소프트웨어실습(2022)
Final Project