Neural Networks : ICP5

Github link: https://github.com/sailikhit0920/Neural-network-ICP5

Video link:
https://drive.google.com/file/d/1l_nTmafk5EMyMHVeGgCwiHhDXXAWttQP/view?usp=drive_link

Code screenshots:

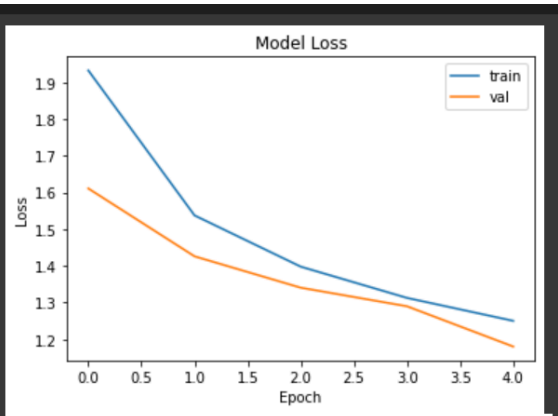
```python
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
from keras.optimizers import SGD

# 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 = 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()
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(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(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 = SGD(lr=learning_rate, momentum=0.9, decay=decay_rate, nesterov=False)
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))
```

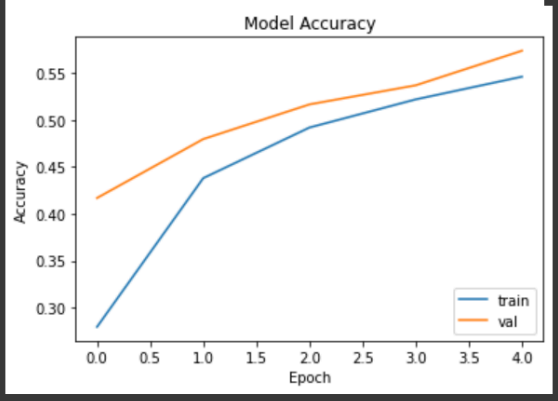
```
conv2d_4 (Conv2D)           (None, 32, 32, 32)        896

dropout_4 (Dropout)         (None, 32, 32, 32)        0

conv2d_5 (Conv2D)           (None, 32, 32, 32)        9248

max_pooling2d_2 (MaxPooling  (None, 16, 16, 32)       0
2D)

conv2d_6 (Conv2D)           (None, 16, 16, 64)        18496

dropout_5 (Dropout)         (None, 16, 16, 64)        0

conv2d_7 (Conv2D)           (None, 16, 16, 64)        36928

max_pooling2d_3 (MaxPooling  (None, 8, 8, 64)         0
2D)

conv2d_8 (Conv2D)           (None, 8, 8, 128)         73856

dropout_6 (Dropout)         (None, 8, 8, 128)         0

conv2d_9 (Conv2D)           (None, 8, 8, 128)         147584

max_pooling2d_4 (MaxPooling  (None, 4, 4, 128)        0
2D)

flatten_2 (Flatten)         (None, 2048)              0

dropout_7 (Dropout)         (None, 2048)              0

dense_4 (Dense)             (None, 1024)              2098176

dropout_8 (Dropout)         (None, 1024)              0

dense_5 (Dense)             (None, 512)               524800

dropout_9 (Dropout)         (None, 512)               0

dense_6 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 15s 9ms/step - loss: 1.9322 - accuracy: 0.2796 - val_loss: 1.6108 - val_accuracy: 0.4168
Epoch 2/5
1563/1563 [==============================] - 13s 9ms/step - loss: 1.5375 - accuracy: 0.4379 - val_loss: 1.4261 - val_accuracy: 0.4795
Epoch 3/5
1563/1563 [==============================] - 13s 9ms/step - loss: 1.3979 - accuracy: 0.4918 - val_loss: 1.3406 - val_accuracy: 0.5164
Epoch 4/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.3128 - accuracy: 0.5217 - val_loss: 1.2901 - val_accuracy: 0.5367
Epoch 5/5
1563/1563 [==============================] - 13s 9ms/step - loss: 1.2504 - accuracy: 0.5459 - val_loss: 1.1804 - val_accuracy: 0.5735
Accuracy: 57.35%
```

```python
# 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 21ms/step
Predicted labels: [3 8 8 8]
Actual labels:    [3 8 8 0]
```

```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()
```


Model Loss
train
val
Loss
Epoch
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Model Accuracy
train
val
Accuracy
Epoch
0.55
0.50
0.45
0.40
0.35
0.30
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0