NEURAL NETWORKS AND DEEP LEARNING – ICP4

N. Manesh
700756918

GitHub link: https://github.com/Manesh1712/ICP4/tree/main

Video Link:
https://drive.google.com/file/d/17uTXb1U4hHQpOcM_k248wcr9vmVpEQ6o/view?usp=sharing

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

#Neural Networks & Deep Learning - ICP-3 Given code with rectified bugs

[11]:
```python
# Simple CNN model for CIFAR-10
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.constraints import MaxNorm
from keras.optimizers.legacy import SGD
from keras.utils import to_categorical
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
np.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 = 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=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 = 25
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))
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_12 (Conv2D)          (None, 32, 32, 32)        896       
                                                                 
 dropout_12 (Dropout)        (None, 32, 32, 32)        0         
                                                                 
 conv2d_13 (Conv2D)          (None, 32, 32, 32)        9248      
                                                                 
 max_pooling2d_6 (MaxPoolin  (None, 16, 16, 32)        0         
 g2D)                                                            
                                                                 
 flatten_2 (Flatten)         (None, 8192)              0         
                                                                 
 dense_6 (Dense)             (None, 512)               4194816   
```

```
1563/1563 [==============================] - 9s 6ms/step - loss: 0.8373 - accuracy: 0.7033 - val_loss: 0.9520 - val_accuracy: 0.6682
Epoch 9/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7805 - accuracy: 0.7234 - val_loss: 0.9462 - val_accuracy: 0.6680
Epoch 10/25
1563/1563 [==============================] - 10s 7ms/step - loss: 0.7269 - accuracy: 0.7421 - val_loss: 0.9450 - val_accuracy: 0.6703
Epoch 11/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.6791 - accuracy: 0.7596 - val_loss: 0.9295 - val_accuracy: 0.6819
Epoch 12/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6302 - accuracy: 0.7752 - val_loss: 0.9197 - val_accuracy: 0.6843
Epoch 13/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5899 - accuracy: 0.7904 - val_loss: 0.9236 - val_accuracy: 0.6880
Epoch 14/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5589 - accuracy: 0.8027 - val_loss: 0.9225 - val_accuracy: 0.6940
Epoch 15/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5148 - accuracy: 0.8186 - val_loss: 0.9430 - val_accuracy: 0.6933
Epoch 16/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4858 - accuracy: 0.8281 - val_loss: 0.9579 - val_accuracy: 0.6857
Epoch 17/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4550 - accuracy: 0.8381 - val_loss: 0.9405 - val_accuracy: 0.7005
Epoch 18/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4325 - accuracy: 0.8464 - val_loss: 0.9630 - val_accuracy: 0.6967
Epoch 19/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4129 - accuracy: 0.8541 - val_loss: 0.9671 - val_accuracy: 0.6978
Epoch 20/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3862 - accuracy: 0.8639 - val_loss: 0.9628 - val_accuracy: 0.7012
Epoch 21/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3603 - accuracy: 0.8731 - val_loss: 0.9678 - val_accuracy: 0.7045
Epoch 22/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3459 - accuracy: 0.8777 - val_loss: 0.9869 - val_accuracy: 0.7043
Epoch 23/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3304 - accuracy: 0.8829 - val_loss: 0.9980 - val_accuracy: 0.7011
Epoch 24/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3116 - accuracy: 0.8901 - val_loss: 1.0140 - val_accuracy: 0.7031
Epoch 25/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3029 - accuracy: 0.8928 - val_loss: 0.9989 - val_accuracy: 0.7047
Accuracy: 70.47%
```

Note: Accuracy of with initilal model is 70.47%.

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.
3. Visualize Loss and Accuracy using the history object.

[3]:

```python
# Simple CNN model for CIFAR-10
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.constraints import MaxNorm
from keras.optimizers.legacy import SGD
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# fix random seed for reproducibility
seed = 7
np.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 /= 255.0
X_test /= 255.0

# one hot encode outputs
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
num_classes = y_test.shape[1]
```

```python
# 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 = 25
lrate = 0.01
decay = lrate / epochs
sgd = SGD(learning_rate=lrate, momentum=0.9, decay=decay, 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)
```

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

# Predict the first 4 images
predictions = model.predict(X_test[:4])
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = np.argmax(y_test[:4], axis=1)
print("Predicted classes:", predicted_classes)
print("Actual classes:   ", actual_classes)

# Check if predictions are correct
correct_predictions = predicted_classes == actual_classes
print("Correct predictions:", correct_predictions)

fig, axes = plt.subplots(1, 4, figsize=(10, 2.5))
for i in range(4):
    axes[i].imshow(X_test[i])
    axes[i].set_title(f"Pred: {predicted_classes[i]}\nActual: {actual_classes[i]}")
    axes[i].axis('off')
plt.show()


# Visualize Loss and Accuracy
plt.figure(figsize=(12, 4))
```

```python
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```

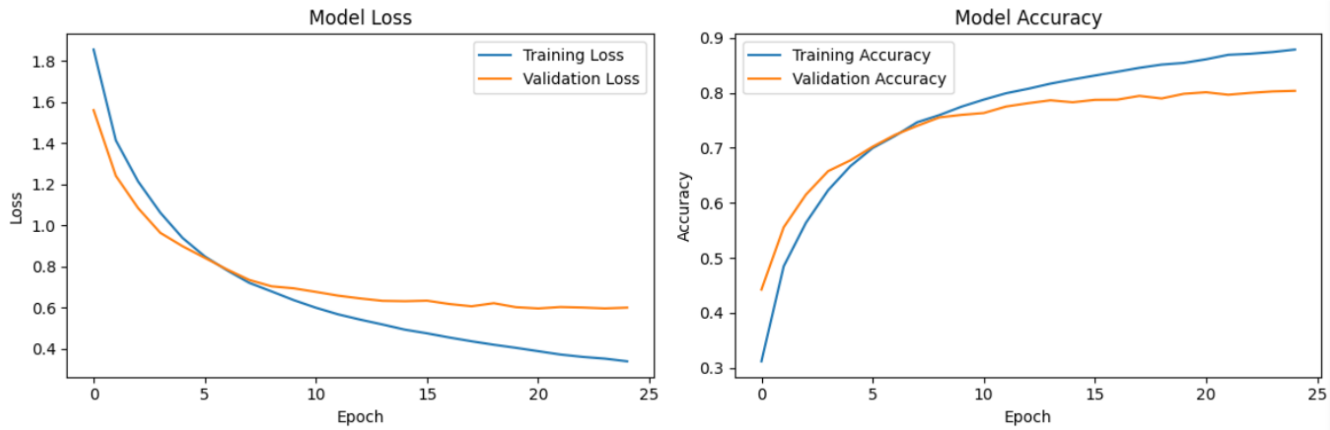
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 32, 32, 32)        896

 dropout_6 (Dropout)         (None, 32, 32, 32)        0

 conv2d_7 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d_3 (MaxPoolin  (None, 16, 16, 32)        0
 g2D)
```

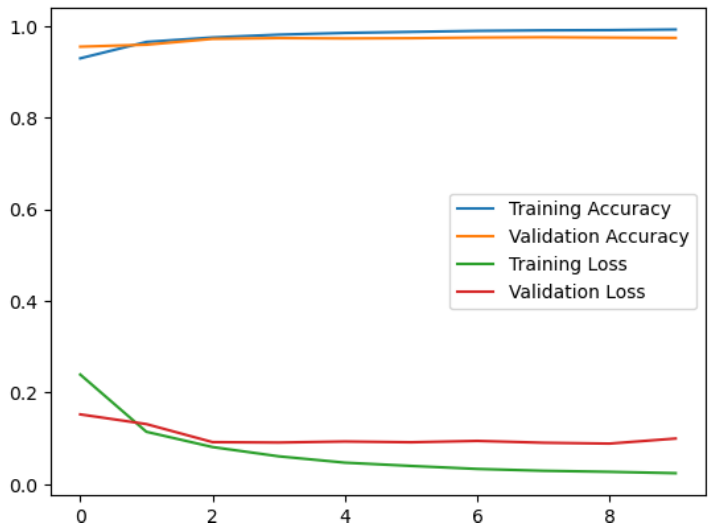
```
Epoch 22/25
1563/1563 [==============================] - 12s 8ms/step - loss: 0.3732 - accuracy: 0.8691 - val_loss: 0.6040 - val_accuracy: 0.7965
Epoch 23/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3617 - accuracy: 0.8710 - val_loss: 0.6015 - val_accuracy: 0.7999
Epoch 24/25
1563/1563 [==============================] - 12s 8ms/step - loss: 0.3533 - accuracy: 0.8741 - val_loss: 0.5972 - val_accuracy: 0.8025
Epoch 25/25
1563/1563 [==============================] - 13s 8ms/step - loss: 0.3403 - accuracy: 0.8786 - val_loss: 0.6010 - val_accuracy: 0.8036
Accuracy: 80.36%
1/1 [==============================] - 0s 370ms/step
Predicted classes: [3 8 8 0]
Actual classes:    [3 8 8 0]
Correct predictions: [ True  True  True  True]
```



Note: Changing the model architecture has boosted model accuracy to 80.36%.

There can be more imporvemnt by trying different hyperperameters.

y: 0.9759



1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
Training Accuracy
Validation Accuracy
Training Loss
Validation Loss