Chart Image Classification using CNN

```
import numpy as np
import tensorflow as tf
from time import time
import math
from include .data import get_data_set
from include.model import model, lr
train_x, train_y= get_data_set("train")
test_x, test_y = get_data_set("test")
tf. set_random_seed(21)
x, y, output, y_pred_cls, global_step, learning_rate=model()
global_accuracy =0
epoch_start=0
#PARAM
_BATCH_SIZE=128
_EPOCH=60
_SAVE_PATH="./tensorboard/cifar-10-v1.0.0/"
#LOSS AND OPTIMIZER
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=output, labels=y))
optimizer=tf.train.AdamOptimizer(learning_rate= learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-08). Minimize(loss, global_step=global_step)
#PREDICTION AND ACCURACY CALCULATION
correct_prediction=tf.equal(y_pred_cls, tf.argmax(y, axis=1))
accuracy = tf.reduce_mean(tf.cast(correct_predictiction, tf.float32))
# SAVER
merged = tf.summary.merge_all()
saver = tf.train.Saver()
sess = tf.Session()
train_writer= tf.summary.FileWriter(_SAVE_PATH, sess.graph)
```

```
try:
print(" Trying to restore last checkpoint?")
last_chk_path= tf.train.latest_checkpoint(checkpoint_dir=SAVE_PATH)
saver.restore(sess, save_path=last_chk_path)
print("Restored checkpoint from:", last_chk_path)
except ValueError:
print("Failed to restore checkpoint. Initializing variable instead.")
sess.run(tf.global_variables_initializer())
def train(epoch):
    global epoch_start
    epoch_start= time()
    batch_size=int(math.ceil(len(train_x)/_BATCH_SIZE))
    i_global = 0
for s in range(batch_size):
    batch_xs= train_x[s*_BATCH_SIZE: (s+1)*_BATCH_SIZE]
    batch_ys = train_y[s*_BATCH_SIZE: (s+1)*_BATCH_SIZE]
   start_time= time()
i_global, _, batch_loss, batch_acc=sess.run( [global_step, optimizer, loss, accuracy],
feed_dict={x: batch_xs, y: batch_ys, learning_rate: lr(epoch)})
duration = time() - start_time
if s% 10== 0:
percentage = int(round((s/batch_size)*100))
```

```
bar_len=29
filled_len= int ((bar_len*int(percentage))/100)
bar='=' *filled_len + ?>' + ?-? * (bar_len _filled_len)
msg= "Global step: { :>5} - [{}] {:>3}% -acc: {:.{:>4f} - loss: {:.4f} -{:.1f} sample/sec"
print(msg.format(i_global, bar, percentage, batch_acc, batch_loss, _BATCH_SIZE/duration))
test_and_save(i_global, epoch)
def test_and_save(_global_step, epoch):
global global_accuracy
global epoch_start
i=0
predicted_class=np.zeroes(shape=len(test_x), dtype=np.int)
while i< len (test_x) : j=min(i+_BATCH_SIZE, len(test_x)) batch_xs=test_x[I:j, :] batch_ys=test_y[i:j,:] predicted_class[i:j]=sess.run(y_pred_cls, feed_dict=x:b
Epoch {} - accuracy: {: .2f}% ({}/{})- time: {:0>2}:{:0.2}:{:05.2f}"
print(mes.format((epoch+1), acc, correct_numbers, len(test_x), int(hours), int(minutes), seconds))
if global_accuracy != 0 and global_accuracy < acc: summary = tf.Summary(value=
[ tf.Summary.Value(tag="Accuracy/test", simple_value=acc), ]) train_writer.add_summary(summary, _global_step) saver.save(sess, save_path=_SAVE_PA
 print(mes.format((acc, global_accuracy))
global_accuracy = acc
elif global_accuracy==0:
global_accuracy=acc
print("#################################################################
def main():
train_start=time()
for i in range(_EPOCH):
print(" Epoch: {}/{}".format(( i+1),_EPOCH))
train(i)
hours, rem=divmod(time()-train_start, 3600 minutes, seconds=divmod(rem,60)
mes= "Best accuracy per session: {:.2f}, time: {:0>2}:{:0>2}:{:05.2f}"
```

```
print(mes.format(global_accuracy, int(hours), int(minutes), seconds))
if _name_ =="_main_":
main()
sess.close()
```

Output:

```
Epoch: 60/60

Global step: 23070 - [>------------------------------]   0% - acc: 0.9531 - loss: 1.5081 - 7045.4 sample/sec
Global step: 23080 - [>------------------------------]   3% - acc: 0.9453 - loss: 1.5159 - 7147.6 sample/sec
Global step: 23090 - [=>-----------------------------]   5% - acc: 0.9844 - loss: 1.4764 - 7154.6 sample/sec
Global step: 23100 - [==>----------------------------]   8% - acc: 0.9297 - loss: 1.5307 - 7104.4 sample/sec
Global step: 23110 - [==>----------------------------]  10% - acc: 0.9141 - loss: 1.5462 - 7091.4 sample/sec
Global step: 23120 - [===>---------------------------]  13% - acc: 0.9297 - loss: 1.5314 - 7162.9 sample/sec
Global step: 23130 - [====>--------------------------]  15% - acc: 0.9297 - loss: 1.5307 - 7174.8 sample/sec
Global step: 23140 - [=====>-------------------------]  18% - acc: 0.9375 - loss: 1.5231 - 7140.0 sample/sec
Global step: 23150 - [=====>-------------------------]  20% - acc: 0.9297 - loss: 1.5301 - 7152.8 sample/sec
Global step: 23160 - [======>------------------------]  23% - acc: 0.9531 - loss: 1.5080 - 7112.3 sample/sec
Global step: 23170 - [=======>-----------------------]  26% - acc: 0.9609 - loss: 1.5000 - 7154.0 sample/sec
Global step: 23180 - [========>----------------------]  28% - acc: 0.9531 - loss: 1.5074 - 6862.2 sample/sec
Global step: 23190 - [========>----------------------]  31% - acc: 0.9609 - loss: 1.4993 - 7134.5 sample/sec
Global step: 23200 - [=========>---------------------]  33% - acc: 0.9609 - loss: 1.4995 - 7166.0 sample/sec
Global step: 23210 - [==========>--------------------]  36% - acc: 0.9375 - loss: 1.5231 - 7116.7 sample/sec
Global step: 23220 - [===========>-------------------]  38% - acc: 0.9453 - loss: 1.5153 - 7134.1 sample/sec
Global step: 23230 - [===========>-------------------]  41% - acc: 0.9375 - loss: 1.5233 - 7074.5 sample/sec
Global step: 23240 - [============>------------------]  43% - acc: 0.9219 - loss: 1.5387 - 7176.9 sample/sec
Global step: 23250 - [=============>-----------------]  46% - acc: 0.8828 - loss: 1.5769 - 7144.1 sample/sec
Global step: 23260 - [==============>----------------]  49% - acc: 0.9219 - loss: 1.5383 - 7059.7 sample/sec
Global step: 23270 - [==============>----------------]  51% - acc: 0.8984 - loss: 1.5618 - 6638.6 sample/sec
Global step: 23280 - [===============>---------------]  54% - acc: 0.9453 - loss: 1.5151 - 7035.7 sample/sec
Global step: 23290 - [================>--------------]  56% - acc: 0.9609 - loss: 1.4996 - 7129.0 sample/sec
Global step: 23300 - [=================>-------------]  59% - acc: 0.9609 - loss: 1.4997 - 7075.4 sample/sec
Global step: 23310 - [=================>-------------]  61% - acc: 0.8750 - loss:1.5842 - 7117.8 sample/sec
```

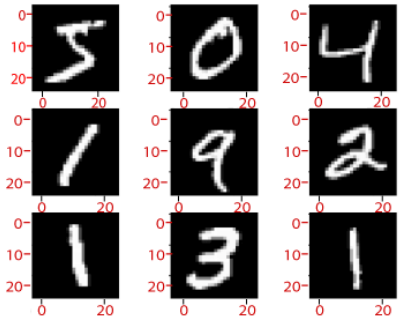
```
Global step: 23320 - [==================>-----------]  64% - acc: 0.9141 - loss:1.5463 - 7157.2 sample/sec
Global step: 23330 - [==================>----------]  66% - acc: 0.9062 - loss: 1.5549 - 7169.3 sample/sec
Global step: 23340 - [====================>---------]  69% - acc: 0.9219 - loss: 1.5389 - 7164.4 sample/sec
Global step: 23350 - [====================>---------]  72% - acc: 0.9609 - loss: 1.5002 - 7135.4 sample/sec
Global step: 23360 - [======================>-------]  74% - acc: 0.9766 - loss: 1.4842 - 7124.2 sample/sec
Global step: 23370 - [======================>-------]  77% - acc: 0.9375 - loss: 1.5231 - 7168.5 sample/sec
Global step: 23380 - [========================>-----]  79% - acc: 0.8906 - loss: 1.5695 - 7175.2 sample/sec
Global step: 23390 - [========================>-----]  82% - acc: 0.9375 - loss: 1.5225 - 7132.1 sample/sec
Global step: 23400 - [==========================>---]  84% - acc: 0.9844 - loss: 1.4768 - 7100.1 sample/sec
Global step: 23410 - [==========================>---]  87% - acc: 0.9766 - loss: 1.4840 - 7172.0 sample/sec
Global step: 23420 - [============================>-]  90% - acc: 0.9062 - loss: 1.5542 - 7122.1 sample/sec
Global step: 23430 - [============================>-]  92% - acc: 0.9297 - loss: 1.5313 - 7145.3 sample/sec
Global step: 23440 - [============================>-]  95% - acc: 0.9297 - loss: 1.5301 - 7133.3 sample/sec
Global step: 23450 - [============================>-]  97% - acc: 0.9375 - loss: 1.5231 - 7135.7 sample/sec
Global step: 23460 - [==============================>] 100% - acc: 0.9250 - loss: 1.5362 - 10297.5 sample/sec

Epoch 60 - accuracy: 78.81% (7881/10000)
This epoch receive better accuracy: 78.81 > 78.78. Saving session...
########################################################################################################
```

Run Network on Test Dataset:

```
import numpy as np
import tensorflow as tf
from include.data import get_data_set
from include.model import model
test_x, test_y= get_data_set("test")
x, y, output, y_pred_cls, global_step, learning_rate =model()
_BATCH_SIZE = 128
_CLASS_SIZE = 10
_SAVE_PATH = "./tensorboard/cifar-10-v1.0.0/"
saver= tf.train.Saver()
Sess=tf.Session()
try;
 print(" Trying to restore last checkpoint ...")
last_chk_path = tf.train.latest_checkpoint(checkpoint_dir=_SAVE_PATH
saver.restore(sess, save_path=last_chk_path)
print("Restored checkpoint from:", last_chk_path)
expect ValueError:
print("
Failed to restore checkpoint. Initializing variables instead.")
sess.run(tf.global_variables_initializer())
def main():
i=0
predicted_class= np.zeros(shape=len(test_x), dtype=np.int)
while i< lens(test_x):
j=min(i+_BATCH_SIZE, len(test_x))
```

```
batch_xs=test_x[i:j,:]
batch_xs=test_y[i:j,:]
pre dicted_class[i:j] = sess.run(y_pred_cls, feed_dict={x: batch_xs, y: batch_ys})
i=j
corr ect = (np.argmax(test_y, axis=1) == predicted_class)
acc=correct.mean()*100
correct_numbers=correct.sum()
print()
print("Accuracy is on Test-Set: {0:.2f}% ({1} / {2})".format(acc, correct_numbers, len(test_x)))
if__name__=="__main__":
main()
sess.close()
```

```
Trying to restore last checkpoint ...
Restored checkpoint from: ./tensorboard/cifar-10-v1.0.0/-23460

Accuracy on Test-Set: 78.81% (7881 / 10000)
```

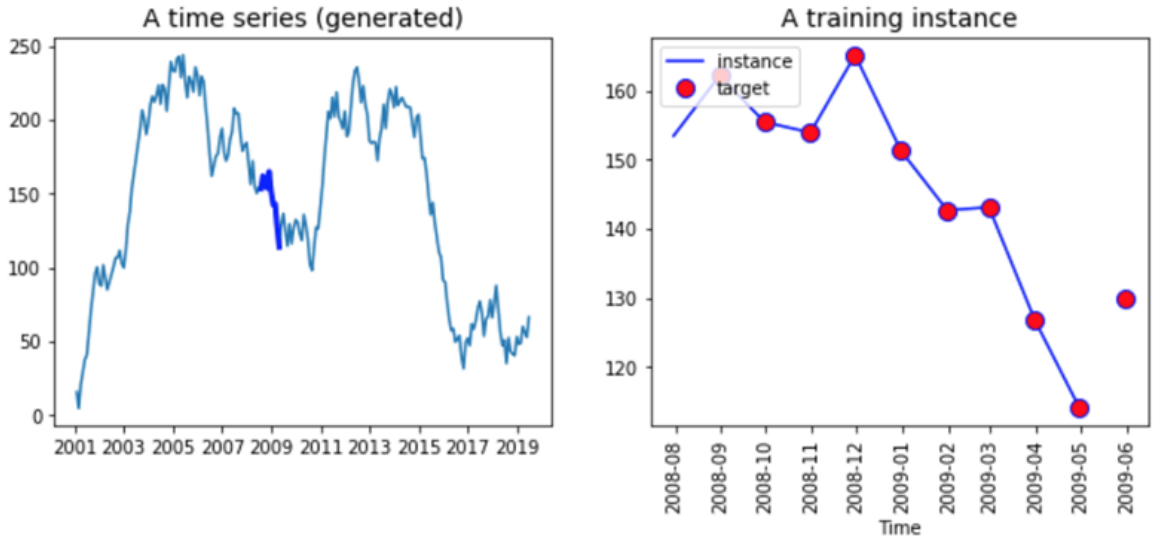


```
# To plotting amazing figure
%matplotlib inline
import matplotlib
import pandas as pd
import matplotlib.pyplot as plt
def create_ts(start = '2001', n = 201, freq = 'M'):
ring = pd.date_range(start=start, periods=n, freq=freq)
ts =pd.Series(np.random.uniform(-18, 18, size=len(rng)), ring).cumsum()
return ts
ts= create_ts(start = '2001', n = 192, freq = 'M')
ts.tail(5)
```

Output:

```
2016-08-31    -93.459631
2016-09-30    -95.264791
2016-10-31    -95.551935
2016-11-30   -105.879611
2016-12-31   -123.729319
Freq: M, dtype: float64

ts = create_ts(start = '2001', n = 222)
```

```
# Left plotting diagram
plt.figure(figsize=(11,4))
plt.subplot(121)
plt.plot(ts.index, ts)
plt.plot(ts.index[90:100], ts[90:100], "b-",linewidth=3, label="A train illustration in the plotting area")
plt.title("A time series (generated)", fontsize=14)

## Right side plotted Diagram
plt.subplot(122)
plt.title("A training instance", fontsize=14)
plt.plot(ts.index[90:100], ts[90:100], "b-", markersize=8, label="instance")
plt.plot(ts.index[91:101], ts[91:101], "bo", markersize=10, label="target", markerfacecolor='red')
plt.legend(loc="upper left")
plt.xlabel("Time")
plt.show()
```



```
series = np.array(ts)
n_windows = 20
n_input =  1
n_output = 1
size_train = 201
```

```
# Split data
train = series[:size_train]
test = series[size_train:]
print(train.shape, test.shape)
(201) (21)
```

```
x_data = train[:size_train-1]: Select the training instance.
X_batches = x_data.reshape(-1, Windows, input): creating the right shape for the batch.
def create_batches(df, Windows, input, output):
   ## Create X
      x_data = train[:size_train-1] # Select the data
      X_batches = x_data.reshape(-1, windows, input)  # Reshaping the data in this line of code
   ## Create y
      y_data = train[n_output:size_train]
      y_batches = y_data.reshape(-1, Windows, output)
      return X_batches, y_batches #return the function
```

```
Windows = n_
Windows, # Creating windows
                    input = n_input,
                            output = n_output)
```

```
print(X_batches.shape, y_batches.shape)
(10, 20, 1) (10, 20, 1)
```

```
X_test, y_test = create_batches(df = test, windows = 20,input = 1, output = 1)
print(X_test.shape, y_test.shape)
(10, 20, 1) (10, 20, 1)
```

```
tf.placeholder(tf.float32, [None, n_windows, n_input])
## 1. Construct the tensors
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])
```

```
## 2. create the model
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron, activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)
```

```
stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

```
tf.reduce_sum(tf.square(outputs - y))
```

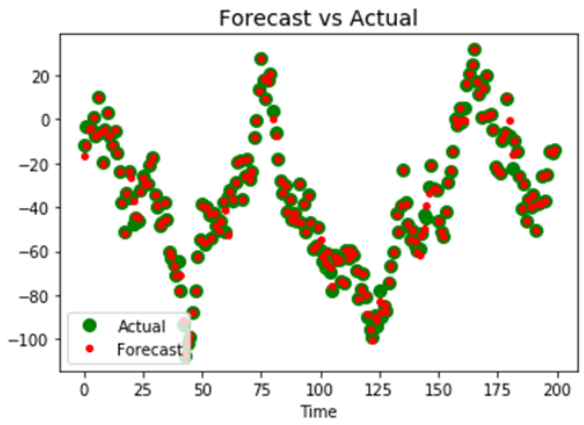
```
tf.train.AdamOptimizer(learning_rate=learning_rate)
optimizer.minimize(loss)
```

```
tf.reset_default_graph()
r_neuron = 120

## 1. Constructing the tensors
X = tf.placeholder(tf.float32, [None, n_windows, n_input])
y = tf.placeholder(tf.float32, [None, n_windows, n_output])
```

```
## 2. creating our models
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron, activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])

## 3. Loss optimization of RNN
learning_rate = 0.001

loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)

init = tf.global_variables_initializer()
```

```
iteration = 1500
with tf.Session() as sess:
    init.run()
    for iters in range(iteration):
        sess.run(training_op, feed_dict={X: X_batches, y: y_batches})
        if iters % 150 == 0:
            mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
            print(iters, "\tMSE:", mse)
    y_pred = sess.run(outputs, feed_dict={X: X_test})
"0  MSE: 502893.34
150     MSE: 13839.129
300     MSE: 3964.835
450     MSE: 2619.885
600     MSE: 2418.772
750     MSE: 2110.5923
900     MSE: 1887.9644
1050    MSE: 1747.1377
1200    MSE: 1556.3398
1350  MSE: 1384.6113"
```

```
plt.title("Forecast vs Actual", fontsize=14)
plt.plot(pd.Series(np.ravel(y_test)), "bo", markersize=8, label="actual", color='green')
plt.plot(pd.Series(np.ravel(y_pred)), "r.", markersize=8, label="forecast", color='red')
plt.legend(loc="lower left")
plt.xlabel("Time")
plt.show()
```



Parameters of the model

```
n_windows = 20
n_input =  1
n_output = 1
size_train = 201
```

Define the model

```
X = tf.placeholder(tf.float32, [none, n_windows, n_input])
y = tf.placeholder(tf.float32, [none, n_windows, n_output])
basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=r_neuron, activation=tf.nn.relu)
rnn_output, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)
stacked_rnn_output = tf.reshape(rnn_output, [-1, r_neuron])
stacked_outputs = tf.layers.dense(stacked_rnn_output, n_output)
outputs = tf.reshape(stacked_outputs, [-1, n_windows, n_output])
```

Constructing the optimization function

```
learning_rate = 0.001
loss = tf.reduce_sum(tf.square(outputs - y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
```

Training the model

```
init = tf.global_variables_initializer()
iteration = 1500

with tf.Session() as sess:
    init.run()
for iters in range(iteration):
sess.run(training_op, feed_dict={X: X_batches, y: y_batches})
        if iters % 150 == 0:
            mse = loss.eval(feed_dict={X: X_batches, y: y_batches})
            print(iters, "\tMSE:", mse)
 y_pred = sess.run(outputs, feed_dict={X: X_test})
```

```
def build_lstm_layers(lstm_sizes, embed, keep_prob_, batch_size):
    """
    Create the LSTM layers
    """
    lstms = [tf.contrib.rnn.BasicLSTMCell(size) for size in lstm_sizes]
    # Add dropout to the cell
    drops = [tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob_) for lstm in lstms]
    # Stacking up multiple LSTM layers, for deep learning
    cell = tf.contrib.rnn.MultiRNNCell(drops)
# Getting an initial state of all zeros
    initial_state = cell.zero_state(batch_size, tf.float32)
    lstm_outputs, final_state = tf.nn.dynamic_rnn(cell, embed, initial_state=initial_state)
```

```
def build_cost_fn_and_opt(lstm_outputs, labels_, learning_rate):
    """
    Creating the Loss function and Optimizer
    """
    predictions = tf.contrib.layers.fully_connected(lstm_outputs[:, -1], 1, activation_fn=tf.sigmoid)
    loss = tf.losses.mean_squared_error(labels_, predictions)
    optimzer = tf.train.AdadeltaOptimizer (learning_rate).minimize(loss)
def build_accuracy(predictions, labels_):
    """
    Create accuracy
    """
    correct_pred = tf.equal(tf.cast(tf.round(predictions), tf.int32), labels_)
    accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

```
def build_and_train_network(lstm_sizes, vocab_size, embed_size, epochs, batch_size,
                            learning_rate, keep_prob, train_x, val_x, train_y, val_y):

    # Build Graph

    with tf.Session() as sess:

        # Train Network
        # Save Network
```

```
Epoch: 1/50 Batch: 303/303 Train Loss: 0.247 Train Accuracy: 0.562 Val Accuracy: 0.578
Epoch: 2/50 Batch: 303/303 Train Loss: 0.245 Train Accuracy: 0.583 Val Accuracy: 0.596
Epoch: 3/50 Batch: 303/303 Train Loss: 0.247 Train Accuracy: 0.597 Val Accuracy: 0.617
Epoch: 4/50 Batch: 303/303 Train Loss: 0.240 Train Accuracy: 0.610 Val Accuracy: 0.627
Epoch: 5/50 Batch: 303/303 Train Loss: 0.238 Train Accuracy: 0.620 Val Accuracy: 0.632
Epoch: 6/50 Batch: 303/303 Train Loss: 0.234 Train Accuracy: 0.632 Val Accuracy: 0.642
Epoch: 7/50 Batch: 303/303 Train Loss: 0.230 Train Accuracy: 0.636 Val Accuracy: 0.648
Epoch: 8/50 Batch: 303/303 Train Loss: 0.227 Train Accuracy: 0.641 Val Accuracy: 0.653
Epoch: 9/50 Batch: 303/303 Train Loss: 0.223 Train Accuracy: 0.646 Val Accuracy: 0.656
Epoch: 10/50 Batch: 303/303 Train Loss: 0.221 Train Accuracy: 0.652 Val Accuracy: 0.659
```

```
def test_network(model_dir, batch_size, test_x, test_y):

    # Build Network

with tf.Session() as sess:

    # Restore Model
    # Test Model
```

```
INFO:tensorflow:Restoring parameters from checkpoints/sentiment.ckpt
Test Accuracy: 0.717
```