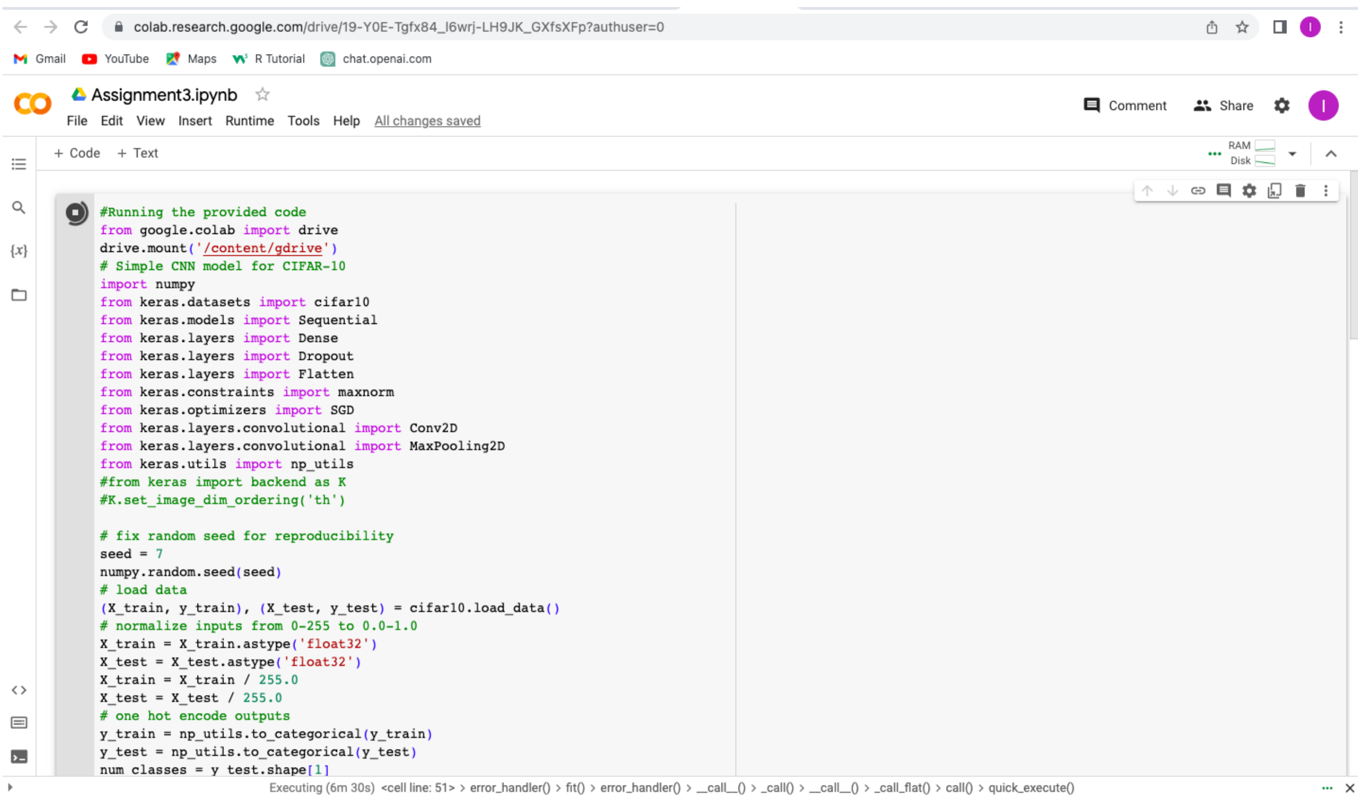
Deep Learning Image Classification with CNN

Use Case Description:

Image Classification with CNN
1. Training the model
2. Evaluating the model

Running the provided code from keras_Example2.pynb




```
#Running the provided code
from google.colab import drive
drive.mount('/content/gdrive')
# Simple CNN model for CIFAR-10
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

```python
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 5
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Output:

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

```
Mounted at /content/gdrive
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/gradient_descent.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` in
  super().__init__(name, **kwargs)
None
Epoch 1/5
1563/1563 [==============================] - 327s 208ms/step - loss: 1.6697 - accuracy: 0.3959 - val_loss: 1.3805 - val_accuracy: 0.5079
Epoch 2/5
1563/1563 [==============================] - ETA: 0s - loss: 1.3613 - accuracy: 0.5106
```

Executing (11m 13s) <cell line: 51> > error_handler() > fit() > error_handler() > evaluate() > error_handler() > __call__() > _call() > __call__() > _call_flat() > call() > quick_execute()

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

+ Code + Text

```
Epoch 3/5
1563/1563 [==============================] - 347s 222ms/step - loss: 1.2445 - accuracy: 0.5544 - val_loss: 1.1784 - val_accuracy: 0.5803
Epoch 4/5
1563/1563 [==============================] - 324s 207ms/step - loss: 1.1671 - accuracy: 0.5840 - val_loss: 1.1245 - val_accuracy: 0.5987
Epoch 5/5
1563/1563 [==============================] - 318s 203ms/step - loss: 1.1106 - accuracy: 0.6040 - val_loss: 1.0970 - val_accuracy: 0.6127
Accuracy: 61.27%
```

In class programming:

1. Follow the instruction below and then report how the performance changed.(apply all at once)
• Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
• Dropout layer at 20%.
• Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
• Max Pool layer with size 2×2.
• Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
• Dropout layer at 20%.
• Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
• Max Pool layer with size 2×2.
• Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
• Dropout layer at 20%.
• Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.
• Max Pool layer with size 2×2.
• Flatten layer.
• Dropout layer at 20%.
• Fully connected layer with 1024 units and a rectifier activation function.
• Dropout layer at 20%.
• Fully connected layer with 512 units and a rectifier activation function.
• Dropout layer at 20%.
• Fully connected output layer with 10 units and a Softmax activation function
Did the performance change?

Answer:

1. Created the model with the required specifications
2. Compiled the model
3. Fit the training data into the model
4. Evaluated the model

Code is:

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

\+ Code + Text

```
#1. Performing all the required changes
from google.colab import drive
drive.mount('/content/gdrive')
# Simple CNN model for CIFAR-10
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
#from keras import backend as K
#K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# Create the model
model = Sequential()
```

Executing (25m 36s) <cell line: 51> > error_handler() > fit() > error_handler() > __call__() > _call() > __call__() > _call_flat() > call() > quick_execute()

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

\+ Code + Text

```
# Create the model
model = Sequential()
#Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
#Dropout layer at 20%.
model.add(Dropout(0.2))
#Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
#Max Pool layer with size 2×2.
model.add(MaxPooling2D(pool_size=(2, 2)))
#Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
#• Dropout layer at 20%.
model.add(Dropout(0.2))
#Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
#Max Pool layer with size 2×2.
model.add(MaxPooling2D(pool_size=(2, 2)))
#Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
#Dropout layer at 20%.
model.add(Dropout(0.2))
#Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
#Max Pool layer with size 2×2.
model.add(MaxPooling2D(pool_size=(2, 2)))
#Flatten layer.
model.add(Flatten())
#Dropout layer at 20%.
model.add(Dropout(0.2))
#Fully connected layer with 1024 units and a rectifier activation function.
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
#Dropout layer at 20%.
model.add(Dropout(0.2))
```

Executing (9m 43s) <cell line: 61> > error_handler() > fit() > error_handler() > __call__() > _call() > __call__() > _call_flat() > call() > quick_execute()

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```python
#Dropout layer at 20%.
model.add(Dropout(0.2))
#Fully connected layer with 512 units and a rectifier activation function
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
#Dropout layer at 20%.
model.add(Dropout(0.2))
#Fully connected output layer with 10 units and a Softmax activation function
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 5
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Output for the above code is:

Assignment3.ipynb

File Edit View Insert Runtime Tools Help All changes saved

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 32, 32, 32)        896

 dropout_2 (Dropout)         (None, 32, 32, 32)        0

 conv2d_3 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_1 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 16, 16, 64)        18496

 dropout_3 (Dropout)         (None, 16, 16, 64)        0

 conv2d_5 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_6 (Conv2D)           (None, 8, 8, 128)         73856

 dropout_4 (Dropout)         (None, 8, 8, 128)         0

 conv2d_7 (Conv2D)           (None, 8, 8, 128)         147584

 max_pooling2d_3 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dropout_5 (Dropout)         (None, 2048)              0
```

Executing (12m 50s) <cell line: 61> > error_handler() > fit() > error_handler() > __call__() > _call() > __call__() > _call_flat() > call() > quick_execute()

```
dense_2 (Dense)             (None, 1024)              2098176

dropout_6 (Dropout)         (None, 1024)              0

dense_3 (Dense)             (None, 512)               524800

dropout_7 (Dropout)         (None, 512)               0

dense_4 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 663s 423ms/step - loss: 1.8820 - accuracy: 0.3041 - val_loss: 1.6018 - val_accuracy: 0.4080
Epoch 2/5
1563/1563 [==============================] - 621s 397ms/step - loss: 1.5268 - accuracy: 0.4426 - val_loss: 1.4512 - val_accuracy: 0.4782
Epoch 3/5
1563/1563 [==============================] - 638s 408ms/step - loss: 1.3983 - accuracy: 0.4925 - val_loss: 1.3425 - val_accuracy: 0.5189
Epoch 4/5
1563/1563 [==============================] - 616s 394ms/step - loss: 1.3256 - accuracy: 0.5179 - val_loss: 1.3175 - val_accuracy: 0.5193
Epoch 5/5
1563/1563 [==============================] - 596s 382ms/step - loss: 1.2680 - accuracy: 0.5412 - val_loss: 1.2493 - val_accuracy: 0.5434
Accuracy: 54.34%
```

- Performance has decreased from **61.27 to 54.34**.

2. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly.

Answer:

1. Predicted the first four images of the test data using the above model.
2. Converted the predictions to class labels.
3. Converted the actual labels to class labels.
4. Printed the predicted labels and actual labels.

```
#2.Predict the first 4 images of the test data
predictions = model.predict(X_test[:4])
# Convert the predictions to class labels
predicted_labels = numpy.argmax(predictions, axis=1)
# Convert the actual labels to class labels
actual_labels = numpy.argmax(y_test[:4], axis=1)

# Print the predicted and actual labels for the first 4 images
print("Predicted labels:", predicted_labels)
print("Actual labels:   ", actual_labels)
```

```
1/1 [==============================] - 1s 557ms/step
Predicted labels: [3 8 8 8]
Actual labels:    [3 8 8 0]
```

3. Visualize Loss and Accuracy using the history object.

Answer:
Visualized the loss and accuracy using the history object.

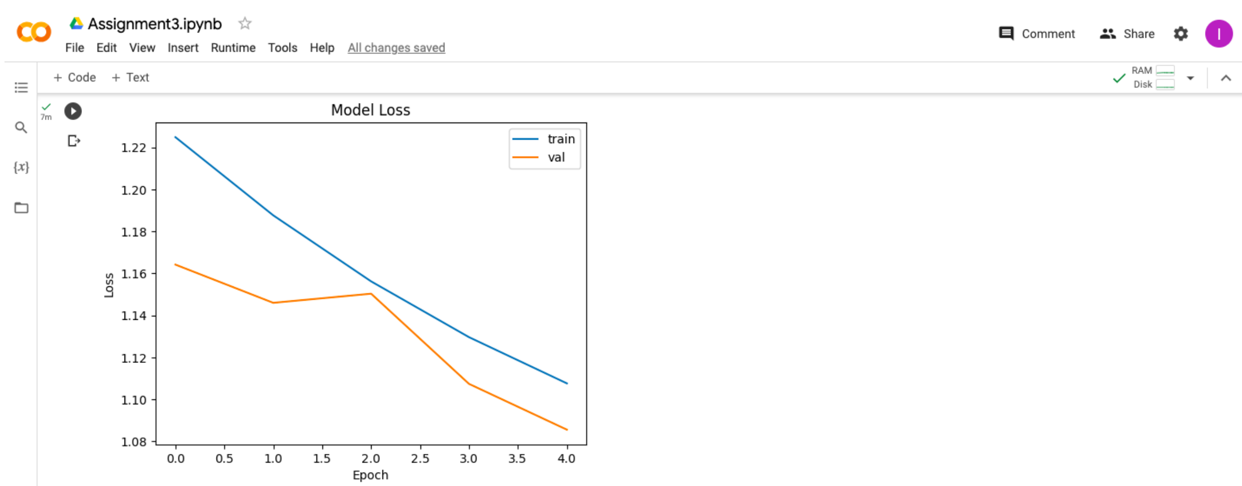
```python
#3.Visualize Loss and Accuracy using the history object
import matplotlib.pyplot as plt
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```

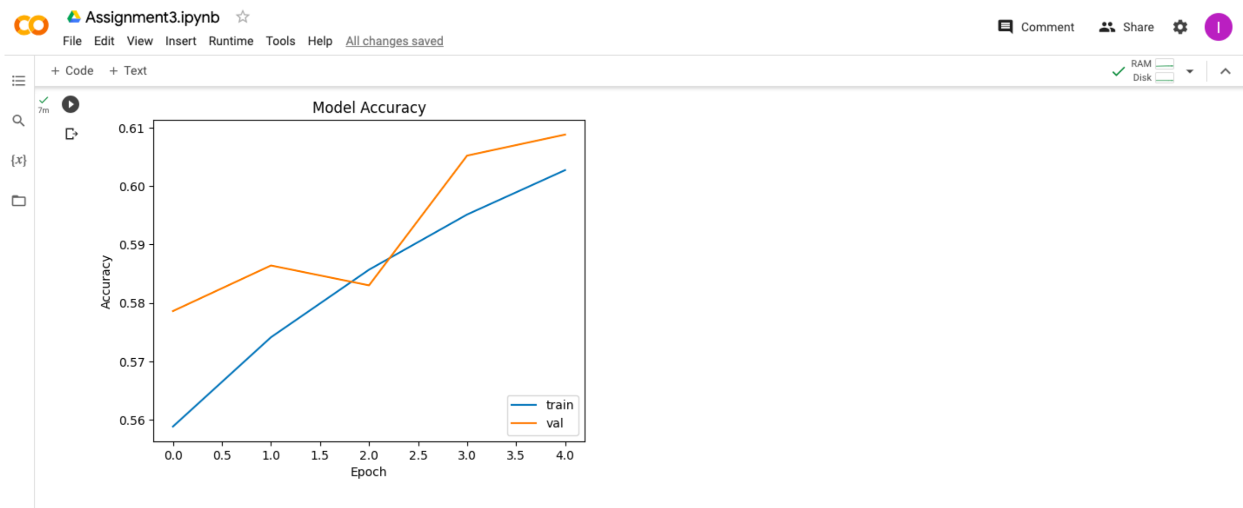
Output:

```
Epoch 1/5
1563/1563 [==============================] - 623s 399ms/step - loss: 1.2249 - accuracy: 0.5588 - val_loss: 1.1642 - val_accuracy: 0.5786
Epoch 2/5
1563/1563 [==============================] - 612s 392ms/step - loss: 1.1876 - accuracy: 0.5741 - val_loss: 1.1460 - val_accuracy: 0.5864
Epoch 3/5
1563/1563 [==============================] - 611s 391ms/step - loss: 1.1562 - accuracy: 0.5857 - val_loss: 1.1503 - val_accuracy: 0.5830
Epoch 4/5
1563/1563 [==============================] - 615s 394ms/step - loss: 1.1296 - accuracy: 0.5951 - val_loss: 1.1074 - val_accuracy: 0.6052
Epoch 5/5
1563/1563 [==============================] - 586s 375ms/step - loss: 1.1076 - accuracy: 0.6027 - val_loss: 1.0856 - val_accuracy: 0.6088
```

Loss Plot:



Accuracy plot:



Github link:
https://github.com/LahariKollipara/NNDL_Assignment3