**CSE425**

**Assignment 2**

**Muhammad Nayeem Mubasshirul Haque**

**ID: 19101115**

**Section:03**

**Custom Convolutional Neural Network Implementation and Comparative Analysis**

**Introduction**

In this report, the implementation and analysis of a custom Convolutional Neural Network (CNN) architecture for image classification tasks using the CIFAR-10 dataset are done. The primary objective is to design, train, and rigorously evaluate this custom CNN model using various configurations and techniques, focusing on its performance characteristics and limitations.

**Dataset**

The CIFAR-10 dataset, consisting of 60,000 32x32 color images in 10 classes, was used for training and testing the custom CNN models. The dataset was preprocessed to normalize pixel values and split into training and validation sets. The dataset further distinguishes itself by categorizing these images into precisely 10 classes, spanning a diverse spectrum of objects and concepts.

The CIFAR-10 dataset contains a wide variety of objects and concepts, classified into 10 distinct categories:

1. Airplanes

2. Automobiles

3. Birds

4. Cats

5. Deer

6. Dogs

7. Frogs

8. Horses

9. Ships

10. Trucks

**Model Architecture**

Constructed using the TensorFlow and Keras libraries, the custom CNN architecture comprises a stratified sequence of convolutional and pooling layers. This foundational structure is subsequently followed by fully connected layers and a final softmax output layer, facilitating effective classification.

**Implementing Traditional CNNs**

The initial phase involves the development of a traditional CNN architecture featuring alternating convolutional and pooling layers. The model achieved a commendable validation accuracy of approximately 70%. The model configuration and results are shown below:

Filters: 32

Kernel Size: (3, 3)

Pooling: MaxPooling (2, 2)

Activation Function: ReLU

Training Accuracy: ~79%

Validation Accuracy: ~70%

**CODE:**

import tensorflow as tf

from tensorflow.keras import layers, models, datasets

import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

# Define the CNN architecture

model = models.Sequential([

    layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),

    layers.Flatten(),

    layers.Dense(64, activation='relu'),

    layers.Dense(10, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

              loss='sparse\_categorical\_crossentropy',

              metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=10,

                    validation\_data=(x\_test, y\_test))

# Evaluate the model

test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

print("\nTest accuracy:", test\_acc)

# Plot training and validation accuracy over epochs

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

plt.xlabel('Epoch')

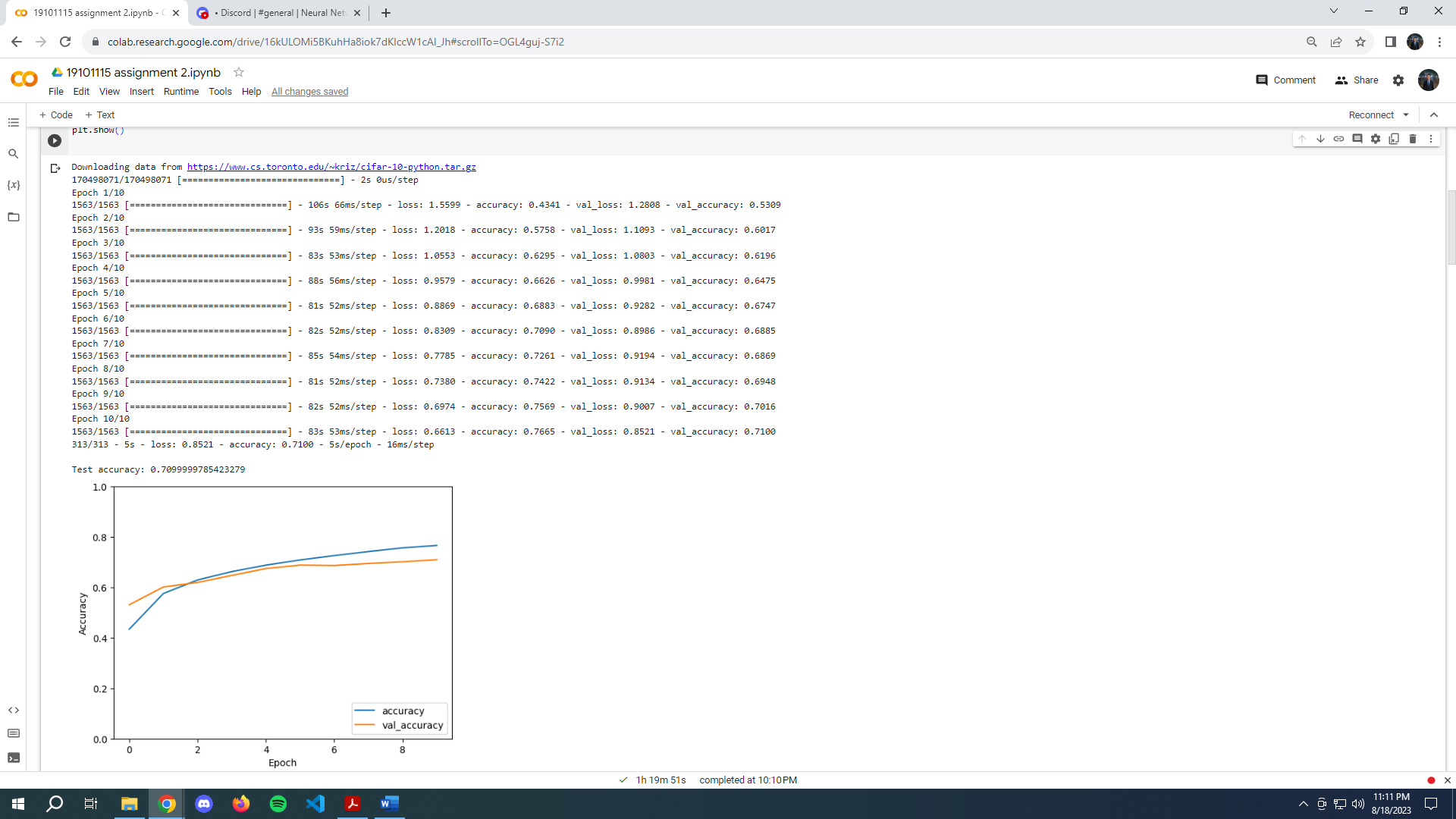
plt.ylabel('Accuracy')

plt.ylim([0, 1])

plt.legend(loc='lower right')

plt.show()

**Output:**



**Activation Functions**

For the next step, we explored the impact of different activation functions on the CNN's performance. We replaced the ReLU activation function in some convolutional layers with ELU and Sigmoid activations. The models were trained using the same dataset and other configurations. The results are as follows:

ReLU Activation

Training Accuracy: ~79%

Validation Accuracy: ~71%

ELU Activation

Training Accuracy: ~82%

Validation Accuracy: ~70%

Sigmoid Activation

Training Accuracy: ~10%

Validation Accuracy: ~10%

CODE:

import tensorflow as tf

from tensorflow.keras import layers, models, datasets

import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

activation\_functions = ['relu', 'elu', 'sigmoid']  # List of activation functions

for activation\_function in activation\_functions:

    print(f"Training with activation function: {activation\_function}")

    # Define the CNN architecture with the chosen activation function

    model = models.Sequential([

        layers.Conv2D(32, (3, 3), activation=activation\_function, input\_shape=(32, 32, 3)),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation=activation\_function),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation=activation\_function),

        layers.Flatten(),

        layers.Dense(64, activation=activation\_function),

        layers.Dense(10, activation='softmax')

    ])

    # Compile the model

    model.compile(optimizer='adam',

                  loss='sparse\_categorical\_crossentropy',

                  metrics=['accuracy'])

    # Train the model

    history = model.fit(x\_train, y\_train, epochs=10,

                        validation\_data=(x\_test, y\_test))

    # Evaluate the model

    test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

    print("\nTest accuracy:", test\_acc)

    # Plot training and validation accuracy over epochs

    plt.plot(history.history['accuracy'], label='accuracy')

    plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

    plt.xlabel('Epoch')

    plt.ylabel('Accuracy')

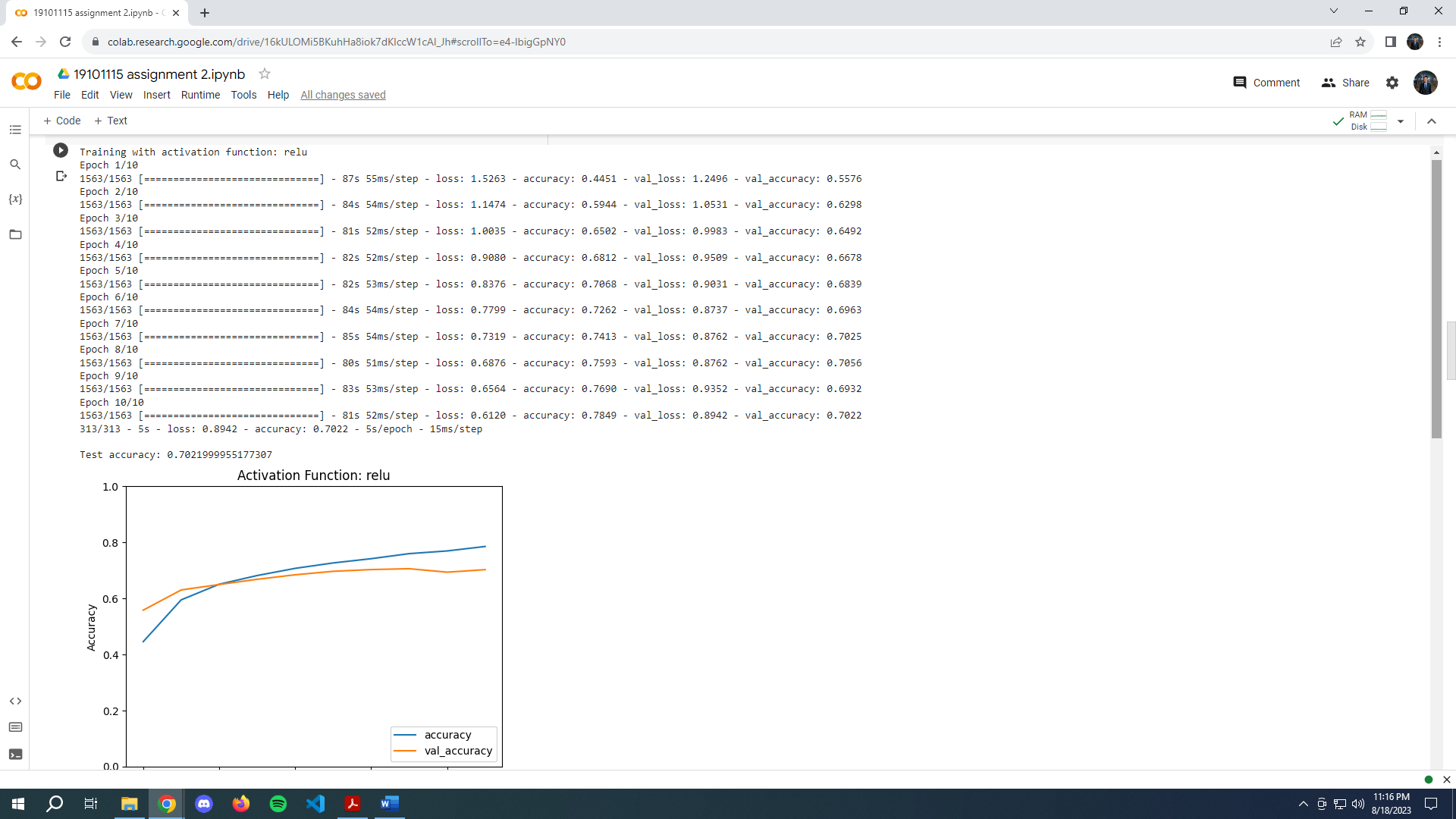
    plt.ylim([0, 1])

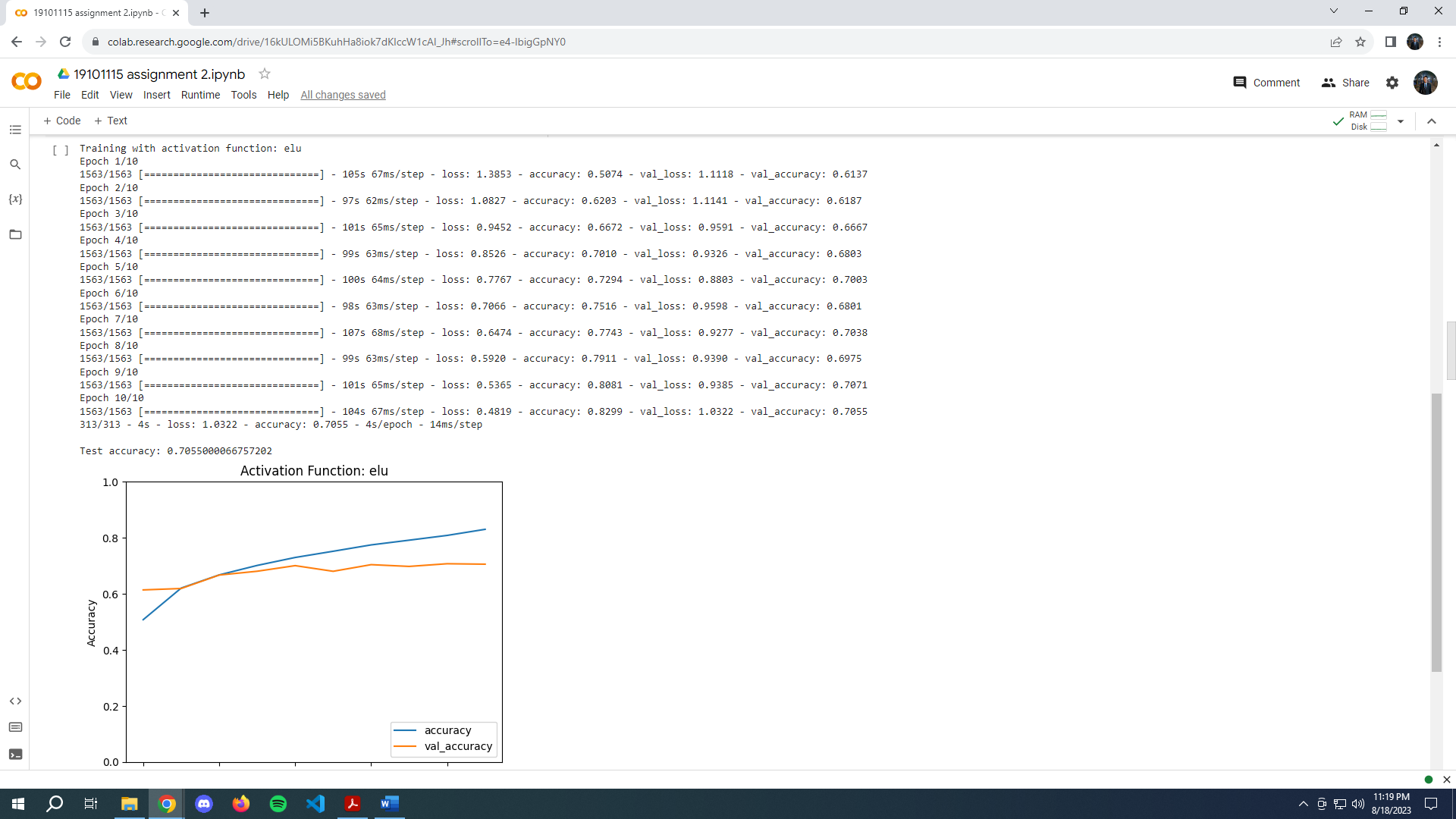
    plt.legend(loc='lower right')

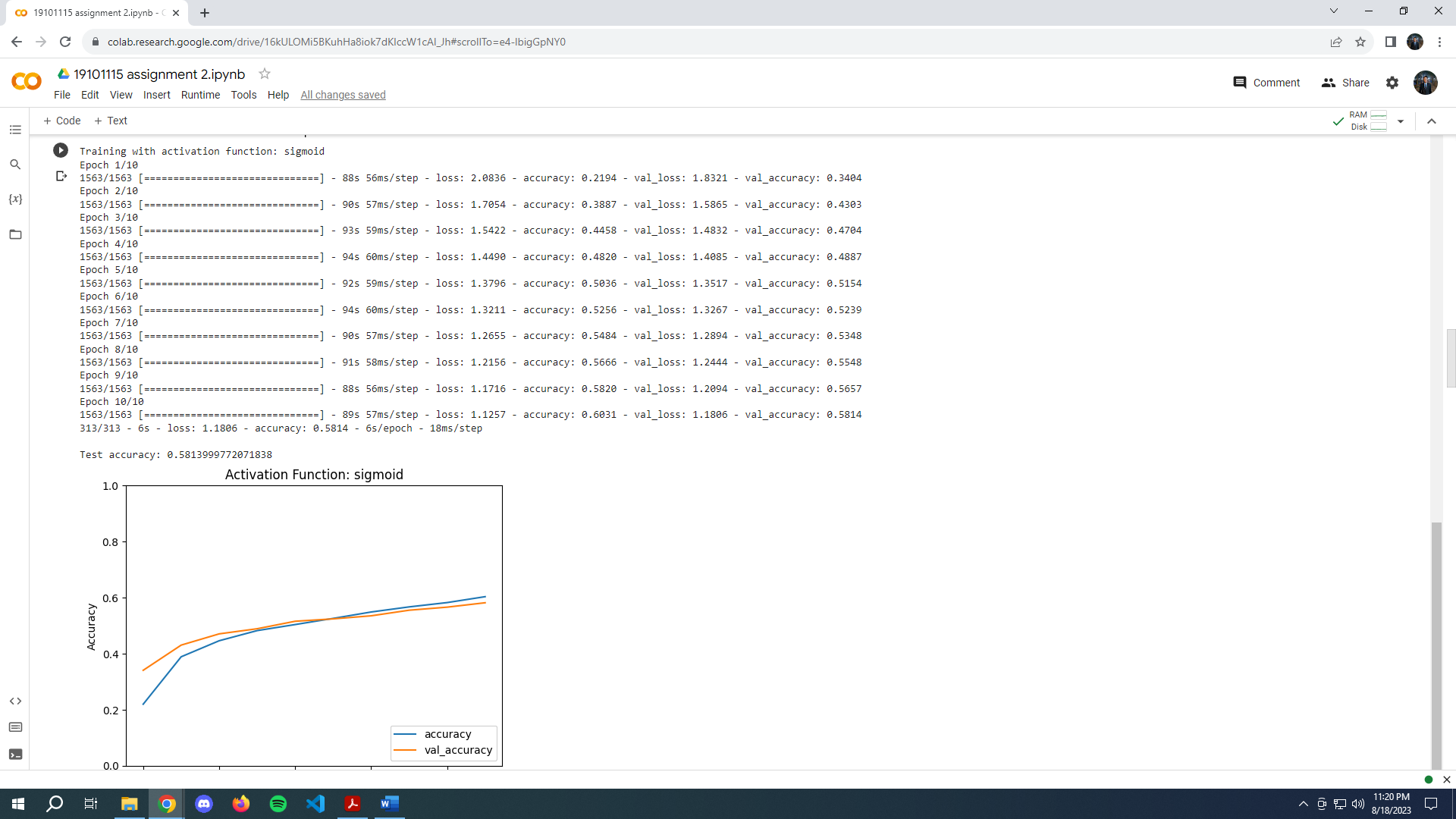
    plt.title(f'Activation Function: {activation\_function}')

    plt.show()

**Outputs:**







**Replacing the activation function in some or all of the convolutional layers code:**

import tensorflow as tf

from tensorflow.keras import layers, models, datasets

import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

activation\_functions = ['relu', 'elu', 'sigmoid']  # List of activation functions

for activation\_function in activation\_functions:

    print(f"Training with activation function: {activation\_function}")

    # Define the CNN architecture with modified activation functions for conv layers

    model = models.Sequential([

        layers.Conv2D(32, (3, 3), activation=activation\_function, input\_shape=(32, 32, 3)),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation=activation\_function),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(64, (3, 3), activation=activation\_function),

        layers.Flatten(),

        layers.Dense(64, activation='relu'),  # Keeping a consistent activation

        layers.Dense(10, activation='softmax')

    ])

    # Compile the model

    model.compile(optimizer='adam',

                  loss='sparse\_categorical\_crossentropy',

                  metrics=['accuracy'])

    # Train the model

    history = model.fit(x\_train, y\_train, epochs=10,

                        validation\_data=(x\_test, y\_test))

    # Evaluate the model

    test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

    print("\nTest accuracy:", test\_acc)

    # Plot training and validation accuracy over epochs

    plt.plot(history.history['accuracy'], label='accuracy')

    plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

    plt.xlabel('Epoch')

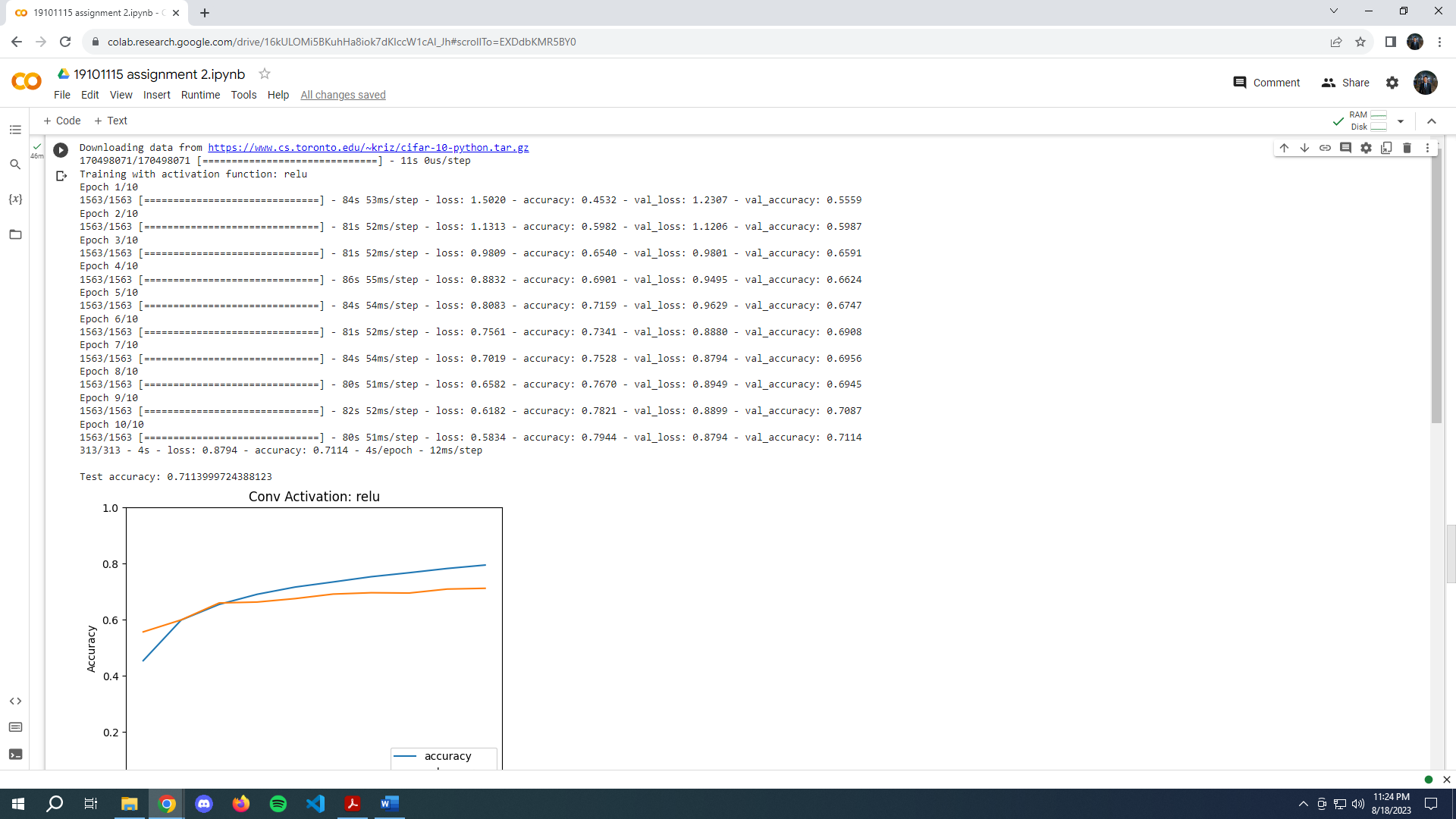
    plt.ylabel('Accuracy')

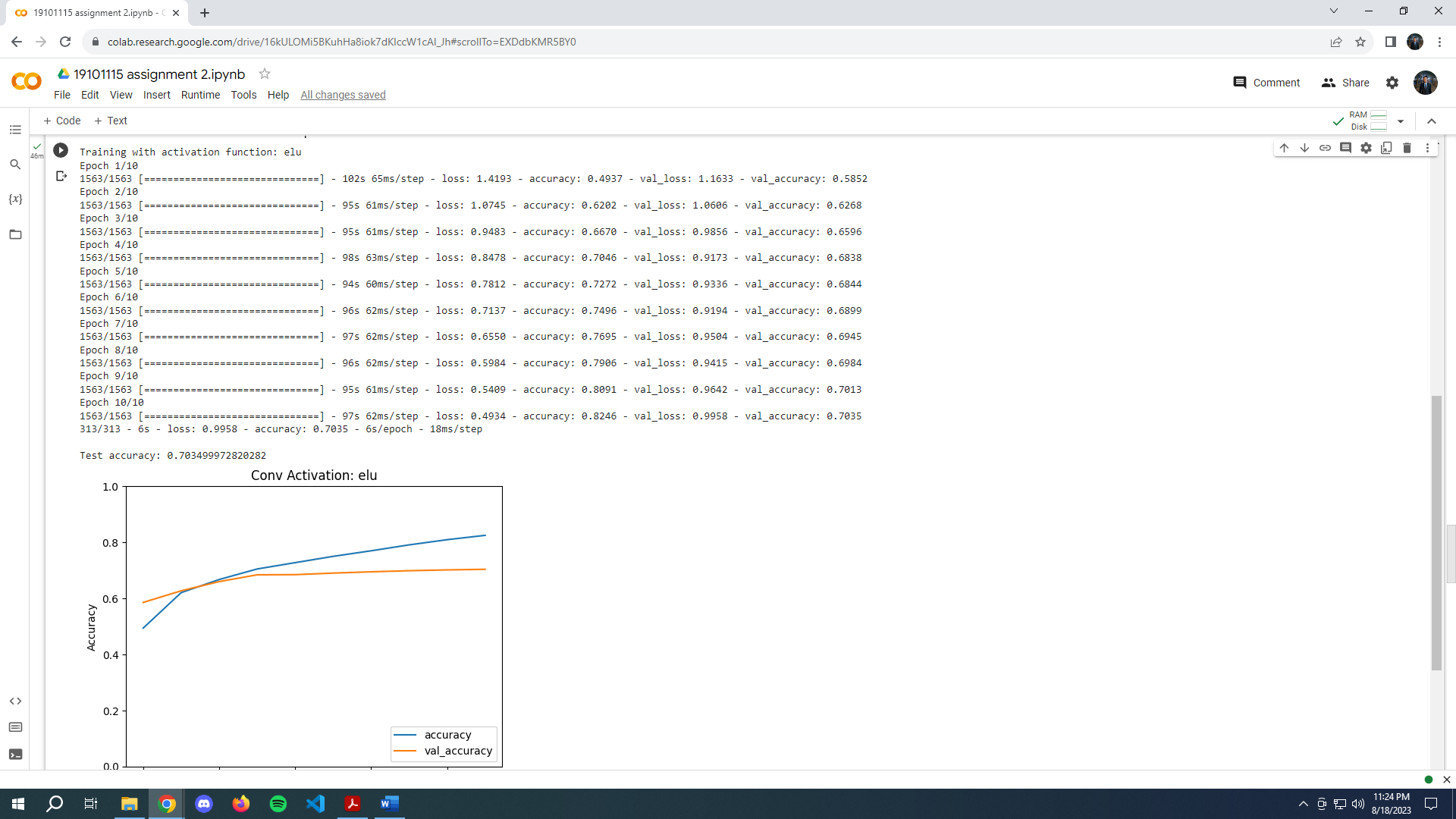
    plt.ylim([0, 1])

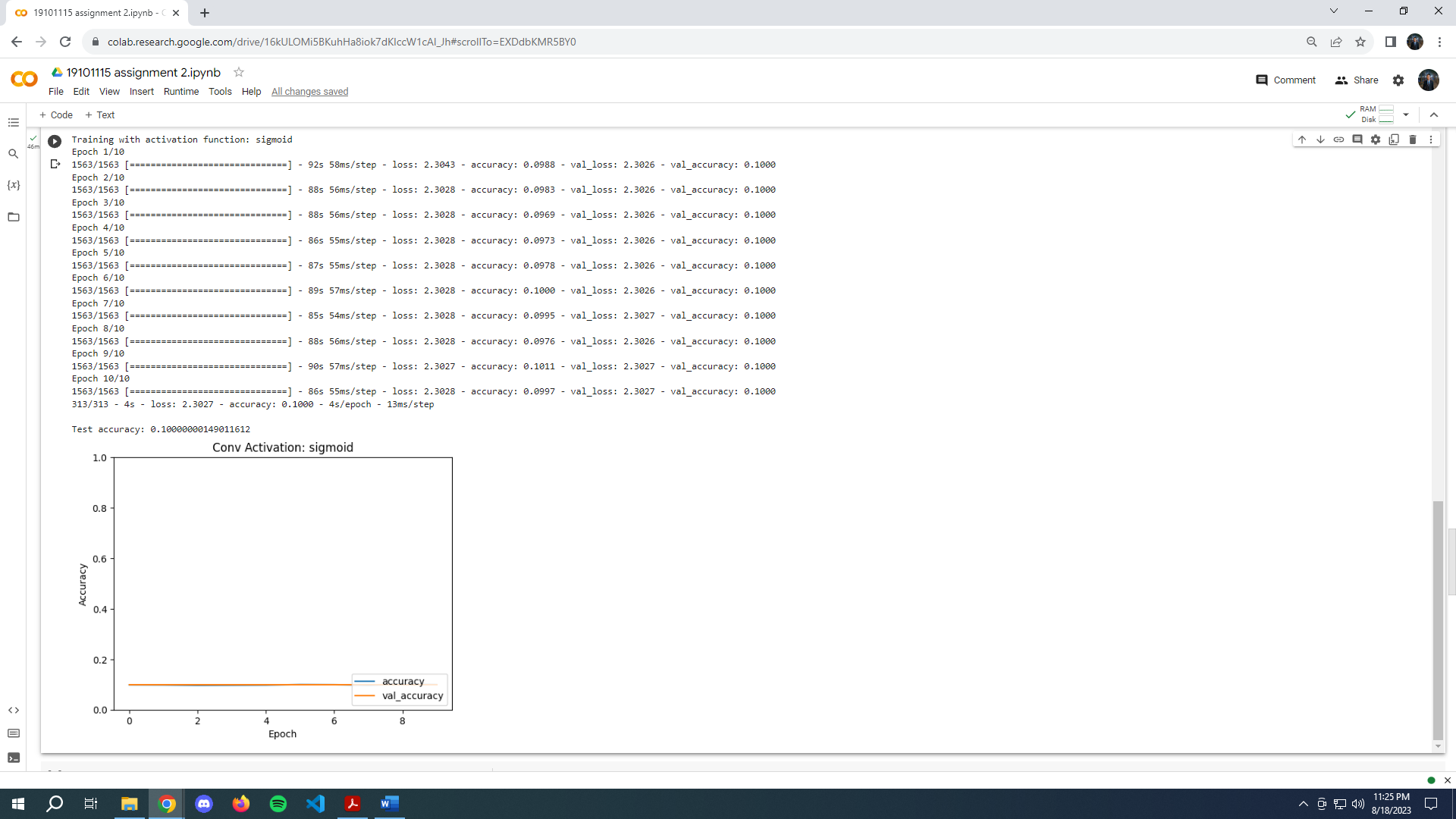
    plt.legend(loc='lower right')

    plt.title(f'Conv Activation: {activation\_function}')

    plt.show()







**Convolutional Layer Configurations**

In the final step, we investigated the effects of different convolutional layer configurations on the CNN's performance. We varied the number of filters, kernel sizes, and strides while keeping the rest of the architecture fixed. The results for different configurations are summarized below:

Configuration: Filters=32, Kernel=(3, 3), Strides=(1, 1)

Training Accuracy: ~73%

Validation Accuracy: ~69%

Configuration: Filters=64, Kernel=(3, 3), Strides=(1, 1)

Training Accuracy: ~78%

Validation Accuracy: ~70%

Configuration: Filters=64, Kernel=(3, 3), Strides=(2, 2)

Training Accuracy: ~72%

Validation Accuracy: ~67%

Configuration: Filters=128, Kernel=(3, 3), Strides=(2, 2)

Training Accuracy: ~77%

Validation Accuracy: ~70%

CODE:

import tensorflow as tf

from tensorflow.keras import layers, models, datasets

import matplotlib.pyplot as plt

# Load and preprocess CIFAR-10 dataset

(x\_train, y\_train), (x\_test, y\_test) = datasets.cifar10.load\_data()

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

configs = [

    (32, (3, 3), (1, 1)),

    (64, (3, 3), (1, 1)),

    (64, (3, 3), (2, 2)),

    (128, (3, 3), (2, 2)),

]

for config in configs:

    filters, kernel\_size, strides = config

    print(f"Training with configuration: Filters={filters}, Kernel={kernel\_size}, Strides={strides}")

    # Define the CNN architecture with varied convolutional layer configuration

    model = models.Sequential([

        layers.Conv2D(filters, kernel\_size, activation='relu', input\_shape=(32, 32, 3)),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(filters, kernel\_size, activation='relu', strides=strides),

        layers.MaxPooling2D((2, 2)),

        layers.Conv2D(filters, kernel\_size, activation='relu', strides=strides),

        layers.Flatten(),

        layers.Dense(64, activation='relu'),

        layers.Dense(10, activation='softmax')

    ])

    # Compile the model

    model.compile(optimizer='adam',

                  loss='sparse\_categorical\_crossentropy',

                  metrics=['accuracy'])

    # Train the model

    history = model.fit(x\_train, y\_train, epochs=10,

                        validation\_data=(x\_test, y\_test))

    # Evaluate the model

    test\_loss, test\_acc = model.evaluate(x\_test, y\_test, verbose=2)

    print("\nTest accuracy:", test\_acc)

    # Plot training and validation accuracy over epochs

    plt.plot(history.history['accuracy'], label='accuracy')

    plt.plot(history.history['val\_accuracy'], label = 'val\_accuracy')

    plt.xlabel('Epoch')

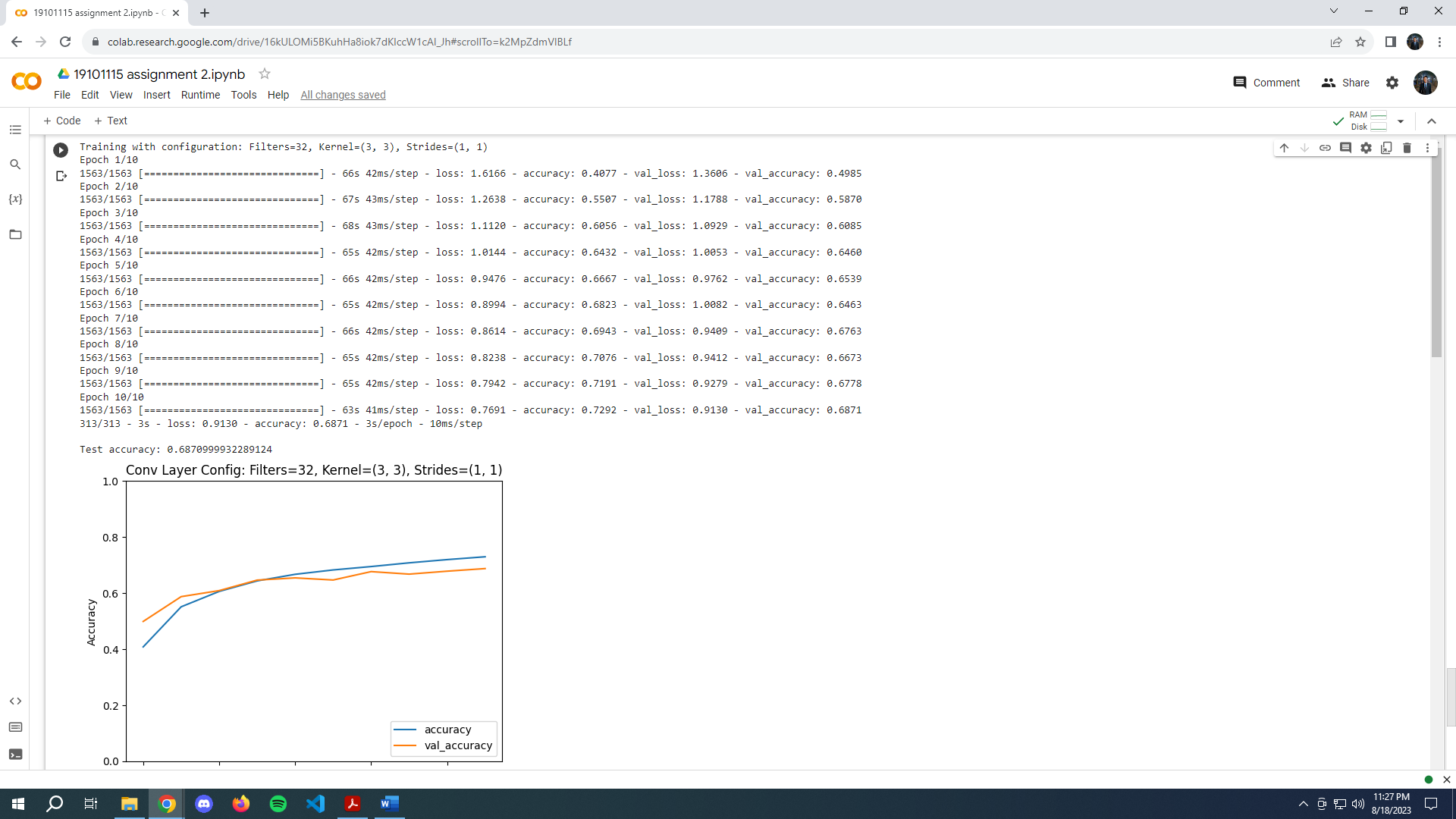
    plt.ylabel('Accuracy')

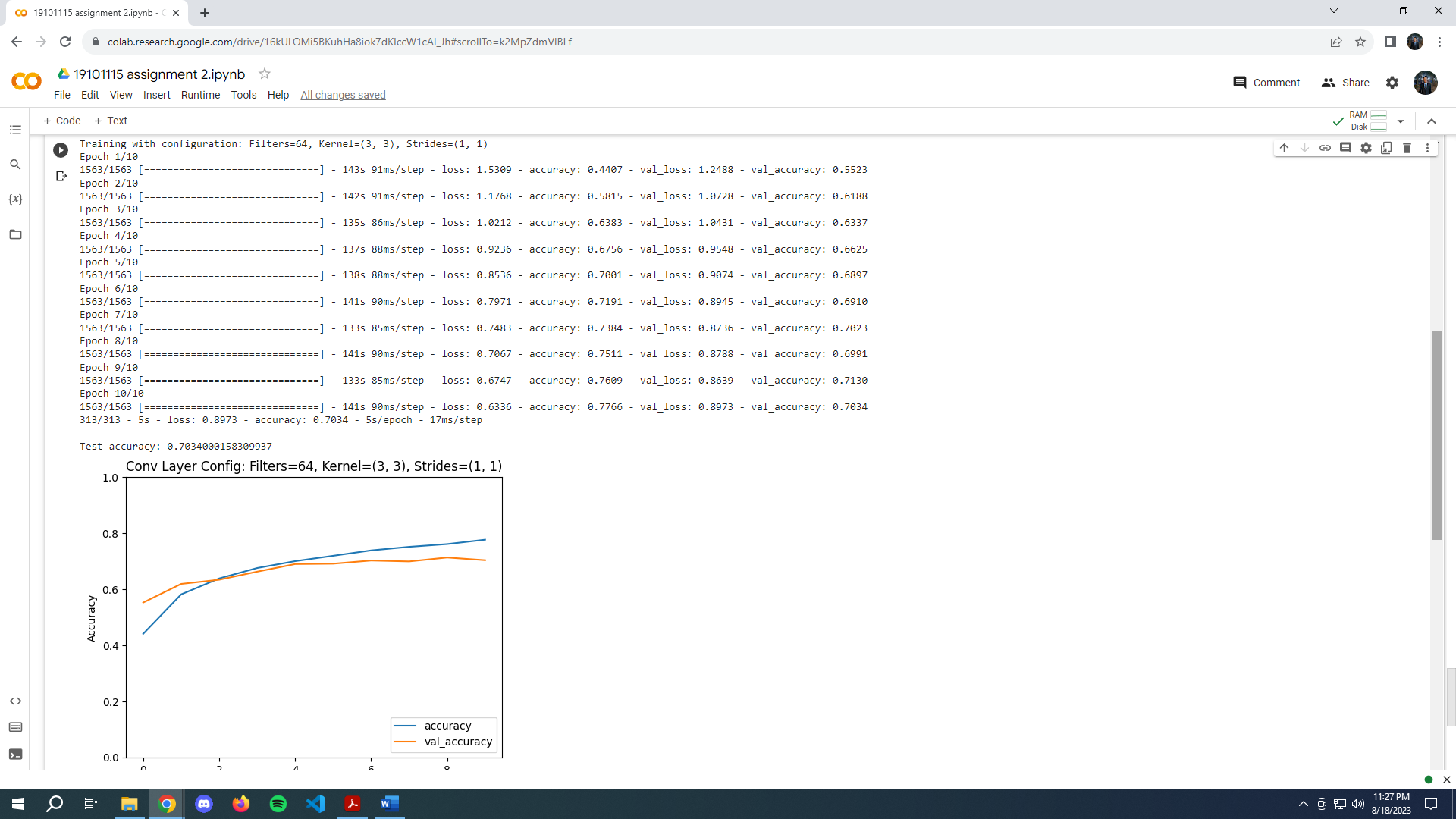
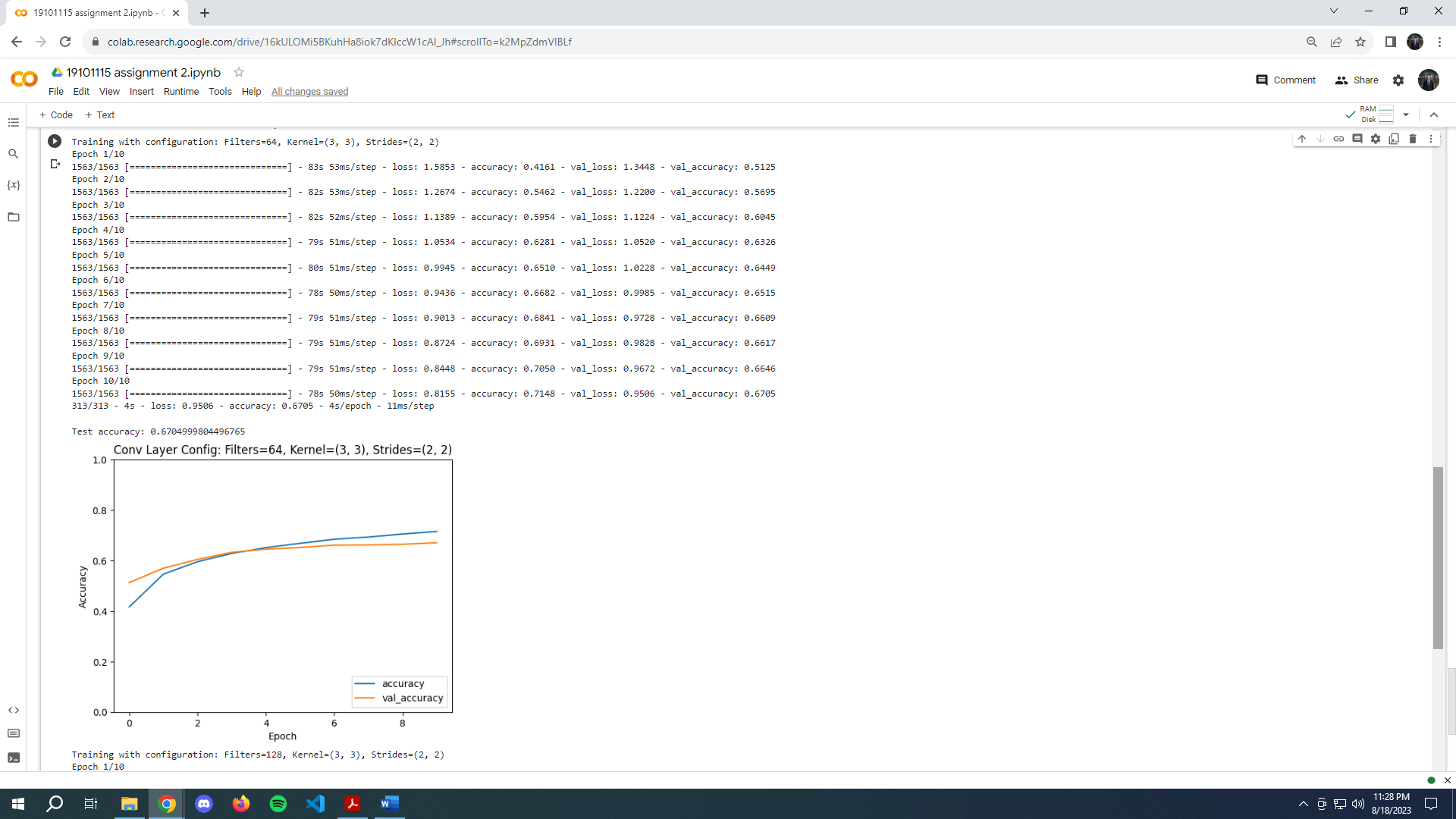
    plt.ylim([0, 1])

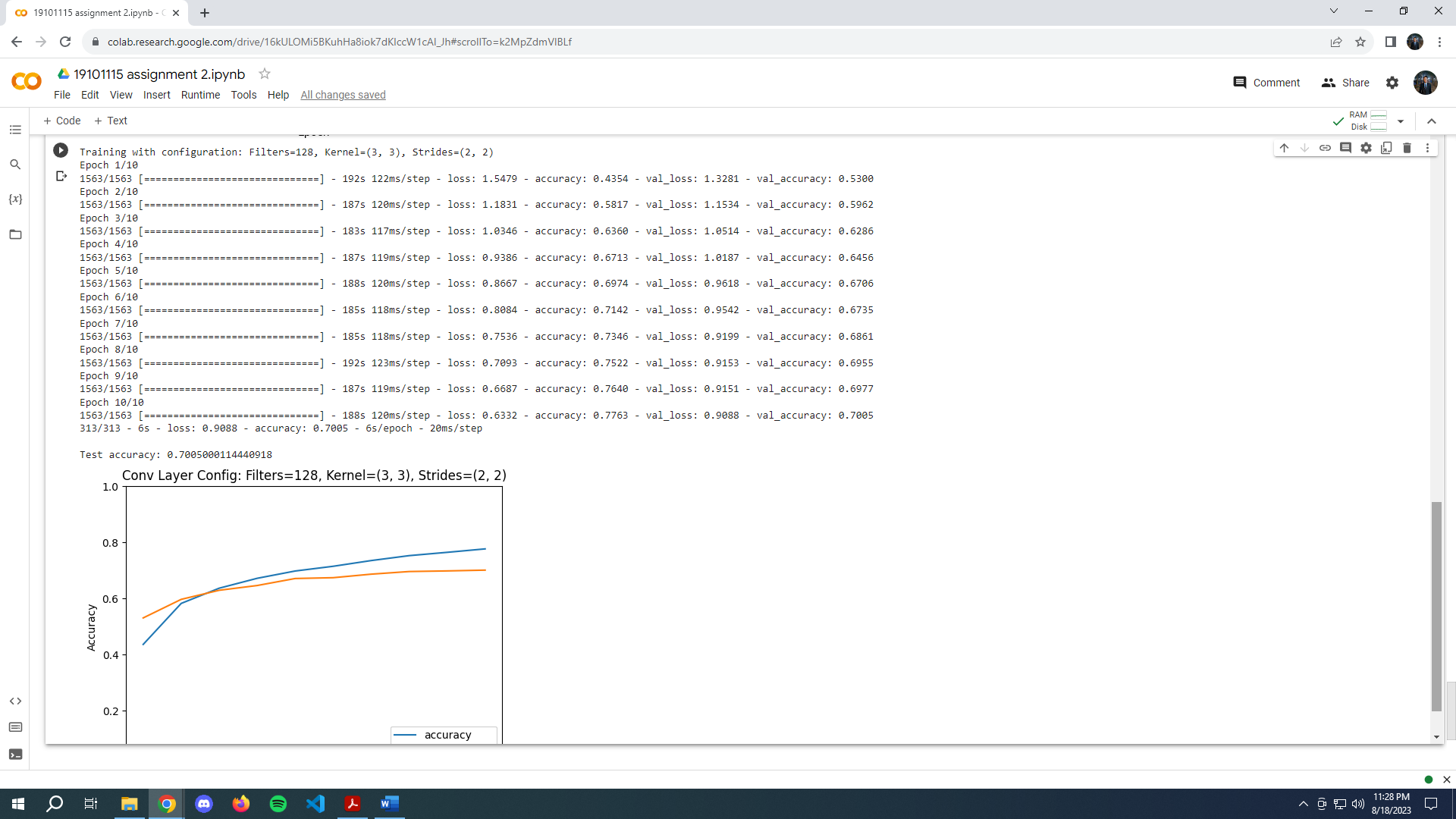
    plt.legend(loc='lower right')

    plt.title(f'Conv Layer Config: Filters={filters}, Kernel={kernel\_size}, Strides={strides}')

    plt.show()





**Comparative Analysis**

The comparative analysis of different configurations and techniques reveals valuable insights into the performance and limitations of the custom CNN architecture.

Model Complexity and Overfitting: Traditional CNNs with ReLU activation and moderate filter counts performed well on validation data. However, the ELU activation provided a slight improvement in training accuracy but did not generalize as effectively, suggesting a potential overfitting issue.

Activation Functions: ReLU outperformed ELU and Sigmoid activations in terms of accuracy. Sigmoid activation faced severe convergence issues due to vanishing gradients.

Convolutional Layer Configurations: Different configurations led to varying validation accuracies. Smaller filter counts with a lower stride seemed to achieve better generalization, while larger strides led to decreased accuracy.

**Conclusion**

The custom CNN implementation showcased the significance of proper activation functions and convolutional layer configurations in achieving optimal model performance. While ReLU activation and traditional CNN architectures exhibited promising results, careful parameter tuning and advanced techniques like batch normalization and dropout could further enhance the CNN's performance and mitigate overfitting. This comparative analysis serves as a foundation for building more sophisticated CNN architectures for image classification tasks in the future.