```
eta name="MySite">
le>My Site</title>

yle>

animation_container {
    position:absolute;
    margin:auto;
    left:0; right:0;
    top:0; bottom:0;

tyle>

cript>
    function makeResponsive(isResp, respDim, isScale, scaleType) {
        var lastW, lastH, lastS=1;
        window.addEventListener('resize', resizeCanvas);
        resizeCanvas();
        function resizeCanvas() {
            var w = lib.properties.width, h = lib.properties.height;
            var iw = window.innerWidth, ih=window.innerHeight;
            var pRatio = window.devicePixelRatio || 1, xRatio=iw/w, yRatio=ih/h, sRatio=1;
            if(isResp) {
                if((respDim=='width'&&lastW==iw) || (respDim=='height'&&lastH==ih)) {
                    sRatio = lastS;
                }
                else if(!isScale) {
                    if(iw<w || ih<h)
                        sRatio = Math.min(xRatio, yRatio);
                }
                else if(scaleType==1) {
                    sRatio = Math.min(xRatio, yRatio);
                }
                else if(scaleType==2) {
                    sRatio = Math.max(xRatio, yRatio);
                }
            }
    }
    makeResponsive(true,'both',false,1);
    AdobeAn.compositionLoaded(lib.properties.id);
    fnStartAnimation();
```

LTIMindtree

POV

# Catch lightning in a bottle How to use image classification on Snowflake using deep learning

by **Srinivasaraghavan Sundar**

# Using Conventional Neural Networks

Back in my college days, while working on a thesis at the 11th hour (as was the norm), I remember coming upon the work of Dr. Carlos Ordonez and his team. Back in 2004, they had come up with a novel method to implement the K Means Clustering Algorithm using SQL.
I remember being in awe of the intricacies involved in bringing the machine learning algorithm into SQL.

In this post, we will take a look at how Convolutional Neural Network (CNN) can be used for Image Classification in SQL.

Now, to be clear, this would certainly be impossible without the various features developed by Snowflake. While we won't be coding this in pure SQL, we will be making use of Snowflake's support for unstructured data and Python UDFs in order to achieve this otherwise herculean task.

## Dataset Description

The problem that we are looking to solve today happens to be that of metal surface defect detection. The dataset can be found here:
https://www.kaggle.com/datasets/fantacher/neu-metal-surface-defects-data

This dataset has 6 different classes namely rolled-in scale (RS), patches (Pa), crazing (Cr), pitted surface (PS), inclusion (In) and scratches (Sc). For a given image containing a defect, our goal is to predict which defect this belongs to.

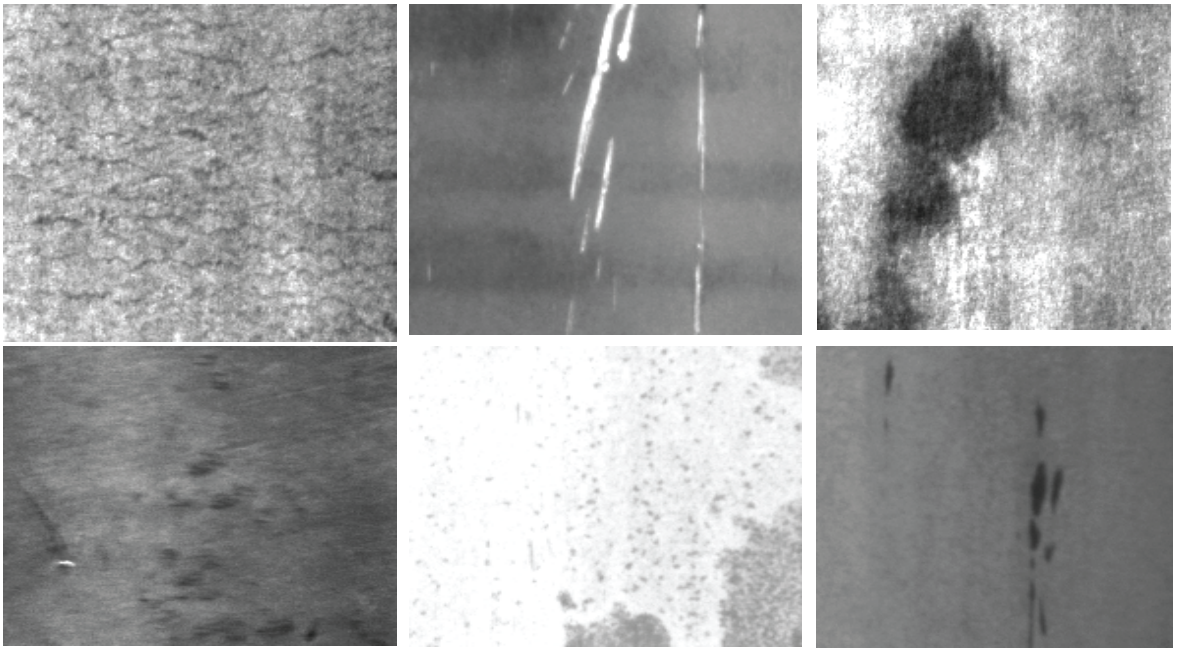

Fig 1: Sample data containing defects

## Model Declaration and Training

Let us start by training the model. This portion is done on a local system.
We make use of the keras and tensorflow libraries in order to construct a CNN model which takes in the image as an input and predicts which class it belongs to.

```
import numpy as np
from sklearn.datasets import load_files
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import img_to_array, load_img
#use image data generator to perform some basic preprocessing.
train_directory = 'train'
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
train_generator = train_datagen.flow_from_directory(
        train_directory,
        target_size=(200, 200),
        batch_size=16,
        class_mode='categorical')
test_directory='test'
test_datagen = ImageDataGenerator(rescale=1./255)
validation_directory = 'valid'
validation_generator = test_datagen.flow_from_directory(
        validation_directory,
        target_size=(200, 200),
        batch_size=16,
        class_mode='categorical')

#define the model.
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=64, kernel_size=(2,2),strides=(2, 2),padding='valid',dila-
tion_rate=(1, 1), activation='elu', input_shape=(200, 200, 3)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=None,padding='valid'),
    tf.keras.layers.Conv2D(filters=128, kernel_size=(2,2),strides=(2, 2),padding='valid',dila-
tion_rate=(1, 1), activation='elu'),
```

```
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=None,padding='valid'),
    tf.keras.layers.Conv2D(filters=256, kernel_size=(2,2),strides=(1, 1),padding='valid',dila-
tion_rate=(2, 2), activation='elu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2),strides=None,padding='valid'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512,activation='elu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(6, activation='softmax')
])

model.summary()
#compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

#fit the model and collect metrics.
history = model.fit(train_generator,
        batch_size = 32,
        epochs=20,
        validation_data=validation_generator,
        verbose=1, shuffle=True)

def load_dataset(path):
    data = load_files(path)
    files = np.array(data['filenames'])
    targets = np.array(data['target'])
    target_labels = np.array(data['target_names'])
    return files,targets,target_labels
x_test, y_test,target_labels = load_dataset(test_directory)
no_of_classes = len(np.unique(y_test))
y_test = to_categorical(y_test,no_of_classes)
def convert_image_to_array(files):
    images_as_array=[]
    for file in files:
        images_as_array.append(img_to_array(load_img(file)))
    return images_as_array
```

```
x_test = np.array(convert_image_to_array(x_test))
x_test = x_test.astype('float32')/255
y_pred = model.evaluate(x_test,y_test)
#save the final model as a h5 file
model.save("metal_defects_final.h5")
```

## Model Architecture and Metrics

The model architecture is as given below:

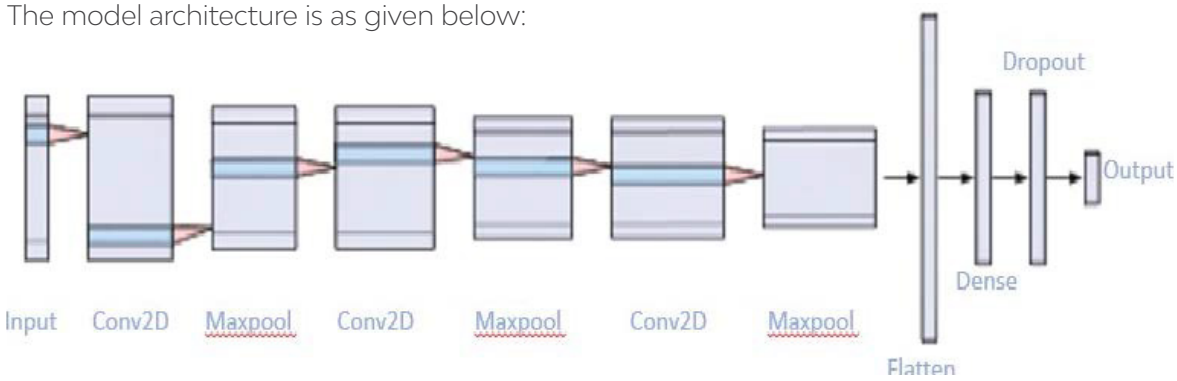


Fig 2: Model Architecture

Without delving into great detail about the model architecture, let me try to summarize the CNN shown above. Over here, we make use of 2D Dilated Convolution layers paired with 2D Maxpooling layers in order to obtain the embeddings of the image. As seen above, we make use of the ELU activation function. We make use of a dropout layer prior to the final output prediction in order to prevent overfitting. Finally the output layer makes use of the softmax activation function in order to determine the probabilities of the defect classes.

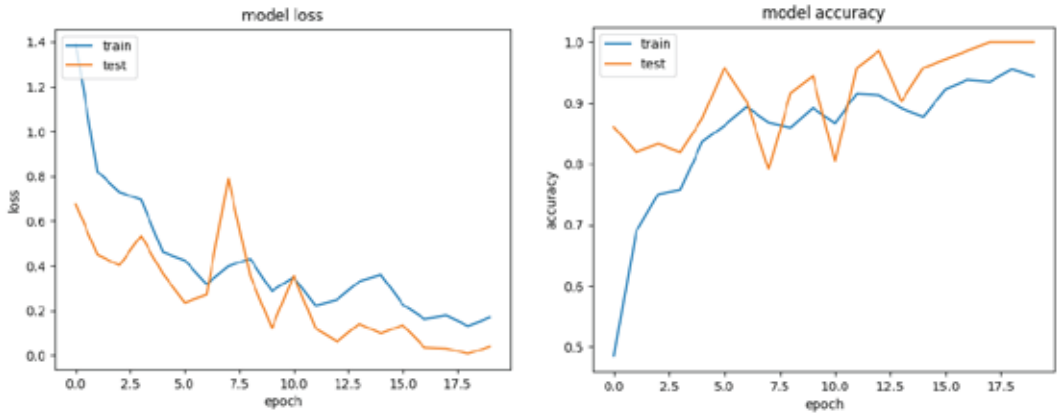


Fig 3: Model Loss and Accuracy vs No. of Epochs

This model has a test accuracy of 94.5%. The loss is observed to be <0.2. For the purposes of this experiment, this model is satisfactory.

For those who have a penchant for deep learning and would like to tweak some parameters and experiment on top of this model, here is a more detailed summary obtained from Netron (https://github.com/lutzroeder/netron)

## Bringing the Model to Snowflake

We save the model in a h5 file and upload onto snowflake stage using the PUT command. Make sure that you set the auto compress to False. Once that is done, we also upload an image from the test set onto the stage using the PUT command. Once again, we set auto_-compress to false.
Now, let us build a UDF in order to load the model and predict on our image.

```
create or replace function metal_defect_detection()
  returns variant
  language python
  runtime_version=3.8
  imports=('@metal_defects/metal_defects_final.h5','@metal_de-
fects/test/crazing/Cr_120.bmp')
  packages = ('pandas==1.2.3','numpy==1.20.1','keras','tensorflow','pillow','h5py==3.2.1')
  handler='compute'
as
$$
import sys
import h5py
import tensorflow as tf
import keras
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing.image import array_to_img, img_to_array, load_img
IMPORT_DIRECTORY_NAME = "snowflake_import_directory"
#declare import_dir to get the h5 file and the test image.
import_dir = sys._xoptions[IMPORT_DIRECTORY_NAME]
import os
#ensure to set file locking to false.
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

#define handler function
def compute():
```

```
#load model
  model=keras.models.load_model(import_dir +'metal_defects_final.h5')
#load image
x=np.array([img_to_array(load_img(import_dir+'Cr_120.bmp'))])
#predict image
  y_pred=model.predict_on_batch(x)
  prediction=y_pred.tolist()
#based on prediction output determine which category of defect this belongs in.
  index = prediction[0].index(1)
  if index==0:
    cat="CRAZING"
  elif index==1:
    cat='Inclusion'
  elif index==2:
    cat='Patches'
  elif index==3:
    cat='Pitted'
  elif index==4:
    cat='Rolled'
  elif index==5:
    cat='Scratches'
  return cat
$$;
```

Be sure to set HDF5 file locking to false in cases where you load a h5 file.

Once the UDF has been created, we run the following command

```
select metal_defect_detection();
```

## Dataflow of the Solution

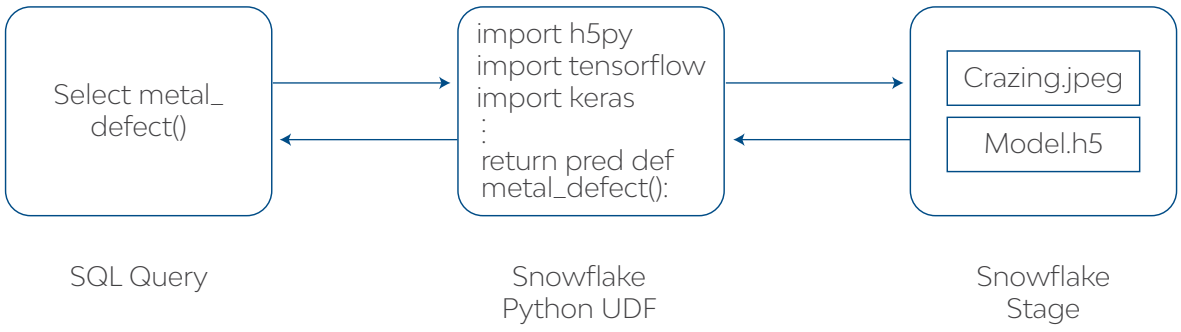


Fig 5: Dataflow Diagram

The above diagram represents our final goal. We take full advantage of the unstructured data support on Snowflake by uploading the h5 file and the defect images onto the data stage. We then write a Python UDF in order to load the model and predict the defect class. This UDF can be called in a SQL Query.

### In Conclusion

And just like that, we catch lightning in a bottle!
We have brought in deep learning, unstructured data and SQL querying into this melting pot – the Snowflake data cloud. What would have been considered all but impossible only a few years ago, can now be done in a matter of minutes. In this era of data renaissance, Snowflake brings with it an avalanche of innovation making many an impossible task possible. I, for one am excited to see what the future holds. What about you?

## About the Author


### Srinivasaraghavan Sundar

Senior Data Engineer, Snowflake Center of Excellence, LTIMindtree


Srinivas usually spends his time either picking up new skills or honing and deep diving into his areas of interest in Data Engineering & Data science. He is currently a part of LTIMindtree's Snowflake COE in the capacity of a Senior Data Engineer with a real penchant for Data Science. When he isn't working, he can be found reading books, learning new languages and practicing the keyboard.

Srinivas has a fiery passion for theatre; aside from presiding over his college's theatre club, he was part of Crea-Shakthi's Malgudi Days troupe which toured South India in 2018.Today, he channels his creativity into finding unique solutions for technical problems, and, well, writing blogs like the one above.


**LTIMindtree** is a global technology consulting and digital solutions company that enables enterprises across industries to reimagine business models, accelerate innovation, and maximize growth by harnessing digital technologies. As a digital transformation partner to more than 700+ clients, LTIMindtree brings extensive domain and technology expertise to help drive superior competitive differentiation, customer experiences, and business outcomes in a converging world. Powered by nearly 90,000 talented and entrepreneurial professionals across more than 30 countries, LTIMindtree — a Larsen & Toubro Group company — combines the industry-acclaimed strengths of erstwhile Larsen and Toubro Infotech and Mindtree in solving the most complex business challenges and delivering transformation at scale. For more information, please visit **www.ltimindtree.com.**

LTIMindtree Limited is a subsidiary of Larsen & Toubro Limited