p3

November 7, 2024

[ ]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential  # Import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense  #␣
↪Import necessary layers
import matplotlib.pyplot as plt

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Display some sample images
fig = plt.figure()
for i in range(9):
    plt.subplot(3, 3, i + 1)
    plt.tight_layout()
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.title("Digit: {}".format(y_train[i]))
    plt.xticks([])
    plt.yticks([])
plt.show()
```

[ ]:
```
from tensorflow.keras import backend as K
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten,␣
↪Dense

# Assuming images are 28x28
img_rows, img_cols = 28, 28

# Reshape the data according to the image data format
if K.image_data_format() == 'channels_first':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
```

```
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

# More reshaping and normalization
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print('X_train shape:', X_train.shape) # Expected: (60000, 28, 28, 1)

# Set number of categories
num_category = 10

# Convert class vectors to binary class matrices
y_train = to_categorical(y_train, num_category)
y_test = to_categorical(y_test, num_category)

# Model building
model = Sequential()
# Convolutional layer with ReLU activation
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',␣
 ↪input_shape=input_shape))
# 64 convolution filters, each of size 3x3
model.add(Conv2D(64, (3, 3), activation='relu'))
# Choose the best features via pooling
model.add(MaxPooling2D(pool_size=(2, 2)))
# Randomly turn neurons on and off to improve convergence
model.add(Dropout(0.25))
# Flatten since too many dimensions; we only want a classification output
model.add(Flatten())
# Fully connected to get all relevant data
model.add(Dense(128, activation='relu'))
# One more dropout for convergence' sake
model.add(Dropout(0.5))
# Output a softmax to squash the matrix into output probabilities
model.add(Dense(num_category, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adadelta',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
# Training parameters
batch_size = 128
num_epoch = 10

# Model training
model_log = model.fit(
```

```
    X_train, y_train,
    batch_size=batch_size,
    epochs=num_epoch,
    verbose=1,
    validation_data=(X_test, y_test)
)

# Evaluate the model
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])  # Example output: Test loss: 0.0296396646054
print('Test accuracy:', score[1])  # Example output: Test accuracy: 0.9904

# Plotting the metrics
fig = plt.figure()
plt.subplot(2, 1, 1)
plt.plot(model_log.history['accuracy'])  # Updated to 'accuracy'
plt.plot(model_log.history['val_accuracy'])  # Updated to 'val_accuracy'
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='lower right')

plt.subplot(2, 1, 2)
plt.plot(model_log.history['loss'])
plt.plot(model_log.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')

plt.tight_layout()
plt.show()

# Save the model architecture to a JSON file
model_digit_json = model.to_json()
with open("model_digit.json", "w") as json_file:
    json_file.write(model_digit_json)

# Serialize weights to HDF5
model.save_weights("model_digit.weights.h5")  # Corrected filename
print("Saved model to disk")
```