ICP-7 Neural Networks & Deep Learning

Nirmala Yarlagadda

700733102

GitHub link: https://github.com/niryarjessy22/ICP7-assign.git

Video link:
https://drive.google.com/file/d/1HNRoc8tJV9bO_w7HKWkOiD8KL7ytyo57/view?usp=share_link

ICP ASSIGNMENT 7

In [ ]:

In [3]:
```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.utils import np_utils
```

In [4]:
```
np.random.seed(7)
```

In [5]:
```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [6]:
```
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0
```

In [7]:
```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

In [8]:
```
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), padding='same'))
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'))
```

In [9]:
```
sgd = SGD(learning_rate=0.01, momentum=0.9, decay=1e-6)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
```

```
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)

 flatten_1 (Flatten)         (None, 8192)              0

 dense_2 (Dense)             (None, 512)               4194816

 dropout_3 (Dropout)         (None, 512)               0
```

Programming elements:

1. About CNN
2. Hyperparameters of CNN
3. Image classification with CNN

```
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
None
```

In [10]:
```
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 [==============================] - 199s 126ms/step - loss: 1.7371 - accuracy: 0.3709 - val_loss: 1.4060 - val_accurac
y: 0.4905
Epoch 2/5
1563/1563 [==============================] - 201s 128ms/step - loss: 1.3607 - accuracy: 0.5144 - val_loss: 1.2087 - val_accurac
y: 0.5706
Epoch 3/5
1563/1563 [==============================] - 161s 103ms/step - loss: 1.1974 - accuracy: 0.5733 - val_loss: 1.0890 - val_accurac
y: 0.6101
Epoch 4/5
1563/1563 [==============================] - 124s 80ms/step - loss: 1.0724 - accuracy: 0.6221 - val_loss: 1.0556 - val_accurac
y: 0.6298
Epoch 5/5
1563/1563 [==============================] - 131s 84ms/step - loss: 0.9600 - accuracy: 0.6594 - val_loss: 0.9788 - val_accurac
y: 0.6557
```

Out[10]:
```
<keras.callbacks.History at 0x1afacee63e0>
```

In [12]:
```
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 65.57%
```

In [13]:
```
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
```

```
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))
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 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
```

Accuracy : 65.57%

Use Case Description:

Image Classification with CNN

1. Training the model
2. Evaluating the model

```
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))
```

```
Model: "sequential_2"
```

Did the performance change?

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.

```
 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 [==============================] - 238s 151ms/step - loss: 1.8680 - accuracy: 0.3060 - val_loss: 1.6120 - val_accurac
y: 0.4154
Epoch 2/5
1563/1563 [==============================] - 194s 124ms/step - loss: 1.5025 - accuracy: 0.4514 - val_loss: 1.3628 - val_accurac
y: 0.5015
Epoch 3/5
1563/1563 [==============================] - 234s 150ms/step - loss: 1.3612 - accuracy: 0.5046 - val_loss: 1.2554 - val_accurac
y: 0.5444
Epoch 4/5
1563/1563 [==============================] - 201s 129ms/step - loss: 1.2777 - accuracy: 0.5390 - val_loss: 1.2120 - val_accurac
y: 0.5626
Epoch 5/5
1563/1563 [==============================] - 218s 139ms/step - loss: 1.2115 - accuracy: 0.5625 - val_loss: 1.1797 - val_accurac
y: 0.5763
Accuracy: 57.63%
```

In [14]:

```
##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.


# 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 349ms/step
Predicted labels: [3 8 8 8]
Actual labels:    [3 8 8 0]
```

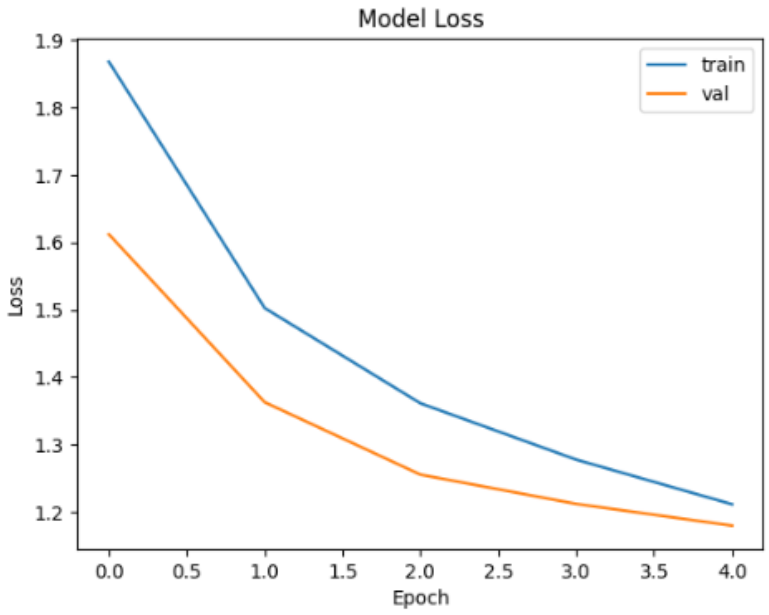
3. Visualize Loss and Accuracy using the history object

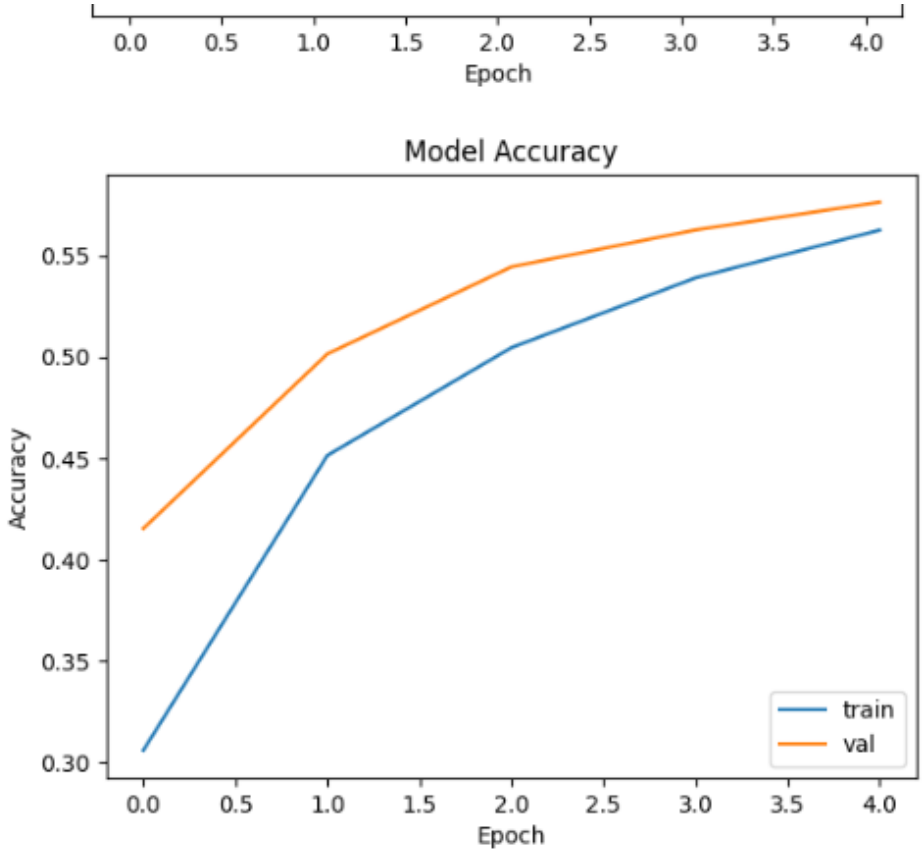
```
In [15]: ##3. Visualize Loss and Accuracy using the history object


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()
```




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