Spring 2025: Neural Networks & Deep Learning – ICP -5

Assignment – Week6

Name: Veera Manikanta Kumar Allada

Student ID: 700756934

Github Link: https://github.com/manialladа9/Neural-Networks-deep-Learning

Video Link:
https://drive.google.com/file/d/1J6D9B4P0O9G7d_wSNYSeL3fyH8rWZCx8/view?usp=drive_link

1)

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

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Normalize images to range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0

# One-hot encode labels
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(32,32,3)),
    Dropout(0.2),
    Conv2D(32, (3,3), activation='relu'),
    MaxPooling2D((2,2),padding='same'),

    Conv2D(64, (3,3), activation='relu'),
    Dropout(0.2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2),padding='same'),

    Conv2D(128, (3,3), activation='relu'),
    Dropout(0.2),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D((2,2),padding='same'),
```

```python
        Flatten(),
        Dropout(0.2),
        Dense(1024, activation='relu'),
        Dropout(0.2),
        Dense(512, activation='relu'),
        Dropout(0.2),
        Dense(10, activation='softmax')
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

validation_data: Any

```python
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=20, batch_size=64)
# Evaluate on test set
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc*100:.2f}%")
print(f"Test Loss: {test_loss:.4f}")
```

```
Epoch 11/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 193s 247ms/step - accuracy: 0.8031 - loss: 0.5783 - val_accuracy: 0.7562 - val_loss: 0.7513
Epoch 12/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 198s 243ms/step - accuracy: 0.8164 - loss: 0.5420 - val_accuracy: 0.7575 - val_loss: 0.7652
Epoch 13/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 204s 245ms/step - accuracy: 0.8177 - loss: 0.5307 - val_accuracy: 0.7614 - val_loss: 0.7545
Epoch 14/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 201s 244ms/step - accuracy: 0.8158 - loss: 0.5339 - val_accuracy: 0.7626 - val_loss: 0.7491
Epoch 15/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 201s 243ms/step - accuracy: 0.8210 - loss: 0.5209 - val_accuracy: 0.7649 - val_loss: 0.7297
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 191s 245ms/step - accuracy: 0.8231 - loss: 0.5149 - val_accuracy: 0.7599 - val_loss: 0.7541
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 191s 244ms/step - accuracy: 0.8260 - loss: 0.5111 - val_accuracy: 0.7610 - val_loss: 0.7544
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 203s 246ms/step - accuracy: 0.8213 - loss: 0.5195 - val_accuracy: 0.7627 - val_loss: 0.7326
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 194s 248ms/step - accuracy: 0.8240 - loss: 0.5092 - val_accuracy: 0.7618 - val_loss: 0.7320
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 199s 245ms/step - accuracy: 0.8247 - loss: 0.5059 - val_accuracy: 0.7684 - val_loss: 0.7300
313/313 ━━━━━━━━━━━━━━━━━━━━ 12s 39ms/step - accuracy: 0.7752 - loss: 0.7117
Test Accuracy: 76.84%
Test Loss: 0.7300
```

2)

[10]

```python
# Get first 4 test images
num_images = 4
predictions = model.predict(x_test[:num_images])
predicted_labels = np.argmax(predictions, axis=1)
actual_labels = np.argmax(y_test[:num_images], axis=1)

# Print predictions vs actual labels
print("Predictions vs Actual Labels:")
for i in range(num_images):
    print(f"Image {i+1}: Predicted={predicted_labels[i]}, Actual={actual_labels[i]}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 288ms/step
Predictions vs Actual Labels:
Image 1: Predicted=3, Actual=3
Image 2: Predicted=8, Actual=8
Image 3: Predicted=8, Actual=8
Image 4: Predicted=0, Actual=0
```

```
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')
plt.show()
```