Akhila Nomula

700759384

NEURAL NETWORK AND DEEP LEARNING ASSIGNMENT-7

GITHUB LINK: https://github.com/aknomula/DL-NN-Assign7.git

RECORDINGLINK: https://drive.google.com/file/d/1PKoyuGA6Y21GqHs0_dZ-Ei2CmrolMiNj/view?usp=drive_link

Use Case Description:

Image Classification with CNN

1. Training the model

2. Evaluating the model

Programming elements:

1. About CNN

2. Hyperparameters of CNN

3. Image classification with CNN

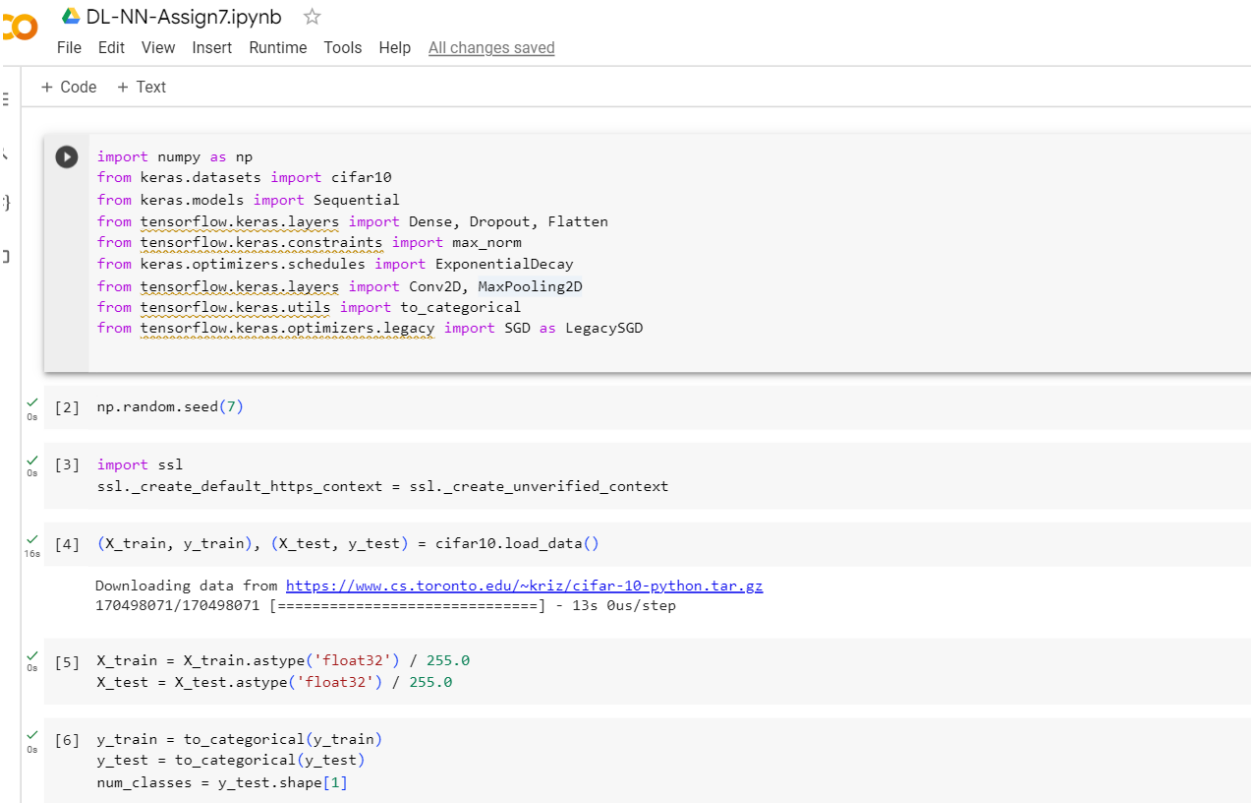
In class programming:

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?



```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2), padding='same'))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
[8] #sgd = SGD(learning_rate=0.01, momentum=0.9, decay=1e-6)


    sgd = LegacySGD(learning_rate=0.01, momentum=0.9, decay=1e-6)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    print(model.summary())
```

```
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 (MaxPooling2  (None, 16, 16, 32)        0
 D)

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 4210090 (16.06 MB)
Trainable params: 4210090 (16.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
[9] epochs = 5
    batch_size = 32
    model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=batch_size)
```

```
Epoch 1/5
1563/1563 [==============================] - 19s 7ms/step - loss: 1.7193 - accuracy: 0.3756 - val_loss: 1.3652 - val_accuracy: 0.5145
Epoch 2/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.3508 - accuracy: 0.5155 - val_loss: 1.2663 - val_accuracy: 0.5463
Epoch 3/5
1563/1563 [==============================] - 9s 6ms/step - loss: 1.1818 - accuracy: 0.5800 - val_loss: 1.1044 - val_accuracy: 0.6070
Epoch 4/5
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0602 - accuracy: 0.6252 - val_loss: 1.0444 - val_accuracy: 0.6300
Epoch 5/5
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9510 - accuracy: 0.6647 - val_loss: 1.0254 - val_accuracy: 0.6429
<keras.src.callbacks.History at 0x7d7f11980550>
```

```
[10] scores = model.evaluate(X_test, y_test, verbose=0)
     print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 64.29%
```

```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.constraints import max_norm
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.optimizers.legacy import SGD as LegacySGD

# Fix random seed for reproducibility
np.random.seed(7)

# 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') / 255.0
X_test = X_test.astype('float32') / 255.0

# One hot encode outputs
y_train = to_categorical(y_train)
y_test = 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=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=max_norm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=max_norm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))
```

```
# Compile model
epochs = 5
learning_rate = 0.01
decay_rate = learning_rate / epochs
sgd = LegacySGD(lr=learning_rate, momentum=0.9, decay=decay_rate, nesterov=False)
#sgd = LegacySGD(learning_rate=0.01, momentum=0.9, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

# Fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)

# Evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1] * 100))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 32, 32, 32)        896

 dropout_14 (Dropout)        (None, 32, 32, 32)        0

 conv2d_15 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_7 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)

 conv2d_16 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_15 (Dropout)        (None, 16, 16, 64)        0

 conv2d_17 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_8 (MaxPoolin  (None, 8, 8, 64)          0
 g2D)

 conv2d_18 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_16 (Dropout)        (None, 8, 8, 128)         0

 conv2d_19 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_9 (MaxPoolin  (None, 4, 4, 128)         0
 g2D)

 flatten_3 (Flatten)         (None, 2048)              0

 dropout_17 (Dropout)        (None, 2048)              0

 dense_8 (Dense)             (None, 1024)              2098176

 dropout_18 (Dropout)        (None, 1024)              0

 dense_9 (Dense)             (None, 512)               524800

 dropout_19 (Dropout)        (None, 512)               0

 dense_10 (Dense)            (None, 10)                5130

=================================================================
Total params: 2915114 (11.12 MB)
Trainable params: 2915114 (11.12 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/legacy/gradient_descent.py:114: UserWarning: The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
None
Epoch 1/5
1563/1563 [==============================] - 16s 9ms/step - loss: 1.9086 - accuracy: 0.2964 - val_loss: 1.5709 - val_accuracy: 0.4328
Epoch 2/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.5088 - accuracy: 0.4494 - val_loss: 1.4358 - val_accuracy: 0.4813
Epoch 3/5
```

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.

```
import numpy
# 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 [==============================] - 0s 20ms/step
Predicted labels: [3 8 8 0]
Actual labels:    [3 8 8 0]
```

3. Visualize Loss and Accuracy using the history object

```python
import matplotlib.pyplot as plt

# 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()
```

[41]