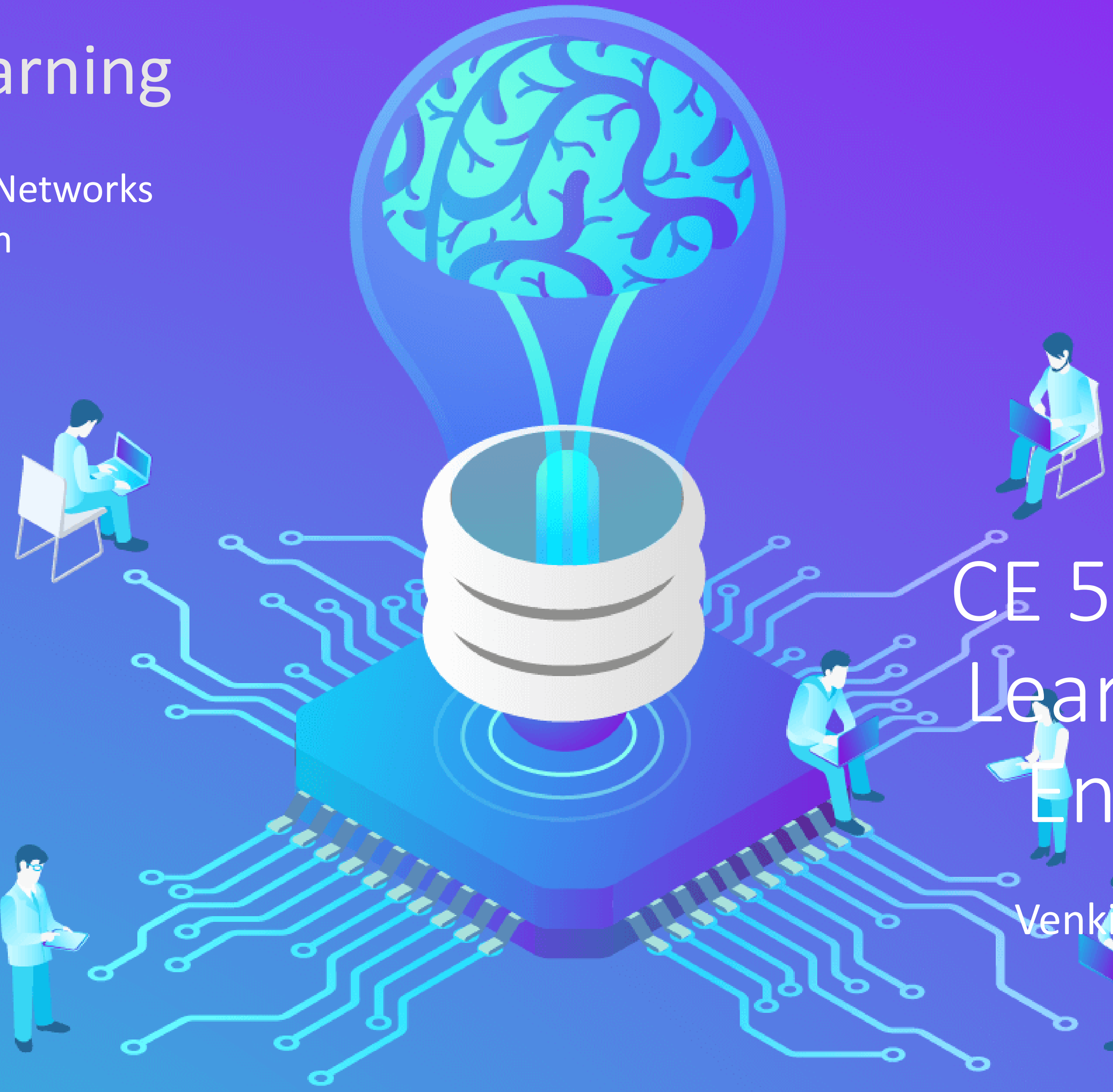
Machine Learning

Convolution Neural Networks
using Python

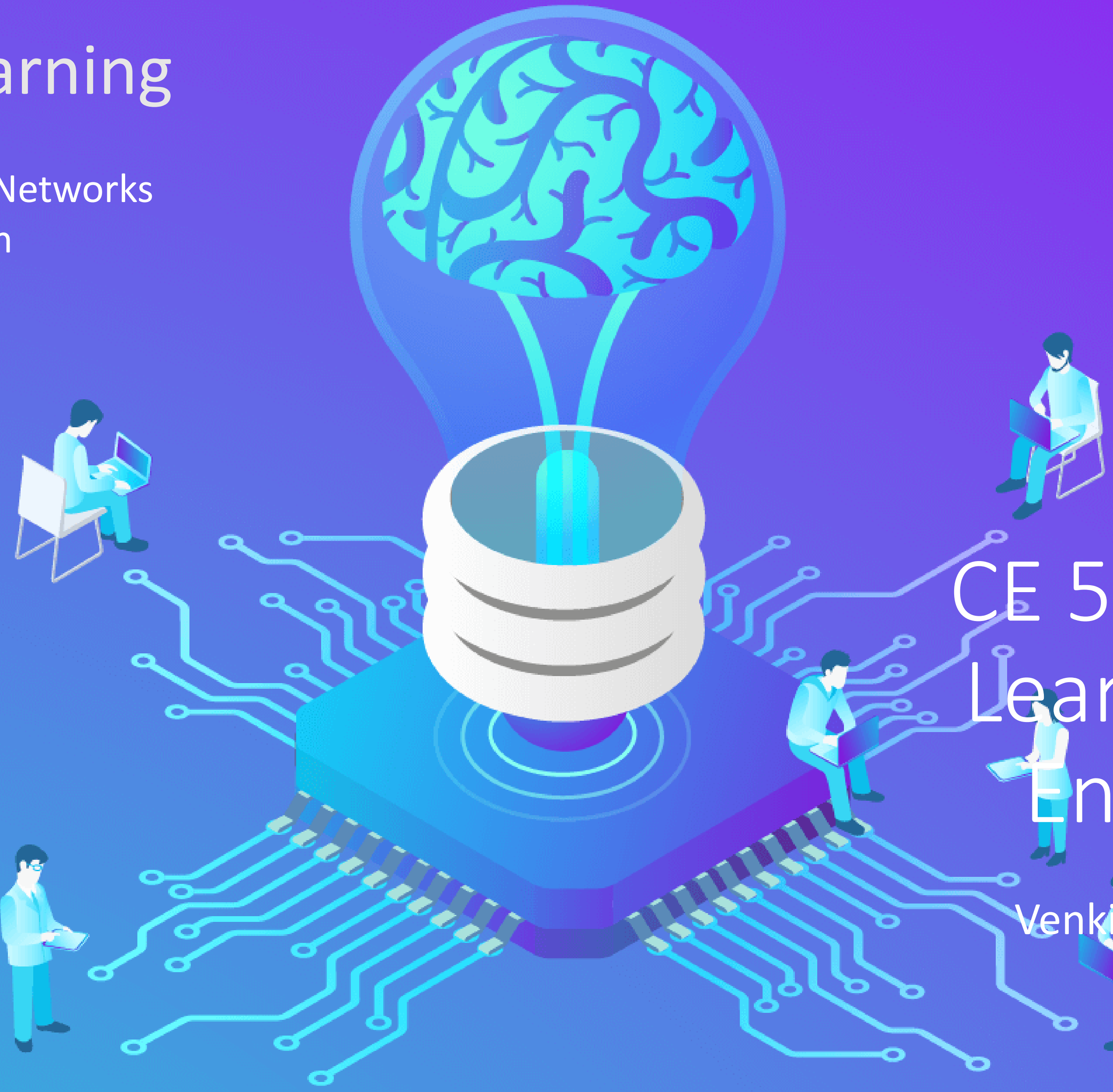


CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Machine Learning

Convolution Neural Networks
using Python



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

Problem Statement

- Develop a CNN Model to use images to classify
  - Images with cracks in pavements (CP)
  - Images with no cracks in pavements (UP)
- Data used in this study is based on SDNET 2018

SDNET2018: A concrete crack image dataset for machine learning applications

Marc Maguire, *Utah State University* Follow
Sattar Dorafshan, *Utah State University* Follow
Robert J. Thomas, *Utah State University* Follow

Description

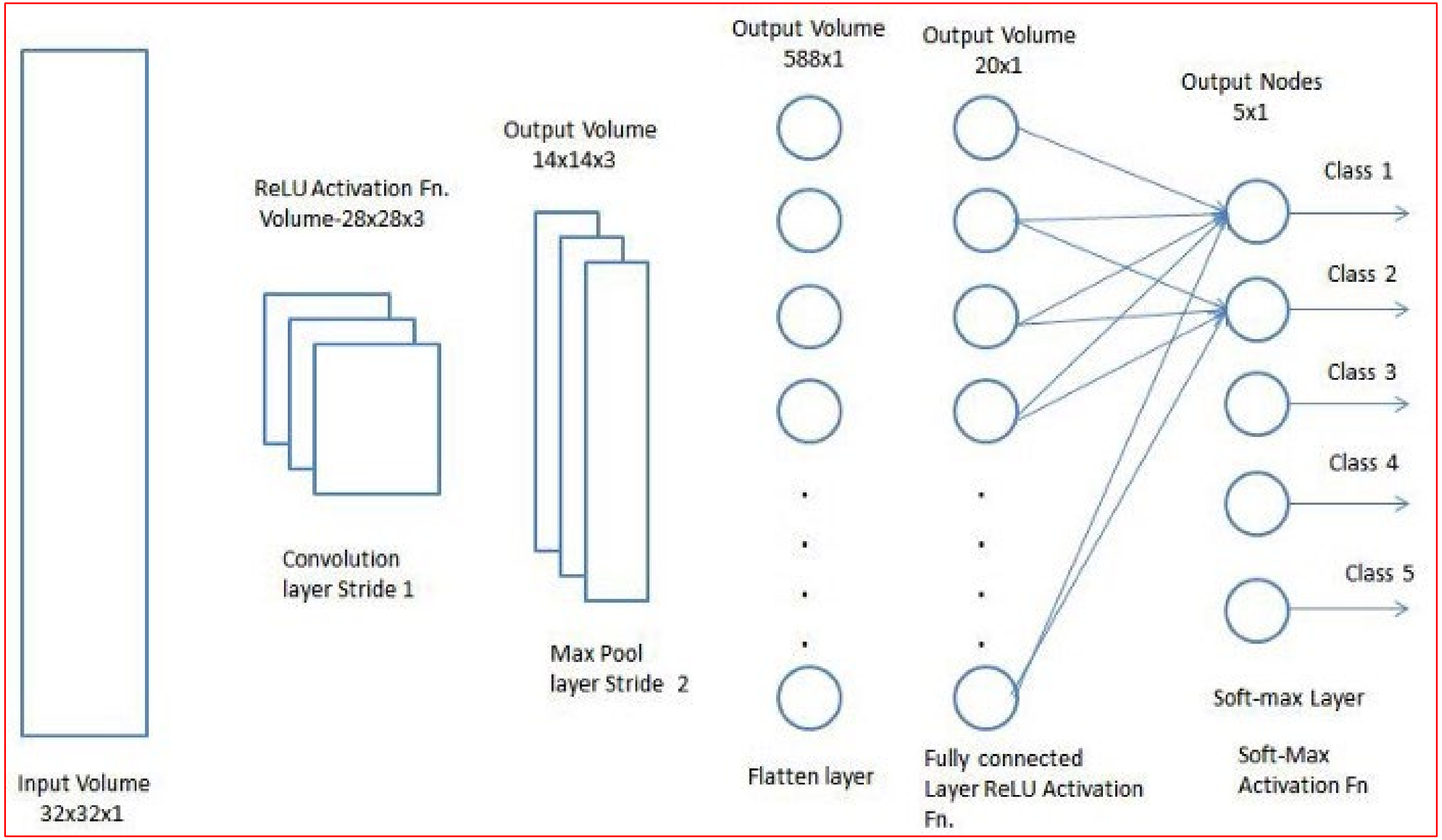
SDNET2018 is an annotated image dataset for training, validation, and benchmarking of artificial intelligence based crack detection algorithms for concrete. SDNET2018 contains over 56,000 images of cracked and non-cracked concrete bridge decks, walls, and pavements. The dataset includes cracks as narrow as 0.06 mm and as wide as 25 mm. The dataset also includes images with a variety of obstructions, including shadows, surface roughness, scaling, edges, holes, and background debris. SDNET2018 will be useful for the continued development of concrete crack detection algorithms based on deep learning convolutional neural networks, which are a subject of continued research in the field of structural health monitoring. .jpe

OCLC

1078404353

https://digitalcommons.usu.edu/all_datasets/48/

Implementation of CNNs



Tensorflow Keras has high-end tools for Implementing CNNs

Data Sampling

- The database comprised of approximately
  - 2600 images of cracks in pavements
  - 22000 images of uncracked pavement surfaces
- Images taken with a 19 MP Nikon Camera
- Approximately 2600 images of uncracked pavements were randomly sampled
  - To match the same number of cracked pavement images
- The images were randomly split into 80% training + 10% resting and 10% validation

Sampled data were organized into two folders CP and UP for Train, Test and Validation datasets which were in separate folders

Python Implementation - Steps

- Step 1: Load Required Libraries and Modules
  - We will need preprocessing layers, models and optimizers modules from tensorflow keras

  Metrics from sklearn and plotting from seaborn and matplotlib

```
## Load Libraries and Modules
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten
from tensorflow.keras.layers import BatchNormalization, Conv2D,
MaxPool2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
import sklearn.metrics
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
import itertools
import os
import random
import seaborn as sns
import matplotlib.pyplot as plt
```

Working Directory and File Paths

- Step 2: Set up working directory and File paths for image data
- In image processing file paths are important
  - These are paths to train, test and validation datasets

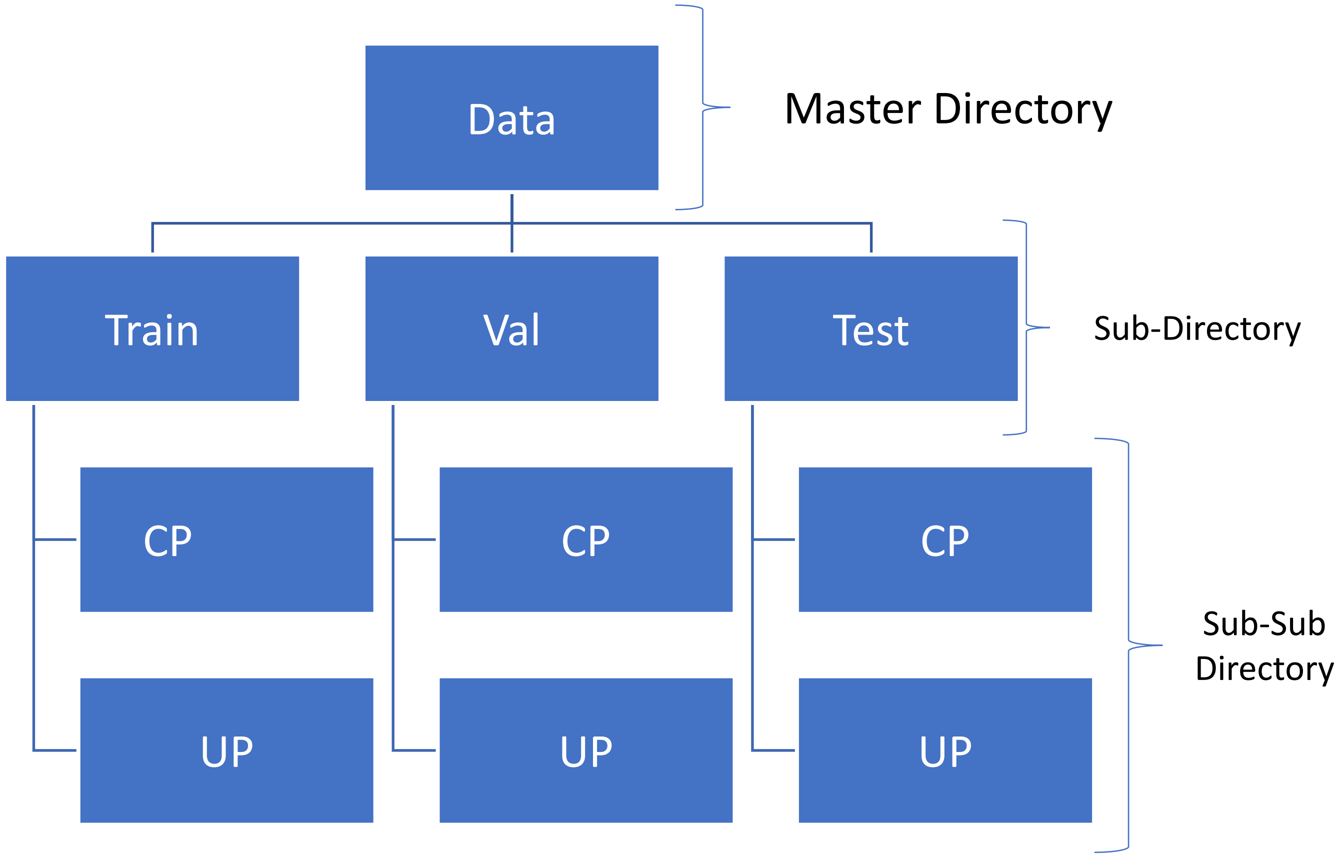
```
# Set working direcgtory
dir = '/home/vuddameri/CE5319/CNN/Pavement/FinalData'
os.chdir(dir)

#set path for train, test and validate datasets
train_path =
"/home/vuddameri/CE5319/CNN/Pavement/FinalData/train"
test_path = "/home/vuddameri/CE5319/CNN/Pavement/FinalData/test"
valid_path = "/home/vuddameri/CE5319/CNN/Pavement/FinalData/val"
```

For Image Classification - Data has to be put into a specific format

Python Implementation - Steps

- Image datasets have to be stored in a hierarchical manner
  - Folder - (Master data folder)
  - Subfolder - (Train, Val and Test sub-folders
  - Sub-Subfolder - (CP and UP)
    - CP Cracked pavement pictures
    - UP Uncracked pavement pictures



Getting Data into a proper directory structure is essential step of CNN modeling

The Directory Setup helps understand the Classes (Y Labels) within the model

Reading Data

- There is an ImageDataGenerator function in keras pre-processing module
  - It will help instantiate an generator object
- The ImageDataGenerator has an iterator that can loop through images and read them into memory
  - flow_from_directory method of ImageDataGenerator

Data are read in batches. This allows data is managed efficiently (not loading a lot of data into memory and also allows manipulating data till it is read into the memory

In addition to reading data we can also perform data processing and Image Augmentation Steps

-- Resize Images, Perform Image Augmentation, etc. --

Reading Data

```
# Preprocess images and make them keras ready
classz = ['CP', 'UP'] # identify classes
batchsize = 10  # Can be changed
train_batches = ImageDataGenerator()
train_batches = train_batches.flow_from_directory(directory=train_path,
                         target_size=(224,224), classes=classz,
                         batch_size=batchsize)

valid_batches = ImageDataGenerator()
valid_batches = valid_batches.flow_from_directory(directory=valid_path,
                         target_size=(224,224), classes=classz,
                         batch_size=10)
test_batches = ImageDataGenerator()
test_batches = test_batches.flow_from_directory(directory=test_path,
                        target_size=(224,224), classes=classz,
                        batch_size=10, shuffle=False)
```

Specify the Model

- This will require some trial-and-error
- Use sequential(),
- Use Conv2D, MaxPool2D in earlier laters
- Flatten() before passing to Dense ANN Layer

```
# Specify architecture
model = Sequential([
   Conv2D(filters=32, kernel_size=(3, 3), activation='relu', padding = 'same',
         input_shape=(224,224,3)),
   MaxPool2D(pool_size=(2, 2), strides=2),
   Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
   MaxPool2D(pool_size=(2, 2), strides=2),
   Flatten(),
   Dense(units=2, activation='softmax')
])
```

Compile Model

- Specify which optimizer to use, which loss function, which metric to track

```
# Compile model
model.compile(optimizer=Adam(learning_rate=0.0001),
            loss='categorical_crossentropy',
            metrics=['accuracy'])
```

Fit the Model

- Make sure all Programs are closed before fitting the model
  - THIS IS A MEMORY INTENSIVE OPERATION!!

```
model.fit(
    x = train_batches,
    steps_per_epoch=train_batches.samples // batchsize,
    epochs=10,
    validation_data=valid_batches,
    validation_steps=valid_batches.samples // batchsize,
    verbose=2)
```

CNN models are susceptible to overfitting so having a validation dataset is good

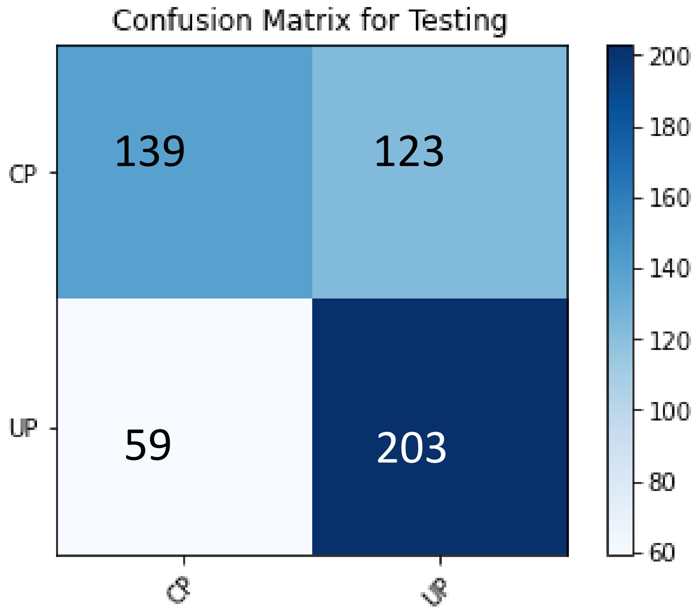
Evaluate the Model

- There are two types of predictions provided
  - Predict provides the probabilities
    - Used for computing AUC
  - Predict_class classifies based on maximum probability
    - Used for computing contingency tables and analysis
    - Deprecated and will be removed after 01-01-2021
      - DO NOT USE

```
# Make Predictions for probabilities and classes
predictions = model.predict(x = test_batches, verbose=0)
predict_class = np.argmax(predictions, axis=-1)
```

Create Confusion Matrix

```
# Make confusion matrix and plot it
cm = sklearn.metrics.confusion_matrix(y_true=test_batches.classes,
                                      y_pred=predict_class)
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1, 0]).ravel()
tn, fp, fn, tp # Write data for each element
cmap = 'Blues'
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.title("Confusion Matrix for Testing")
plt.colorbar()
tick_marks = np.arange(len(classz))
plt.xticks(tick_marks, classz, rotation=45)
plt.yticks(tick_marks, classz)
```



Predicted

CP = 0 (Cracked) and UP = 1 (Uncracked)

Confusion Matrix Metrics

```
# Evaluate usng accuracy, precision, recall
print("Accuracy:",metrics.accuracy_score(Y_test, predict_class))
print("Precision:",metrics.precision_score(Y_test, predict_class))
print("Recall:",metrics.recall_score(Y_test, predict_class))
```

N = (TP+FP+FN+TP)

Accuracy = (TN + TP) /N

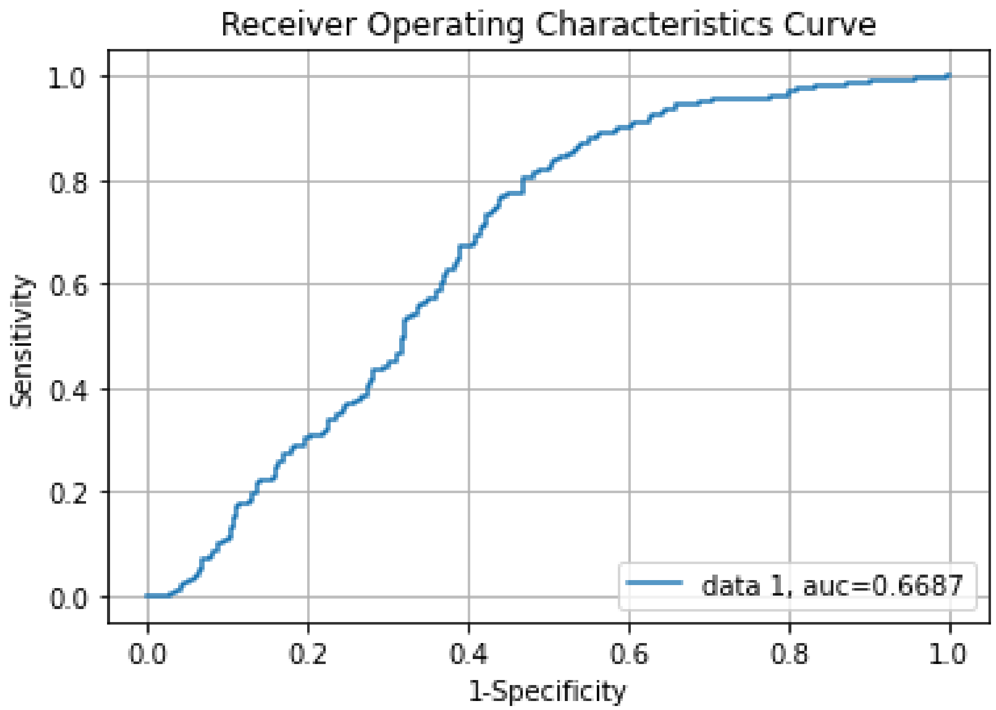
Precision = (TP/(TP + FP)

Recall = TP/(TP + FN)

		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

AUC Curve

```
#Construct ROC Curve and Plot it
y_proba = predictions[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_test,  y_proba)
auc = metrics.roc_auc_score(Y_test, y_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(round(auc,4)))
plt.legend(loc=4)
plt.title('Receiver Operating Characteristics Curve')
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.grid() # Plot the grid
plt.show() #show the curve
```



You should know

- What are CNNs
- How to implement them in Python
- How to arrange the image data into appropriate folders
  - Train, Test and Val
  - Sub-Sub-Folders of each category in each layer
- How to setup the model
  - Model Specification
  - Model Compilation
  - Model Fitting
- How to evaluate the model
  - Contingency table and ROC curve