Spring 2024: CS5720 Neural Networks & Deep Learning

Assignment-5

NAME: Lakkireddy Sriram Reddy

STUDENT ID:700758340

Github link: https://github.com/sriram7040/Neural-network-and-deep-learning/upload/main/WEEK6

Video link: https://drive.google.com/file/d/1DfcmcgDHQCQ5pE0GuYtWrHNnr1bYtPaJ/view?usp=drive_link

Use Case Description:

Image Classification with CNN

1. Training the model

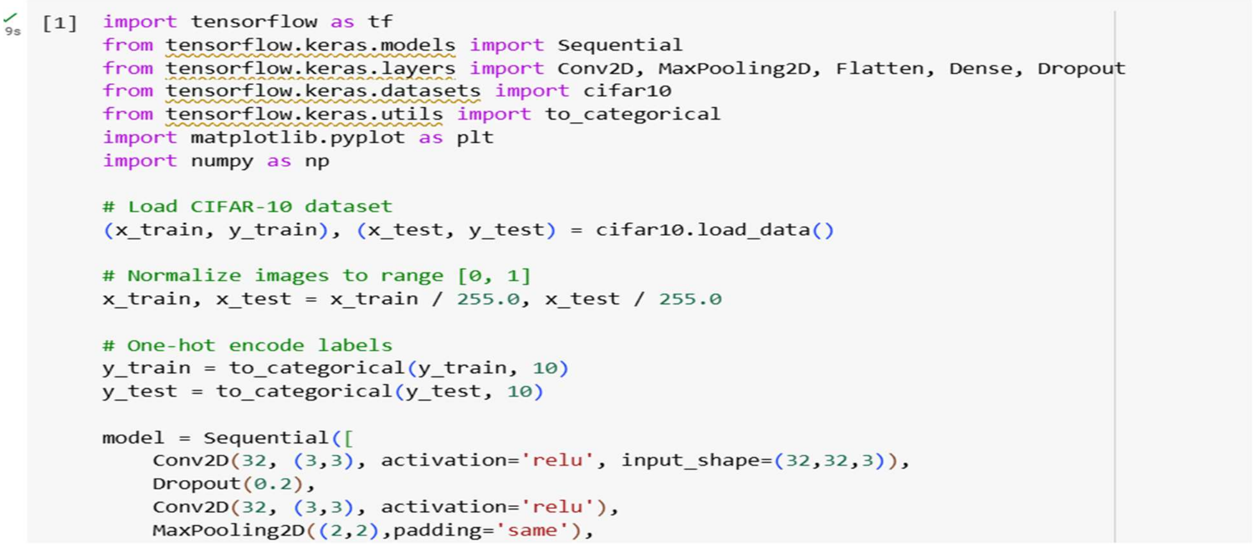
2. Evaluating the model

Programming elements:

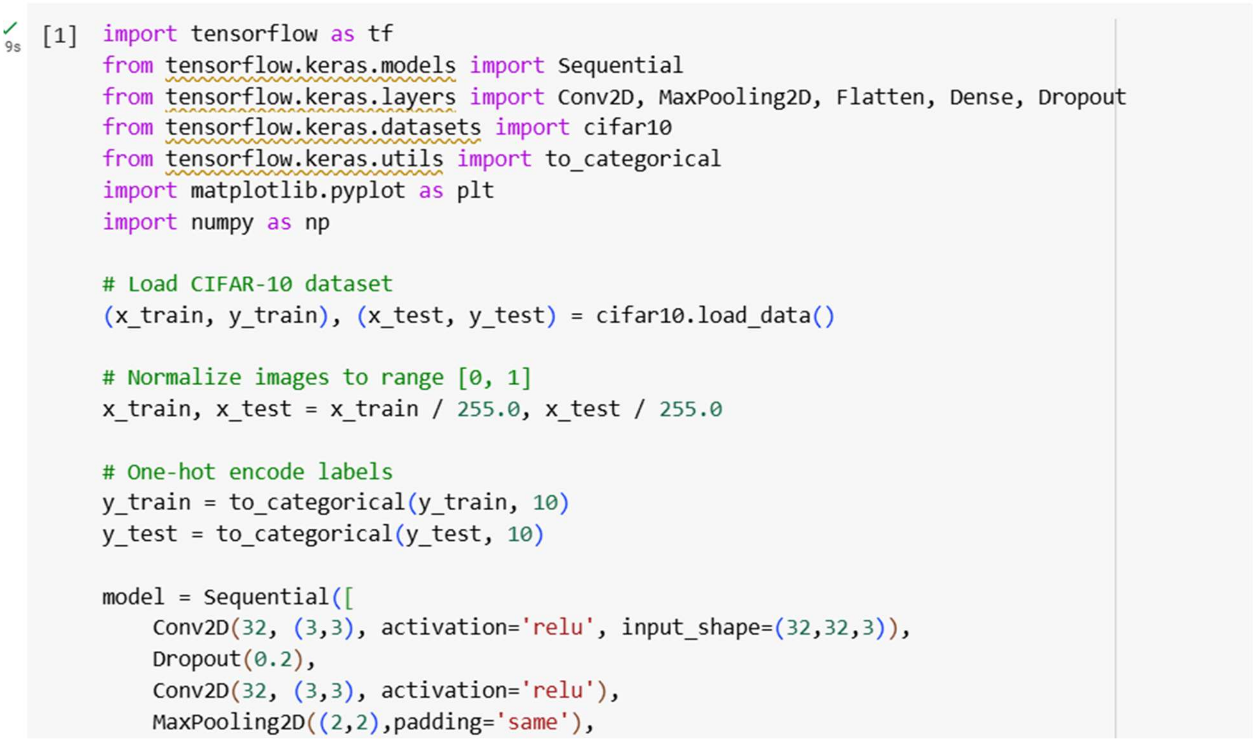
1. About CNN

2. Hyperparameters of CNN

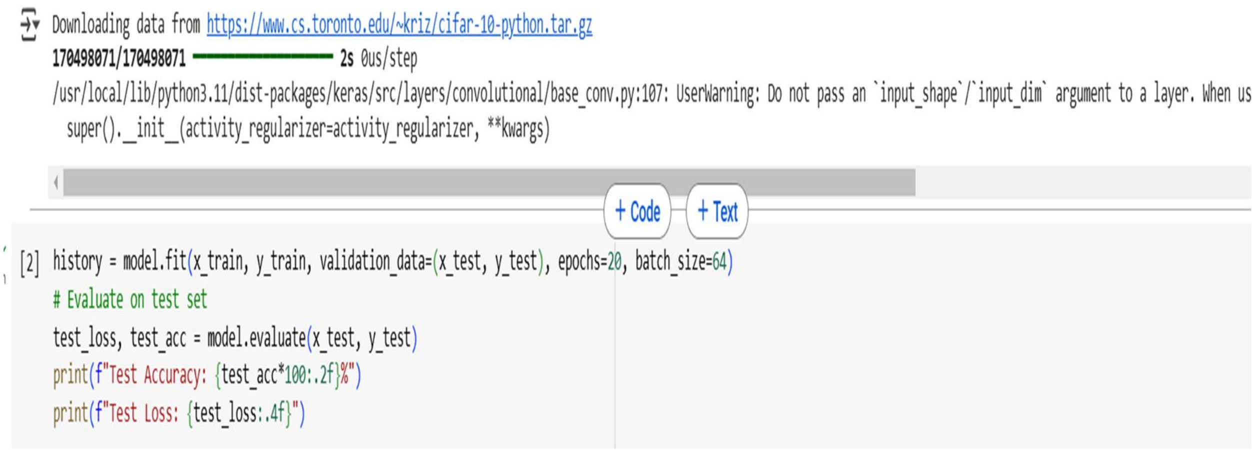
3. Image classification with CNN



```
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'),
```

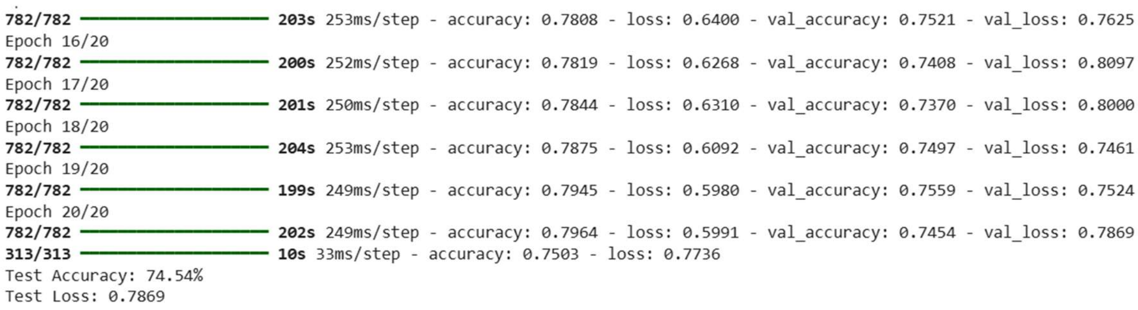


```
[1] import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
    from tensorflow.keras.datasets import cifar10
    from tensorflow.keras.utils import to_categorical
    import matplotlib.pyplot 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'),
```

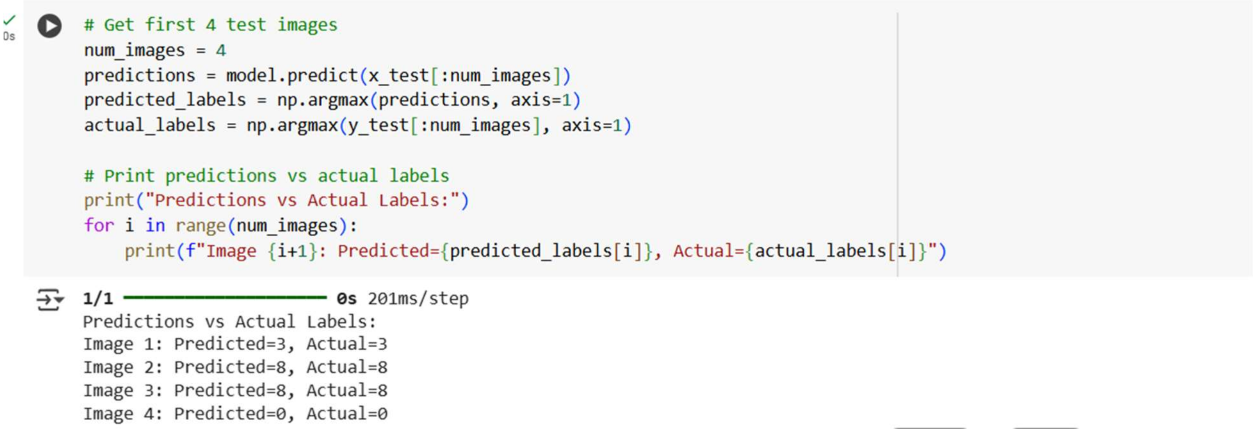


```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When us
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[2] 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}")
```

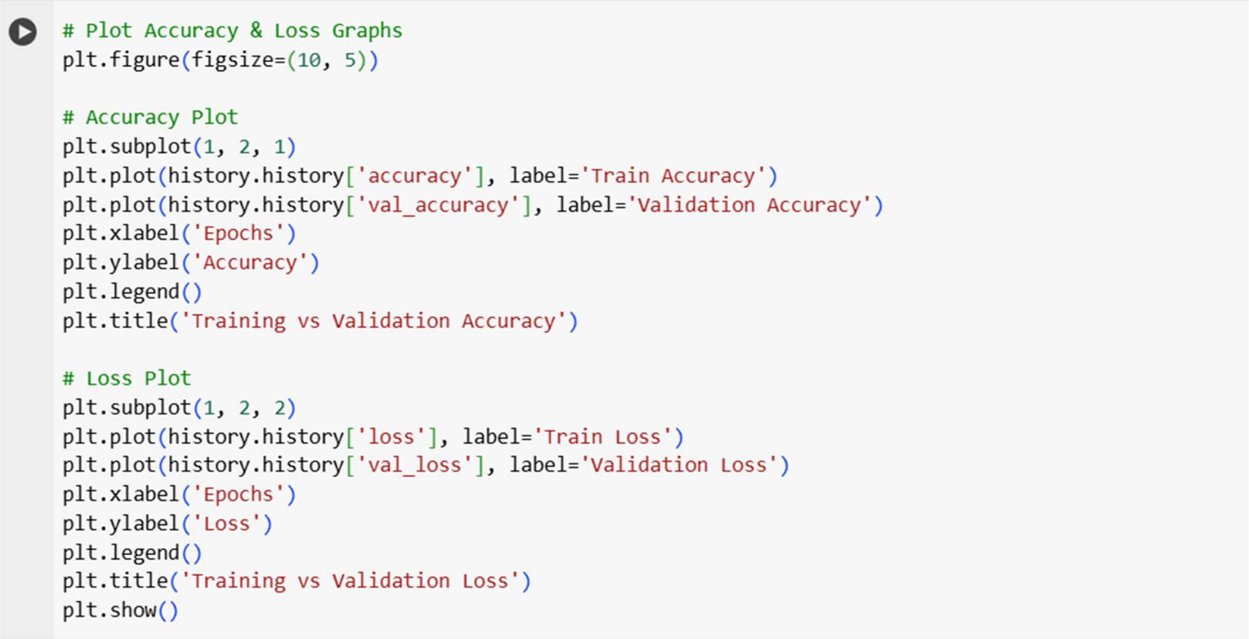


```
782/782 ━━━━━━━━━━━━━━━━━━━━ 203s 253ms/step - accuracy: 0.7808 - loss: 0.6400 - val_accuracy: 0.7521 - val_loss: 0.7625
Epoch 16/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 200s 252ms/step - accuracy: 0.7819 - loss: 0.6268 - val_accuracy: 0.7408 - val_loss: 0.8097
Epoch 17/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 201s 250ms/step - accuracy: 0.7844 - loss: 0.6310 - val_accuracy: 0.7370 - val_loss: 0.8000
Epoch 18/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 204s 253ms/step - accuracy: 0.7875 - loss: 0.6092 - val_accuracy: 0.7497 - val_loss: 0.7461
Epoch 19/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 199s 249ms/step - accuracy: 0.7945 - loss: 0.5980 - val_accuracy: 0.7559 - val_loss: 0.7524
Epoch 20/20
782/782 ━━━━━━━━━━━━━━━━━━━━ 202s 249ms/step - accuracy: 0.7964 - loss: 0.5991 - val_accuracy: 0.7454 - val_loss: 0.7869
313/313 ━━━━━━━━━━━━━━━━━━━━ 10s 33ms/step - accuracy: 0.7503 - loss: 0.7736
Test Accuracy: 74.54%
Test Loss: 0.7869
```

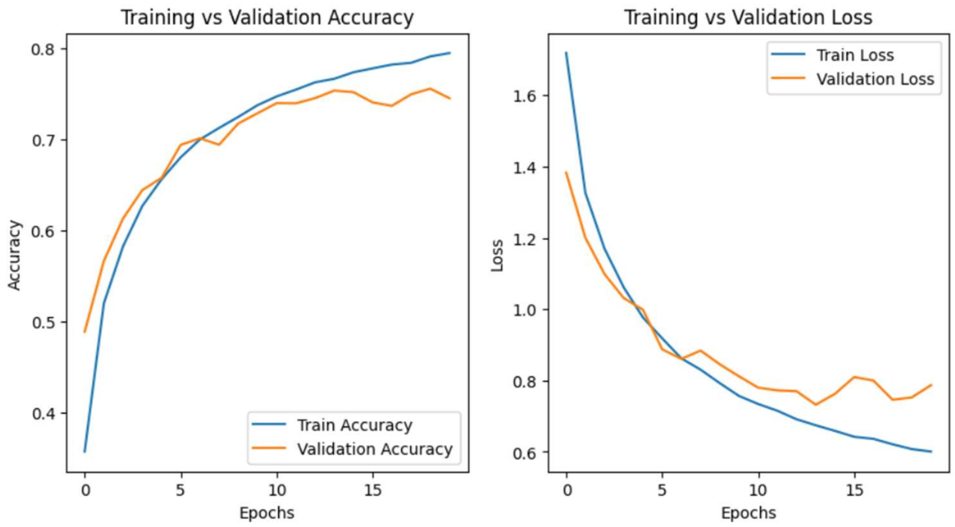


```
# Get first 4 test images
num_images = 4
predictions = model.predict(x_test[:num_images])
predicted_labels = np.argmax(predictions, axis=1)
actual_labels = np.argmax(y_test[:num_images], axis=1)

# 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 201ms/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
```



```
# Plot Accuracy & Loss Graphs
plt.figure(figsize=(10, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training vs Validation Accuracy')

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')
plt.show()
```


Training vs Validation Accuracy
Accuracy
Epochs
Train Accuracy
Validation Accuracy
Training vs Validation Loss
Loss
Epochs
Train Loss
Validation Loss