

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

Assignment -5

Name: Vanitha Chintalapudi

Student ID: 700756782

Github Link: <https://github.com/VanithaChintalapudi10/Neural-network-deep-learning>

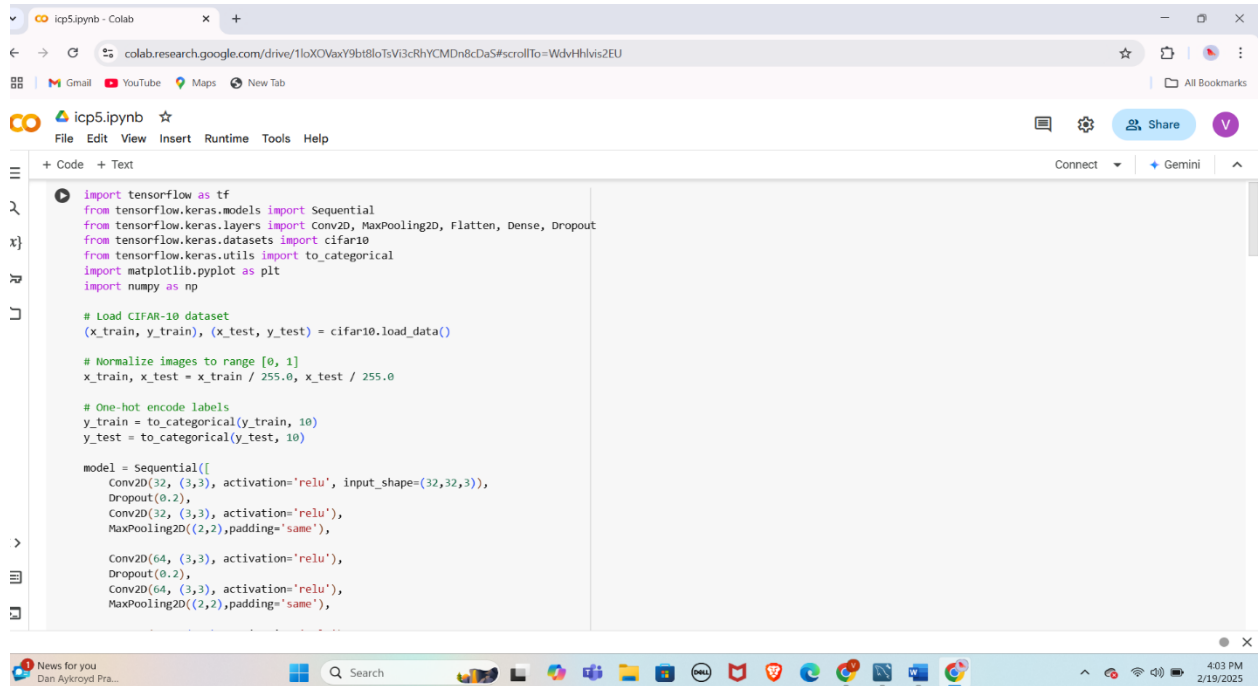
Video Link:

https://drive.google.com/file/d/1Ow8l1IBJn2_Lyb9KH_ZBlrE3j6wQubW/view?usp=drive_link

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

Output:



The screenshot shows a Google Colab notebook interface. The browser address bar displays a Google Drive link. The notebook's toolbar includes options for File, Edit, View, Insert, Runtime, Tools, and Help. The code editor contains the following Python code:

```
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'),
```

The bottom of the image shows a Windows taskbar with various application icons and a system clock indicating 4:03 PM on 2/19/2025.

```
icp5.ipynb - Colab
colab.research.google.com/drive/1loXOVaxY9b18loTsVi3cRhYCMDn8cDaS#scrollTo=WdVHhVw2EU
Gmail YouTube Maps New Tab
icp5.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Connect Gemini
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'),

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'])

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 4s 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 using Sequential:
super().__init__(activity_regularizer=activity_regularizer, **kwargs)

[ ] history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=15, batch_size=64)
```

```
icp5.ipynb - Colab
colab.research.google.com/drive/1loXOVaxY9b18loTsVi3cRhYCMDn8cDaS#scrollTo=WdVHhVw2EU
Gmail YouTube Maps New Tab
icp5.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
Connect Gemini
history = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=15, 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 1/15
782/782 212s 260ms/step - accuracy: 0.2536 - loss: 1.9500 - val_accuracy: 0.4838 - val_loss: 1.4273
Epoch 2/15
782/782 266s 271ms/step - accuracy: 0.5081 - loss: 1.3640 - val_accuracy: 0.5678 - val_loss: 1.1915
Epoch 3/15
782/782 259s 268ms/step - accuracy: 0.5793 - loss: 1.1846 - val_accuracy: 0.6217 - val_loss: 1.0587
Epoch 4/15
782/782 265s 272ms/step - accuracy: 0.6280 - loss: 1.0611 - val_accuracy: 0.6465 - val_loss: 1.0411
Epoch 5/15
782/782 245s 250ms/step - accuracy: 0.6601 - loss: 0.9730 - val_accuracy: 0.6830 - val_loss: 0.9069
Epoch 6/15
782/782 200s 248ms/step - accuracy: 0.6836 - loss: 0.9043 - val_accuracy: 0.6945 - val_loss: 0.8917
Epoch 7/15
782/782 210s 268ms/step - accuracy: 0.7041 - loss: 0.8471 - val_accuracy: 0.7189 - val_loss: 0.8229
Epoch 8/15
782/782 209s 267ms/step - accuracy: 0.7189 - loss: 0.8143 - val_accuracy: 0.6909 - val_loss: 0.8816
Epoch 9/15
782/782 212s 271ms/step - accuracy: 0.7294 - loss: 0.7805 - val_accuracy: 0.7256 - val_loss: 0.8041
Epoch 10/15
782/782 260s 269ms/step - accuracy: 0.7392 - loss: 0.7450 - val_accuracy: 0.7363 - val_loss: 0.7738
Epoch 11/15
782/782 257s 262ms/step - accuracy: 0.7513 - loss: 0.7156 - val_accuracy: 0.7440 - val_loss: 0.7659
Epoch 12/15
```

The screenshot shows a Google Colab notebook interface. The top toolbar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the toolbar, a progress bar displays training metrics for epochs 12 through 15, with a final test accuracy of 74.58% and test loss of 0.7550. The code cell contains the following Python code:

```
# 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]}")
```

The output of the code cell shows the predictions for the first four test images:

```
1/1 0s 249ms/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
```

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.

This screenshot is identical to the one above, showing the same Google Colab notebook interface, training progress bar, and code cell for testing the model. The code and its output are the same as in the first screenshot.

3. Visualize Loss and Accuracy using the history object

Output:

