

## Common Test I. Multi-Class Classification

**Task:** Build a model for classifying the images into lenses using PyTorch or Keras. Pick the most appropriate approach and discuss your strategy.

**Dataset:** [dataset.zip - Google Drive](#)

**Dataset Description:** The Dataset consists of three classes, strong lensing images with no substructure, subhalo substructure, and vortex substructure. The images have been normalized using min-max normalization, but you are free to use any normalization or data augmentation methods to improve your results.

**Evaluation Metrics:** ROC curve (Receiver Operating Characteristic curve) and AUC score (Area Under the ROC Curve)

Downloading the dataset:

[+ Code](#)[+ Text](#)

```
from google.colab import drive
drive.mount('/content/gdrive')
!unzip -qq gdrive/MyDrive/dataset_deeplense.zip
print('Extraction done.')
```

```
Mounted at /content/gdrive
Extraction done.
```

Setting up imports:

```
import os
import numpy as np
import tensorflow as tf
import cv2
from sklearn.metrics import roc_auc_score, roc_curve, auc, classification_report
import matplotlib.pyplot as plt
import random
from sklearn.preprocessing import LabelEncoder
```

Visualizing the data:

```
# Define the input paths
train = os.listdir('./dataset/train')
val = os.listdir('./dataset/val')
train_path1 = './dataset/train/no'
train_files1 = [os.path.join(train_path1, f) for f in os.listdir(train_path1) if f.endswith(".jpg")]
train_path2 = './dataset/train/sphere'
```

```
train_files2 = [os.path.join(train_path2, f) for f in os.listdir(train_path2) if f.endswith("
train_path3 = './dataset/train/vort'
train_files3 = [os.path.join(train_path3, f) for f in os.listdir(train_path3) if f.endswith("

# Number of samples to display per class
n = 5

# Plot the samples
i = 1
print('Samples with no substructure: ')
plt.rcParams['figure.figsize'] = [14, 14]
for image in train_files1[:n]:
    ax = plt.subplot(3,n,i)
    plt.imshow(np.load(image).reshape(150,150), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    i += 1
plt.show()

print('Samples with spherical substructure: ')
plt.rcParams['figure.figsize'] = [14, 14]
for image in train_files2[:n]:
    ax = plt.subplot(3,n,i)
    plt.imshow(np.load(image).reshape(150,150), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    i += 1
plt.show()

print('Samples with vortex substructure: ')
plt.rcParams['figure.figsize'] = [14, 14]
for image in train_files3[:n]:
    ax = plt.subplot(3,n,i)
    plt.imshow(np.load(image).reshape(150,150), cmap='gray')
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
    i += 1
```

Loading the training data:

```
X_train = []
Y_train = []

for i in range(len(train)):
    if os.path.isdir("./dataset/train/" + train[i]):
        for j in range(len(os.listdir('./dataset/train/' + train[i]))):
            img = np.load('./dataset/train/' + train[i] + '/' + os.listdir('./dataset/train/' + train[i])[j] + '.npy')
            img = cv2.resize(img[0], (75, 75))
            img = np.stack([img, img, img], -1)
            X_train.append(img)
            Y_train.append(train[i])
        print(train[i] + ' loaded')

no loaded
vort loaded
sphere loaded
```

Introducing randomness in the training dataset:

```
train = list(zip(X_train, Y_train))
random.shuffle(train)
X_train, Y_train = zip(*train)
del train
```

Loading the validation data:

```
X_test = []
Y_test = []
```

```
for i in range(len(val)):
```

```

if os.path.isdir("./dataset/val/" + val[i]):
    for j in range(len(os.listdir('./dataset/val/' + val[i]))):
        img = np.load('./dataset/val/' + val[i] + '/' + os.listdir('./dataset/val/' + val[i])[j])
        img = cv2.resize(img[0], (75, 75))
        img = np.stack([img, img, img], -1)
        X_test.append(img)
        Y_test.append(val[i])
    print(val[i] + ' loaded')

no loaded
vort loaded
sphere loaded

```

Introducing randomness in the validation dataset:

```

val = list(zip(X_test, Y_test))
random.shuffle(val)
X_test, Y_test = zip(*val)
del val

```

Pre processing the datasets for fitting to the CNN model:

```

le = LabelEncoder()

Y_train = le.fit_transform(Y_train)
Y_test = le.fit_transform(Y_test)

X_train = np.array(X_train)
Y_train = np.array(Y_train)
X_test = np.array(X_test)
Y_test = np.array(Y_test)

X_train.shape, Y_train.shape, X_test.shape, Y_test.shape

((30000, 75, 75, 3), (30000,)), (7500, 75, 75, 3), (7500,))

```

Loading and compiling the model:

```

model = tf.keras.applications.Xception(
    include_top=True,
    weights=None,
    input_shape=(75, 75, 3),
    classes=3,
    classifier_activation='softmax'
)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

```

A callback for saving the weights in order of increased accuracy that can be applied to the model for faster performance:


```
filepath="classifier_weights2-improvement-{epoch:02d}-{val_accuracy:.2f}.hdf5"
checkpoint1 = tf.keras.callbacks.ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1,
callbacks_list = [checkpoint1]
```

Fitting the model with the processed data:

```
history = model.fit(X_train,
                    Y_train,
                    epochs=20,
                    validation_data=(X_test, Y_test)
)
```

```
Epoch 1/20
938/938 [=====] - 99s 89ms/step - loss: 1.1366 - accuracy: 0.33
Epoch 2/20
938/938 [=====] - 82s 88ms/step - loss: 1.1179 - accuracy: 0.33
Epoch 3/20
938/938 [=====] - 84s 90ms/step - loss: 1.1130 - accuracy: 0.34
Epoch 4/20
938/938 [=====] - 85s 91ms/step - loss: 1.0711 - accuracy: 0.46
Epoch 5/20
938/938 [=====] - 86s 92ms/step - loss: 1.0084 - accuracy: 0.46
Epoch 6/20
938/938 [=====] - 87s 93ms/step - loss: 0.9714 - accuracy: 0.49
Epoch 7/20
938/938 [=====] - 87s 92ms/step - loss: 0.9456 - accuracy: 0.51
Epoch 8/20
938/938 [=====] - 87s 93ms/step - loss: 0.9246 - accuracy: 0.53
Epoch 9/20
938/938 [=====] - 86s 92ms/step - loss: 0.9062 - accuracy: 0.54
Epoch 10/20
938/938 [=====] - 86s 92ms/step - loss: 0.8840 - accuracy: 0.55
Epoch 11/20
938/938 [=====] - 86s 92ms/step - loss: 0.8468 - accuracy: 0.58
Epoch 12/20
938/938 [=====] - 86s 92ms/step - loss: 0.7768 - accuracy: 0.63
Epoch 13/20
938/938 [=====] - 86s 92ms/step - loss: 0.6826 - accuracy: 0.69
Epoch 14/20
938/938 [=====] - 91s 97ms/step - loss: 0.6073 - accuracy: 0.73
Epoch 15/20
938/938 [=====] - 91s 97ms/step - loss: 0.5501 - accuracy: 0.76
Epoch 16/20
938/938 [=====] - 91s 97ms/step - loss: 0.4912 - accuracy: 0.79
Epoch 17/20
938/938 [=====] - 86s 91ms/step - loss: 0.4370 - accuracy: 0.82
Epoch 18/20
```

```
938/938 [=====] - 86s 92ms/step - loss: 0.3886 - accuracy: 0.84  
Epoch 19/20  
938/938 [=====] - 86s 91ms/step - loss: 0.3417 - accuracy: 0.86  
Epoch 20/20  
938/938 [=====] - 86s 91ms/step - loss: 0.3079 - accuracy: 0.88
```



Plotting the accuracy metrics of the model:

```
plt.plot(history.history['accuracy'])  
plt.plot(history.history['val_accuracy'])  
plt.title('Accuracy of the model')  
plt.ylabel('Accuracy')  
plt.xlabel('Epochs')  
plt.legend(['Train', 'Test'], loc='lower right')  
plt.show()
```

```
plt.plot(history.history['loss'])  
plt.plot(history.history['val_loss'])  
plt.title('Loss metrics of the model')  
plt.ylabel('Loss')  
plt.xlabel('Epochs')  
plt.legend(['train', 'test'], loc='upper right')  
plt.show()
```



```
best_epoch=np.argmax(history.history['val_accuracy'])
best_acc=np.max(history.history['val_accuracy'])
model.load_weights(f"classifier_weights2-improvement-{best_epoch+1}-{best_acc:.2f}.hdf5")
```

Plotting the ROC AUC curve:

```
predictions = model.predict(X_test)

for pred in predictions:
    k = np.argmax(i)
    for i in range(3):
        if i == k:
            pred[i] = 1
        else:
            pred[i] = 0

temp_test_y = []

for i in range(len(Y_test)):
    a = [0, 0, 0]
    a[Y_test[i]] = 1
    temp_test_y.append(a)

temp_test_y = np.array(temp_test_y)
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(temp_test_y[:, i], predictions[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])
colors = ['red', 'blue', 'green']

for i, color in zip(range(3), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label='ROC curve of class {0} (area = {1:0.2f'

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
```



```
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver Operating Characteristic (Dense Neural Network model)')  
plt.legend(loc="lower right")  
plt.show()
```

