In [28]:
```
from __future__ import print_function
import keras
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.preprocessing import image
from keras.utils import Sequence
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.python.client import device_lib
from keras.callbacks import TensorBoard
from keras.models import model_from_json
from sklearn.model_selection import train_test_split
%matplotlib inline

import glob
import imageio
import pandas as pd
from os import listdir
from datetime import timezone, datetime

import holoviews as hv
from holoviews import opts
hv.extension('bokeh')
```

Initialize

Define constant variable

- data_dir : root directory for date
- label_file : location of label file
- history_csv : location of history output file
- model_file : location of model output file
- weight_file : locaiton of weight output file

Checking processor

Checking CPU and GPU available

In [2]:
```
# input
data_dir = '../../DATASET/Black/data/'
label_file = '../../DATASET/Black/label.csv'

# output
timestamp = int(datetime.now().timestamp()*1000)
history_csv = './results/history_{}.csv'.format(timestamp)
confusion_csv = './results/confusion_{}.csv'.format(timestamp)
model_file = './models/model_{}.json'.format(timestamp)
weight_file = './models/weight_{}.h5'.format(timestamp)
log_tb = './logs/tensorboard/{}/'.format(timestamp)

# config training
batch_size = 5
num_epochs = 10
learning_rate = 0.0005

print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 10718419965069393542
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 3215668019
locality {
  bus_id: 1
  links {
  }
}
incarnation: 2673047097416521548
physical_device_desc: "device: 0, name: GeForce GTX 950M, pci bus id: 0000:01:00.0, compute capability: 5.
0"
]
```

Prepare Model

create simple model for CNN

In [3]:
```
def create_model() :
    model = Sequential()

    # Convolution
    ## 5x5 convolution with 2x2 stride and 32 filters
    model.add(Conv2D(32, (5, 5), strides = (4, 4), padding='same',
                     input_shape=(3000, 3000, 1)))
    model.add(Activation('relu'))

    ## 2x2 max pooling reduces to 3 x 3 x 32
    model.add(MaxPooling2D(pool_size=(4, 4)))

    ## Another 5x5 convolution with 2x2 stride and 32 filters
    model.add(Conv2D(32, (5, 5), strides = (4, 4)))
    model.add(Activation('relu'))

    ## 2x2 max pooling reduces to 3 x 3 x 32
    model.add(MaxPooling2D(pool_size=(4, 4)))

    ## Flatten turns 3x3x32 into 288x1
    model.add(Flatten())

    model.add(Dense(100))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))

    model.add(Dense(100))
    model.add(Activation('relu'))
    model.add(Dropout(0.4))

    model.add(Dense(2))
    model.add(Activation('sigmoid'))

    return model
```

In [4]:
```
model = create_model()
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 750, 750, 32)      832
_________________________________________________________________
activation_1 (Activation)    (None, 750, 750, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 187, 187, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 32)        25632
_________________________________________________________________
activation_2 (Activation)    (None, 46, 46, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 11, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3872)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               387300
_________________________________________________________________
activation_3 (Activation)    (None, 100)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               10100
_________________________________________________________________
activation_4 (Activation)    (None, 100)               0
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 202
_________________________________________________________________
activation_5 (Activation)    (None, 2)                 0
=================================================================
Total params: 424,066
Trainable params: 424,066
Non-trainable params: 0
_________________________________________________________________
```

Prepare data

Prepare label and input file name

In [5]:
```
df  = pd.read_csv(label_file, index_col=0)
df.image = data_dir + df.image
df.head()
```

Out[5]:

	image	label
0	../../DATASET/Black/data/1.jpg	0
1	../../DATASET/Black/data/2.jpg	0
2	../../DATASET/Black/data/3.jpg	0
3	../../DATASET/Black/data/4.jpg	0
4	../../DATASET/Black/data/5.jpg	0

Split data set

split data to train(70%), val(15%) and test(15%) set

In [6]:
```
# Split - train:70, val:15, test:15
x_train, x_test, y_train, y_test = train_test_split(df.image, df.label, test_size=0.30)
x_val, x_test, y_val, y_test = train_test_split(x_test, y_test, test_size=0.50)
```

Create data generator

create data generator for query data from data set

In [7]:
```
class DataGenerator(Sequence) :
    def __init__(self, x, y, batch_size) :
        self.x = x
        self.y = keras.utils.to_categorical(y, 2)
        self.batch_size = batch_size

    def __len__(self) :
        return int(np.ceil(len(self.x) / float(self.batch_size)))

    def __getitem__(self, idx) :
        # get all data in batch number idx
        start = idx * self.batch_size
        end = (idx+1) * self.batch_size
        names = self.x.iloc[start : end]

        batch_x = np.array([ np.array(imageio.imread(f, pilmode='L')).reshape((3000, 3000, 1)) for f in name
s])
        batch_x = batch_x.astype('float16')
        batch_x /= 255
        batch_y = np.array(self.y[start: end])
        return batch_x, batch_y
```

In [8]:
```
train_generator = DataGenerator(x_train, y_train, batch_size)
val_generator = DataGenerator(x_val, y_val, batch_size)
test_generator = DataGenerator(x_test, y_test, batch_size)
```

Training model

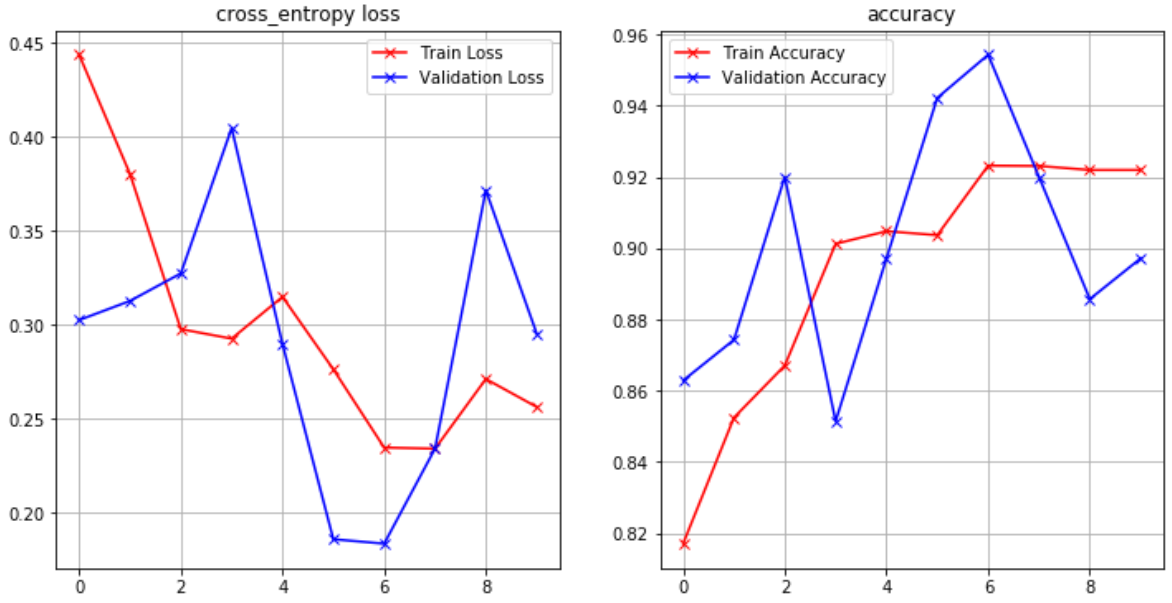
traing model with training set and validate set

In [9]:
```
opt = keras.optimizers.rmsprop(lr=learning_rate, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

tensorboard = TensorBoard(log_dir=log_tb)

history = model.fit_generator(generator=train_generator,
                                  steps_per_epoch=(len(train_generator) // batch_size),
                                  epochs=num_epochs,
                                  verbose=1,
                                  validation_data=val_generator,
                                  validation_steps=(len(val_generator) // batch_size),
                                  callbacks=[tensorboard])
```

```
Epoch 1/10
164/164 [==============================] - 186s 1s/step - loss: 0.4437 - acc: 0.8171 - val_
loss: 0.3024 - val_acc: 0.8629
Epoch 2/10
164/164 [==============================] - 167s 1s/step - loss: 0.3802 - acc: 0.8524 - val_
loss: 0.3126 - val_acc: 0.8743
Epoch 3/10
164/164 [==============================] - 165s 1s/step - loss: 0.2976 - acc: 0.8671 - val_
loss: 0.3272 - val_acc: 0.9200
Epoch 4/10
164/164 [==============================] - 160s 975ms/step - loss: 0.2925 - acc: 0.9012 -
val_loss: 0.4046 - val_acc: 0.8514
Epoch 5/10
164/164 [==============================] - 164s 1s/step - loss: 0.3145 - acc: 0.9049 - val_
loss: 0.2896 - val_acc: 0.8971
Epoch 6/10
164/164 [==============================] - 148s 905ms/step - loss: 0.2762 - acc: 0.9037 -
val_loss: 0.1857 - val_acc: 0.9422
Epoch 7/10
164/164 [==============================] - 143s 871ms/step - loss: 0.2345 - acc: 0.9232 -
val_loss: 0.1833 - val_acc: 0.9543
Epoch 8/10
164/164 [==============================] - 142s 866ms/step - loss: 0.2337 - acc: 0.9232 -
val_loss: 0.2339 - val_acc: 0.9200
Epoch 9/10
164/164 [==============================] - 144s 875ms/step - loss: 0.2710 - acc: 0.9220 -
val_loss: 0.3715 - val_acc: 0.8857
Epoch 10/10
164/164 [==============================] - 144s 880ms/step - loss: 0.2561 - acc: 0.9220 -
val_loss: 0.2952 - val_acc: 0.8971
```

plot loss and acc graph

In [10]:
```
def plot_loss_accuracy(history):
    fig = plt.figure(figsize=(12, 6))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(history.history["loss"],'r-x', label="Train Loss")
    ax.plot(history.history["val_loss"],'b-x', label="Validation Loss")
    ax.legend()
    ax.set_title('cross_entropy loss')
    ax.grid(True)


    ax = fig.add_subplot(1, 2, 2)
    ax.plot(history.history["acc"],'r-x', label="Train Accuracy")
    ax.plot(history.history["val_acc"],'b-x', label="Validation Accuracy")
    ax.legend()
    ax.set_title('accuracy')
    ax.grid(True)


plot_loss_accuracy(history)
```



Save model

- save model to .json file
- save weight to .h5 file
- save history to .csv file

In [11]:
```
model_json = model.to_json()
with open(model_file, "w+") as json_file:
    json_file.write(model_json)
model.save_weights(weight_file)
```

In [12]:
```
history_df = pd.DataFrame(history.history)
history_df.to_csv(history_csv)
history_df
```

Out[12]:

	val_loss	val_acc	loss	acc
0	0.302399	0.862857	0.443654	0.817073
1	0.312582	0.874286	0.380213	0.852439
2	0.327236	0.920000	0.297565	0.867073
3	0.404608	0.851429	0.292503	0.901220
4	0.289629	0.897143	0.314770	0.904762
5	0.185703	0.942197	0.276181	0.903659
6	0.183348	0.954286	0.234515	0.923171
7	0.233929	0.920000	0.233914	0.923077
8	0.371512	0.885714	0.271019	0.921951
9	0.295235	0.897143	0.256132	0.921951

Eavalute

predict Y

predict Y with test set

In [13]:
```
# predict
_y_pred = model.predict_generator(test_generator, verbose=1)
y_pred = _y_pred.argmax(axis=1)
```

```
176/176 [==============================] - 177s 1s/step
```

Confusion matrix

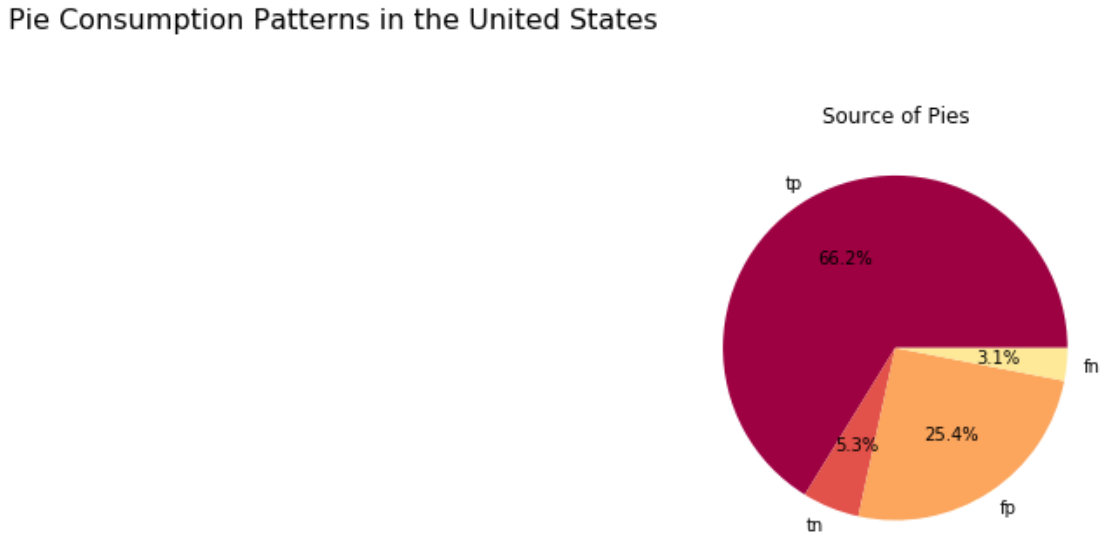
calculate confusion matrix and accuracy

In [40]:
```
result_df = pd.concat([y_test.reset_index(), pd.Series(y_pred, name='predict')], axis=1)
false_positive_df = result_df[(result_df.predict - result_df.label) == 1]
false_negative_df = result_df[(result_df.predict - result_df.label) == -1]
true_positive_df = result_df[((result_df.predict - result_df.label) == 0) & (result_df.label == 1)]
true_negative_df = result_df[((result_df.predict - result_df.label) == 0) & (result_df.label == 0)]

ap = len(result_df[result_df.label == 1]) # all positive label
an = len(result_df[result_df.label == 0]) # all negative label

fp = len(false_positive_df)
fn = len(false_negative_df)
tp = len(true_positive_df)
tn = len(true_negative_df)
tr = tp+tn

# calculate percent from expect label
fp_percent = fp / an
fn_percent = fn / ap
tp_percent = tp / ap
tn_percent = tn / an
tr_percent = tr / len(result_df)
total = len(result_df)

confusion_mat = pd.DataFrame({'false_positive': [fp, fp_percent],
                              'false_negative': [fn, fn_percent],
                              'true_positive': [tp, tp_percent],
                              'true_negative': [tn, tn_percent],
                              'true_result':[tr, tr_percent],
                              'total': [total, 1.0]}, index=['amount', 'percent_group'])
confusion_mat.to_csv(confusion_csv)
confusion_mat
```

Out[40]:

	false_positive	false_negative	true_positive	true_negative	true_result	total
amount	27.000	47.000000	582.000000	223.000	805.000000	879.0
percent_group	0.108	0.074722	0.925278	0.892	0.915813	1.0

In [39]:
```
import matplotlib.ticker as ticker
import matplotlib.cm as cm
import matplotlib as mpl
from matplotlib.gridspec import GridSpec

# Make square figures and axes
plt.figure(1, figsize=(20,10))
the_grid = GridSpec(2, 2)


cmap = plt.get_cmap('Spectral')
colors = [cmap(i) for i in np.linspace(0, 1, 8)]


plt.subplot(the_grid[0, 1], aspect=1, title='Source of Pies')

source_pie = plt.pie([tp, fn, tn, fp], labels=['tp', 'tn', 'fp', 'fn'], autopct='%1.1f%%', colors=colors)

plt.suptitle('Pie Consumption Patterns in the United States', fontsize=16)

plt.show()
```



In [15]:
```
# load json and create model

# json_file = open(model_file, 'r')
# loaded_model_json = json_file.read()
# json_file.close()
# loaded_model = model_from_json(loaded_model_json)
# # load weights into new model
# loaded_model.load_weights(weight_file)
# print("Loaded model from disk")

# opt = keras.optimizers.rmsprop(lr=0.0005, decay=1e-6)
# loaded_model.compile(loss='categorical_crossentropy',
#              optimizer=opt,
#              metrics=['accuracy'])
# result = loaded_model.evaluate_generator(test_generator, verbose=1)
```