Convolutional Neural Networks (CNN)

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
# import warnings
import warnings
# filter warnings
warnings.filterwarnings('ignore')
import os
print(os.listdir("../input"))
```

```
['sample_submission.csv', 'train.csv', 'test.csv']
```

Loading the Data Set

- In this part we load and visualize the data.

```
# read train
train = pd.read_csv("../input/train.csv")
print(train.shape)
train.head()
```

```
(42000, 785)
```

```
   label  pixel0  pixel1    ...     pixel781  pixel782  pixel783
0      1       0       0    ...            0         0         0
1      0       0       0    ...            0         0         0
2      1       0       0    ...            0         0         0
3      4       0       0    ...            0         0         0
4      0       0       0    ...            0         0         0

[5 rows x 785 columns]
```

```
# read test
test= pd.read_csv("../input/test.csv")
print(test.shape)
test.head()
```

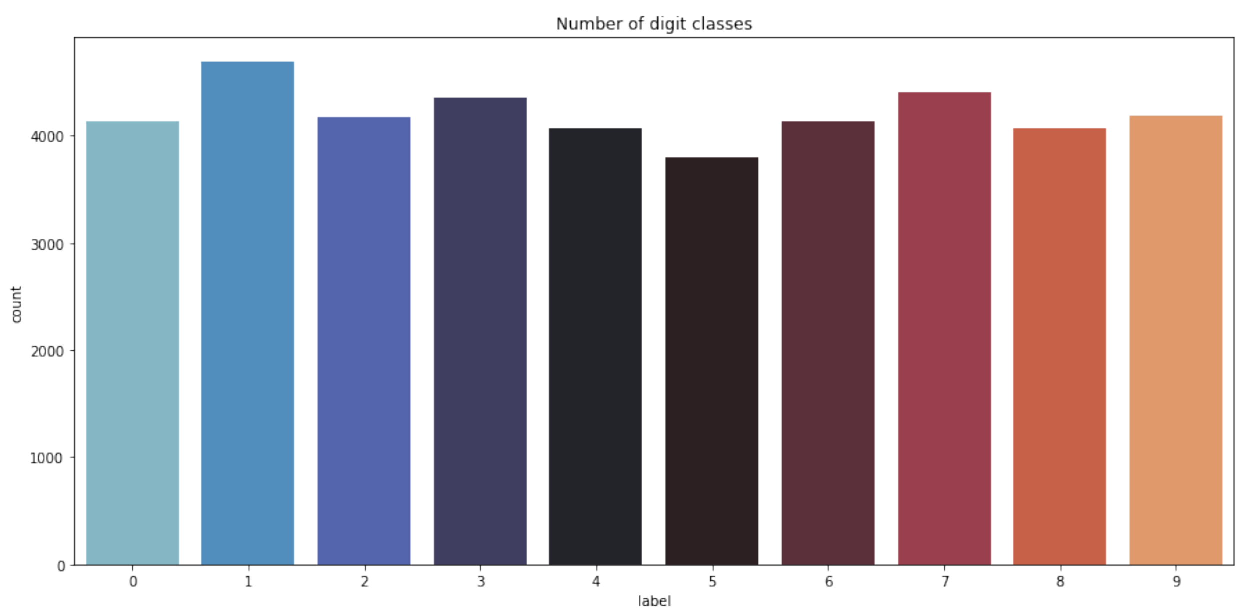
```
(28000, 784)
```

```
   pixel0  pixel1  pixel2    ...     pixel781  pixel782  pixel783
0       0       0       0    ...            0         0         0
1       0       0       0    ...            0         0         0
2       0       0       0    ...            0         0         0
```

```
3        0        0        0    ...              0          0          0
4        0        0        0    ...              0          0          0

[5 rows x 784 columns]
```

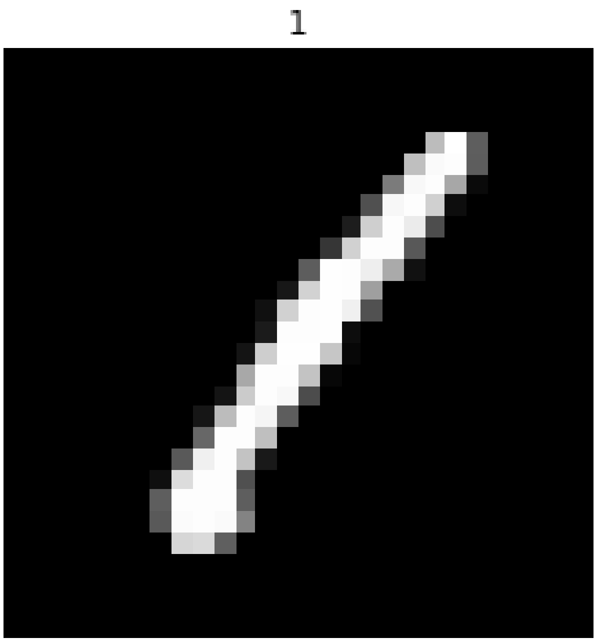
```
# put labels into y_train variable
Y_train = train["label"]
# Drop 'label' column
X_train = train.drop(labels = ["label"],axis = 1)

# visualize number of digits classes
plt.figure(figsize=(15,7))
g = sns.countplot(Y_train, palette="icefire")
plt.title("Number of digit classes")
Y_train.value_counts()
```

```
1    4684
7    4401
3    4351
9    4188
2    4177
6    4137
0    4132
4    4072
8    4063
5    3795
Name: label, dtype: int64
```



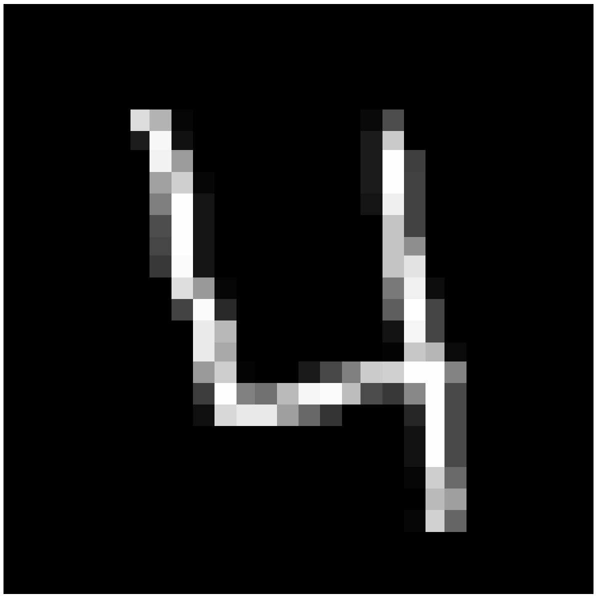

```
# plot some samples
img = X_train.iloc[0].as_matrix()
img = img.reshape((28,28))
```

```
plt.imshow(img,cmap='gray')
plt.title(train.iloc[0,0])
plt.axis("off")
plt.show()
```




```
# plot some samples
img = X_train.iloc[3].as_matrix()
img = img.reshape((28,28))
plt.imshow(img,cmap='gray')
plt.title(train.iloc[3,0])
plt.axis("off")
plt.show()
```

4



Normalization, Reshape and Label Encoding

```
# Normalize the data
X_train = X_train / 255.0
test = test / 255.0
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 784)
test shape:  (28000, 784)
```

```
# Reshape
X_train = X_train.values.reshape(-1,28,28,1)
test = test.values.reshape(-1,28,28,1)
print("x_train shape: ",X_train.shape)
print("test shape: ",test.shape)
```

```
x_train shape:  (42000, 28, 28, 1)
test shape:  (28000, 28, 28, 1)
```

```
# Label Encoding
from keras.utils.np_utils import to_categorical # convert to one-hot-encoding
Y_train = to_categorical(Y_train, num_classes = 10)
```

```
Using TensorFlow backend.
```

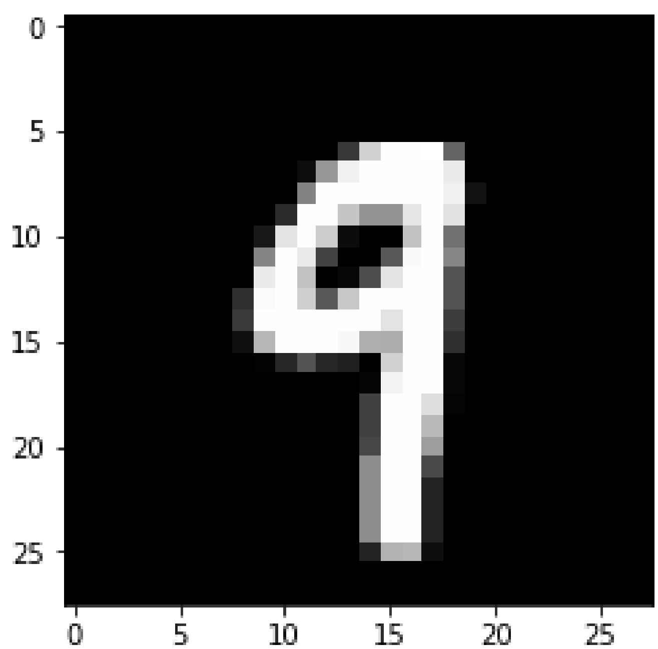
Train Test Split

- We split the data into train and test sets.
- test size is 10%.
- train size is 90%.

```
# Split the train and the validation set for the fitting
from sklearn.model_selection import train_test_split
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train,
test_size = 0.1, random_state=2)
print("x_train shape",X_train.shape)
print("x_test shape",X_val.shape)
print("y_train shape",Y_train.shape)
print("y_test shape",Y_val.shape)
```

```
x_train shape (37800, 28, 28, 1)
x_test shape (4200, 28, 28, 1)
y_train shape (37800, 10)
y_test shape (4200, 10)
```

```
# Some examples
plt.imshow(X_train[2][:,:,0],cmap='gray')
plt.show()
```



Convolutional Neural Network

- CNN is used for image classification, object detection

What is Convolution Operation?

- We have some image and feature detector(3*3)
- Feature detector does not need to be 3 by 3 matrix. It can be 5 by 5 or 7 by 7.
- Feature detector = kernel = filter
- Feauture detector detects features like edges or convex shapes. Example, if out input is dog, feature detector can detect features like ear or tail of the dog.
- feature map = conv(input image, feature detector). Element wise multiplication of matrices.
- feature map = convolved feature
- Stride = navigating in input image.
- We reduce the size of image. This is important code runs faster. However, we lost information.
- We create multiple feature maps bc we use multiple feature detectors(filters).

Same Padding

- As we keep applying conv layers, the size of the volume will decrease faster than we would like. In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features.

Max Pooling

- It makes down-sampling or sub-sampling (Reduces the number of parameters)
- It makes the detection of features invariant to scale or orientation changes.
- It reduce the amount of parameters and computation in the network, and hence to also control overfitting.

Flattening

Full Connection

- Neurons in a fully connected layer have connections to all activations in the previous layer
- Artificial Neural Network

Implementing with Keras

Create Model

```
#
from sklearn.metrics import confusion_matrix
import itertools

from keras.utils.np_utils import to_categorical # convert to one-hot-
encoding
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.optimizers import RMSprop,Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau

model = Sequential()
#
model.add(Conv2D(filters = 8, kernel_size = (5,5),padding = 'Same',
                 activation ='relu', input_shape = (28,28,1)))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#
model.add(Conv2D(filters = 16, kernel_size = (3,3),padding = 'Same',
                 activation ='relu'))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
model.add(Dropout(0.25))
# fully connected
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.5))
model.add(Dense(10, activation = "softmax"))
```

Define Optimizer

- Adam optimizer: Change the learning rate

```
# Define the optimizer
optimizer = Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Compile Model

- categorical crossentropy

```
# Compile the model
model.compile(optimizer = optimizer , loss =
"categorical_crossentropy", metrics=["accuracy"])
```

Epochs and Batch Size

- Say you have a dataset of 10 examples (or samples). You have a **batch size** of 2, and you've specified you want the algorithm to run for 3 **epochs**. Therefore, in each epoch, you have 5 **batches** (10/2 = 5). Each batch gets passed through the algorithm, therefore you have 5 iterations **per epoch**.
- reference: https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks

```
epochs = 10  # for better result increase the epochs
batch_size = 250
```

Data Augmentation

- To avoid overfitting problem, we need to expand artificially our handwritten digit dataset
- Alter the training data with small transformations to reproduce the variations of digit.
- For example, the number is not centered The scale is not the same (some who write with big/small numbers) The image is rotated.

```
# data augmentation
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the
dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std
of the dataset
        samplewise_std_normalization=False,  # divide each input by
its std
        zca_whitening=False,  # dimesion reduction
        rotation_range=5,  # randomly rotate images in the range 5
degrees
        zoom_range = 0.1, # Randomly zoom image 10%
        width_shift_range=0.1,  # randomly shift images horizontally
10%
        height_shift_range=0.1,  # randomly shift images vertically
10%
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images

datagen.fit(X_train)
```

Fit the model

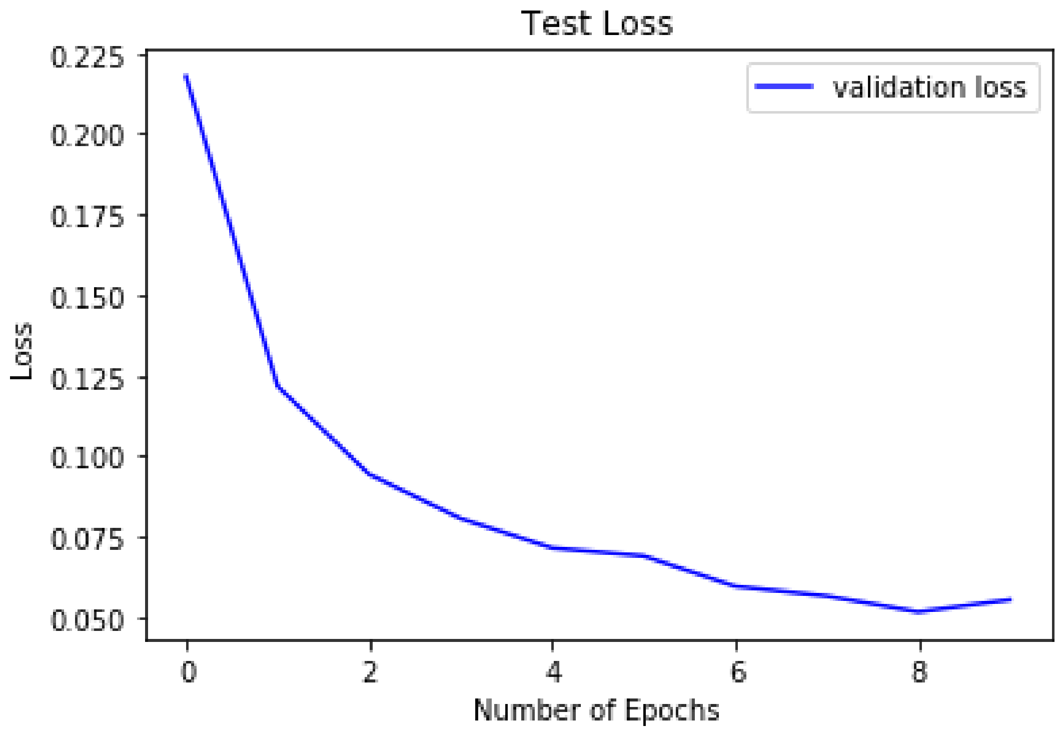
```
# Fit the model
history = model.fit_generator(datagen.flow(X_train,Y_train, 
batch_size=batch_size),
                              epochs = epochs, validation_data = 
(X_val,Y_val), steps_per_epoch=X_train.shape[0] // batch_size)

Epoch 1/10
151/151 [==============================] - 29s 192ms/step - loss: 
1.0913 - acc: 0.6364 - val_loss: 0.2178 - val_acc: 0.9443
Epoch 2/10
151/151 [==============================] - 29s 190ms/step - loss: 
0.4216 - acc: 0.8646 - val_loss: 0.1217 - val_acc: 0.9648
Epoch 3/10
151/151 [==============================] - 29s 190ms/step - loss: 
0.3057 - acc: 0.9011 - val_loss: 0.0943 - val_acc: 0.9726
Epoch 4/10
151/151 [==============================] - 29s 190ms/step - loss: 
0.2571 - acc: 0.9207 - val_loss: 0.0806 - val_acc: 0.9750
Epoch 5/10
151/151 [==============================] - 28s 189ms/step - loss: 
0.2274 - acc: 0.9276 - val_loss: 0.0715 - val_acc: 0.9793
Epoch 6/10
151/151 [==============================] - 29s 189ms/step - loss: 
0.2024 - acc: 0.9372 - val_loss: 0.0690 - val_acc: 0.9798
Epoch 7/10
151/151 [==============================] - 28s 189ms/step - loss: 
0.1821 - acc: 0.9437 - val_loss: 0.0596 - val_acc: 0.9829
Epoch 8/10
151/151 [==============================] - 29s 189ms/step - loss: 
0.1734 - acc: 0.9454 - val_loss: 0.0565 - val_acc: 0.9831
Epoch 9/10
151/151 [==============================] - 28s 186ms/step - loss: 
0.1624 - acc: 0.9491 - val_loss: 0.0516 - val_acc: 0.9869
Epoch 10/10
151/151 [==============================] - 29s 189ms/step - loss: 
0.1632 - acc: 0.9486 - val_loss: 0.0554 - val_acc: 0.9840
```

Evaluate the model

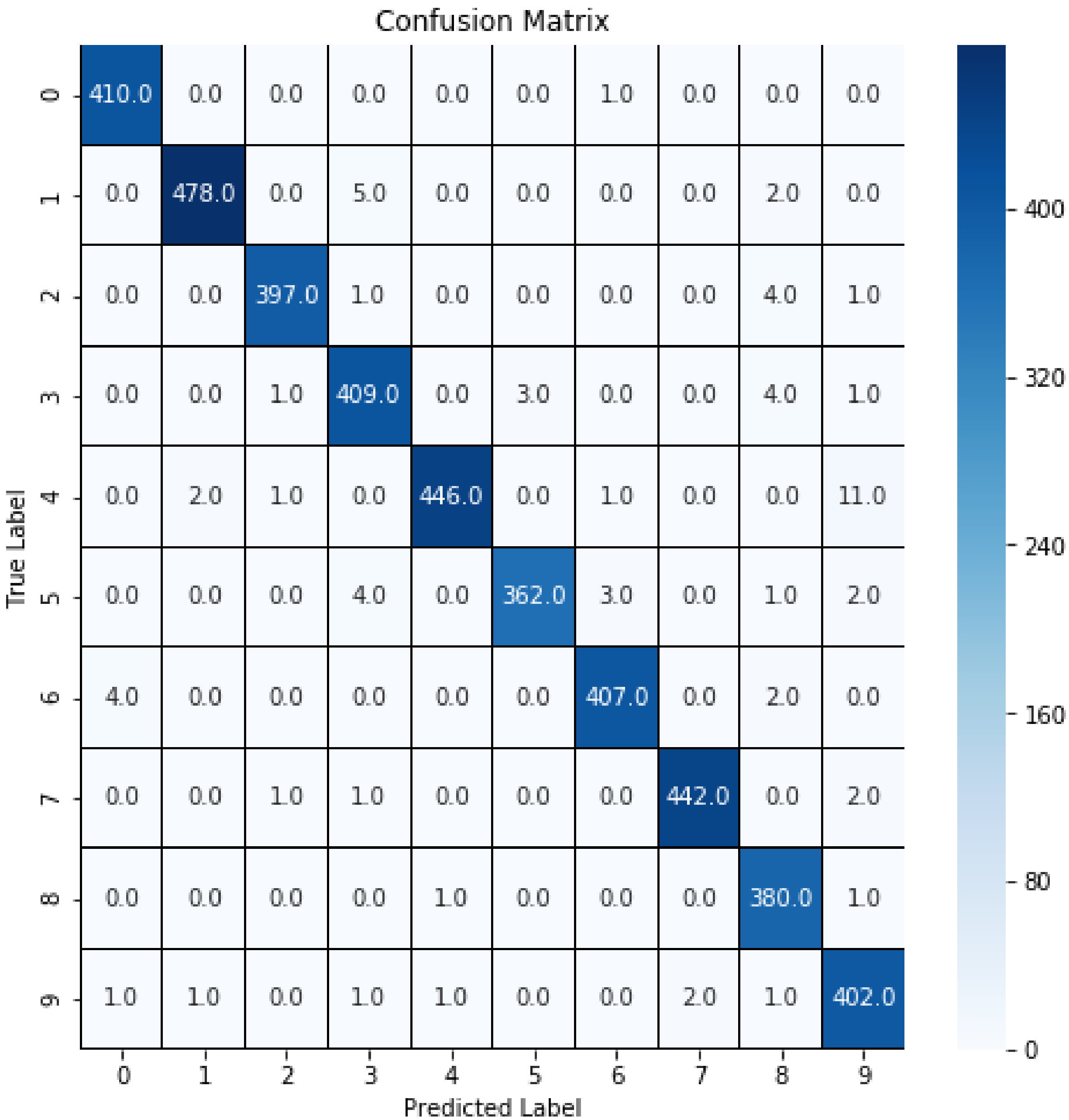
- Test Loss visualization
- Confusion matrix

```
# Plot the loss and accuracy curves for training and validation
plt.plot(history.history['val_loss'], color='b', label="validation 
loss")
plt.title("Test Loss")
plt.xlabel("Number of Epochs")
```

```
plt.ylabel("Loss")
plt.legend()
plt.show()
```




```
# confusion matrix
import seaborn as sns
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
f,ax = plt.subplots(figsize=(8, 8))
sns.heatmap(confusion_mtx, annot=True,
linewidths=0.01,cmap="Blues",linecolor="black", fmt= '.1f',ax=ax)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```


Confusion Matrix
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9
410.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0
0.0 478.0 0.0 5.0 0.0 0.0 0.0 0.0 2.0 0.0
0.0 0.0 397.0 1.0 0.0 0.0 0.0 0.0 4.0 1.0
0.0 0.0 1.0 409.0 0.0 3.0 0.0 0.0 4.0 1.0
0.0 2.0 1.0 0.0 446.0 0.0 1.0 0.0 0.0 11.0
0.0 0.0 0.0 4.0 0.0 362.0 3.0 0.0 1.0 2.0
4.0 0.0 0.0 0.0 0.0 0.0 407.0 0.0 2.0 0.0
0.0 0.0 1.0 1.0 0.0 0.0 0.0 442.0 0.0 2.0
0.0 0.0 0.0 0.0 1.0 0.0 0.0 0.0 380.0 1.0
1.0 1.0 0.0 1.0 1.0 0.0 0.0 2.0 1.0 402.0
0 80 160 240 320 400