**SAT4-INFRARED**

**import scipy.io**

**import numpy as np**

**import pandas as pd**

**import sklearn**

**from sklearn.preprocessing import Normalizer**

**import keras**

**from keras.models import Sequential**

**import tensorflow as tf**

**import matplotlib.pyplot as plt**

**import matplotlib.image as implt**

**from sklearn.model\_selection import train\_test\_split**

**from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,Activation,MaxPool2D**

**mat1 = scipy.io.loadmat("sat-4-full.mat")**

**x\_train1=mat1['train\_x']**

**x\_test1=mat1['test\_x']**

**y\_train1=mat1['train\_y']**

**y\_test1=mat1['test\_y']**

**x\_train1=np.array(x\_train1)**

**x\_test1=np.array(x\_test1)**

**def resh(x1,a,b):**

**x2=np.empty([100000,28,28,1])**

**t=0**

**for i in range(a,b):**

**for j in range(0,4):**

**#if(j==0 ):**

**#x2[t,:,:,0]=x1[:,:,j,i]**

**if(j==3):**

**x2[t,:,:,0]=x1[:,:,j,i]**

**t=t+1**

**return x2;**

**k1=np.empty([400000,28,28,1])**

**k1[0:100000,:,:,:]=resh(x\_train1,0,100000)**

**k1[100000:200000,:,:,:]=resh(x\_train1,100000,200000)**

**k1[200000:300000,:,:,:]=resh(x\_train1,200000,300000)**

**k1[300000:400000,:,:,:]=resh(x\_train1,300000,400000)**

**print('-------------------------------------------')**

**print("K shape=",np.shape(k1))**

**z=np.transpose(y\_train1)**

**x\_train,x\_vvalid,y\_train,y\_valid=train\_test\_split(k1,z,test\_size=0.2)**

**x\_train=(x\_train - x\_train.min())/(x\_train.max() - x\_train.min())**

**x\_valid=(x\_vvalid - x\_vvalid.min())/(x\_vvalid.max() - x\_vvalid.min())**

**model = Sequential()**

**model.add(Conv2D(16, (3,3), activation='relu', input\_shape=(28,28,1)))**

**model.add(Conv2D(32, (3,3), activation='relu'))**

**model.add(MaxPool2D(pool\_size=(2,2)))**

**model.add(Dropout(0.5))**

**model.add(Conv2D(32, (3,3), activation='relu'))**

**model.add(Conv2D(64, (3,3), activation='relu'))**

**model.add(MaxPool2D(pool\_size=(2,2)))**

**model.add(Dropout(0.5))**

**model.add(Flatten())**

**model.add(Dense(128, activation='relu'))**

**model.add(Dropout(0.5))**

**model.add(Dense(4, activation='softmax'))**

**model.summary()**

**model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**model.fit(x\_train, y\_train, batch\_size=200, epochs=50, verbose=1, validation\_data=(x\_valid, y\_valid))**

**te=np.empty([100000,28,28,1])**

**te[0:100000,:,:,:]=resh(x\_test1,0,100000)**

**te=(te - te.min())/(te.max() - te.min())**

**y\_test1=np.transpose(y\_test1)**

**loss,accuracy=model.evaluate(te,y\_test1)**

**print("\nLoss: %.2f,Accuracy: %.2f%%" % (loss,accuracy\*100))**

**y\_predict = model.predict(te)**

**y\_exp1 = np.argmax(y\_predict,axis=1)**

**y\_exp=np.argmax(y\_test1,axis=1)**

**from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,confusion\_matrix**

**accuracy=accuracy\_score(y\_exp,y\_exp1)**

**print(accuracy)**

**precision=precision\_score(y\_exp, y\_exp1, average=None)**

**print(precision)**

**recall=recall\_score(y\_exp, y\_exp1, average=None)**

**print(recall)**

**f1score=f1\_score(y\_exp, y\_exp1, average=None)**

**print(f1score)**

**from sklearn.metrics import classification\_report**

**sklearn.metrics.classification\_report(y\_exp, y\_exp1)**

**target\_names = ['class 0', 'class 1', 'class 2','class 3']**

**print(classification\_report(y\_exp, y\_exp1, target\_names=target\_names))**

**import itertools**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn import svm, datasets**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import confusion\_matrix**

**def plot\_confusion\_matrix(cm, classes,**

**normalize=False,**

**title='Confusion matrix',**

**cmap=plt.cm.Blues):**

**"""**

**This function prints and plots the confusion matrix.**

**Normalization can be applied by setting `normalize=True`.**

**"""**

**if normalize:**

**cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]**

**print("Normalized confusion matrix")**

**else:**

**print('Confusion matrix, without normalization')**

**#print(cm)**

**#plt.imshow(cm, interpolation='nearest', cmap=cmap)**

**plt.title(title)**

**#plt.colorbar()**

**tick\_marks = np.arange(len(classes))**

**plt.xticks(tick\_marks, classes, rotation=45)**

**plt.yticks(tick\_marks, classes)**

**fmt = '.2f' if normalize else 'd'**

**thresh = cm.max() / 2.**

**for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):**

**plt.text(j, i, format(cm[i, j], fmt),**

**horizontalalignment="center",**

**color="white" if cm[i, j] > thresh else "black")**

**plt.ylabel('True label')**

**plt.xlabel('Predicted label')**

**plt.tight\_layout()**

**# Compute confusion matrix**

**cnf\_matrix = confusion\_matrix(y\_exp, y\_exp1)**

**np.set\_printoptions(precision=2)**

**# Plot non-normalized confusion matrix**

**fig1 = plt.figure()**

**plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3'],**

**title='Confusion matrix, without normalization')**

**# Plot normalized confusion matrix**

**fig2 = plt.figure()**

**plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3'], normalize=True,**

**title='Normalized confusion matrix')**

**#fig1.savefig("nonnormalized.png")**

**#fig2.savefig("normalized.png")**

**import scikitplot as skplt**

**skplt.metrics.plot\_confusion\_matrix(y\_exp, y\_exp1, normalize=False)**

**plt.savefig("nonnormalized1.png")**

**# In[67]:**

**# Compute ROC curve and ROC area for each class**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from itertools import cycle**

**from sklearn import svm, datasets**

**from sklearn.metrics import roc\_curve, auc**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import label\_binarize**

**from sklearn.multiclass import OneVsRestClassifier**

**from scipy import interp**

**fpr = dict()**

**tpr = dict()**

**roc\_auc = dict()**

**n\_classes=4**

**for i in range(n\_classes):**

**fpr[i], tpr[i], \_ = roc\_curve(y\_test1[:,i], y\_predict[:,i])**

**roc\_auc[i] = auc(fpr[i], tpr[i])**

**# Compute micro-average ROC curve and ROC area**

**fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test1.ravel(), y\_predict.ravel())**

**roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])**

**lw=2**

**# Compute macro-average ROC curve and ROC area**

**# First aggregate all false positive rates**

**all\_fpr = np.unique(np.concatenate([fpr[i] for i in range(n\_classes)]))**

**# Then interpolate all ROC curves at this points**

**mean\_tpr = np.zeros\_like(all\_fpr)**

**for i in range(n\_classes):**

**mean\_tpr += interp(all\_fpr, fpr[i], tpr[i])**

**# Finally average it and compute AUC**

**mean\_tpr /= n\_classes**

**fpr["macro"] = all\_fpr**

**tpr["macro"] = mean\_tpr**

**roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])**

**# Plot all ROC curves**

**fig3 = plt.figure()**

**colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])**

**for i, color in zip(range(n\_classes), colors):**

**plt.plot(fpr[i], tpr[i], color=color, lw=lw,**

**label='ROC curve of class {0} (area = {1:0.2f})'**

**''.format(i, roc\_auc[i]))**

**plt.plot([0, 1], [0, 1], 'k--', lw=lw)**

**plt.xlim([0.0, 1.0])**

**plt.ylim([0.0, 1.05])**

**plt.xlabel('False Positive Rate')**

**plt.ylabel('True Positive Rate')**

**#plt.title('Some extension of Receiver operating characteristic to multi-class')**

**plt.legend(loc="lower right")**

**#plt.show()**

**fig3.savefig("roc.png")**

(cv) namosvision@gpuserver:~/SAT4and6$ python3 sat4\_6.py

Using TensorFlow backend.

-------------------------------------------

K shape= (400000, 28, 28, 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 16) 160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 24, 24, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 10, 10, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 131200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 4) 516

=================================================================

Total params: 164,260

Trainable params: 164,260

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 320000 samples, validate on 80000 samples

Epoch 1/50

2019-03-19 16:09:46.191128: I tensorflow/core/platform/cpu\_feature\_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA

320000/320000 [==============================] - 319s 998us/step - loss: 0.5602 - acc: 0.7567 - val\_loss: 0.3304 - val\_acc: 0.8720

Epoch 2/50

320000/320000 [==============================] - 319s 997us/step - loss: 0.3150 - acc: 0.8828 - val\_loss: 0.2379 - val\_acc: 0.9094

Epoch 3/50

320000/320000 [==============================] - 320s 999us/step - loss: 0.2623 - acc: 0.9023 - val\_loss: 0.2229 - val\_acc: 0.9149

Epoch 4/50

320000/320000 [==============================] - 320s 1ms/step - loss: 0.2329 - acc: 0.9132 - val\_loss: 0.1804 - val\_acc: 0.9313

Epoch 5/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.2113 - acc: 0.9208 - val\_loss: 0.1883 - val\_acc: 0.9303

Epoch 6/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1961 - acc: 0.9270 - val\_loss: 0.1640 - val\_acc: 0.9359

Epoch 7/50

320000/320000 [==============================] - 319s 998us/step - loss: 0.1833 - acc: 0.9310 - val\_loss: 0.1465 - val\_acc: 0.9427

Epoch 8/50

320000/320000 [==============================] - 319s 998us/step - loss: 0.1745 - acc: 0.9348 - val\_loss: 0.1635 - val\_acc: 0.9373

Epoch 9/50

320000/320000 [==============================] - 318s 995us/step - loss: 0.1703 - acc: 0.9365 - val\_loss: 0.1636 - val\_acc: 0.9367

Epoch 10/50

320000/320000 [==============================] - 319s 996us/step - loss: 0.1613 - acc: 0.9395 - val\_loss: 0.1260 - val\_acc: 0.9516

Epoch 11/50

320000/320000 [==============================] - 319s 997us/step - loss: 0.1604 - acc: 0.9401 - val\_loss: 0.1496 - val\_acc: 0.9462

Epoch 12/50

320000/320000 [==============================] - 318s 994us/step - loss: 0.1530 - acc: 0.9428 - val\_loss: 0.1386 - val\_acc: 0.9486

Epoch 13/50

320000/320000 [==============================] - 319s 998us/step - loss: 0.1509 - acc: 0.9434 - val\_loss: 0.1253 - val\_acc: 0.9529

Epoch 14/50

320000/320000 [==============================] - 318s 992us/step - loss: 0.1497 - acc: 0.9441 - val\_loss: 0.1231 - val\_acc: 0.9551

Epoch 15/50

320000/320000 [==============================] - 318s 993us/step - loss: 0.1451 - acc: 0.9459 - val\_loss: 0.1285 - val\_acc: 0.9504

Epoch 16/50

320000/320000 [==============================] - 318s 994us/step - loss: 0.1423 - acc: 0.9469 - val\_loss: 0.1152 - val\_acc: 0.9581

Epoch 17/50

320000/320000 [==============================] - 320s 999us/step - loss: 0.1400 - acc: 0.9474 - val\_loss: 0.1159 - val\_acc: 0.9565

Epoch 18/50

320000/320000 [==============================] - 319s 996us/step - loss: 0.1369 - acc: 0.9489 - val\_loss: 0.1132 - val\_acc: 0.9583

Epoch 19/50

320000/320000 [==============================] - 319s 995us/step - loss: 0.1362 - acc: 0.9493 - val\_loss: 0.1416 - val\_acc: 0.9467

Epoch 20/50

320000/320000 [==============================] - 319s 997us/step - loss: 0.1363 - acc: 0.9496 - val\_loss: 0.1108 - val\_acc: 0.9590

Epoch 21/50

320000/320000 [==============================] - 319s 997us/step - loss: 0.1340 - acc: 0.9498 - val\_loss: 0.1338 - val\_acc: 0.9503

Epoch 22/50

320000/320000 [==============================] - 318s 994us/step - loss: 0.1293 - acc: 0.9521 - val\_loss: 0.1130 - val\_acc: 0.9581

Epoch 23/50

320000/320000 [==============================] - 319s 997us/step - loss: 0.1291 - acc: 0.9517 - val\_loss: 0.1121 - val\_acc: 0.9604

Epoch 24/50

320000/320000 [==============================] - 320s 999us/step - loss: 0.1275 - acc: 0.9531 - val\_loss: 0.1145 - val\_acc: 0.9595

Epoch 25/50

320000/320000 [==============================] - 320s 1ms/step - loss: 0.1275 - acc: 0.9529 - val\_loss: 0.1155 - val\_acc: 0.9596

Epoch 26/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1259 - acc: 0.9532 - val\_loss: 0.1173 - val\_acc: 0.9556

Epoch 27/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1253 - acc: 0.9536 - val\_loss: 0.1157 - val\_acc: 0.9600

Epoch 28/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1244 - acc: 0.9539 - val\_loss: 0.1114 - val\_acc: 0.9607

Epoch 29/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1231 - acc: 0.9544 - val\_loss: 0.1021 - val\_acc: 0.9604

Epoch 30/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1213 - acc: 0.9551 - val\_loss: 0.1137 - val\_acc: 0.9579

Epoch 31/50

320000/320000 [==============================] - 320s 1ms/step - loss: 0.1206 - acc: 0.9556 - val\_loss: 0.1148 - val\_acc: 0.9597

Epoch 32/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1200 - acc: 0.9556 - val\_loss: 0.1243 - val\_acc: 0.9541

Epoch 33/50

320000/320000 [==============================] - 320s 1ms/step - loss: 0.1207 - acc: 0.9552 - val\_loss: 0.1077 - val\_acc: 0.9618

Epoch 34/50

320000/320000 [==============================] - 321s 1ms/step - loss: 0.1187 - acc: 0.9565 - val\_loss: 0.1392 - val\_acc: 0.9475

Epoch 35/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1183 - acc: 0.9563 - val\_loss: 0.0997 - val\_acc: 0.9636

Epoch 36/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1173 - acc: 0.9566 - val\_loss: 0.0937 - val\_acc: 0.9666

Epoch 37/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1156 - acc: 0.9574 - val\_loss: 0.0945 - val\_acc: 0.9658

Epoch 38/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1151 - acc: 0.9573 - val\_loss: 0.1082 - val\_acc: 0.9600

Epoch 39/50

320000/320000 [==============================] - 322s 1ms/step - loss: 0.1155 - acc: 0.9573 - val\_loss: 0.0979 - val\_acc: 0.9631

Epoch 40/50

320000/320000 [==============================] - 324s 1ms/step - loss: 0.1145 - acc: 0.9574 - val\_loss: 0.1026 - val\_acc: 0.9633

Epoch 41/50

320000/320000 [==============================] - 324s 1ms/step - loss: 0.1149 - acc: 0.9576 - val\_loss: 0.1022 - val\_acc: 0.9603

Epoch 42/50

320000/320000 [==============================] - 324s 1ms/step - loss: 0.1135 - acc: 0.9580 - val\_loss: 0.1106 - val\_acc: 0.9585

Epoch 43/50

320000/320000 [==============================] - 323s 1ms/step - loss: 0.1142 - acc: 0.9582 - val\_loss: 0.0963 - val\_acc: 0.9644

Epoch 44/50

320000/320000 [==============================] - 323s 1ms/step - loss: 0.1114 - acc: 0.9593 - val\_loss: 0.1052 - val\_acc: 0.9620

Epoch 45/50

320000/320000 [==============================] - 323s 1ms/step - loss: 0.1117 - acc: 0.9587 - val\_loss: 0.1006 - val\_acc: 0.9621

Epoch 46/50

320000/320000 [==============================] - 325s 1ms/step - loss: 0.1116 - acc: 0.9588 - val\_loss: 0.0973 - val\_acc: 0.9649

Epoch 47/50

320000/320000 [==============================] - 323s 1ms/step - loss: 0.1122 - acc: 0.9581 - val\_loss: 0.0981 - val\_acc: 0.9644

Epoch 48/50

320000/320000 [==============================] - 323s 1ms/step - loss: 0.1114 - acc: 0.9588 - val\_loss: 0.1006 - val\_acc: 0.9643

Epoch 49/50

320000/320000 [==============================] - 324s 1ms/step - loss: 0.1096 - acc: 0.9594 - val\_loss: 0.0990 - val\_acc: 0.9653

Epoch 50/50

320000/320000 [==============================] - 324s 1ms/step - loss: 0.1087 - acc: 0.9600 - val\_loss: 0.0953 - val\_acc: 0.9669

100000/100000 [==============================] - 28s 280us/step

Loss: 0.09,Accuracy: 96.60%

0.96599

[0.97074479 0.97302654 0.89369031 0.99816034]

[0.94012753 0.99495823 0.95419592 0.97449065]

[0.95519088 0.98387018 0.92295254 0.98618349]

precision recall f1-score support

class 0 0.97 0.94 0.96 26189

class 1 0.97 0.99 0.98 20231

class 2 0.89 0.95 0.92 17946

class 3 1.00 0.97 0.99 35634

micro avg 0.97 0.97 0.97 100000

macro avg 0.96 0.97 0.96 100000

weighted avg 0.97 0.97 0.97 100000

Confusion matrix, without normalization

Normalized confusion matrix

**SAT 4 Red & Infra**

**import scipy.io**

**import numpy as np**

**import pandas as pd**

**import sklearn**

**from sklearn.preprocessing import Normalizer**

**import keras**

**from keras.models import Sequential**

**import tensorflow as tf**

**import matplotlib.pyplot as plt**

**import matplotlib.image as implt**

**from sklearn.model\_selection import train\_test\_split**

**from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,Activation,MaxPool2D**

**mat1 = scipy.io.loadmat("sat-4-full.mat")**

**x\_train1=mat1['train\_x']**

**x\_test1=mat1['test\_x']**

**y\_train1=mat1['train\_y']**

**y\_test1=mat1['test\_y']**

**x\_train1=np.array(x\_train1)**

**x\_test1=np.array(x\_test1)**

**def resh(x1,a,b):**

**x2=np.empty([100000,28,28,2])**

**t=0**

**for i in range(a,b):**

**for j in range(0,4):**

**if(j==0 ):**

**x2[t,:,:,0]=x1[:,:,j,i]**

**if(j==3):**

**x2[t,:,:,1]=x1[:,:,j,i]**

**t=t+1**

**return x2;**

**k1=np.empty([400000,28,28,2])**

**k1[0:100000,:,:,:]=resh(x\_train1,0,100000)**

**k1[100000:200000,:,:,:]=resh(x\_train1,100000,200000)**

**k1[200000:300000,:,:,:]=resh(x\_train1,200000,300000)**

**k1[300000:400000,:,:,:]=resh(x\_train1,300000,400000)**

**print('-------------------------------------------')**

**print("K shape=",np.shape(k1))**

**z=np.transpose(y\_train1)**

**x\_train,x\_vvalid,y\_train,y\_valid=train\_test\_split(k1,z,test\_size=0.2)**

**x\_train=(x\_train - x\_train.min())/(x\_train.max() - x\_train.min())**

**x\_valid=(x\_vvalid - x\_vvalid.min())/(x\_vvalid.max() - x\_vvalid.min())**

**model = Sequential()**

**model.add(Conv2D(16, (3,3), activation='relu', input\_shape=(28,28,2)))**

**model.add(Conv2D(32, (3,3), activation='relu'))**

**model.add(MaxPool2D(pool\_size=(2,2)))**

**model.add(Dropout(0.5))**

**model.add(Conv2D(32, (3,3), activation='relu'))**

**model.add(Conv2D(64, (3,3), activation='relu'))**

**model.add(MaxPool2D(pool\_size=(2,2)))**

**model.add(Dropout(0.5))**

**model.add(Flatten())**

**model.add(Dense(128, activation='relu'))**

**model.add(Dropout(0.5))**

**model.add(Dense(4, activation='softmax'))**

**model.summary()**

**model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])**

**model.fit(x\_train, y\_train, batch\_size=200, epochs=50, verbose=1, validation\_data=(x\_valid, y\_valid))**

**te=np.empty([100000,28,28,2])**

**te[0:100000,:,:,:]=resh(x\_test1,0,100000)**

**te=(te - te.min())/(te.max() - te.min())**

**y\_test1=np.transpose(y\_test1)**

**loss,accuracy=model.evaluate(te,y\_test1)**

**print("\nLoss: %.2f,Accuracy: %.2f%%" % (loss,accuracy\*100))**

**y\_predict = model.predict(te)**

**y\_exp1 = np.argmax(y\_predict,axis=1)**

**y\_exp=np.argmax(y\_test1,axis=1)**

**from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,confusion\_matrix**

**accuracy=accuracy\_score(y\_exp,y\_exp1)**

**print(accuracy)**

**precision=precision\_score(y\_exp, y\_exp1, average=None)**

**print(precision)**

**recall=recall\_score(y\_exp, y\_exp1, average=None)**

**print(recall)**

**f1score=f1\_score(y\_exp, y\_exp1, average=None)**

**print(f1score)**

**from sklearn.metrics import classification\_report**

**sklearn.metrics.classification\_report(y\_exp, y\_exp1)**

**target\_names = ['class 0', 'class 1', 'class 2','class 3']**

**print(classification\_report(y\_exp, y\_exp1, target\_names=target\_names))**

**import itertools**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from sklearn import svm, datasets**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import confusion\_matrix**

**def plot\_confusion\_matrix(cm, classes,**

**normalize=False,**

**title='Confusion matrix',**

**cmap=plt.cm.Blues):**

**"""**

**This function prints and plots the confusion matrix.**

**Normalization can be applied by setting `normalize=True`.**

**"""**

**if normalize:**

**cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]**

**print("Normalized confusion matrix")**

**else:**

**print('Confusion matrix, without normalization')**

**#print(cm)**

**#plt.imshow(cm, interpolation='nearest', cmap=cmap)**

**plt.title(title)**

**#plt.colorbar()**

**tick\_marks = np.arange(len(classes))**

**plt.xticks(tick\_marks, classes, rotation=45)**

**plt.yticks(tick\_marks, classes)**

**fmt = '.2f' if normalize else 'd'**

**thresh = cm.max() / 2.**

**for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):**

**plt.text(j, i, format(cm[i, j], fmt),**

**horizontalalignment="center",**

**color="white" if cm[i, j] > thresh else "black")**

**plt.ylabel('True label')**

**plt.xlabel('Predicted label')**

**plt.tight\_layout()**

**# Compute confusion matrix**

**cnf\_matrix = confusion\_matrix(y\_exp, y\_exp1)**

**np.set\_printoptions(precision=2)**

**# Plot non-normalized confusion matrix**

**fig1 = plt.figure()**

**plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3'],**

**title='Confusion matrix, without normalization')**

**# Plot normalized confusion matrix**

**fig2 = plt.figure()**

**plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3'], normalize=True,**

**title='Normalized confusion matrix')**

**#fig1.savefig("nonnormalized.png")**

**#fig2.savefig("normalized.png")**

**import scikitplot as skplt**

**skplt.metrics.plot\_confusion\_matrix(y\_exp, y\_exp1, normalize=False)**

**plt.savefig("nonnormalized1.png")**

**# In[67]:**

**# Compute ROC curve and ROC area for each class**

**import numpy as np**

**import matplotlib.pyplot as plt**

**from itertools import cycle**

**from sklearn import svm, datasets**

**from sklearn.metrics import roc\_curve, auc**

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.preprocessing import label\_binarize**

**from sklearn.multiclass import OneVsRestClassifier**

**from scipy import interp**

**fpr = dict()**

**tpr = dict()**

**roc\_auc = dict()**

**n\_classes=4**

**for i in range(n\_classes):**

**fpr[i], tpr[i], \_ = roc\_curve(y\_test1[:,i], y\_predict[:,i])**

**roc\_auc[i] = auc(fpr[i], tpr[i])**

**# Compute micro-average ROC curve and ROC area**

**fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test1.ravel(), y\_predict.ravel())**

**roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])**

**lw=2**

**# Compute macro-average ROC curve and ROC area**

**# First aggregate all false positive rates**

**all\_fpr = np.unique(np.concatenate([fpr[i] for i in range(n\_classes)]))**

**# Then interpolate all ROC curves at this points**

**mean\_tpr = np.zeros\_like(all\_fpr)**

**for i in range(n\_classes):**

**mean\_tpr += interp(all\_fpr, fpr[i], tpr[i])**

**# Finally average it and compute AUC**

**mean\_tpr /= n\_classes**

**fpr["macro"] = all\_fpr**

**tpr["macro"] = mean\_tpr**

**roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])**

**# Plot all ROC curves**

**fig3 = plt.figure()**

**colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])**

**for i, color in zip(range(n\_classes), colors):**

**plt.plot(fpr[i], tpr[i], color=color, lw=lw,**

**label='ROC curve of class {0} (area = {1:0.2f})'**

**''.format(i, roc\_auc[i]))**

**plt.plot([0, 1], [0, 1], 'k--', lw=lw)**

**plt.xlim([0.0, 1.0])**

**plt.ylim([0.0, 1.05])**

**plt.xlabel('False Positive Rate')**

**plt.ylabel('True Positive Rate')**

**#plt.title('Some extension of Receiver operating characteristic to multi-class')**

**plt.legend(loc="lower right")**

**#plt.show()**

**fig3.savefig("roc.png")**

(cv) namosvision@gpuserver:~/SAT4and6$ python3 sat4\_6.py

Using TensorFlow backend.

-------------------------------------------

K shape= (400000, 28, 28, 2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 16) 304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 24, 24, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 10, 10, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 131200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 4) 516

=================================================================

Total params: 164,404

Trainable params: 164,404

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 320000 samples, validate on 80000 samples

Epoch 1/50

2019-03-20 14:17:45.714274: I tensorflow/core/platform/cpu\_feature\_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA

320000/320000 [==============================] - 272s 852us/step - loss: 0.2719 - acc: 0.9032 - val\_loss: 0.1330 - val\_acc: 0.9532

Epoch 2/50

320000/320000 [==============================] - 275s 858us/step - loss: 0.1368 - acc: 0.9530 - val\_loss: 0.0904 - val\_acc: 0.9694

Epoch 3/50

320000/320000 [==============================] - 280s 875us/step - loss: 0.1039 - acc: 0.9643 - val\_loss: 0.0867 - val\_acc: 0.9664

Epoch 4/50

320000/320000 [==============================] - 283s 884us/step - loss: 0.0895 - acc: 0.9686 - val\_loss: 0.0775 - val\_acc: 0.9726

Epoch 5/50

320000/320000 [==============================] - 288s 901us/step - loss: 0.0748 - acc: 0.9741 - val\_loss: 0.0607 - val\_acc: 0.9787

Epoch 6/50

320000/320000 [==============================] - 291s 911us/step - loss: 0.0705 - acc: 0.9759 - val\_loss: 0.0568 - val\_acc: 0.9803

Epoch 7/50

320000/320000 [==============================] - 314s 981us/step - loss: 0.0632 - acc: 0.9787 - val\_loss: 0.0390 - val\_acc: 0.9867

Epoch 8/50

320000/320000 [==============================] - 328s 1ms/step - loss: 0.0600 - acc: 0.9798 - val\_loss: 0.0472 - val\_acc: 0.9835

Epoch 9/50

320000/320000 [==============================] - 328s 1ms/step - loss: 0.0572 - acc: 0.9810 - val\_loss: 0.0528 - val\_acc: 0.9801

Epoch 10/50

320000/320000 [==============================] - 329s 1ms/step - loss: 0.0532 - acc: 0.9823 - val\_loss: 0.0390 - val\_acc: 0.9867

Epoch 11/50

320000/320000 [==============================] - 328s 1ms/step - loss: 0.0509 - acc: 0.9829 - val\_loss: 0.0423 - val\_acc: 0.9845

Epoch 12/50

320000/320000 [==============================] - 330s 1ms/step - loss: 0.0479 - acc: 0.9837 - val\_loss: 0.0337 - val\_acc: 0.9879

Epoch 13/50

320000/320000 [==============================] - 330s 1ms/step - loss: 0.0472 - acc: 0.9842 - val\_loss: 0.0473 - val\_acc: 0.9840

Epoch 14/50

320000/320000 [==============================] - 331s 1ms/step - loss: 0.0442 - acc: 0.9855 - val\_loss: 0.0443 - val\_acc: 0.9836

Epoch 15/50

320000/320000 [==============================] - 334s 1ms/step - loss: 0.0444 - acc: 0.9850 - val\_loss: 0.0663 - val\_acc: 0.9780

Epoch 16/50

320000/320000 [==============================] - 333s 1ms/step - loss: 0.0421 - acc: 0.9859 - val\_loss: 0.0336 - val\_acc: 0.9885

Epoch 17/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0417 - acc: 0.9860 - val\_loss: 0.0244 - val\_acc: 0.9920

Epoch 18/50

320000/320000 [==============================] - 337s 1ms/step - loss: 0.0392 - acc: 0.9869 - val\_loss: 0.0362 - val\_acc: 0.9872

Epoch 19/50

320000/320000 [==============================] - 337s 1ms/step - loss: 0.0396 - acc: 0.9868 - val\_loss: 0.0262 - val\_acc: 0.9917

Epoch 20/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0383 - acc: 0.9873 - val\_loss: 0.0304 - val\_acc: 0.9891

Epoch 21/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0370 - acc: 0.9875 - val\_loss: 0.0575 - val\_acc: 0.9814

Epoch 22/50

320000/320000 [==============================] - 337s 1ms/step - loss: 0.0363 - acc: 0.9878 - val\_loss: 0.0380 - val\_acc: 0.9877

Epoch 23/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0374 - acc: 0.9875 - val\_loss: 0.0282 - val\_acc: 0.9909

Epoch 24/50

320000/320000 [==============================] - 337s 1ms/step - loss: 0.0353 - acc: 0.9882 - val\_loss: 0.0261 - val\_acc: 0.9913

Epoch 25/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0350 - acc: 0.9883 - val\_loss: 0.0262 - val\_acc: 0.9902

Epoch 26/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0334 - acc: 0.9889 - val\_loss: 0.0304 - val\_acc: 0.9889

Epoch 27/50

320000/320000 [==============================] - 334s 1ms/step - loss: 0.0342 - acc: 0.9885 - val\_loss: 0.0373 - val\_acc: 0.9868

Epoch 28/50

320000/320000 [==============================] - 337s 1ms/step - loss: 0.0328 - acc: 0.9889 - val\_loss: 0.0282 - val\_acc: 0.9903

Epoch 29/50

320000/320000 [==============================] - 333s 1ms/step - loss: 0.0325 - acc: 0.9891 - val\_loss: 0.0276 - val\_acc: 0.9899

Epoch 30/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0319 - acc: 0.9893 - val\_loss: 0.0301 - val\_acc: 0.9902

Epoch 31/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0330 - acc: 0.9889 - val\_loss: 0.0276 - val\_acc: 0.9905

Epoch 32/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0303 - acc: 0.9900 - val\_loss: 0.0231 - val\_acc: 0.9920

Epoch 33/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0319 - acc: 0.9893 - val\_loss: 0.0268 - val\_acc: 0.9907

Epoch 34/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0307 - acc: 0.9900 - val\_loss: 0.0258 - val\_acc: 0.9910

Epoch 35/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0299 - acc: 0.9900 - val\_loss: 0.0335 - val\_acc: 0.9890

Epoch 36/50

320000/320000 [==============================] - 334s 1ms/step - loss: 0.0304 - acc: 0.9899 - val\_loss: 0.0361 - val\_acc: 0.9863

Epoch 37/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0295 - acc: 0.9902 - val\_loss: 0.0233 - val\_acc: 0.9924

Epoch 38/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0288 - acc: 0.9902 - val\_loss: 0.0226 - val\_acc: 0.9920

Epoch 39/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0282 - acc: 0.9904 - val\_loss: 0.0191 - val\_acc: 0.9932

Epoch 40/50

320000/320000 [==============================] - 334s 1ms/step - loss: 0.0290 - acc: 0.9903 - val\_loss: 0.0262 - val\_acc: 0.9914

Epoch 41/50

320000/320000 [==============================] - 335s 1ms/step - loss: 0.0287 - acc: 0.9905 - val\_loss: 0.0239 - val\_acc: 0.9922

Epoch 42/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0277 - acc: 0.9907 - val\_loss: 0.0256 - val\_acc: 0.9916

Epoch 43/50

320000/320000 [==============================] - 367s 1ms/step - loss: 0.0281 - acc: 0.9906 - val\_loss: 0.0202 - val\_acc: 0.9932

Epoch 44/50

320000/320000 [==============================] - 399s 1ms/step - loss: 0.0272 - acc: 0.9909 - val\_loss: 0.0196 - val\_acc: 0.9929

Epoch 45/50

320000/320000 [==============================] - 336s 1ms/step - loss: 0.0275 - acc: 0.9909 - val\_loss: 0.0192 - val\_acc: 0.9930

Epoch 46/50

320000/320000 [==============================] - 351s 1ms/step - loss: 0.0274 - acc: 0.9908 - val\_loss: 0.0238 - val\_acc: 0.9922

Epoch 47/50

320000/320000 [==============================] - 396s 1ms/step - loss: 0.0267 - acc: 0.9910 - val\_loss: 0.0209 - val\_acc: 0.9925

Epoch 48/50

320000/320000 [==============================] - 396s 1ms/step - loss: 0.0265 - acc: 0.9909 - val\_loss: 0.0210 - val\_acc: 0.9922

Epoch 49/50

320000/320000 [==============================] - 396s 1ms/step - loss: 0.0275 - acc: 0.9909 - val\_loss: 0.0277 - val\_acc: 0.9904

Epoch 50/50

320000/320000 [==============================] - 393s 1ms/step - loss: 0.0258 - acc: 0.9915 - val\_loss: 0.0357 - val\_acc: 0.9903

100000/100000 [==============================] - 33s 326us/step

Loss: 0.03,Accuracy: 99.01%

0.99010

[0.97430327 0.99653637 0.98304707 0.99997156]

[0.99316507 0.99550195 0.98228017 0.98672616]

[0.98364376 0.99601889 0.98266347 0.99330471]

precision recall f1-score support

class 0 0.98 0.99 0.98 26189

class 1 1.00 1.00 1.00 20231

class 2 0.98 0.98 0.98 17946

class 3 1.00 0.99 0.99 35634

micro avg 0.99 0.99 0.99 100000

macro avg 0.99 0.99 0.99 100000

weighted avg 0.99 0.99 0.99 100000

Confusion matrix, without normalization

Normalized confusion matrix

**SAT6- InfraRed**

import scipy.io

import numpy as np

import pandas as pd

import sklearn

from sklearn.preprocessing import Normalizer

import keras

from keras.models import Sequential

import tensorflow as tf

import matplotlib.pyplot as plt

import matplotlib.image as implt

from sklearn.model\_selection import train\_test\_split

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,Activation,MaxPool2D

mat1 = scipy.io.loadmat("sat-6-full.mat")

x\_train1=mat1['train\_x']

x\_test1=mat1['test\_x']

y\_train1=mat1['train\_y']

y\_test1=mat1['test\_y']

x\_train1=np.array(x\_train1)

x\_test1=np.array(x\_test1)

print(np.shape(x\_test1))

print(np.shape(x\_train1))

def resh(x1,a,b):

x2=np.empty([(b-a),28,28,1])

t=0

for i in range(a,b):

for j in range(0,4):

#if(j==0 ):

#x2[t,:,:,0]=x1[:,:,j,i]

if(j==3):

x2[t,:,:,0]=x1[:,:,j,i]

t=t+1

print(t)

return x2;

k1=np.empty([324000,28,28,1])

k1[0:100000,:,:,:]=resh(x\_train1,0,100000)

k1[100000:200000,:,:,:]=resh(x\_train1,100000,200000)

k1[200000:300000,:,:,:]=resh(x\_train1,200000,300000)

k1[300000:324000,:,:,:]=resh(x\_train1,300000,324000)

print('-------------------------------------------')

print("K shape=",np.shape(k1))

z=np.transpose(y\_train1)

x\_train,x\_vvalid,y\_train,y\_valid=train\_test\_split(k1,z,test\_size=0.2)

x\_train=(x\_train - x\_train.min())/(x\_train.max() - x\_train.min())

x\_valid=(x\_vvalid - x\_vvalid.min())/(x\_vvalid.max() - x\_vvalid.min())

model = Sequential()

model.add(Conv2D(16, (3,3), activation='relu', input\_shape=(28,28,1)))

model.add(Conv2D(32, (3,3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Dropout(0.5))

model.add(Conv2D(32, (3,3), activation='relu'))

model.add(Conv2D(64, (3,3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(6, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(x\_train, y\_train, batch\_size=200, epochs=50, verbose=1, validation\_data=(x\_valid, y\_valid))

te=np.empty([81000,28,28,1])

te[0:81000,:,:,:]=resh(x\_test1,0,81000)

te=(te - te.min())/(te.max() - te.min())

y\_test1=np.transpose(y\_test1)

loss,accuracy=model.evaluate(te,y\_test1)

print("\nLoss: %.2f,Accuracy: %.2f%%" % (loss,accuracy\*100))

y\_predict = model.predict(te)

y\_exp1 = np.argmax(y\_predict,axis=1)

y\_exp=np.argmax(y\_test1,axis=1)

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,confusion\_matrix

accuracy=accuracy\_score(y\_exp,y\_exp1)

print(accuracy)

precision=precision\_score(y\_exp, y\_exp1, average=None)

print(precision)

recall=recall\_score(y\_exp, y\_exp1, average=None)

print(recall)

f1score=f1\_score(y\_exp, y\_exp1, average=None)

print(f1score)

from sklearn.metrics import classification\_report

sklearn.metrics.classification\_report(y\_exp, y\_exp1)

target\_names = ['class 0', 'class 1', 'class 2','class 3','class 4','class 5']

print(classification\_report(y\_exp, y\_exp1, target\_names=target\_names))

import itertools

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

#print(cm)

#plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

#plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.tight\_layout()

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_exp, y\_exp1)

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

fig1 = plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3','4','5'],

title='Confusion matrix, without normalization')

# Plot normalized confusion matrix

fig2 = plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3','4','5'], normalize=True,

title='Normalized confusion matrix')

#fig1.savefig("nonnormalized.png")

#fig2.savefig("normalized.png")

import scikitplot as skplt

skplt.metrics.plot\_confusion\_matrix(y\_exp, y\_exp1, normalize=False)

plt.savefig("nonnormalized1.png")

# In[67]:

# Compute ROC curve and ROC area for each class

import numpy as np

import matplotlib.pyplot as plt

from itertools import cycle

from sklearn import svm, datasets

from sklearn.metrics import roc\_curve, auc

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import label\_binarize

from sklearn.multiclass import OneVsRestClassifier

from scipy import interp

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes=6

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test1[:,i], y\_predict[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test1.ravel(), y\_predict.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

lw=2

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates

all\_fpr = np.unique(np.concatenate([fpr[i] for i in range(n\_classes)]))

# Then interpolate all ROC curves at this points

mean\_tpr = np.zeros\_like(all\_fpr)

for i in range(n\_classes):

mean\_tpr += interp(all\_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC

mean\_tpr /= n\_classes

fpr["macro"] = all\_fpr

tpr["macro"] = mean\_tpr

roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves

fig3 = plt.figure()

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])

for i, color in zip(range(n\_classes), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=lw,

label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

#plt.title('Some extension of Receiver operating characteristic to multi-class')

plt.legend(loc="lower right")

#plt.show()

fig3.savefig("roc.png")

(cv) namosvision@gpuserver:~/SAT4and6$ python3 sat4\_6.py

Using TensorFlow backend.

(28, 28, 4, 81000)

(28, 28, 4, 324000)

100000

100000

100000

24000

-------------------------------------------

K shape= (324000, 28, 28, 1)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 16) 160

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 24, 24, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 10, 10, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 131200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 6) 774

=================================================================

Total params: 164,518

Trainable params: 164,518

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 259200 samples, validate on 64800 samples

Epoch 1/50

2019-03-21 16:05:58.295276: I tensorflow/core/platform/cpu\_feature\_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA

259200/259200 [==============================] - 225s 868us/step - loss: 0.4739 - acc: 0.7919 - val\_loss: 0.2561 - val\_acc: 0.8954

Epoch 2/50

259200/259200 [==============================] - 217s 837us/step - loss: 0.2572 - acc: 0.8997 - val\_loss: 0.2303 - val\_acc: 0.9012

Epoch 3/50

259200/259200 [==============================] - 224s 864us/step - loss: 0.2230 - acc: 0.9134 - val\_loss: 0.1796 - val\_acc: 0.9281

Epoch 4/50

259200/259200 [==============================] - 247s 952us/step - loss: 0.2010 - acc: 0.9215 - val\_loss: 0.1676 - val\_acc: 0.9322

Epoch 5/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1866 - acc: 0.9264 - val\_loss: 0.1651 - val\_acc: 0.9326

Epoch 6/50

259200/259200 [==============================] - 285s 1ms/step - loss: 0.1709 - acc: 0.9320 - val\_loss: 0.1479 - val\_acc: 0.9377

Epoch 7/50

259200/259200 [==============================] - 279s 1ms/step - loss: 0.1591 - acc: 0.9377 - val\_loss: 0.1356 - val\_acc: 0.9496

Epoch 8/50

259200/259200 [==============================] - 289s 1ms/step - loss: 0.1517 - acc: 0.9412 - val\_loss: 0.1155 - val\_acc: 0.9553

Epoch 9/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1424 - acc: 0.9445 - val\_loss: 0.1144 - val\_acc: 0.9552

Epoch 10/50

259200/259200 [==============================] - 287s 1ms/step - loss: 0.1379 - acc: 0.9461 - val\_loss: 0.1257 - val\_acc: 0.9473

Epoch 11/50

259200/259200 [==============================] - 283s 1ms/step - loss: 0.1319 - acc: 0.9486 - val\_loss: 0.1161 - val\_acc: 0.9560

Epoch 12/50

259200/259200 [==============================] - 283s 1ms/step - loss: 0.1276 - acc: 0.9509 - val\_loss: 0.0994 - val\_acc: 0.9617

Epoch 13/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1264 - acc: 0.9513 - val\_loss: 0.1101 - val\_acc: 0.9554

Epoch 14/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1233 - acc: 0.9526 - val\_loss: 0.1005 - val\_acc: 0.9617

Epoch 15/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1176 - acc: 0.9551 - val\_loss: 0.1198 - val\_acc: 0.9544

Epoch 16/50

259200/259200 [==============================] - 280s 1ms/step - loss: 0.1145 - acc: 0.9558 - val\_loss: 0.0909 - val\_acc: 0.9654

Epoch 17/50

259200/259200 [==============================] - 287s 1ms/step - loss: 0.1161 - acc: 0.9554 - val\_loss: 0.1010 - val\_acc: 0.9619

Epoch 18/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1110 - acc: 0.9575 - val\_loss: 0.0882 - val\_acc: 0.9647

Epoch 19/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1081 - acc: 0.9585 - val\_loss: 0.0870 - val\_acc: 0.9654

Epoch 20/50

259200/259200 [==============================] - 287s 1ms/step - loss: 0.1083 - acc: 0.9585 - val\_loss: 0.1152 - val\_acc: 0.9545

Epoch 21/50

259200/259200 [==============================] - 279s 1ms/step - loss: 0.1081 - acc: 0.9587 - val\_loss: 0.1117 - val\_acc: 0.9565

Epoch 22/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1073 - acc: 0.9587 - val\_loss: 0.0861 - val\_acc: 0.9664

Epoch 23/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1039 - acc: 0.9602 - val\_loss: 0.0939 - val\_acc: 0.9641

Epoch 24/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1076 - acc: 0.9591 - val\_loss: 0.0971 - val\_acc: 0.9603

Epoch 25/50

259200/259200 [==============================] - 278s 1ms/step - loss: 0.1051 - acc: 0.9603 - val\_loss: 0.0819 - val\_acc: 0.9697

Epoch 26/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1041 - acc: 0.9602 - val\_loss: 0.0895 - val\_acc: 0.9654

Epoch 27/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1021 - acc: 0.9607 - val\_loss: 0.0989 - val\_acc: 0.9622

Epoch 28/50

259200/259200 [==============================] - 286s 1ms/step - loss: 0.1006 - acc: 0.9617 - val\_loss: 0.0951 - val\_acc: 0.9624

Epoch 29/50

259200/259200 [==============================] - 287s 1ms/step - loss: 0.0972 - acc: 0.9628 - val\_loss: 0.0780 - val\_acc: 0.9707

Epoch 30/50

259200/259200 [==============================] - 278s 1ms/step - loss: 0.1007 - acc: 0.9620 - val\_loss: 0.0774 - val\_acc: 0.9705

Epoch 31/50

259200/259200 [==============================] - 227s 876us/step - loss: 0.0966 - acc: 0.9632 - val\_loss: 0.0798 - val\_acc: 0.9701

Epoch 32/50

259200/259200 [==============================] - 227s 877us/step - loss: 0.0988 - acc: 0.9626 - val\_loss: 0.0835 - val\_acc: 0.9686

Epoch 33/50

259200/259200 [==============================] - 228s 878us/step - loss: 0.0975 - acc: 0.9629 - val\_loss: 0.0951 - val\_acc: 0.9694

Epoch 34/50

259200/259200 [==============================] - 224s 863us/step - loss: 0.0946 - acc: 0.9646 - val\_loss: 0.1010 - val\_acc: 0.9607

Epoch 35/50

259200/259200 [==============================] - 224s 865us/step - loss: 0.0943 - acc: 0.9641 - val\_loss: 0.0770 - val\_acc: 0.9710

Epoch 36/50

259200/259200 [==============================] - 227s 878us/step - loss: 0.0923 - acc: 0.9648 - val\_loss: 0.0857 - val\_acc: 0.9665

Epoch 37/50

259200/259200 [==============================] - 227s 874us/step - loss: 0.0955 - acc: 0.9638 - val\_loss: 0.0781 - val\_acc: 0.9708

Epoch 38/50

259200/259200 [==============================] - 227s 877us/step - loss: 0.0942 - acc: 0.9644 - val\_loss: 0.0743 - val\_acc: 0.9717

Epoch 39/50

259200/259200 [==============================] - 221s 851us/step - loss: 0.0930 - acc: 0.9648 - val\_loss: 0.0767 - val\_acc: 0.9707

Epoch 40/50

259200/259200 [==============================] - 227s 877us/step - loss: 0.0918 - acc: 0.9652 - val\_loss: 0.0807 - val\_acc: 0.9680

Epoch 41/50

259200/259200 [==============================] - 227s 875us/step - loss: 0.0894 - acc: 0.9661 - val\_loss: 0.0680 - val\_acc: 0.9748

Epoch 42/50

259200/259200 [==============================] - 227s 877us/step - loss: 0.0917 - acc: 0.9655 - val\_loss: 0.0832 - val\_acc: 0.9680

Epoch 43/50

259200/259200 [==============================] - 228s 879us/step - loss: 0.0902 - acc: 0.9658 - val\_loss: 0.0794 - val\_acc: 0.9704

Epoch 44/50

259200/259200 [==============================] - 222s 856us/step - loss: 0.0887 - acc: 0.9668 - val\_loss: 0.0910 - val\_acc: 0.9667

Epoch 45/50

259200/259200 [==============================] - 228s 881us/step - loss: 0.0896 - acc: 0.9663 - val\_loss: 0.0812 - val\_acc: 0.9681

Epoch 46/50

259200/259200 [==============================] - 228s 878us/step - loss: 0.0895 - acc: 0.9660 - val\_loss: 0.0687 - val\_acc: 0.9749

Epoch 47/50

259200/259200 [==============================] - 227s 875us/step - loss: 0.0885 - acc: 0.9663 - val\_loss: 0.0713 - val\_acc: 0.9724

Epoch 48/50

259200/259200 [==============================] - 221s 851us/step - loss: 0.0862 - acc: 0.9674 - val\_loss: 0.0678 - val\_acc: 0.9751

Epoch 49/50

259200/259200 [==============================] - 227s 877us/step - loss: 0.0849 - acc: 0.9676 - val\_loss: 0.0774 - val\_acc: 0.9702

Epoch 50/50

259200/259200 [==============================] - 228s 878us/step - loss: 0.0876 - acc: 0.9672 - val\_loss: 0.0690 - val\_acc: 0.9743

81000

81000/81000 [==============================] - 19s 239us/step

Loss: 0.07,Accuracy: 97.57%

0.9756913580246913

[0.99272931 0.96050181 0.97728532 0.93465457 0.96520496 1. ]

[0.95584276 0.95459248 0.99485372 0.93228009 0.97826087 1. ]

[0.9739369 0.95753802 0.98599127 0.93346582 0.97168906 1. ]

precision recall f1-score support

class 0 0.99 0.96 0.97 3714

class 1 0.96 0.95 0.96 18367

class 2 0.98 0.99 0.99 14185

class 3 0.93 0.93 0.93 12596

class 4 0.97 0.98 0.97 2070

class 5 1.00 1.00 1.00 30068

micro avg 0.98 0.98 0.98 81000

macro avg 0.97 0.97 0.97 81000

weighted avg 0.98 0.98 0.98 81000

Confusion matrix, without normalization

Normalized confusion matrix

**SAT6 – RED AND INFRA**

import scipy.io

import numpy as np

import pandas as pd

import sklearn

from sklearn.preprocessing import Normalizer

import keras

from keras.models import Sequential

import tensorflow as tf

import matplotlib.pyplot as plt

import matplotlib.image as implt

from sklearn.model\_selection import train\_test\_split

from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D,Activation,MaxPool2D

mat1 = scipy.io.loadmat("sat-6-full.mat")

x\_train1=mat1['train\_x']

x\_test1=mat1['test\_x']

y\_train1=mat1['train\_y']

y\_test1=mat1['test\_y']

x\_train1=np.array(x\_train1)

x\_test1=np.array(x\_test1)

print(np.shape(x\_test1))

print(np.shape(x\_train1))

def resh(x1,a,b):

x2=np.empty([(b-a),28,28,2])

t=0

for i in range(a,b):

for j in range(0,4):

if(j==0 ):

x2[t,:,:,0]=x1[:,:,j,i]

if(j==3):

x2[t,:,:,1]=x1[:,:,j,i]

t=t+1

print(t)

return x2;

k1=np.empty([324000,28,28,2])

k1[0:100000,:,:,:]=resh(x\_train1,0,100000)

k1[100000:200000,:,:,:]=resh(x\_train1,100000,200000)

k1[200000:300000,:,:,:]=resh(x\_train1,200000,300000)

k1[300000:324000,:,:,:]=resh(x\_train1,300000,324000)

print('-------------------------------------------')

print("K shape=",np.shape(k1))

z=np.transpose(y\_train1)

x\_train,x\_vvalid,y\_train,y\_valid=train\_test\_split(k1,z,test\_size=0.2)

x\_train=(x\_train - x\_train.min())/(x\_train.max() - x\_train.min())

x\_valid=(x\_vvalid - x\_vvalid.min())/(x\_vvalid.max() - x\_vvalid.min())

model = Sequential()

model.add(Conv2D(16, (3,3), activation='relu', input\_shape=(28,28,2)))

model.add(Conv2D(32, (3,3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Dropout(0.5))

model.add(Conv2D(32, (3,3), activation='relu'))

model.add(Conv2D(64, (3,3), activation='relu'))

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(6, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(x\_train, y\_train, batch\_size=200, epochs=50, verbose=1, validation\_data=(x\_valid, y\_valid))

te=np.empty([81000,28,28,2])

te[0:81000,:,:,:]=resh(x\_test1,0,81000)

te=(te - te.min())/(te.max() - te.min())

y\_test1=np.transpose(y\_test1)

loss,accuracy=model.evaluate(te,y\_test1)

print("\nLoss: %.2f,Accuracy: %.2f%%" % (loss,accuracy\*100))

y\_predict = model.predict(te)

y\_exp1 = np.argmax(y\_predict,axis=1)

y\_exp=np.argmax(y\_test1,axis=1)

from sklearn.metrics import accuracy\_score,precision\_score,recall\_score,f1\_score,confusion\_matrix

accuracy=accuracy\_score(y\_exp,y\_exp1)

print(accuracy)

precision=precision\_score(y\_exp, y\_exp1, average=None)

print(precision)

recall=recall\_score(y\_exp, y\_exp1, average=None)

print(recall)

f1score=f1\_score(y\_exp, y\_exp1, average=None)

print(f1score)

from sklearn.metrics import classification\_report

sklearn.metrics.classification\_report(y\_exp, y\_exp1)

target\_names = ['class 0', 'class 1', 'class 2','class 3','class 4','class 5']

print(classification\_report(y\_exp, y\_exp1, target\_names=target\_names))

import itertools

import numpy as np

import matplotlib.pyplot as plt

from sklearn import svm, datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

#print(cm)

#plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

#plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt),

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.tight\_layout()

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_exp, y\_exp1)

np.set\_printoptions(precision=2)

# Plot non-normalized confusion matrix

fig1 = plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3','4','5'],

title='Confusion matrix, without normalization')

# Plot normalized confusion matrix

fig2 = plt.figure()

plot\_confusion\_matrix(cnf\_matrix, classes=['0','1','2','3','4','5'], normalize=True,

title='Normalized confusion matrix')

#fig1.savefig("nonnormalized.png")

#fig2.savefig("normalized.png")

import scikitplot as skplt

skplt.metrics.plot\_confusion\_matrix(y\_exp, y\_exp1, normalize=False)

plt.savefig("nonnormalized1.png")

# In[67]:

# Compute ROC curve and ROC area for each class

import numpy as np

import matplotlib.pyplot as plt

from itertools import cycle

from sklearn import svm, datasets

from sklearn.metrics import roc\_curve, auc

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import label\_binarize

from sklearn.multiclass import OneVsRestClassifier

from scipy import interp

fpr = dict()

tpr = dict()

roc\_auc = dict()

n\_classes=6

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y\_test1[:,i], y\_predict[:,i])

roc\_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area

fpr["micro"], tpr["micro"], \_ = roc\_curve(y\_test1.ravel(), y\_predict.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

lw=2

# Compute macro-average ROC curve and ROC area

# First aggregate all false positive rates

all\_fpr = np.unique(np.concatenate([fpr[i] for i in range(n\_classes)]))

# Then interpolate all ROC curves at this points

mean\_tpr = np.zeros\_like(all\_fpr)

for i in range(n\_classes):

mean\_tpr += interp(all\_fpr, fpr[i], tpr[i])

# Finally average it and compute AUC

mean\_tpr /= n\_classes

fpr["macro"] = all\_fpr

tpr["macro"] = mean\_tpr

roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])

# Plot all ROC curves

fig3 = plt.figure()

colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])

for i, color in zip(range(n\_classes), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=lw,

label='ROC curve of class {0} (area = {1:0.2f})'

''.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

#plt.title('Some extension of Receiver operating characteristic to multi-class')

plt.legend(loc="lower right")

#plt.show()

fig3.savefig("roc.png")

(cv) namosvision@gpuserver:~/SAT4and6$ python3 sat4\_6.py

Using TensorFlow backend.

(28, 28, 4, 81000)

(28, 28, 4, 324000)

100000

100000

100000

24000

-------------------------------------------

K shape= (324000, 28, 28, 2)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv2d\_1 (Conv2D) (None, 26, 26, 16) 304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_2 (Conv2D) (None, 24, 24, 32) 4640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_1 (MaxPooling2 (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_1 (Dropout) (None, 12, 12, 32) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_3 (Conv2D) (None, 10, 10, 32) 9248

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv2d\_4 (Conv2D) (None, 8, 8, 64) 18496

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling2d\_2 (MaxPooling2 (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_2 (Dropout) (None, 4, 4, 64) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten\_1 (Flatten) (None, 1024) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_1 (Dense) (None, 128) 131200

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout\_3 (Dropout) (None, 128) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense\_2 (Dense) (None, 6) 774

=================================================================

Total params: 164,662

Trainable params: 164,662

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Train on 259200 samples, validate on 64800 samples

Epoch 1/50

2019-03-22 11:06:39.792050: I tensorflow/core/platform/cpu\_feature\_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2 FMA

259200/259200 [==============================] - 154s 593us/step - loss: 0.2246 - acc: 0.9118 - val\_loss: 0.1117 - val\_acc: 0.9601

Epoch 2/50

259200/259200 [==============================] - 154s 596us/step - loss: 0.1118 - acc: 0.9601 - val\_loss: 0.0889 - val\_acc: 0.9698

Epoch 3/50

259200/259200 [==============================] - 154s 595us/step - loss: 0.0885 - acc: 0.9683 - val\_loss: 0.0692 - val\_acc: 0.9738

Epoch 4/50

259200/259200 [==============================] - 154s 594us/step - loss: 0.0790 - acc: 0.9721 - val\_loss: 0.0573 - val\_acc: 0.9789

Epoch 5/50

259200/259200 [==============================] - 157s 604us/step - loss: 0.0699 - acc: 0.9751 - val\_loss: 0.0554 - val\_acc: 0.9802

Epoch 6/50

259200/259200 [==============================] - 155s 598us/step - loss: 0.0640 - acc: 0.9775 - val\_loss: 0.0598 - val\_acc: 0.9786

Epoch 7/50

259200/259200 [==============================] - 156s 601us/step - loss: 0.0601 - acc: 0.9789 - val\_loss: 0.0520 - val\_acc: 0.9816

Epoch 8/50

259200/259200 [==============================] - 156s 602us/step - loss: 0.0564 - acc: 0.9805 - val\_loss: 0.0433 - val\_acc: 0.9855

Epoch 9/50

259200/259200 [==============================] - 157s 604us/step - loss: 0.0517 - acc: 0.9824 - val\_loss: 0.0366 - val\_acc: 0.9870

Epoch 10/50

259200/259200 [==============================] - 156s 603us/step - loss: 0.0498 - acc: 0.9829 - val\_loss: 0.0474 - val\_acc: 0.9841

Epoch 11/50

259200/259200 [==============================] - 156s 601us/step - loss: 0.0466 - acc: 0.9838 - val\_loss: 0.0339 - val\_acc: 0.9887

Epoch 12/50

259200/259200 [==============================] - 155s 599us/step - loss: 0.0456 - acc: 0.9844 - val\_loss: 0.0382 - val\_acc: 0.9869

Epoch 13/50

259200/259200 [==============================] - 155s 596us/step - loss: 0.0438 - acc: 0.9852 - val\_loss: 0.0350 - val\_acc: 0.9882

Epoch 14/50

259200/259200 [==============================] - 154s 592us/step - loss: 0.0417 - acc: 0.9858 - val\_loss: 0.0363 - val\_acc: 0.9872

Epoch 15/50

259200/259200 [==============================] - 154s 594us/step - loss: 0.0422 - acc: 0.9856 - val\_loss: 0.0321 - val\_acc: 0.9883

Epoch 16/50

259200/259200 [==============================] - 167s 644us/step - loss: 0.0386 - acc: 0.9873 - val\_loss: 0.0317 - val\_acc: 0.9890

Epoch 17/50

259200/259200 [==============================] - 192s 742us/step - loss: 0.0380 - acc: 0.9871 - val\_loss: 0.0295 - val\_acc: 0.9897

Epoch 18/50

259200/259200 [==============================] - 199s 767us/step - loss: 0.0374 - acc: 0.9873 - val\_loss: 0.0378 - val\_acc: 0.9873

Epoch 19/50

259200/259200 [==============================] - 199s 770us/step - loss: 0.0375 - acc: 0.9876 - val\_loss: 0.0393 - val\_acc: 0.9868

Epoch 20/50

259200/259200 [==============================] - 198s 765us/step - loss: 0.0360 - acc: 0.9879 - val\_loss: 0.0251 - val\_acc: 0.9917

Epoch 21/50

259200/259200 [==============================] - 197s 758us/step - loss: 0.0369 - acc: 0.9880 - val\_loss: 0.0364 - val\_acc: 0.9886

Epoch 22/50

259200/259200 [==============================] - 199s 766us/step - loss: 0.0338 - acc: 0.9885 - val\_loss: 0.0290 - val\_acc: 0.9910

Epoch 23/50

259200/259200 [==============================] - 188s 724us/step - loss: 0.0350 - acc: 0.9884 - val\_loss: 0.0496 - val\_acc: 0.9829

Epoch 24/50

259200/259200 [==============================] - 154s 594us/step - loss: 0.0326 - acc: 0.9891 - val\_loss: 0.0326 - val\_acc: 0.9881

Epoch 25/50

259200/259200 [==============================] - 154s 596us/step - loss: 0.0325 - acc: 0.9893 - val\_loss: 0.0285 - val\_acc: 0.9900

Epoch 26/50

259200/259200 [==============================] - 154s 594us/step - loss: 0.0313 - acc: 0.9895 - val\_loss: 0.0290 - val\_acc: 0.9896

Epoch 27/50

259200/259200 [==============================] - 154s 594us/step - loss: 0.0316 - acc: 0.9896 - val\_loss: 0.0289 - val\_acc: 0.9919

Epoch 28/50

259200/259200 [==============================] - 154s 596us/step - loss: 0.0315 - acc: 0.9895 - val\_loss: 0.0217 - val\_acc: 0.9924

Epoch 29/50

259200/259200 [==============================] - 154s 593us/step - loss: 0.0320 - acc: 0.9895 - val\_loss: 0.0300 - val\_acc: 0.9896

Epoch 30/50

259200/259200 [==============================] - 154s 593us/step - loss: 0.0303 - acc: 0.9899 - val\_loss: 0.0236 - val\_acc: 0.9920

Epoch 31/50

259200/259200 [==============================] - 154s 592us/step - loss: 0.0302 - acc: 0.9898 - val\_loss: 0.0342 - val\_acc: 0.9890

Epoch 32/50

259200/259200 [==============================] - 153s 592us/step - loss: 0.0295 - acc: 0.9900 - val\_loss: 0.0209 - val\_acc: 0.9927

Epoch 33/50

259200/259200 [==============================] - 153s 591us/step - loss: 0.0305 - acc: 0.9899 - val\_loss: 0.0244 - val\_acc: 0.9915

Epoch 34/50

259200/259200 [==============================] - 153s 591us/step - loss: 0.0286 - acc: 0.9903 - val\_loss: 0.0202 - val\_acc: 0.9933

Epoch 35/50

259200/259200 [==============================] - 154s 593us/step - loss: 0.0279 - acc: 0.9908 - val\_loss: 0.0261 - val\_acc: 0.9893

Epoch 36/50

259200/259200 [==============================] - 179s 692us/step - loss: 0.0283 - acc: 0.9905 - val\_loss: 0.0269 - val\_acc: 0.9917

Epoch 37/50

259200/259200 [==============================] - 213s 823us/step - loss: 0.0275 - acc: 0.9908 - val\_loss: 0.0171 - val\_acc: 0.9944

Epoch 38/50

259200/259200 [==============================] - 211s 814us/step - loss: 0.0276 - acc: 0.9908 - val\_loss: 0.0315 - val\_acc: 0.9889

Epoch 39/50

259200/259200 [==============================] - 217s 838us/step - loss: 0.0276 - acc: 0.9911 - val\_loss: 0.0237 - val\_acc: 0.9926

Epoch 40/50

259200/259200 [==============================] - 218s 841us/step - loss: 0.0268 - acc: 0.9912 - val\_loss: 0.0190 - val\_acc: 0.9934

Epoch 41/50

259200/259200 [==============================] - 219s 846us/step - loss: 0.0268 - acc: 0.9911 - val\_loss: 0.0354 - val\_acc: 0.9874

Epoch 42/50

259200/259200 [==============================] - 216s 834us/step - loss: 0.0262 - acc: 0.9912 - val\_loss: 0.0231 - val\_acc: 0.9917

Epoch 43/50

259200/259200 [==============================] - 181s 699us/step - loss: 0.0260 - acc: 0.9914 - val\_loss: 0.0266 - val\_acc: 0.9916

Epoch 44/50

259200/259200 [==============================] - 155s 597us/step - loss: 0.0258 - acc: 0.9917 - val\_loss: 0.0179 - val\_acc: 0.9938

Epoch 45/50

259200/259200 [==============================] - 154s 596us/step - loss: 0.0254 - acc: 0.9917 - val\_loss: 0.0165 - val\_acc: 0.9944

Epoch 46/50

259200/259200 [==============================] - 155s 597us/step - loss: 0.0250 - acc: 0.9918 - val\_loss: 0.0213 - val\_acc: 0.9921

Epoch 47/50

259200/259200 [==============================] - 154s 596us/step - loss: 0.0251 - acc: 0.9917 - val\_loss: 0.0285 - val\_acc: 0.9899

Epoch 48/50

259200/259200 [==============================] - 155s 597us/step - loss: 0.0247 - acc: 0.9920 - val\_loss: 0.0190 - val\_acc: 0.9932

Epoch 49/50

259200/259200 [==============================] - 154s 595us/step - loss: 0.0259 - acc: 0.9915 - val\_loss: 0.0162 - val\_acc: 0.9944

Epoch 50/50

259200/259200 [==============================] - 155s 597us/step - loss: 0.0234 - acc: 0.9923 - val\_loss: 0.0162 - val\_acc: 0.9941

81000

81000/81000 [==============================] - 13s 156us/step

Loss: 0.02,Accuracy: 99.44%

0.9943703703703703

[0.99356223 0.99790437 0.99745457 0.97334266 0.99272198 1. ]

[0.99730749 0.98519083 0.99450123 0.9942839 0.9884058 1. ]

[0.99543134 0.99150685 0.99597571 0.98370184 0.99055919 1. ]

precision recall f1-score support

class 0 0.99 1.00 1.00 3714

class 1 1.00 0.99 0.99 18367

class 2 1.00 0.99 1.00 14185

class 3 0.97 0.99 0.98 12596

class 4 0.99 0.99 0.99 2070

class 5 1.00 1.00 1.00 30068

micro avg 0.99 0.99 0.99 81000

macro avg 0.99 0.99 0.99 81000

weighted avg 0.99 0.99 0.99 81000

Confusion matrix, without normalization

Normalized confusion matrix