Akash T

ADIT | NSTI CALICUT

**Weekly Assessment**

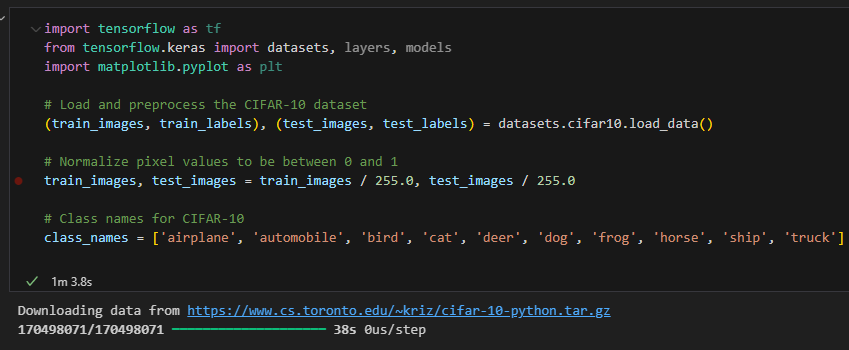
**AIM**

Using a deep learning framework of your choice (TensorFlow, PyTorch, etc.), implement a CNN to classify images from the CIFAR-10 dataset. Ensure your network includes convolutional layers, pooling layers, and fully connected layers. Evaluate the performance of your model and discuss any improvements you could make.  
  
**SOFTWARE/HARDWARE REQUIREMENTS**

* Windows PC
* VS Code/Jupyter notebook

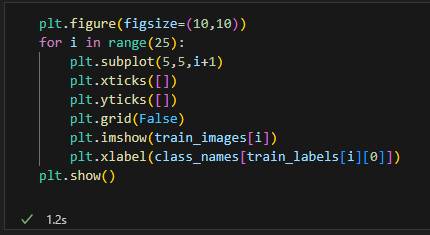
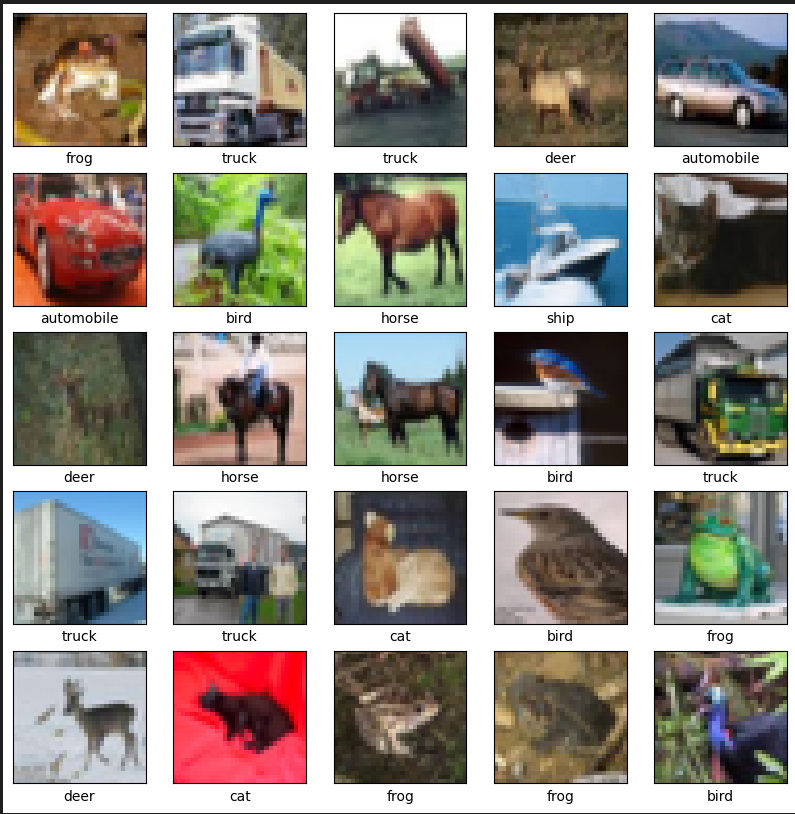
**PROCEDURE**

**Step 1**

****

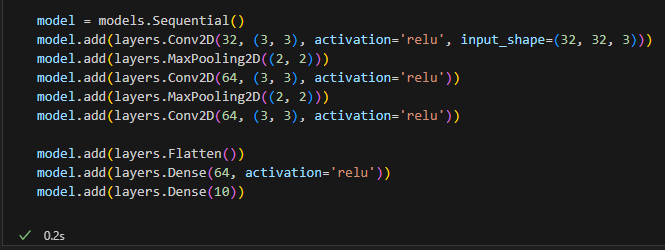
* tensorflow: A popular deep learning framework.
* datasets, layers, models: Submodules of Keras (which is integrated into TensorFlow), used for loading data, defining layers, and creating models.
* matplotlib.pyplot: A plotting library used for visualizing data.

**Step 2**

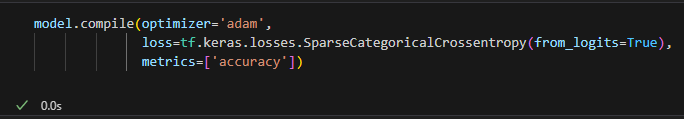
* datasets.cifar10.load\_data(): Loads the CIFAR-10 dataset, which contains 60,000 32x32 color images in 10 classes, with 6,000 images per class.
* train\_images, train\_labels: Training data and labels.
* test\_images, test\_labels: Testing data and labels.
* train\_images / 255.0, test\_images / 255.0: Normalizes the image pixel values to be between 0 and 1 for faster convergence during training.
* class\_names: List of class names corresponding to the CIFAR-10 dataset.

**Step 3**

****

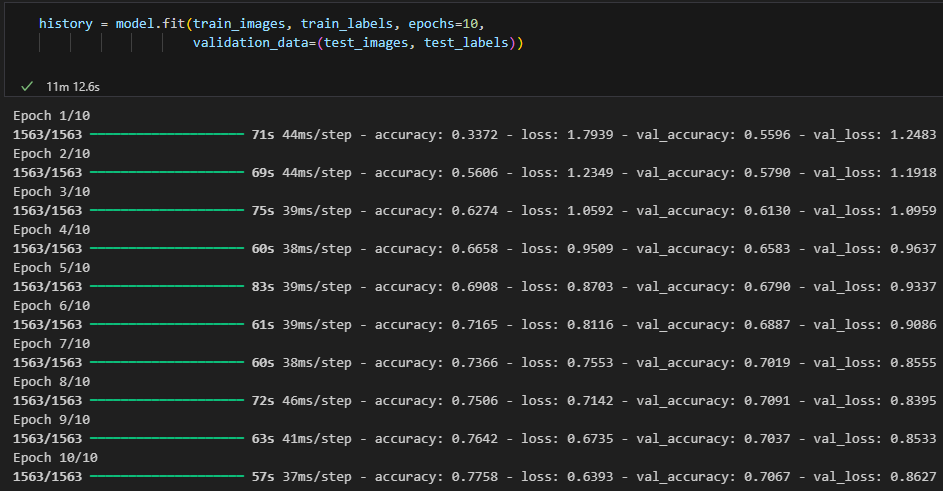
* models.Sequential(): Initializes a sequential model.
* layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(32, 32, 3)): Adds a 2D convolutional layer with 32 filters, a 3x3 kernel size, ReLU activation, and an input shape of 32x32x3 (height, width, channels).
* layers.MaxPooling2D((2, 2)): Adds a max pooling layer with a 2x2 pool size to reduce the spatial dimensions of the feature maps.
* layers.Conv2D(64, (3, 3), activation='relu'): Adds another convolutional layer with 64 filters.
* layers.Flatten(): Flattens the 3D feature maps to 1D feature vectors.
* layers.Dense(64, activation='relu'): Adds a fully connected (dense) layer with 64 units and ReLU activation.
* layers.Dense(10): Adds the output layer with 10 units (one for each class).

**Step 4**



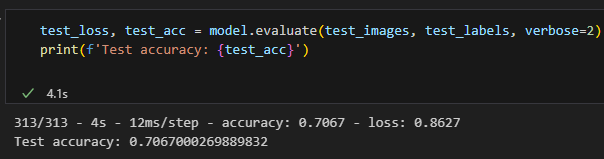
* model.compile(): Configures the model for training.
* optimizer='adam': Uses the Adam optimization algorithm.
* loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True): Uses sparse categorical crossentropy loss (from\_logits=True indicates that the output values are not normalized).
* metrics=['accuracy']: Monitors accuracy during training and evaluation.

**Step 5**



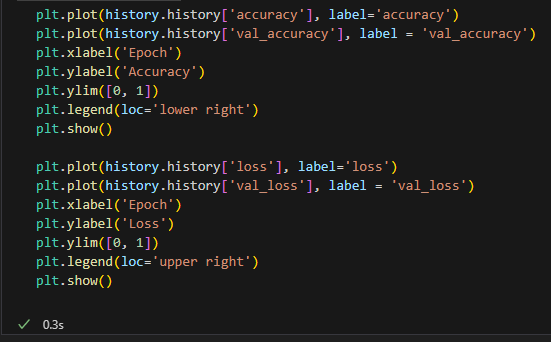
* model.fit(): Trains the model on the training data.
* epochs=10: Specifies the number of epochs (full passes through the training dataset).
* validation\_data=(test\_images, test\_labels): Uses the test data for validation during training.

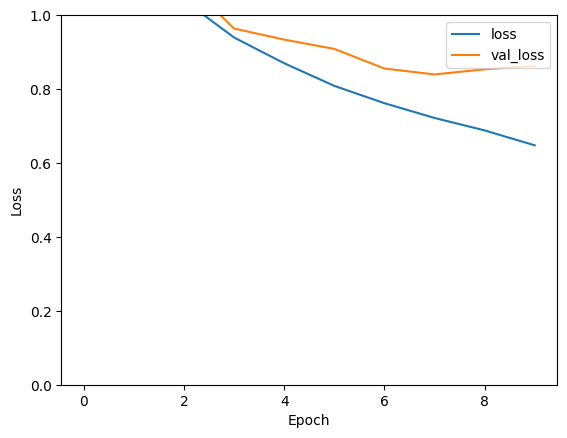
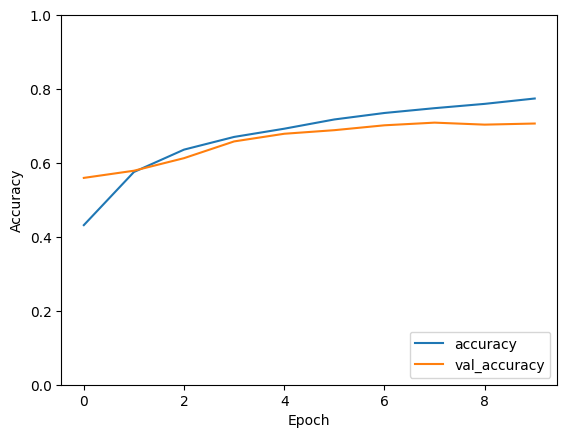
**Step 6**



* model.evaluate(): Evaluates the model on the test data.
* verbose=2: Prints the evaluation progress.
* test\_acc: Stores the test accuracy.

**Step 7**





* history.history['accuracy']: Training accuracy for each epoch.
* history.history['val\_accuracy']: Validation accuracy for each epoch.
* plt.plot(): Plots the training and validation accuracy.
* plt.xlabel('Epoch'), plt.ylabel('Accuracy'): Labels the x and y axes.
* plt.ylim([0, 1]): Sets the y-axis limits.
* plt.legend(loc='lower right'): Adds a legend in the lower right corner.
* plt.show(): Displays the plot.
* Similar plotting is done for the training and validation loss.

**Possible Improvements**

* Deeper Network: Adding more layers to capture more complex features.
* Data Augmentation: Increases the diversity of the training data to improve generalization.
* Dropout Layers: Prevents overfitting by randomly dropping units during training.

**AIM**

Construct a feedforward neural network to predict housing prices based on the provided

dataset. Include input normalization, hidden layers with appropriate activation

functions, and an output layer. Train the network using backpropagation and evaluate its

performance using Mean Squared Error (MSE).

Bedrooms,Bathrooms,SquareFootage,Location,Age,Price

3,2,1500,Urban,10,300000

4,3,2000,Suburban,5,400000

2,1,800,Rural,20,150000

3,2,1600,Urban,12,310000

4,3,2200,Suburban,8,420000

2,1,900,Rural,25,160000

5,4,3000,Urban,3,600000

3,2,1400,Suburban,15,290000

3,2,1300,Rural,30,180000

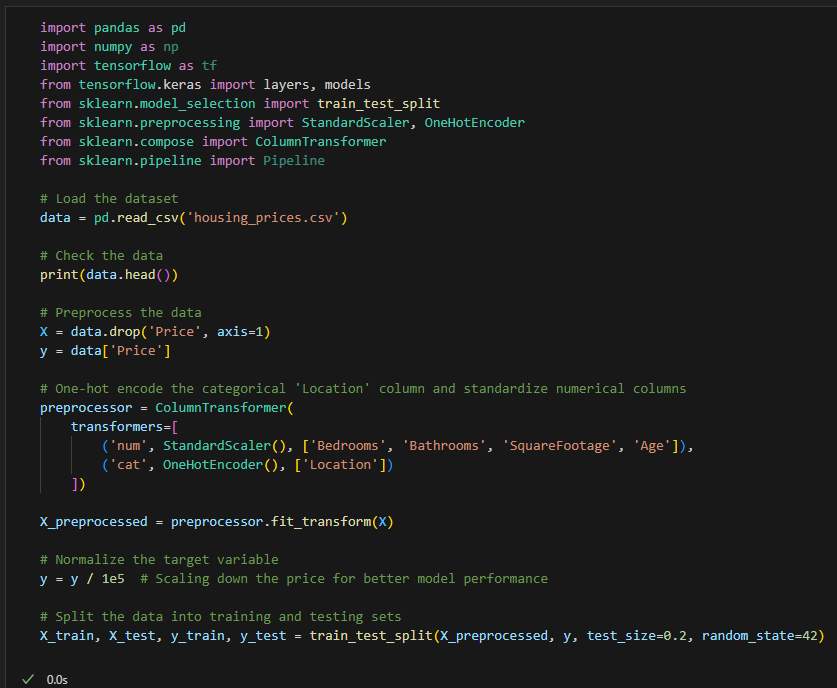
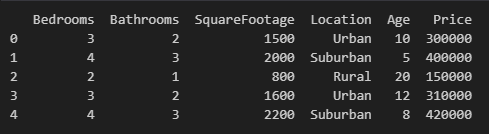
4,3,2500,Urban,7,500000

You can copy this data into a CSV file named housing\_prices.csv.  
  
**SOFTWARE/HARDWARE REQUIREMENTS**

* Windows PC
* VS Code/Jupyter notebook

**PROCEDURE**

**Step 1**

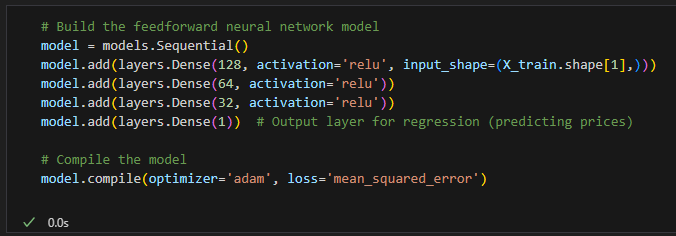
**  
**

* Library Imports: We import the necessary libraries for data manipulation, model building, and preprocessing.
* Load the Dataset: We load the dataset from the CSV file using pd.read\_csv('housing\_prices.csv').
* Check the Data: Print the first few rows of the dataset to ensure it is loaded correctly.

Preprocess the Data:

* We separate the features (X) from the target variable (y).
* We use ColumnTransformer to preprocess the data:
* StandardScaler standardizes the numerical columns (Bedrooms, Bathrooms, SquareFootage, Age).
* OneHotEncoder encodes the categorical 'Location' column.
* We normalize the target variable (price) by dividing it by 100,000 to bring it to a similar scale as the input features.
* Split the Data: We split the data into training and testing sets using train\_test\_split.

**Step 2**



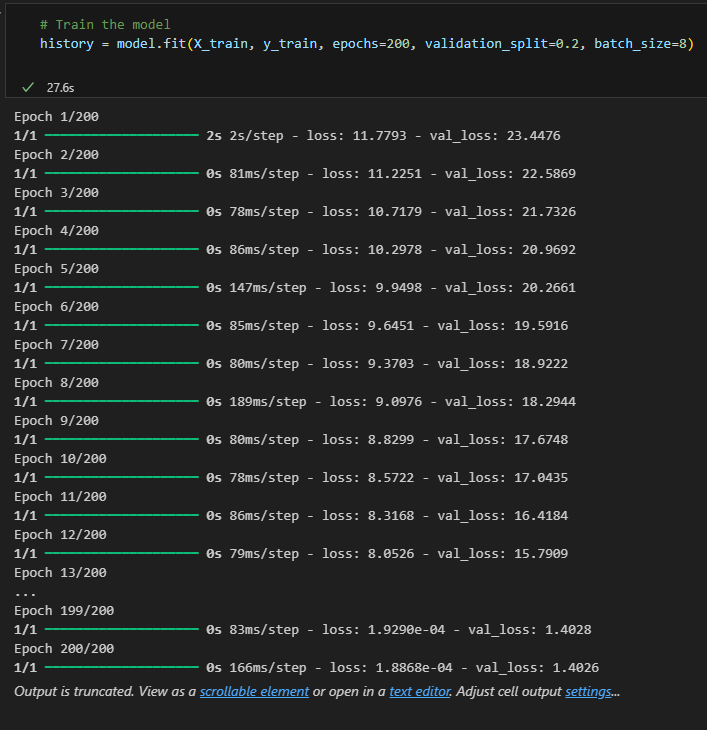
**Define the Model**:

* models.Sequential(): Initializes a sequential model.
* layers.Dense(128, activation='relu', input\_shape=(X\_train.shape[1],)): Adds a dense layer with 128 units and ReLU activation. The input\_shape is set to the number of features in the training set.
* layers.Dense(64, activation='relu'): Adds a dense layer with 64 units and ReLU activation.
* layers.Dense(32, activation='relu'): Adds a dense layer with 32 units and ReLU activation.
* layers.Dense(1): Adds the output layer with a single unit to predict the price.

**Compile the Model**:

* model.compile(optimizer='adam', loss='mean\_squared\_error'): Compiles the model with the Adam optimizer and Mean Squared Error (MSE) loss function.

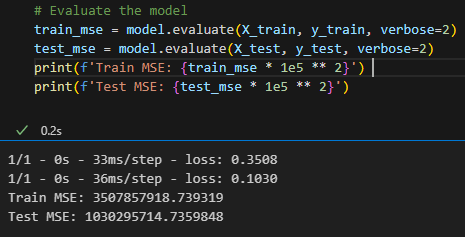
**Step 3**



**Train the Model**:

* model.fit(X\_train, y\_train, epochs=200, validation\_split=0.2, batch\_size=8): Trains the model on the training data.
  + epochs=200: Specifies the number of epochs (full passes through the training dataset).
  + validation\_split=0.2: Uses 20% of the training data for validation.
  + batch\_size=8: Sets the number of samples per gradient update.

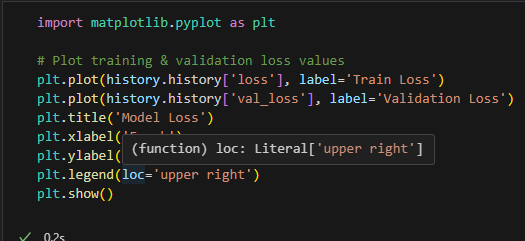
**Step 4**

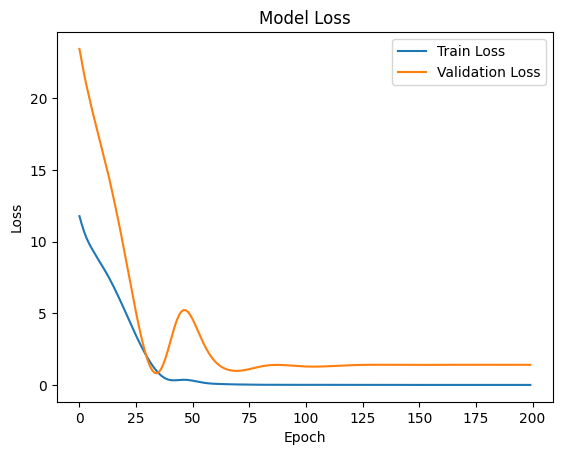
****

**Evaluate the Model:**

* model.evaluate(X\_train, y\_train, verbose=2): Evaluates the model on the training data.
* model.evaluate(X\_test, y\_test, verbose=2): Evaluates the model on the testing data.
* print(f'Train MSE: {train\_mse \* 1e5 \*\* 2}'): Prints the training MSE, converting it back to the original scale by multiplying by 100,000 squared.
* print(f'Test MSE: {test\_mse \* 1e5 \*\* 2}'): Prints the testing MSE, converting it back to the original scale.

**Step 5**





**Plot Training History**:

* plt.plot(history.history['loss'], label='Train Loss'): Plots the training loss over epochs.
* plt.plot(history.history['val\_loss'], label='Validation Loss'): Plots the validation loss over epochs.
* plt.title('Model Loss'): Sets the title of the plot.
* plt.xlabel('Epoch'), plt.ylabel('Loss'): Labels the x and y axes.
* plt.legend(loc='upper right'): Adds a legend in the upper right corner.
* plt.show(): Displays the plot.

**\*** The error here is high because the amount of data is very small