Table Structure Detection and Data Extraction

1. Business Problem

↳ *6 cells hidden*

2. Data Information :

- We will be using the **MARMOT Dataset** for training and evaluating our models.
- It consists of about **500 images** and their corresponding table & column annotations. Annotation files are present in xml format.
- We will use these annotations to generate the table & column segmentation masks for each image.

3. ML Problem Formulation :

Type of Machine Learning Problem :

For extracting data from table(s) present in an Image :

- The columns and tables have to be segmented from the image. Thus it is an **Image Segmentation** i.e pixel-wise classification task.
- Then we need to pass the table and column segments through an OCR(Object Character Recognition) tool in order to retrieve the text present in the table cells.

Performance Metrics :

The evaluation metric used in this classification task is F1-score :

F1 is calculated as follows:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

where:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

- *Precison is the measure of the correctly identified positive cases from all the predicted positive cases*
- *Recall is the measure of the correctly identified positive cases from all the actual positive cases*
- *F1-Score is the harmonic mean of Precision and Recall. This used as a evalutation metric since it penalizes false positives and false negatives equally. It is a **preferred metric** in cases where class imbalance exists.*

4. Exploratory Data Analysis

[ ] ↳ *22 cells hidden*

Preparing Train and Test Data

[ ] ↳ *6 cells hidden*

Model Building :

```
from datetime import datetime
from tensorflow.keras import Model, Sequential
from tensorflow.keras.layers import Activation, Dense, Dropout, Input, Embedding, Flatten, Conv2DTranspose, concatenate, UpS
```

TABLENET :

```
class tbl_decoder(tf.keras.layers.Layer):
  def __init__(self, name = "Table_mask"):
    super().__init__(name = name)
    self.conv1 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.umsample1 = UpSampling2D(size = (2,2),)
    self.umsample2 = UpSampling2D(size = (2,2),)
    self.umsample3 = UpSampling2D(size = (2,2),)
    self.umsample4 = UpSampling2D(size = (2,2),)
    self.convtranspose = Conv2DTranspose( filters=3, kernel_size=3, strides=2, padding = 'same')

  def call(self, X):

    input,pool_3,pool_4 = X[0],X[1],X[2]
    x = self.conv1(input)
    x = self.umsample1(x)
    x = concatenate([x, pool_4])
    x = self.umsample2(x)
    x = concatenate([x, pool_3])
    x = self.umsample3(x)
    x = self.umsample4(x)
    x = self.convtranspose(x)

    return x


class col_decoder(tf.keras.layers.Layer):
```

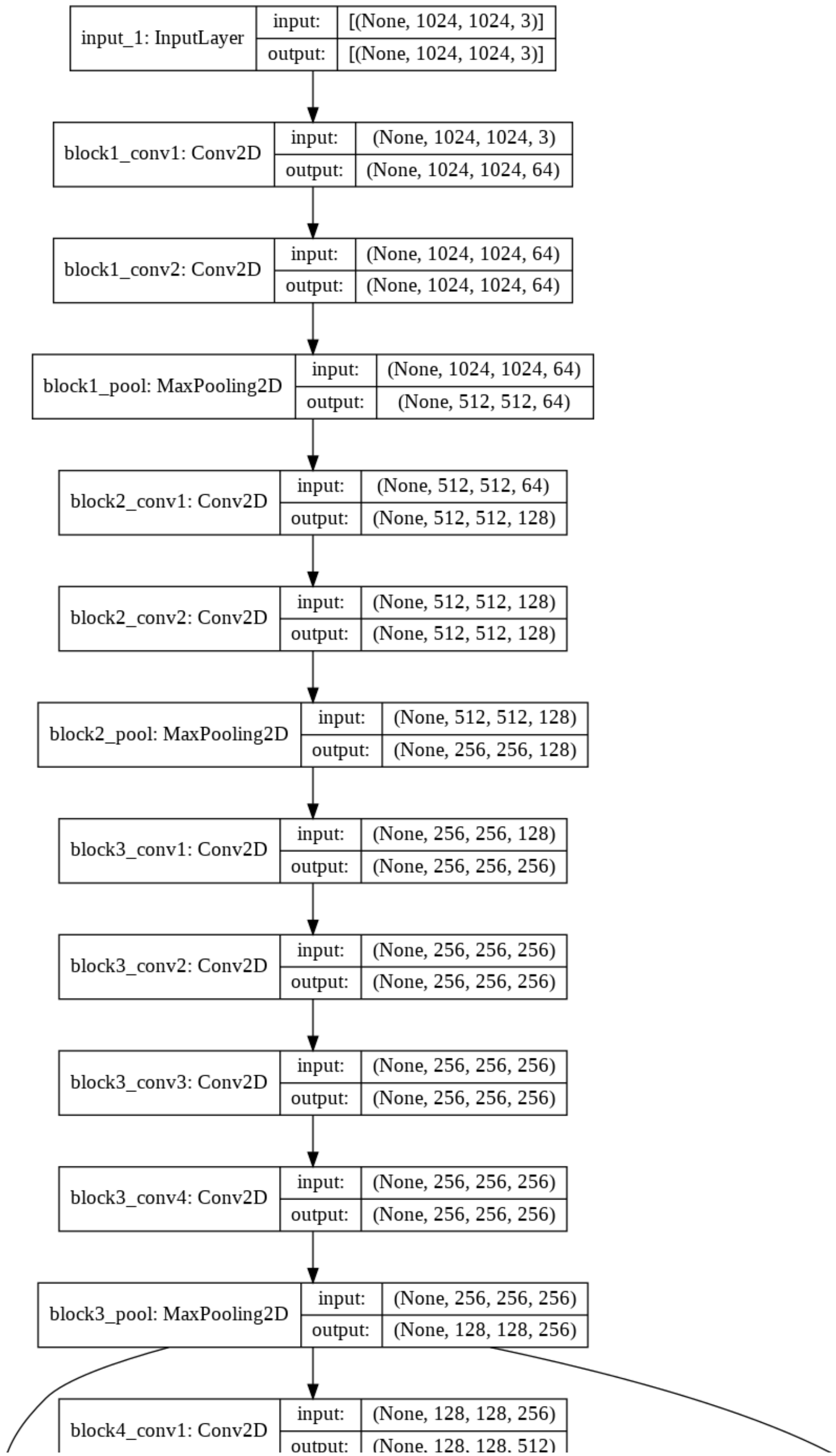
```python
  def __init__(self, name = "Column_mask"):
    super().__init__(name = name)
    self.conv1 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.drop  = Dropout(0.8)
    self.conv2 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.umsample1 = UpSampling2D(size = (2,2),)
    self.umsample2 = UpSampling2D(size = (2,2),)
    self.umsample3 = UpSampling2D(size = (2,2),)
    self.umsample4 = UpSampling2D(size = (2,2),)
    self.convtranspose = Conv2DTranspose( filters=3, kernel_size=3, strides=2, padding = 'same')

  def call(self, X):

    input,pool_3,pool_4 = X[0],X[1],X[2]
    x = self.conv1(input)
    x = self.drop(x)
    x = self.conv2(x)
    x = self.umsample1(x)
    x = concatenate([x, pool_4])
    x = self.umsample2(x)
    x = concatenate([x, pool_3])
    x = self.umsample3(x)
    x = self.umsample4(x)
    x = self.convtranspose(x)

    return x


input = Input(shape=(1024,1024,3))
vgg19 = tf.keras.applications.VGG19(include_top=False, weights = 'imagenet', input_tensor=input, classes= 1000)

x = vgg19.output
pool_3 = vgg19.get_layer('block3_pool').output
pool_4 = vgg19.get_layer('block4_pool').output

x = Conv2D(512, (1, 1), activation = 'relu', name='block6_conv1')(x)
x = Dropout(0.8, name='block6_dropout1')(x)
x = Conv2D(512, (1, 1), activation = 'relu', name='block6_conv2')(x)
x = Dropout(0.8, name = 'block6_dropout2')(x)
```

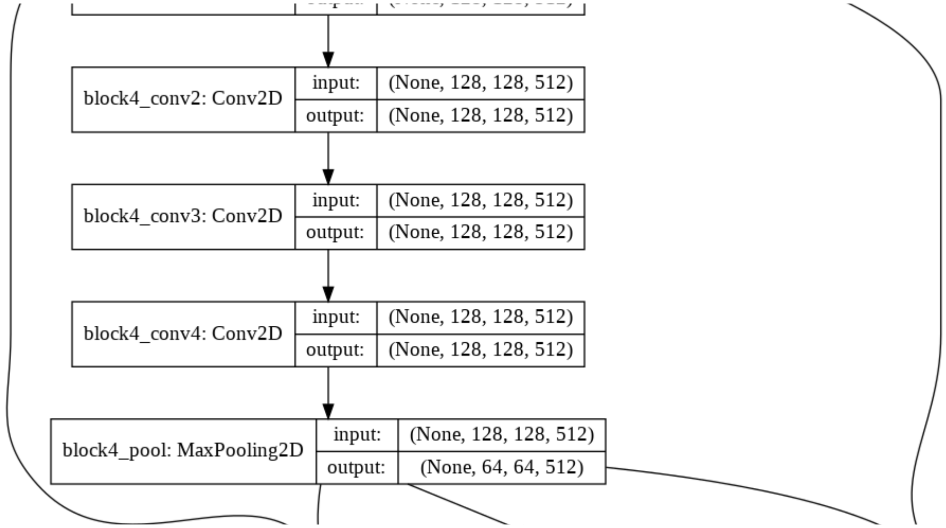
```
Table_Decoder  = tbl_decoder()
Column_Decoder = col_decoder()

output1 = Table_Decoder([x, pool_3, pool_4])
#output1 = Activation(activation='relu', name = "Table_mask")(output1)
output2 = Column_Decoder([x, pool_3, pool_4])
#output2 = Activation(activation='relu', name = "Column_mask")(output2)

model = Model(inputs = input, outputs= [output1,output2], name = "TableNet")
model.summary()
```

```
Model: "TableNet"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1024, 1024,  0
__________________________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 1024, 1024, 6 1792        input_1[0][0]
__________________________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 1024, 1024, 6 36928       block1_conv1[0][0]
__________________________________________________________________________________________________
block1_pool (MaxPooling2D)      (None, 512, 512, 64) 0           block1_conv2[0][0]
__________________________________________________________________________________________________
block2_conv1 (Conv2D)           (None, 512, 512, 128 73856       block1_pool[0][0]
__________________________________________________________________________________________________
block2_conv2 (Conv2D)           (None, 512, 512, 128 147584      block2_conv1[0][0]
__________________________________________________________________________________________________
block2_pool (MaxPooling2D)      (None, 256, 256, 128 0           block2_conv2[0][0]
__________________________________________________________________________________________________
block3_conv1 (Conv2D)           (None, 256, 256, 256 295168      block2_pool[0][0]
__________________________________________________________________________________________________
block3_conv2 (Conv2D)           (None, 256, 256, 256 590080      block3_conv1[0][0]
__________________________________________________________________________________________________
block3_conv3 (Conv2D)           (None, 256, 256, 256 590080      block3_conv2[0][0]
__________________________________________________________________________________________________
block3_conv4 (Conv2D)           (None, 256, 256, 256 590080      block3_conv3[0][0]
__________________________________________________________________________________________________
block3_pool (MaxPooling2D)      (None, 128, 128, 256 0           block3_conv4[0][0]
__________________________________________________________________________________________________
block4_conv1 (Conv2D)           (None, 128, 128, 512 1180160     block3_pool[0][0]
__________________________________________________________________________________________________
```

block4_conv2 (Conv2D)	(None, 128, 128, 512	2359808	block4_conv1[0][0]
block4_conv3 (Conv2D)	(None, 128, 128, 512	2359808	block4_conv2[0][0]
block4_conv4 (Conv2D)	(None, 128, 128, 512	2359808	block4_conv3[0][0]
block4_pool (MaxPooling2D)	(None, 64, 64, 512)	0	block4_conv4[0][0]
block5_conv1 (Conv2D)	(None, 64, 64, 512)	2359808	block4_pool[0][0]
block5_conv2 (Conv2D)	(None, 64, 64, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Conv2D)	(None, 64, 64, 512)	2359808	block5_conv2[0][0]
block5_conv4 (Conv2D)	(None, 64, 64, 512)	2359808	block5_conv3[0][0]
block5_pool (MaxPooling2D)	(None, 32, 32, 512)	0	block5_conv4[0][0]
block6_conv1 (Conv2D)	(None, 32, 32, 512)	262656	block5_pool[0][0]
block6_dropout1 (Dropout)	(None, 32, 32, 512)	0	block6_conv1[0][0]
block6_conv2 (Conv2D)	(None, 32, 32, 512)	262656	block6_dropout1[0][0]
block6_dropout2 (Dropout)	(None, 32, 32, 512)	0	block6_conv2[0][0]
Table_mask (tbl_decoder)	(None, 1024, 1024, 3	297219	block6_dropout2[0][0] block3_pool[0][0] block4_pool[0][0]

```
tf.keras.utils.plot_model(model,show_shapes=True,show_layer_names=True)
```


input_1: InputLayer
input: [(None, 1024, 1024, 3)]
output: [(None, 1024, 1024, 3)]
block1_conv1: Conv2D
input: (None, 1024, 1024, 3)
output: (None, 1024, 1024, 64)
block1_conv2: Conv2D
input: (None, 1024, 1024, 64)
output: (None, 1024, 1024, 64)
block1_pool: MaxPooling2D
input: (None, 1024, 1024, 64)
output: (None, 512, 512, 64)
block2_conv1: Conv2D
input: (None, 512, 512, 64)
output: (None, 512, 512, 128)
block2_conv2: Conv2D
input: (None, 512, 512, 128)
output: (None, 512, 512, 128)
block2_pool: MaxPooling2D
input: (None, 512, 512, 128)
output: (None, 256, 256, 128)
block3_conv1: Conv2D
input: (None, 256, 256, 128)
output: (None, 256, 256, 256)
block3_conv2: Conv2D
input: (None, 256, 256, 256)
output: (None, 256, 256, 256)
block3_conv3: Conv2D
input: (None, 256, 256, 256)
output: (None, 256, 256, 256)
block3_conv4: Conv2D
input: (None, 256, 256, 256)
output: (None, 256, 256, 256)
block3_pool: MaxPooling2D
input: (None, 256, 256, 256)
output: (None, 128, 128, 256)
block4_conv1: Conv2D
input: (None, 128, 128, 256)
output: (None, 128, 128, 512)


block4_conv2: Conv2D
input:
(None, 128, 128, 512)
output:
(None, 128, 128, 512)
block4_conv3: Conv2D
input:
(None, 128, 128, 512)
output:
(None, 128, 128, 512)
block4_conv4: Conv2D
input:
(None, 128, 128, 512)
output:
(None, 128, 128, 512)
block4_pool: MaxPooling2D
input:
(None, 128, 128, 512)
output:
(None, 64, 64, 512)

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remo
```

Model Training:

```
!rm -rf ./logs/fit/
```

```
!rm -rf ./model_save/
```

```
train_dataloader, test_dataloader, train_steps, val_steps = load_data(BATCH_SIZE = 1, BUFFER_SIZE = 10)
```

```
losses = {"Table_mask" : tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
          "Column_mask" : tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True)}

loss_weights = {"Table_mask" : 2.0,
                "Column_mask" : 1.0}

model.compile(optimizer= tf.keras.optimizers.Adam(0.0001, beta_1=0.8), loss = losses, loss_weights=loss_weights, metrics = [
```

```
filepath="model_save/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath ,  monitor='val_loss' ,save_best_only=True, mode='auto', v


hist = model.fit(train_dataloader, epochs =50, steps_per_epoch=train_steps, \
          validation_data=test_dataloader, validation_steps=val_steps, callbacks=[checkpoint])
```

```
395/395 [==============================] - 145s 367ms/step - loss: 0.1995 - Table_mask_loss: 0.0598 - Column_mask_lo

Epoch 00028: val_loss did not improve from 0.69926
Epoch 29/50
395/395 [==============================] - 145s 367ms/step - loss: 0.1928 - Table_mask_loss: 0.0571 - Column_mask_lo

Epoch 00029: val_loss did not improve from 0.69926
Epoch 30/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1930 - Table_mask_loss: 0.0569 - Column_mask_lo

Epoch 00030: val_loss did not improve from 0.69926
Epoch 31/50
395/395 [==============================] - 145s 367ms/step - loss: 0.2848 - Table_mask_loss: 0.0893 - Column_mask_lo

Epoch 00031: val_loss did not improve from 0.69926
Epoch 32/50
395/395 [==============================] - 145s 368ms/step - loss: 0.2041 - Table_mask_loss: 0.0616 - Column_mask_lo

Epoch 00032: val_loss did not improve from 0.69926
Epoch 33/50
395/395 [==============================] - 145s 367ms/step - loss: 0.1874 - Table_mask_loss: 0.0544 - Column_mask_lo

Epoch 00033: val_loss did not improve from 0.69926
Epoch 34/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1993 - Table_mask_loss: 0.0595 - Column_mask_lo

Epoch 00034: val_loss did not improve from 0.69926
Epoch 35/50
395/395 [==============================] - 145s 367ms/step - loss: 0.1981 - Table_mask_loss: 0.0576 - Column_mask_lo

Epoch 00035: val_loss did not improve from 0.69926
Epoch 36/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1861 - Table_mask_loss: 0.0534 - Column_mask_lo

Epoch 00036: val_loss improved from 0.69926 to 0.64264, saving model to model_save/weights-36-0.6426.hdf5
```

```
Epoch 37/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1664 - Table_mask_loss: 0.0464 - Column_mask_lo

Epoch 00037: val_loss improved from 0.64264 to 0.63377, saving model to model_save/weights-37-0.6338.hdf5
Epoch 38/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1547 - Table_mask_loss: 0.0411 - Column_mask_lo

Epoch 00038: val_loss did not improve from 0.63377
Epoch 39/50
395/395 [==============================] - 145s 367ms/step - loss: 0.1557 - Table_mask_loss: 0.0412 - Column_mask_lo

Epoch 00039: val_loss improved from 0.63377 to 0.60328, saving model to model_save/weights-39-0.6033.hdf5
Epoch 40/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1539 - Table_mask_loss: 0.0394 - Column_mask_lo

Epoch 00040: val_loss did not improve from 0.60328
Epoch 41/50
395/395 [==============================] - 145s 368ms/step - loss: 0.1732 - Table_mask_loss: 0.0491 - Column_mask_lo

Epoch 00041: val_loss did not improve from 0.60328

Epoch 42/50
395/395 [==============================] - 146s 368ms/step - loss: 0.1369 - Table_mask_loss: 0.0334 - Column_mask_lo
```

```
losses = {"Table_mask" : tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
          "Column_mask" : tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True)}

loss_weights = {"Table_mask" : 1.0,
                "Column_mask" : 1.0}

model.compile(optimizer= tf.keras.optimizers.Adam(0.0001, beta_1=0.8), loss = losses, loss_weights=loss_weights, metrics = [


filepath="model_save/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath ,  monitor='val_loss' ,save_best_only=True, mode='auto', v

csvlog = tf.keras.callbacks.CSVLogger("/content/results.csv")

hist = model.fit(train_dataloader, epochs =80, steps_per_epoch=train_steps, \
          validation_data=test_dataloader, validation_steps=val_steps, callbacks=[checkpoint, csvlog])
```

```
Epoch 13/80
395/395 [==============================] - 281s 712ms/step - loss: 0.1111 - Table_mask_loss: 0.0403 - Column_mask_lo

Epoch 00013: val_loss improved from 0.29042 to 0.25956, saving model to model_save/weights-13-0.2596.hdf5
Epoch 14/80
395/395 [==============================] - 281s 711ms/step - loss: 0.0850 - Table_mask_loss: 0.0270 - Column_mask_lo

Epoch 00014: val_loss improved from 0.25956 to 0.24460, saving model to model_save/weights-14-0.2446.hdf5
Epoch 15/80
395/395 [==============================] - 281s 711ms/step - loss: 0.0786 - Table_mask_loss: 0.0236 - Column_mask_lo

Epoch 00015: val_loss did not improve from 0.24460
Epoch 16/80
395/395 [==============================] - 280s 710ms/step - loss: 0.0747 - Table_mask_loss: 0.0216 - Column_mask_lo

Epoch 00016: val_loss did not improve from 0.24460
Epoch 17/80
395/395 [==============================] - 280s 708ms/step - loss: 0.0723 - Table_mask_loss: 0.0206 - Column_mask_lo

Epoch 00017: val_loss did not improve from 0.24460
Epoch 18/80
395/395 [==============================] - 280s 710ms/step - loss: 0.0722 - Table_mask_loss: 0.0203 - Column_mask_lo

Epoch 00018: val_loss did not improve from 0.24460
Epoch 19/80
395/395 [==============================] - 281s 713ms/step - loss: 0.1537 - Table_mask_loss: 0.0702 - Column_mask_lo

Epoch 00019: val_loss improved from 0.24460 to 0.18593, saving model to model_save/weights-19-0.1859.hdf5
Epoch 20/80
395/395 [==============================] - 280s 709ms/step - loss: 0.0865 - Table_mask_loss: 0.0285 - Column_mask_lo

Epoch 00020: val_loss did not improve from 0.18593
Epoch 21/80
395/395 [==============================] - 280s 710ms/step - loss: 0.0714 - Table_mask_loss: 0.0208 - Column_mask_lo

Epoch 00021: val_loss did not improve from 0.18593
Epoch 22/80
395/395 [==============================] - 281s 712ms/step - loss: 0.0673 - Table_mask_loss: 0.0190 - Column_mask_lo

Epoch 00022: val_loss did not improve from 0.18593
Epoch 23/80
395/395 [==============================] - 280s 709ms/step - loss: 0.0647 - Table_mask_loss: 0.0178 - Column_mask_lo

Epoch 00023: val_loss did not improve from 0.18593
```

```
Epoch 00023: val_loss did not improve from 0.18593
Epoch 24/80
395/395 [==============================] - 280s 708ms/step - loss: 0.0724 - Table_mask_loss: 0.0229 - Column_mask_lo

Epoch 00024: val_loss did not improve from 0.18593
Epoch 25/80
395/395 [==============================] - 280s 710ms/step - loss: 0.0879 - Table_mask_loss: 0.0328 - Column_mask_lo

Epoch 00025: val_loss did not improve from 0.18593
Epoch 26/80
395/395 [==============================] - 281s 712ms/step - loss: 0.0643 - Table_mask_loss: 0.0182 - Column_mask_lo

Epoch 00026: val_loss did not improve from 0.18593
Epoch 27/80
395/395 [==============================] - ETA: 0s - loss: 0.0610 - Table_mask_loss: 0.0167 - Column_mask_loss: 0.04
```

```
filepath="model_save/weights-{epoch:02d}-{val_loss:.4f}.hdf5"
checkpoint = tf.keras.callbacks.ModelCheckpoint(filepath=filepath ,  monitor='val_loss' ,save_best_only=True, mode='auto', v


hist = model.fit(train_dataloader, epochs =60, steps_per_epoch=train_steps, \
          validation_data=test_dataloader, validation_steps=val_steps, callbacks=[checkpoint])
```

```
Epoch 00002: val_loss improved from 0.14850 to 0.14367, saving model to model_save/weights-02-0.1437.hdf5
Epoch 3/60
395/395 [==============================] - 280s 709ms/step - loss: 0.1029 - Table_mask_loss: 0.0391 - Column_mask_lo

Epoch 00003: val_loss did not improve from 0.14367
Epoch 4/60
395/395 [==============================] - 280s 709ms/step - loss: 0.0981 - Table_mask_loss: 0.0345 - Column_mask_lo

Epoch 00004: val_loss did not improve from 0.14367
Epoch 5/60

395/395 [==============================] - 280s 708ms/step - loss: 0.0762 - Table_mask_loss: 0.0222 - Column_mask_lo

Epoch 00005: val_loss improved from 0.14367 to 0.13915, saving model to model_save/weights-05-0.1391.hdf5
Epoch 6/60
395/395 [==============================] - 279s 706ms/step - loss: 0.0707 - Table_mask_loss: 0.0197 - Column_mask_lo

Epoch 00006: val_loss did not improve from 0.13915
```

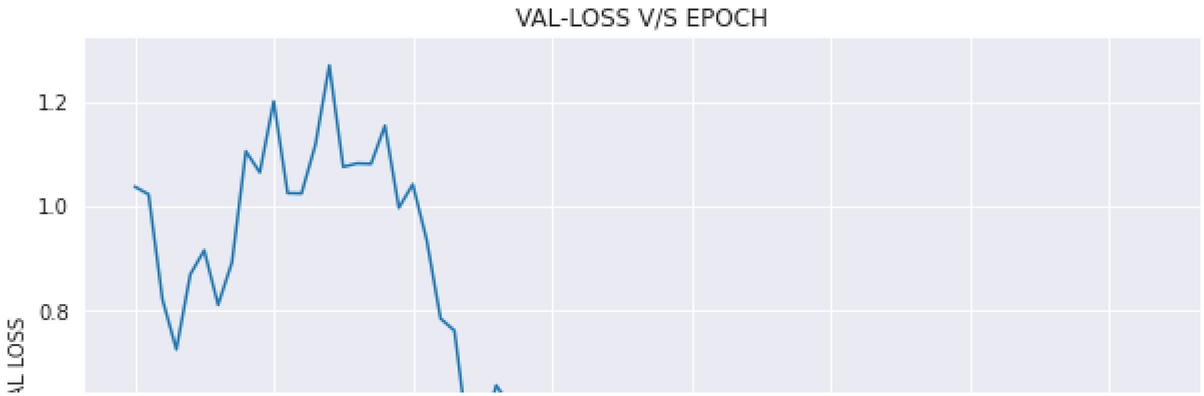
```
Epoch 7/60
395/395 [==============================] - 279s 707ms/step - loss: 0.0679 - Table_mask_loss: 0.0185 - Column_mask_lo

Epoch 00007: val_loss improved from 0.13915 to 0.13549, saving model to model_save/weights-07-0.1355.hdf5
Epoch 8/60
395/395 [==============================] - 279s 707ms/step - loss: 0.0661 - Table_mask_loss: 0.0180 - Column_mask_lo

Epoch 00008: val_loss did not improve from 0.13549
Epoch 9/60
395/395 [==============================] - 279s 708ms/step - loss: 0.0680 - Table_mask_loss: 0.0192 - Column_mask_lo

Epoch 00009: val_loss did not improve from 0.13549
Epoch 10/60
395/395 [==============================] - 279s 707ms/step - loss: 0.0932 - Table_mask_loss: 0.0337 - Column_mask_lo

Epoch 00010: val_loss did not improve from 0.13549
Epoch 11/60
395/395 [==============================] - 280s 709ms/step - loss: 0.0794 - Table_mask_loss: 0.0265 - Column_mask_lo

Epoch 00011: val_loss did not improve from 0.13549
Epoch 12/60
395/395 [==============================] - 279s 707ms/step - loss: 0.0651 - Table_mask_loss: 0.0180 - Column_mask_lo

Epoch 00012: val_loss did not improve from 0.13549
Epoch 13/60
395/395 [==============================] - 279s 707ms/step - loss: 0.0604 - Table_mask_loss: 0.0163 - Column_mask_lo

Epoch 00013: val_loss did not improve from 0.13549
Epoch 14/60
395/395 [==============================] - 279s 706ms/step - loss: 0.0585 - Table_mask_loss: 0.0154 - Column_mask_lo

Epoch 00014: val_loss did not improve from 0.13549
Epoch 15/60
395/395 [==============================] - 279s 706ms/step - loss: 0.0573 - Table_mask_loss: 0.0151 - Column_mask_lo

Epoch 00015: val_loss did not improve from 0.13549

Epoch 16/60
395/395 [==============================] - 280s 709ms/step - loss: 0.1412 - Table_mask_loss: 0.0665 - Column_mask_lo

Epoch 00016: val_loss did not improve from 0.13549
```

Plotting the metric curves :

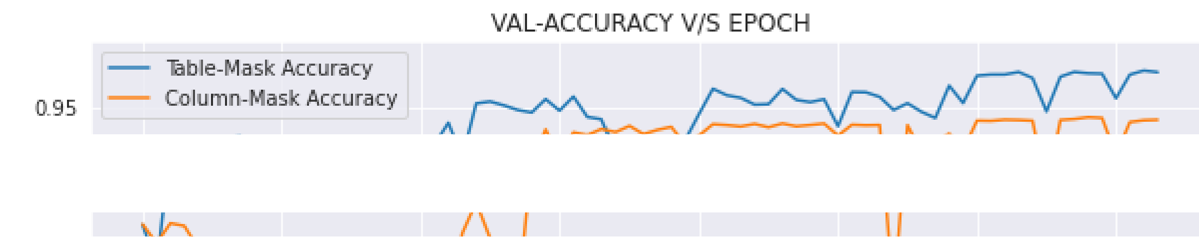
```
df = pd.read_csv("/content/TrainingLogs.txt")
df.head()
```

	epoch	Column_mask_accuracy	Column_mask_loss	Table_mask_accuracy	Table_mask_loss	loss	val_Column_mask_accu
0	0	0.865139	0.314912	0.863160	0.352070	1.019052	0.88
1	1	0.882519	0.259748	0.881721	0.296116	0.851981	0.87
2	2	0.898247	0.189354	0.911400	0.199551	0.588457	0.88
3	3	0.905912	0.170238	0.924257	0.169564	0.509367	0.88
4	4	0.909222	0.160504	0.930152	0.152773	0.466050	0.87

```
sns.set_style('darkgrid')
plt.figure(figsize=(10,6))
plt.plot("epoch", "val_loss", data = df)
plt.xlabel("EPOCHS")
plt.ylabel("VAL LOSS")
plt.title("VAL-LOSS V/S EPOCH")
plt.show()
```




```
plt.figure(figsize=(10,6))

plt.plot("epoch", "val_Table_mask_accuracy", data = df, label ="Table-Mask Accuracy")
plt.plot("epoch", "val_Column_mask_accuracy", data = df, label ="Column-Mask Accuracy")
plt.xlabel("EPOCHS")
plt.ylabel("VAL ACCURACY")
plt.title("VAL-ACCURACY V/S EPOCH")
plt.legend()
plt.show()
```



Getting Predictions :

```
def get_mask(mask):
  mask = tf.argmax(mask, axis=-1)
  mask = mask[..., tf.newaxis]
  return mask[0]
```

```
number = 10
for image, mask  in test_dataloader.take(number):
  plt.figure(figsize=(15, 15))
  mask1, mask2 = model.predict(image)
  table_mask, column_mask = get_mask(mask1),get_mask(mask2)
  image = image[0]
  lists = [image, table_mask, column_mask]
  title = ['Image', 'Table Mask', 'Column Mask']
  for i in range(len(lists)):
    plt.subplot(1, len(lists), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.preprocessing.image.array_to_img(lists[i]))
    plt.axis('off')
  plt.show()
```


Image
Table Mask
Column Mask
Image
Table Mask
Column Mask
Image
Table Mask
Column Mask

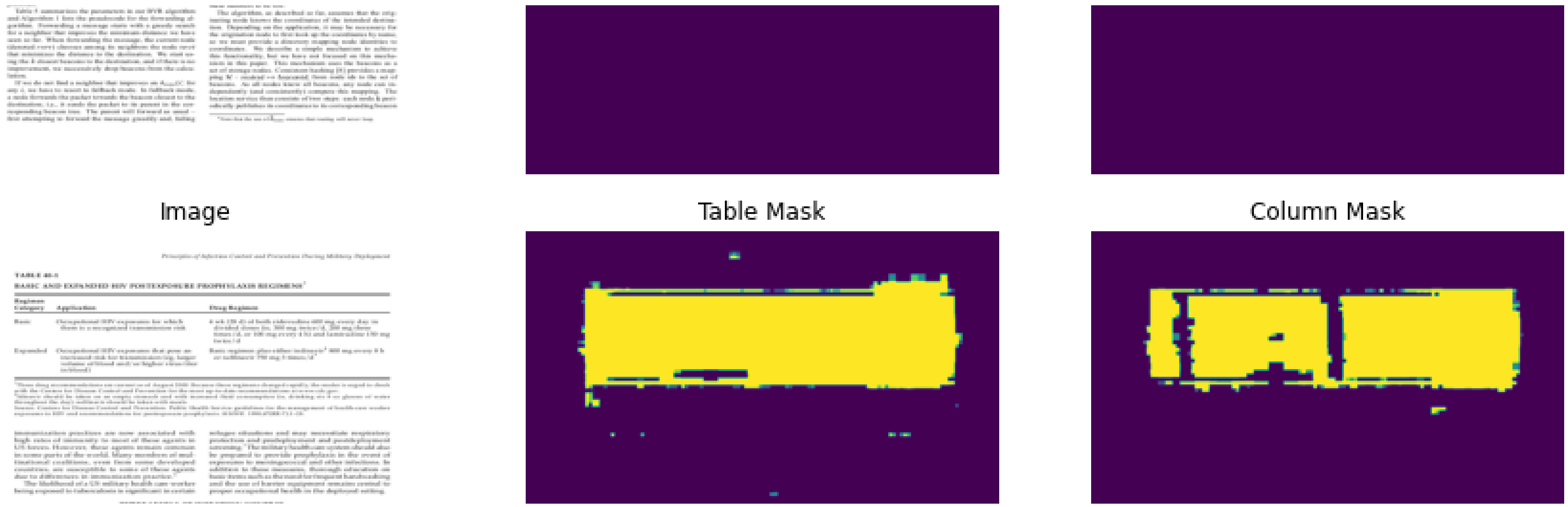


Image
Table Mask
Column Mask

Evaluating Performance :

```
model.load_weights("/content/model_save/weights-08-0.1101.hdf5")
```

```
def f1_score(true, pred):
  ''' Returns F1-Score '''
  re = tf.keras.metrics.Recall()
  re.update_state(true, pred)
  re = re.result().numpy()

  pr = tf.keras.metrics.Precision()
  pr.update_state(true, pred)
```

```
  pr = pr.result().numpy()

  f1 = 2*(re * pr)/(re + pr)
  return f1
```

```
table , column = [] , []
predicted_table_mask , predicted_column_mask = list() , list()
for image, mask  in test_dataloader.take(Test.shape[0]):
  table.append(mask['Table_mask'][0])
  column.append(mask['Column_mask'][0])

  mask1, mask2 = model.predict(image)
  table_mask, column_mask = get_mask(mask1),get_mask(mask2)

  predicted_table_mask.append(table_mask)
  predicted_column_mask.append(column_mask)


print("F1-Score for Table Masks : ",f1_score(table , predicted_table_mask))
print("-"*70)
print("F1-Score for Column Masks : ",f1_score(column , predicted_column_mask))
```

```
F1-Score for Table Masks :  0.9420742650539058
----------------------------------------------------------------------
F1-Score for Column Masks :  0.8512543315886713
```

Extracting Data from Tables :

```
def get_mask(mask):
  mask = tf.argmax(mask, axis=-1)
  mask = mask[..., tf.newaxis]
  return mask[0]
```

```
def table_detection(path) :
```

```
"""Detects and returns the table(s) in an image"""
#reading  , resizing and normalizing for image data
image = tf.io.read_file(path)
image = tf.image.decode_bmp(image, channels=3)
image = tf.image.resize(image, [1024, 1024])  #Decode a JPEG-encoded image to a uint8 tensor
image = tf.cast(image, tf.float32) / 255.0 # normalizing image

mask1, mask2 = model.predict(image[np.newaxis,:,:,:])
table_mask, column_mask = get_mask(mask1), get_mask(mask2)

im1=tf.keras.preprocessing.image.array_to_img(image)
im1.save('/content/Testing/image.png')

im2=tf.keras.preprocessing.image.array_to_img(table_mask)
im2.save('/content/Testing/table_mask.png')

im3=tf.keras.preprocessing.image.array_to_img(column_mask)
im3.save('/content/Testing/column_mask.png')

img_org = Image.open('/content/Testing/image.png')
img_org = img_org.resize((1024,1024),Image.ANTIALIAS)

print("\n")
print('\033[1m' + "INPUT IMAGE :" + '\033[0m')
print("\n")

plt.figure(figsize=(10,40))
plotting = plt.imshow(img_org,cmap='gray')
plt.show()


print("\n")
print("-"*90)
print("\n")

print("\n")
print('\033[1m' + "OUTPUT IMAGE :" + '\033[0m')
print("\n")
```

```
    table_mask = Image.open('/content/Testing/table_mask.png')
    table_mask = table_mask.resize((1024,1024),Image.ANTIALIAS)
    col_mask = Image.open('/content/Testing/column_mask.png')
    #col_mask = col_mask.resize((1024,1024),Image.ANTIALIAS)

    img_mask = table_mask.convert('L')
    # img_mask = col_mask.convert('L')

    img_org.putalpha(img_mask)

    plt.figure(figsize=(10,40))
    plotting = plt.imshow(img_org,cmap='gray')
    plt.show()


    img_org.save('/content/Testing/output.png')

    return


def get_text():

    #read your file
    file=r'/content/Testing/output.png'
    img = cv2.imread(file,0)


    #thresholding the image to a binary image
    thresh,img_bin = cv2.threshold(img,128,255,cv2.THRESH_BINARY |cv2.THRESH_OTSU)
    #inverting the image
    img_bin = 255-img_bin
    cv2.imwrite('/content/Testing/cv_inverted.png',img_bin)

    # Length(width) of kernel as 100th of total width
    kernel_len = np.array(img).shape[1]//100
    # Defining a vertical kernel to detect all vertical lines of image
    ver_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1, kernel_len))
    # Defining a horizontal kernel to detect all horizontal lines of image
    hor_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_len, 1))
```

```
hor_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_len, 1))
# A kernel of 2x2
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2, 2))

#Use vertical kernel to detect and save the vertical lines in a jpg
image_1 = cv2.erode(img_bin, ver_kernel, iterations=3)
vertical_lines = cv2.dilate(image_1, ver_kernel, iterations=3)
cv2.imwrite("/content/Testing/vertical.jpg",vertical_lines)

#Use horizontal kernel to detect and save the horizontal lines in a jpg
image_2 = cv2.erode(img_bin, hor_kernel, iterations=3)
horizontal_lines = cv2.dilate(image_2, hor_kernel, iterations=3)
cv2.imwrite("/content/Testing/horizontal.jpg",horizontal_lines)

# Combine horizontal and vertical lines in a new third image, with both having same weight.
img_vh = cv2.addWeighted(vertical_lines, 0.9, horizontal_lines, 0.1, 0.0 )
#Eroding and thesholding the image
img_vh = cv2.erode(~img_vh, kernel, iterations=2)
thresh, img_vh = cv2.threshold(img_vh,128,255, cv2.THRESH_BINARY)
cv2.imwrite("/content/Testing/img_vh.jpg", img_vh)
bitxor = cv2.bitwise_xor(img,img_vh)
bitnot = cv2.bitwise_not(bitxor)

im1=tf.keras.preprocessing.image.array_to_img(bitnot[:,:,np.newaxis])
im1.save('/content/Testing/image1.png')


img_mask = Image.open('/content/Testing/column_mask.png')
img_mask = img_mask.resize((1024,1024),Image.ANTIALIAS)

img_mask = img_mask.convert('L')
im1 = Image.open('/content/Testing/image1.png')
im1 = im1.resize((1024,1024),Image.ANTIALIAS)

im1.putalpha(img_mask)
im1.save('/content/Testing/image1.png')


print("\n")
print("-"*90)
```

```
  print("-"*90)
  print("\n")
  print('\033[1m' + "RETRIEVED TEXT :" + '\033[0m')
  print("\n")

  text_list = pytesseract.image_to_string(Image.open('/content/Testing/image1.png'), lang='eng' )
  text_list = text_list.split('\n')
  while("" in text_list)  :
    text_list.remove("")
  while(" " in text_list)  :
    text_list.remove(" ")
  while("  " in text_list) :
    text_list.remove("  ")

  for i in text_list:
    print(i)
```

```
!rm -rf ./content/Testing
```

EXAMPLE 1 :

```
table_detection('/content/Data/10.1.1.1.2139_44.bmp')

get_text()
```

Statistics	Unigrams	Bigrams	Trigrams
# n-grams seen in corpus	102,625,259	102,025,258	67,319,946
# distinct n-grams seen in corpus	126,725	4,553,400	13,847,674
# distinct non-singleton n-grams	58,616	2,254,454	4,842,677
Max abs. frequency	5,905,550	735,878	253,921
Ave abs. freq $\bar{x}$	1749.67	44.23	12.04
# distinct n-grams with freq $> \bar{x}$	3773	199,219	582,357
Lower quartile bound	3	2	2
Median frequency	8	4	3
Upper quartile bound	62	11	6
# n-grams in each quartile	~14,680	~560,000	~1,200,000

Table 3.1: Corpus Frequency Statistics

absolute corpus frequency previously recorded. General corpus frequency statistics can be found in table 3.1.

Many typical measures of frequency do not effectively express the sentence n-gram distribution. n-gram frequency arithmetic or geometric means are heavily affected by the appearance of a few commonly seen n-grams, regardless of the remainder of the distribution. As can be seen in table 3.1, only 5.7%, 8.8%, and 12% of non-singleton unigrams, bigrams, and trigrams respectively were seen more frequently than the mean count. A second possible approach of simply noting the median n-gram frequency of a sentence would not give enough detail about the sentence as a whole. Taking this into consideration, we decided to use a quartile range measure. We divided the corpus n-grams into four quartiles containing approximately an equivalent number of distinct n-grams. The number of source sentence n-grams within each frequency quartile was then counted and normalized over the number of n-grams in the sentence.

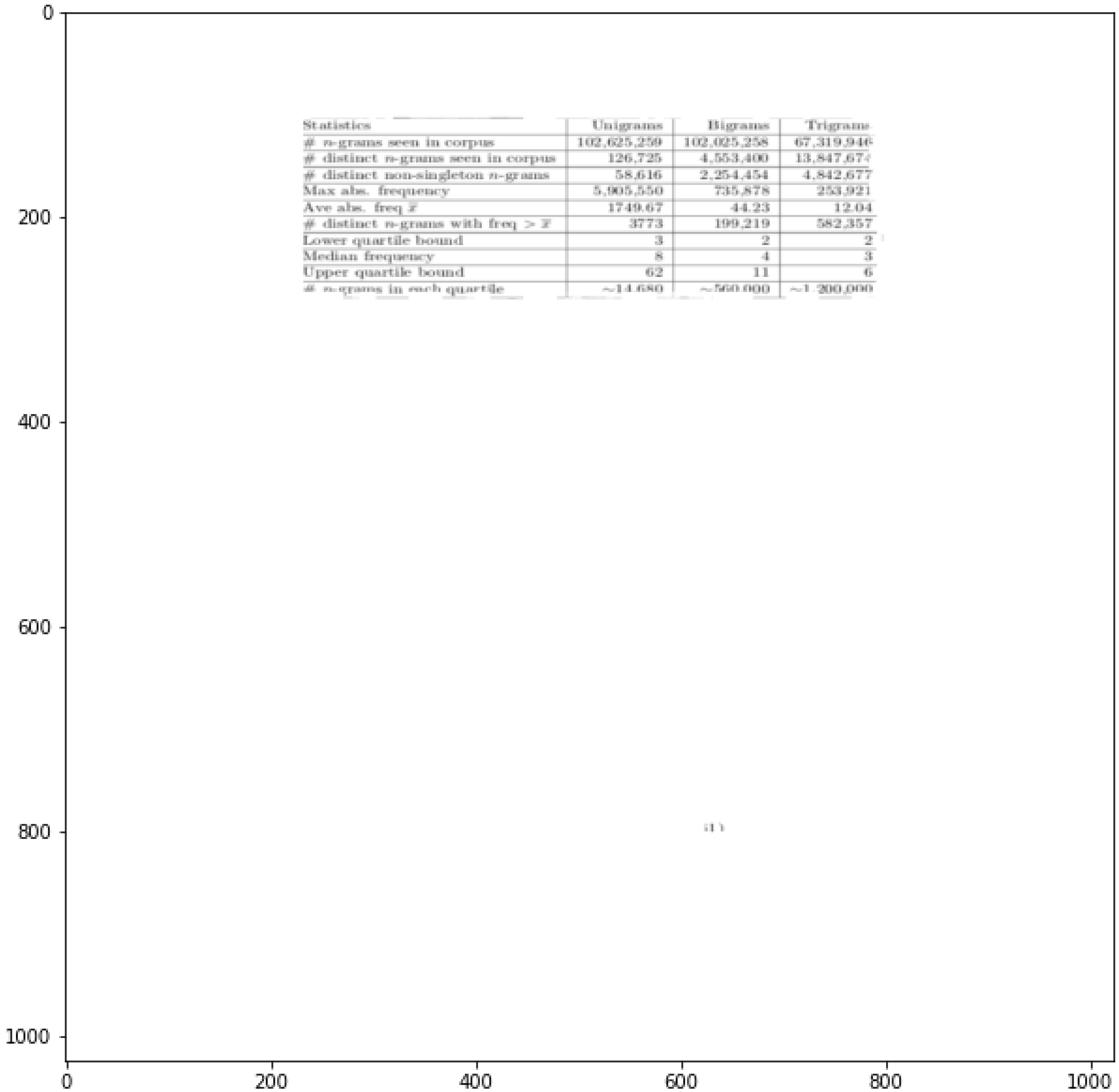
The final result of this approach was 12 parameters per source sentence showing the percentage of 1-, 2-, and 3-grams in each of the four frequency quartile ranges. (N, S, 49–60)

3.1.6 Source Language Model Features

Using the same Chinese corpus as in section 3.1.5, a trigram language model with Kneser-Ney discounting as backoff was built using the SRI Toolkit. These features were also used in our sub-sentential CE experiments. For each source sentence, three features were calculated:

- Log-probability of the source sentence (N, S, 61)

OUTPUT IMAGE :



Statistics	Unigrams	Bigrams	Trigrams
# n-grams seen in corpus	102,625,259	102,025,258	67,319,946
# distinct n-grams seen in corpus	126,725	4,553,400	13,847,67[illegible]
# distinct non-singleton n-grams	58,616	2,254,454	4,842,677
Max abs. frequency	5,905,550	735,878	253,921
Ave abs. freq $\bar{x}$	1749.67	44.23	12.04
# distinct n-grams with freq $> \bar{x}$	3773	199,219	582,357
Lower quartile bound	3	2	2
Median frequency	8	4	3
Upper quartile bound	62	11	6
# n-grams in each quartile	$\sim$14,680	$\sim$560,000	$\sim$1,200,000

```
--------------------------------------------------------------------------------------------
```

```
RETRIEVED TEXT :
```

EXAMPLE 2:

```
# distinct [illegible] grams seen in corpa [illegible]
# distinct non-singleton n-grams SSC18   0.254451 4,842,677
```

```
table_detection('/content/Data/10.1.1.1.2076_85.bmp')

get_text()
```

INPUT IMAGE :

	Maximum	Minimum	Mean	Std. dev.	C.V.
Versus/IR	29,341	3,093	15,531	6,993	45.0%
Versus/RF	11,395	6,347	8,730	1,095	12.5%
Campus/syslog	563,940	83,344	299,650	81,130	27.1%
Campus/SNMP	365,290	48,896	314,320	51,541	16.4%

Table 6.2: Statistics of daily location-update traffic. C.V. is the standard deviation divided by the mean value (coefficient of variation).

concentrator to answer queries such as "Where is badge A?". *In summary, data pre-processing plays a vital role in localization systems.*

6.3.2 Data volume

We compare the location-update traffic generated by the four systems in Table 6.2.

We see a relatively small variation in the daily location updates for Versus/RF and Campus/SNMP. Considering that the update rate was more-or-less fixed for each (pushed by badge every 2 minutes in Versus/RF and pulled by the SNMP poller every 5 minutes in Campus/SNMP), this indicates that the number of badges and cards present daily had small variation too. On a typical day, Campus/SNMP produced more than 3.6 messages per second, given a total 6006 cards and 543 APs seen during the whole trace.

Campus/syslog produced a smaller amount of update traffic than Campus/SNMP, given that the two traces were collected for same period. It is not surprising, since syslog only produced updates when the card associated and disassociated with an access point, and such visits often lasted longer than the 5-minute polling interval.

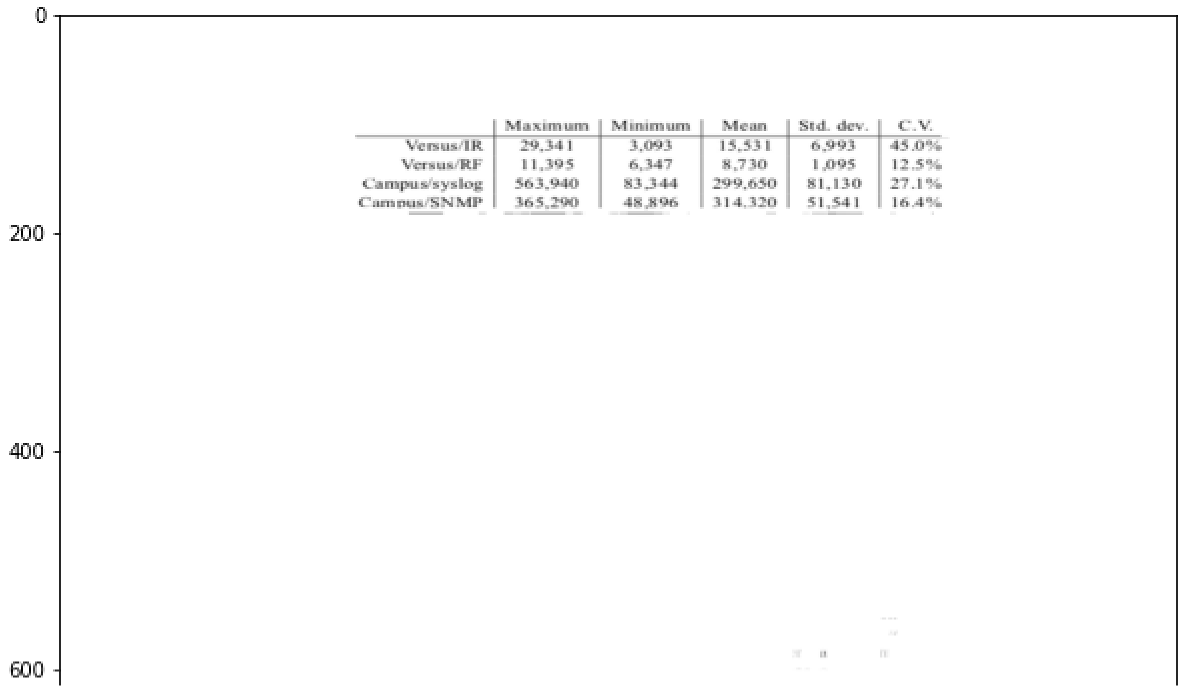
Versus/IR had a relatively large variance on daily updates, due to the significant difference in update rate when the badge is in motion (every 3.5 seconds) or stationary (every 4 minutes). In theory, N badges can produce $(N/3.5)$ updates per second if there are no missed pings caused by interference and line-of-sight problems. That is 10 updates per second with 350 active badges. Clearly, tracking all the assets and people in a multi-site large organization will be a challenge for a location service.

Location-aware applications resident on a mobile device will not have the resources to handle this amount of traffic. *Instead, a software infrastructure is necessary to collect, process, and disseminate the location updates to applications.* This infrastructure needs to keep up with the update arrival rate and control the amount of information delivered to applications. It can shield the data pre-processing from applications, and share the results with multiple applications. Several major research efforts specifically target this direction [119, 50, 68]. In the case where the data rate still may outrun the capability of the infrastructure, approximation techniques have to be applied to reduce the data stream (Chapter 3).

6.3.3 Load disparity

Many location-aware applications may only care about location updates from a particular zone or from a particular user, to answer queries such as "Who is in zone A?" or "Where is user X?". We decompose the trace to show the load distribution of such queries. We compute the fraction of

OUTPUT IMAGE :



	Maximum	Minimum	Mean	Std. dev.	C.V.
Versus/IR	29,341	3,093	15,531	6,993	45.0%
Versus/RF	11,395	6,347	8,730	1,095	12.5%
Campus/syslog	563,940	83,344	299,650	81,130	27.1%
Campus/SNMP	365,290	48,896	314,320	51,541	16.4%

EXAMPLE 3:

```
table_detection('/content/Data/10.1.1.1.2084_18.bmp')

get_text()
```

Table 3—Fiscal condition and investment in 1999 (billion yuan)

	Province	County total	Rural county total
Total revenue	68.0	61.6	28.6
Total expenditure	52.9	39.2	20.8
Capital construction	3.8	1.9	0.3
Farming and rural water conservation	3.5	1.5	0.9
Science, education, culture, and health care	15.4	8.3	6.0
Other	30.1	27.5	13.5
Total investment in fixed assets	274.3	-	-
Capital construction	74.9	-	-
By state-owned units	64.3	-	-
Real estate development	33.1	-	-
By state-owned units	17.2	-	-
Rural collective	48.2	-	-
By state-owned units	0.0	-	-
Provincial GDP	769.8	-	-

Source: Jiangsu Statistical Yearbook.
Note: State-owned units encompass the government sector.

To begin with, it is assumed that local cadres, as representatives of local people, allocate the formal budget expenditure (E) to maximize their utility:

$$\begin{aligned} \max_{I_1, I_2, I_3, S} \quad & U = [I_1 - \bar{I}_1]^{\alpha_1} [I_2 - \bar{I}_2]^{\alpha_2} [I_3 - \bar{I}_3]^{\alpha_3} S^{1-\alpha_1-\alpha_2-\alpha_3} \\ \text{s.t.} \quad & I_1 + I_2 + I_3 + S = E \end{aligned} , \qquad (2)$$

where I_j, $\bar{I}_j$ ($j = 1, 2, 3$), denote investments in agriculture, capital construction, and social welfare (such as science, education, culture, and health care), respectively, beyond required minimum investments $\bar{I}_j$, (or RMI), in each of these items, and S denotes non-investment expenditures, such as subsidy grants, officers' salaries, and the like. By solving equation (2), we have

$$\frac{I_i}{E} = \alpha_i + \frac{(1-\alpha_i)\bar{I}_i - \alpha_i \bar{I}_j - \alpha_i \bar{I}_k}{E} \qquad (i, j, k = 1, 2, 3; i \neq j, j \neq k, k \neq i), \qquad (3)$$

OUTPUT IMAGE :

[illegible]

	Province	County total	Rural county total
Total revenue	68.0	61.6	28.6
Total expenditure	52.9	39.2	20.8
Capital construction	3.8	1.9	0.3
Farming and rural water conservation	3.5	1.5	0.9
Science, education, culture, and health care	15.4	8.3	6.0
Other	30.1	27.5	13.5
Total investment in fixed assets	274.3	-	-
Capital construction	74.9	-	-
By state-owned units	64.3	-	-
Real estate development	33.1	-	-
By state-owned units	17.2	-	-
Rural collective	48.2	-	-
By state-owned units	0.0	-	-
Provincial GDP	769.8	-	-

Source: Jia[illegible]cal Yearbook.

[illegible]

```
------------------------------------------------------------------------------------------

RETRIEVED TEXT :

unity > ural county
total total
Total revenue 68.0 61.6 28.6
Total expenditure 52.9 39.2 20.8
Capital construction 3.8 1.9 0.3
arming and rural water conservation 3.5 1s 0.9
Science, education, culture, and health care 15.4 8.3 6.0
Other 30.1 27.5 13.5
Total investment in fixed assets 274.3 - -
Capital construction 74.9 - -
```

EXAMPLE 4:

```
Rural collective 48.2 - -
```

```
table_detection('/content/Data/10.1.1.6.2366_6.bmp')

get_text()
```

INPUT IMAGE :

with 50kbps. Note that all the above happen if the mobile device requesting the photo is located at the inner circle.

But the mobile client is also moving within the coverage area. It is logical to assume that when a mobile client is located at the outer circle, it is either leaving or entering the coverage area. In the former case, we assume that the transfer is completed. But in the latter case, the mobile client must receive the photo-series (or the video) as quickly as possible in order to decide what to do. So the mobile server must adjust its transmitting rate accordingly. Below we provide a table of (example) actions taken by the network operator with respect to server's transmitting rate in order for the mobile server to serve the incoming requests.

Mobile Server → Mobile Client ↓	Inner Circle	Middle Circle	Outer Circle
Inner Circle	10kbps	10kbps	20kbps
Middle Circle	10kbps	20kbps	50kbps
Outer Circle	20kbps	20kbps	50kbps

When the mobile client is at the outer circle, it is important that he receives the photo-series quickly. When the mobile server is also at the outer circle then the highest possible rate must be assigned to the particular transmission, since the mobile server is leaving the coverage area. As the client moves toward the inner circle, the transmission rate drops since the information provided by the photos will be less useful. Furthermore, as the mobile server moves outwards, the transmission rates increase since the photo must be given a chance to remain within the coverage area.

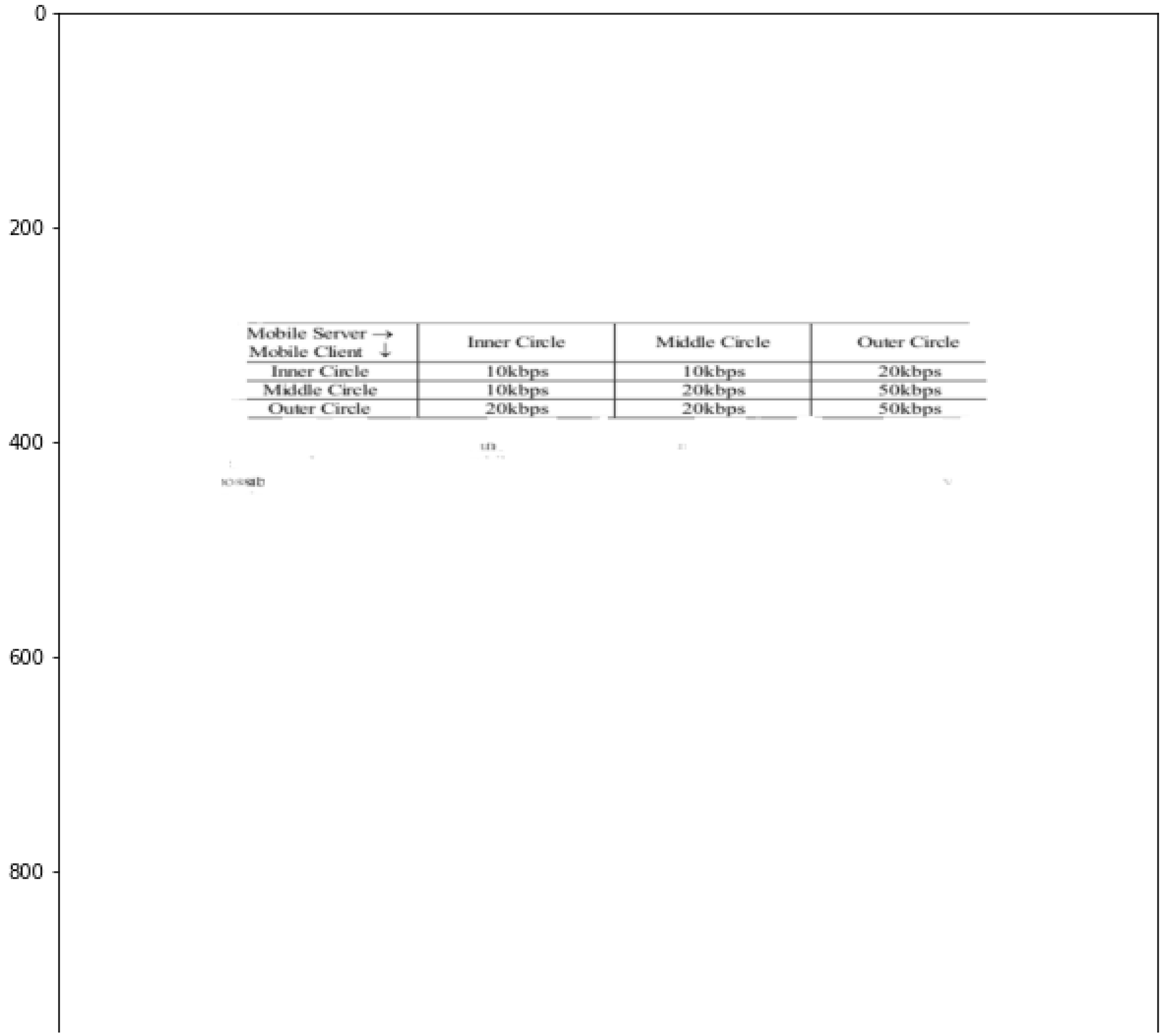
Another important issue for the network operator is how to provide a mechanism so that the content is kept within the coverage area for as long as possible, so that there is some content to be exchanged between the mobile subscribers and not wait for a new server-enabled mobile subscriber to enter the coverage area.

To achieve this, the file-transfer sessions with clients that are themselves registered as mobile servers must be prioritised, especially if the transmitting server is leaving the coverage area. This would generally assist in keeping the content inside the area of interest. If a mobile server requests a photo from another mobile server which is leaving the area, then this exchange must receive enough transmitting rate so that, at the same time, the requesting mobile server can also serve clients that request the specific photograph, even if the mobile server that initially took the photograph is out of reach.

5. Implementation

It is clear that the actions made by the network operator in order to support the architecture mentioned in the previous section assume the existence of a context-aware QoS scheme taken into consideration by the network management module. Unlike existing QoS schemes for wired and wireless networks, the QoS scheme presented here does not apply to all the outgoing traffic of a mobile server but only affects the transfer of a specific file between two mobile subscribers, in the context of an LBS session. Thus, the outgoing flows of a mobile server may receive different treatment by the network management module, relative to the position of the mobile server, the position

OUTPUT IMAGE :



Mobile Server → Mobile Client ↓	Inner Circle	Middle Circle	Outer Circle
Inner Circle	10kbps	10kbps	20kbps
Middle Circle	10kbps	20kbps	50kbps
Outer Circle	20kbps	20kbps	50kbps

EXAMPLE 5:

0 200 400 600 800 1000

```
table_detection('/content/Data/10.1.1.8.2156_5.bmp')
```

```
get_text()
```

INPUT IMAGE :

Table 1: Model checking results on the 1-bit fabric model using FormalCheck

Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.05e+06	63	100.00%	11	6.34
Property 2	4.20e+06	55	99.09%	10	6.34
Property 3	6.78e+07	84	98.81%	12	6.34
Property 4	7.23e+07	76	98.68%	14	6.34
Property 5	1.05e+06	64	98.44%	11	6.34
Property 6	7.29e+07	76	98.68%	14	6.34
Property 7	1.05e+06	63	100.00%	11	6.34

Table 2: Model checking results on the 4-bit fabric model using FormalCheck

Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.94e+06	102	99.02%	20	6.77
Property 2	4.20e+06	56	99.11%	12	6.77
Property 3	1.08e+08	120	97.50%	17	6.78
Property 4	7.23e+07	76	98.68%	16	6.77
Property 5	1.94e+06	103	98.06%	14	6.78
Property 6	7.29e+07	76	98.68%	13	6.78
Property 7	1.94e+06	102	99.02%	14	6.77

Table 3: Model checking results on the 8-bit fabric model using FormalCheck

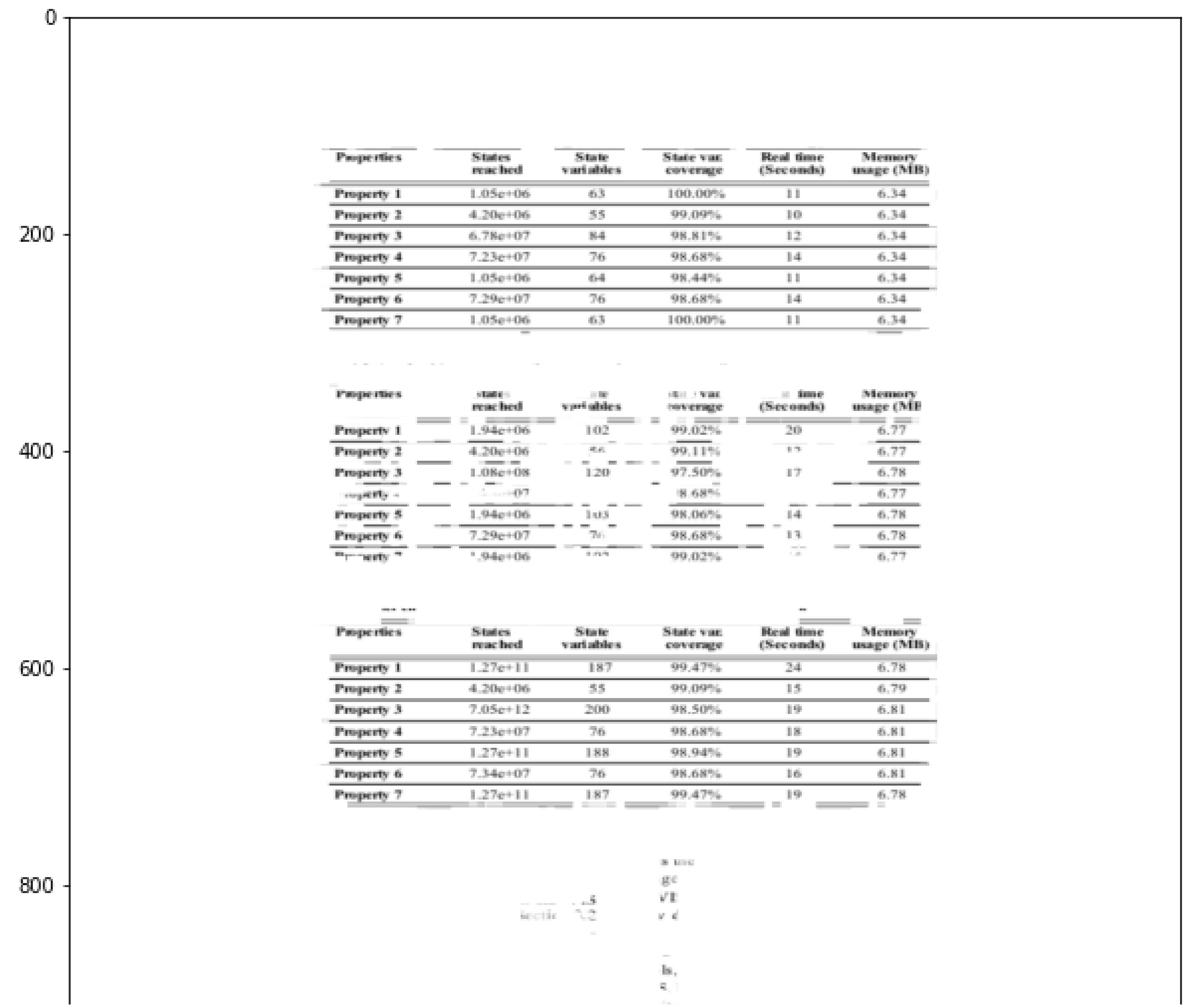
Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.27e+11	187	99.47%	24	6.78
Property 2	4.20e+06	55	99.09%	15	6.79
Property 3	7.05e+12	200	98.50%	19	6.81
Property 4	7.23e+07	76	98.68%	18	6.81
Property 5	1.27e+11	188	98.94%	19	6.81
Property 6	7.34e+07	76	98.68%	16	6.81
Property 7	1.27e+11	187	99.47%	19	6.78

4. COMPARISON WITH THE VERIFICATION IN VIS

To have a fair comparison between FormalCheck and VIS tools, we tried to verify the properties defined in Section 3.2 for the 1-bit fabric model using both tools VIS and FormalCheck. The experimental results of this comparison is shown in Table 4.

FormalCheck, by default, uses 1-step reduction algorithm, which first performs a single reduction, and then verifies the property [3]. As we can see in Table 4, the memory usage of the properties in FormalCheck has been less than in VIS for all of the properties checked. The CPU time usually depends on the amount of work being loaded on the CPU at the time of the verification. So by considering the CPU time we cannot have a fair comparison between the tools, but we can say in general, FormalCheck is faster than VIS. Note also that VIS was unable to check the 4-bit and 8-bit models of the Fairisle switch fabric [9].

OUTPUT IMAGE :



Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.05e+06	63	100.00%	11	6.34
Property 2	4.20e+06	55	99.09%	10	6.34
Property 3	6.78e+07	84	98.81%	12	6.34
Property 4	7.23e+07	76	98.68%	14	6.34
Property 5	1.05e+06	64	98.44%	11	6.34
Property 6	7.29e+07	76	98.68%	14	6.34
Property 7	1.05e+06	63	100.00%	11	6.34

Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.94e+06	102	99.02%	20	6.77
Property 2	4.20e+06	[illegible]	99.11%	[illegible]	6.77
Property 3	1.08e+08	120	97.50%	17	6.78
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	6.77
Property 5	1.94e+06	103	98.06%	14	6.78
Property 6	7.29e+07	[illegible]	98.68%	13	6.78
[illegible]	[illegible]	[illegible]	99.02%	[illegible]	6.77

Properties	States reached	State variables	State var. coverage	Real time (Seconds)	Memory usage (MB)
Property 1	1.27e+11	187	99.47%	24	6.78
Property 2	4.20e+06	55	99.09%	15	6.79
Property 3	7.05e+12	200	98.50%	19	6.81
Property 4	7.23e+07	76	98.68%	18	6.81
Property 5	1.27e+11	188	98.94%	19	6.81
Property 6	7.34e+07	76	98.68%	16	6.81
Property 7	1.27e+11	187	99.47%	19	6.78

EXAMPLE 6:

```
table_detection('/content/Data/10.1.1.12.797_2.bmp')

get_text()
```

INPUT IMAGE :

b. With probability P_{mut} Then Select one parent page P from Pop and let O ← page indicated by a link in P (local exploration with a mutation operator),

4. Evaluate f(O)

5. Insert O in Pop if (|Pop| < Pop_{Max}) or if f(O) is greater than the fitness of the worst page in Pop which is deleted,

6. Go to step 3 or Stop (Pop is the output given to the user).

Table 1: the GA used in GeniMiner

The GA that we propose for Web mining is represented in table 1. An individual in the population is a Web page which can be numerically evaluated with the fitness function f. This function represents the user query and requires to download the page and analyze its content. It can take into account for instance keywords, regular expressions, links, size of document, referenced files (images, mp3, etc).

The creation operator queries standard search engines (Altavista, Google, Lycos, Voila, Yahoo) with the user's keywords. The mutation of a parent page P consists in 1) selecting P in the population (choosing the best between two randomly chosen individuals), 2) choosing one link l going out of P, and 3) proposing the so-pointed page as an offspring. We have sorted P's links according to their distance to the keywords in the text so that the most promising links are explored first.

3. RESULTS AND PERSPECTIVES

Num	Keywords (K_1, K_2, ...)	Should (S_1, S_2, ...)	Should not (Sn_1, Sn_2, ...)
1	telecom crisis analysis	mobile future	France
2	fiber optic technology information	network	
3	text poet flower wind	Baudelaire	Rimbaud
4	wine excellent price good	Bourgueil	Bordeaux
5	buy cd music michael jackson	download mp3	
6	mouse disney movie animation	DVD	
7	artificial ant algorithm		
8	genetic algorithm artificial ant	experimental comparison	
9	javascript window opener	tutorial free	
10	dll export class template	code example	

Table 2: the queries used in our tests

We have compared GeniMiner with $P_{mut} = 0.5$ (GA dominates the search) to GeniMiner with $P_{mut} = 0$ (a meta-search where pages are generated with standard search engines only). We have tested also two evaluation functions for a page P: f_1 is closer to the evaluation functions used in standard search engines (presence of all keywords (K_1, K_2, ...) + maximization of the number of keywords). f_2 is more complex and takes into account the keywords but also the number of links going out of P (with a more important weight when they are close to keywords), additional words (S_1, S_2, ...) and (Sn_1, Sn_2, ...) which should (resp. should not) be present in P, the fact that the proportions of keywords are uniform. Both algorithms are given 1000 pages evaluations for each tested query (see table 2) and

OUTPUT IMAGE :

TableNet(Using ResNet50 Encoder) :

```
class tbl_decoder(tf.keras.layers.Layer):
  def __init__(self, name = "Table_mask"):
    super().__init__(name = name)
    self.conv1 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.umsample1 = UpSampling2D(size = (2,2),)
    self.umsample2 = UpSampling2D(size = (2,2),)
    self.umsample3 = UpSampling2D(size = (2,2),)
    self.umsample4 = UpSampling2D(size = (2,2),)
    self.convtranspose = Conv2DTranspose( filters=3, kernel_size=3, strides=2, padding = 'same')

  def call(self, X):
```

```python
    input,pool_3,pool_4 = X[0],X[1],X[2]
    x = self.conv1(input)
    x = self.umsample1(x)
    x = concatenate([x, pool_4])
    x = self.umsample2(x)
    x = concatenate([x, pool_3])
    x = self.umsample3(x)
    x = self.umsample4(x)
    x = self.convtranspose(x)

    return x

class col_decoder(tf.keras.layers.Layer):
  def __init__(self, name = "Column_mask"):
    super().__init__(name = name)
    self.conv1 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.drop  = Dropout(0.8)
    self.conv2 = Conv2D(filters=512, kernel_size=(1,1), activation='relu')
    self.umsample1 = UpSampling2D(size = (2,2),)
    self.umsample2 = UpSampling2D(size = (2,2),)
    self.umsample3 = UpSampling2D(size = (2,2),)
    self.umsample4 = UpSampling2D(size = (2,2),)
    self.convtranspose = Conv2DTranspose( filters=3, kernel_size=3, strides=2, padding = 'same')

  def call(self, X):

    input,pool_3,pool_4 = X[0],X[1],X[2]
    x = self.conv1(input)
    x = self.drop(x)
    x = self.conv2(x)
    x = self.umsample1(x)
    x = concatenate([x, pool_4])
    x = self.umsample2(x)
    x = concatenate([x, pool_3])
    x = self.umsample3(x)
    x = self.umsample4(x)
    x = self.convtranspose(x)

    return x
```

```
input = Input(shape=(1024,1024,3))

resnet50 = tf.keras.applications.ResNet50(include_top=False, weights='imagenet', input_tensor=input, classes=1000)

x = resnet50.output
pool_3 = resnet50.get_layer('conv3_block4_out').output # (128,128)
pool_4 = resnet50.get_layer('conv4_block6_out').output # (64,64)

x = Conv2D(512, (1, 1), activation = 'relu', name='block6_conv1')(x)
x = Dropout(0.8, name='block6_dropout1')(x)
x = Conv2D(512, (1, 1), activation = 'relu', name='block6_conv2')(x)
x = Dropout(0.8, name = 'block6_dropout2')(x)


Table_Decoder  = tbl_decoder()
Column_Decoder = col_decoder()

output1 = Table_Decoder([x, pool_3, pool_4])
output2 = Column_Decoder([x, pool_3, pool_4])

model = Model(inputs = input, outputs= [output1,output2], name = "TableNet")
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_or
94773248/94765736 [==============================] - 1s 0us/step
Model: "TableNet"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 1024, 1024,  0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 1030, 1030, 3 0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 512, 512, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 512, 512, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 512, 512, 64) 0           conv1_bn[0][0]
```