Neural Networks and Deep Learning ICP-3:

Student Details:

Student Id No: 700745501

Name: Pavan Naga Sai Gullapalli

CRN: 31176

Video link:
https://drive.google.com/file/d/1h4DGJCr7vd0QX5lj90kiEWpIDYt4HEqZ/view?usp=sharing

Github link: https://github.com/PavanNagaSaiG/Neural_Network_DeepLearning

Question: 1

```
#Importing the required Libraries.
# Simple CNN model for CIFAR-10.
import numpy
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.optimizers import SGD
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.utils import np_utils
#from keras import backend as K
#K.set_image_dim_ordering('th')

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
# load data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# normalize inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train = X_train / 255.0
X_test = X_test / 255.0
# one hot encode outputs
```

```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
# Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# Compile model
epochs = 25
lrate = 0.01
decay = lrate/epochs
sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

Explanation:

1. Importing necessary libraries and modules for the implementation of the CNN model.

2. Setting a seed for random number generation to ensure reproducibility.

3. Loading the CIFAR-10 dataset.

4. Normalizing the image data inputs from 0-255 to 0.0-1.0 to make computations easier for the network.

5. One hot encoding the output labels for multi-class classification (10 classes in CIFAR-10).

6. Defining the CNN model, which includes 2 convolutional layers (with 32 filters, 3x3 kernel size, 'relu' activation and dropout), a max pooling layer, a flattening layer to connect the CNN layers to fully connected layers, a dense layer with 512 nodes, another dropout layer to prevent overfitting, and a final dense layer with 10 nodes (for 10 classes) using the 'softmax' activation function.

7. Compiling the model with parameters for training, including the learning rate, decay, Stochastic Gradient Descent (SGD) as the optimizer, and the categorical cross-entropy loss function for multi-class classification.

8. Printing the model summary to show the layers and their order, along with their output shapes and number of parameters.

9. Fitting the model to the training data with the specified number of epochs and batch size, and validating it on the test data.

10. Evaluating the model's performance on the test data and printing the accuracy percentage.

Output:

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 3s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 dropout (Dropout)           (None, 32, 32, 32)        0

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 flatten (Flatten)           (None, 8192)              0

 dense (Dense)               (None, 512)               4194816

 dropout_1 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
_________________________________________________________________
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/gradient_descent.py:114: UserWarning:
The `lr` argument is deprecated, use `learning_rate` instead.
  super().__init__(name, **kwargs)
None
Epoch 1/25
1563/1563 [==============================] - 20s 7ms/step - loss: 1.6790 - accuracy: 0.3900 -
val_loss: 1.4187 - val_accuracy: 0.5006
Epoch 2/25
1563/1563 [==============================] - 10s 7ms/step - loss: 1.3384 - accuracy: 0.5161 -
val_loss: 1.2028 - val_accuracy: 0.5695
Epoch 3/25
1563/1563 [==============================] - 10s 7ms/step - loss: 1.1749 - accuracy: 0.5797 -
val_loss: 1.0972 - val_accuracy: 0.6059
Epoch 4/25
1563/1563 [==============================] - 10s 6ms/step - loss: 1.0546 - accuracy: 0.6243 -
val_loss: 1.0516 - val_accuracy: 0.6232
Epoch 5/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.9610 - accuracy: 0.6595 -
val_loss: 0.9963 - val_accuracy: 0.6452
Epoch 6/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.8756 - accuracy: 0.6903 -
val_loss: 0.9586 - val_accuracy: 0.6585
Epoch 7/25
1563/1563 [==============================] - 11s 7ms/step - loss: 0.8047 - accuracy: 0.7161 -
val_loss: 0.9571 - val_accuracy: 0.6640
```

Epoch 8/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.7413 - accuracy: 0.7371 - val_loss: 0.9306 - val_accuracy: 0.6774
Epoch 9/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6878 - accuracy: 0.7586 - val_loss: 0.9366 - val_accuracy: 0.6790
Epoch 10/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.6393 - accuracy: 0.7744 - val_loss: 0.9088 - val_accuracy: 0.6845
Epoch 11/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5915 - accuracy: 0.7902 - val_loss: 0.9163 - val_accuracy: 0.6922
Epoch 12/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.5475 - accuracy: 0.8059 - val_loss: 0.9220 - val_accuracy: 0.6971
Epoch 13/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.5142 - accuracy: 0.8182 - val_loss: 0.9303 - val_accuracy: 0.6919
Epoch 14/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.4768 - accuracy: 0.8319 - val_loss: 0.9370 - val_accuracy: 0.6948
Epoch 15/25
1563/1563 [==============================] - 10s 7ms/step - loss: 0.4438 - accuracy: 0.8412 - val_loss: 0.9572 - val_accuracy: 0.6919
Epoch 16/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.4145 - accuracy: 0.8533 - val_loss: 0.9766 - val_accuracy: 0.6918
Epoch 17/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3859 - accuracy: 0.8634 - val_loss: 0.9860 - val_accuracy: 0.6955
Epoch 18/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3614 - accuracy: 0.8731 - val_loss: 0.9661 - val_accuracy: 0.6960
Epoch 19/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3419 - accuracy: 0.8816 - val_loss: 0.9945 - val_accuracy: 0.6974
Epoch 20/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.3240 - accuracy: 0.8850 - val_loss: 1.0137 - val_accuracy: 0.7008
Epoch 21/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.3037 - accuracy: 0.8941 - val_loss: 1.0129 - val_accuracy: 0.7001
Epoch 22/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2881 - accuracy: 0.8986 - val_loss: 1.0500 - val_accuracy: 0.6982
Epoch 23/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2758 - accuracy: 0.9039 - val_loss: 1.0372 - val_accuracy: 0.7010
Epoch 24/25
1563/1563 [==============================] - 10s 6ms/step - loss: 0.2578 - accuracy: 0.9097 - val_loss: 1.0618 - val_accuracy: 0.7016
Epoch 25/25
1563/1563 [==============================] - 9s 6ms/step - loss: 0.2477 - accuracy: 0.9142 - val_loss: 1.0681 - val_accuracy: 0.7019
Accuracy: 70.19%

```
import numpy as np
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.constraints import maxnorm
from keras.utils import np_utils
from keras.optimizers import SGD

# to fix the random seed for reproducibility
np.random.seed(7)

# to load the data
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# to normalize the inputs from 0-255 to 0.0-1.0
X_train = X_train.astype('float32') / 255.0
X_test = X_test.astype('float32') / 255.0

# One hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

# to Create the model
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(32, 32, 3), padding='same', activation='relu', kernel_constraint=maxnorm(3)))
```

```
model.add(Dropout(0.2))
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax'))

# to compile the model
epochs = 5
learning_rate = 0.01
decay_rate = learning_rate / epochs
sgd = SGD(lr=learning_rate, momentum=0.9, decay=decay_rate, nesterov=False)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
print(model.summary())

# to fit the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=32)
```

```
# to evaluate the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1] * 100))
```

Explanation:

1. **Importing Libraries**: The code imports various Python libraries required for building and training the CNN, such as numpy for numerical operations and Keras for creating and training the model.

2. **Loading and Preprocessing Data**: The CIFAR-10 dataset is loaded into the variables X_train and y_train for training and X_test and y_test for testing. The pixel values of the images are normalized to a range between 0.0 and 1.0 to make training more efficient.

3. **One-Hot Encoding**: The labels (y_train and y_test) are converted into a one-hot encoded format, which means converting categorical values (class labels) into binary vectors. This step is necessary for training a multiclass classification model.

4. **Creating the CNN Model**: The model is defined as a sequence of layers. It starts with a series of Convolutional layers, each followed by a MaxPooling layer to extract features from the images. Dropout layers are also added to prevent overfitting during training. The final layers are fully connected (Dense) layers that perform the classification based on the learned features.

5. **Compiling the Model**: Before training, the model is compiled with the specified loss function, optimizer, and evaluation metric. The loss function is 'categorical_crossentropy,' which is appropriate for multiclass classification tasks. The optimizer used is Stochastic Gradient Descent (SGD) with a learning rate, momentum, and decay rate.

6. **Training the Model**: The model is trained using the training data (X_train and y_train) with a specified number of epochs (5 in this case) and a batch size of 32. During training, the model learns to classify the images correctly by adjusting its internal parameters (weights) based on the provided data.

7. **Evaluating the Model**: After training, the model is evaluated using the test data (X_test and y_test) to see how well it performs on unseen data. The accuracy metric is used to measure the model's performance, indicating the percentage of correctly classified images out of all the test images.

The final output of the code will be the accuracy achieved by the model on the test data, represented as a percentage. The goal is to obtain a high accuracy, indicating that the CNN has learned to recognize and classify the images from the CIFAR-10 dataset.

Output:

```
Model: "sequential_3"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 32, 32, 32)        896

 dropout_10 (Dropout)        (None, 32, 32, 32)        0

 conv2d_11 (Conv2D)          (None, 32, 32, 32)        9248

 max_pooling2d_5 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_12 (Conv2D)          (None, 16, 16, 64)        18496

 dropout_11 (Dropout)        (None, 16, 16, 64)        0

 conv2d_13 (Conv2D)          (None, 16, 16, 64)        36928

 max_pooling2d_6 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_14 (Conv2D)          (None, 8, 8, 128)         73856

 dropout_12 (Dropout)        (None, 8, 8, 128)         0

 conv2d_15 (Conv2D)          (None, 8, 8, 128)         147584

 max_pooling2d_7 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_3 (Flatten)         (None, 2048)              0

 dropout_13 (Dropout)        (None, 2048)              0

 dense_7 (Dense)             (None, 1024)              2098176

 dropout_14 (Dropout)        (None, 1024)              0

 dense_8 (Dense)             (None, 512)               524800

 dropout_15 (Dropout)        (None, 512)               0

 dense_9 (Dense)             (None, 10)                5130

=================================================================
Total params: 2,915,114
Trainable params: 2,915,114
Non-trainable params: 0

_________________________________________________________________
None
Epoch 1/5
1563/1563 [==============================] - 14s 9ms/step - loss: 1.8945 -
accuracy: 0.3012 - val_loss: 1.6138 - val_accuracy: 0.4223
Epoch 2/5
```

```
1563/1563 [==============================] - 13s 8ms/step - loss: 1.5147 -
accuracy: 0.4478 - val_loss: 1.4179 - val_accuracy: 0.4900
Epoch 3/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.3861 -
accuracy: 0.4972 - val_loss: 1.3966 - val_accuracy: 0.4998
Epoch 4/5
1563/1563 [==============================] - 12s 8ms/step - loss: 1.3160 -
accuracy: 0.5215 - val_loss: 1.2847 - val_accuracy: 0.5323
Epoch 5/5
1563/1563 [==============================] - 13s 8ms/step - loss: 1.2652 -
accuracy: 0.5431 - val_loss: 1.2342 - val_accuracy: 0.5508
Accuracy: 55.08%
```

Question: 2

```
[ ]  # to predict the first 4 images of the test data
     predictions = model.predict(X_test[:4])
     # Converting the predictions to class labels
     predicted_labels = np.argmax(predictions, axis=1)
     # Converting the actual labels to class labels
     actual_labels = np.argmax(y_test[:4], axis=1)

     # Printing the predicted and actual labels for the first 4 images
     print("Predicted labels:", predicted_labels)
     print("Actual labels:   ", actual_labels)
```

Explanation:

1. **Predicting Class Labels**: The line `predictions = model.predict(X_test[:4])` uses the trained CNN model (`model`) to make predictions on the first 4 images in the test dataset (`X_test[:4]`). The `model.predict()` function returns an array of predicted probabilities for each class for each image.

2. **Converting to Class Labels**: The line `predicted_labels = np.argmax(predictions, axis=1)` converts the predicted probabilities into class labels. For each image, it finds the class label with the highest probability (the index with the maximum value) and stores it in the `predicted_labels` array.

3. **Converting Actual Labels**: Similarly, the line `actual_labels = np.argmax(y_test[:4], axis=1)` converts the one-hot encoded actual labels (ground truth) for the first 4 images in the test dataset (`y_test[:4]`) into class labels. It finds the index of the 1 (the class label) in each one-hot encoded vector and stores the class labels in the `actual_labels` array.

4. **Printing Predicted and Actual Labels**: Finally, the code prints the predicted and actual class labels for the first 4 images. The line `print("Predicted labels:", predicted_labels)` prints

the array of predicted class labels, and the line `print("Actual labels:   ", actual_labels)` prints the array of actual class labels.

By comparing the predicted and actual labels, you can visually check how well the model is performing on these specific images. Ideally, you would hope that the predicted labels closely match the actual labels, indicating that the CNN is making accurate predictions for these images.

Output:

```
1/1 [==============================] - 0s 333ms/step
Predicted labels: [3 8 8 0]
Actual labels:    [3 8 8 0]
```

Question-3

```
import matplotlib.pyplot as plt

# Plot the training and validation loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper right')
plt.show()

# Plot the training and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='lower right')
plt.show()
```

Explanation:

This part of the code is responsible for plotting two graphs to visualize the training process and the performance of the Convolutional Neural Network (CNN) model during training.

1. **Plotting Training and Validation Loss**:

   - The line `plt.plot(history.history['loss'])` plots the training loss over each epoch on the graph.

   - The line `plt.plot(history.history['val_loss'])` plots the validation loss over each epoch on the same graph.

   - `history.history` is a dictionary that holds the training and validation loss values recorded during training.

   - The `title`, `ylabel`, and `xlabel` functions set the title and labels for the graph.

   - The line `plt.legend(['train', 'val'], loc='upper right')` adds a legend to the graph, indicating which line corresponds to training loss and which one corresponds to validation loss.

   - Finally, `plt.show()` displays the graph.

2. **Plotting Training and Validation Accuracy**:

   - The code for plotting training and validation accuracy is similar to the one for plotting loss, but this time it focuses on the accuracy metric.

   - The line `plt.plot(history.history['accuracy'])` plots the training accuracy over each epoch on the graph.

   - The line `plt.plot(history.history['val_accuracy'])` plots the validation accuracy over each epoch on the same graph.

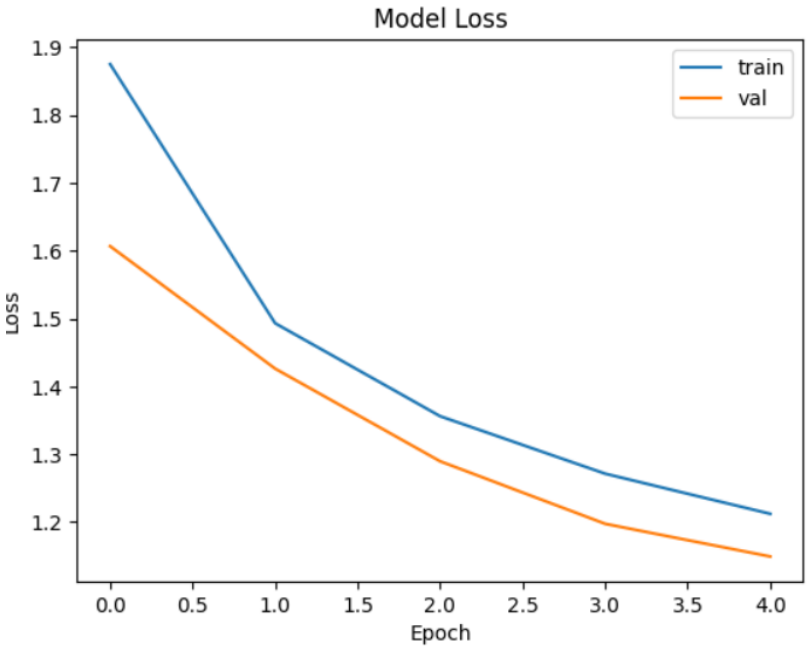
   - `history.history` contains the recorded accuracy values during training and validation.

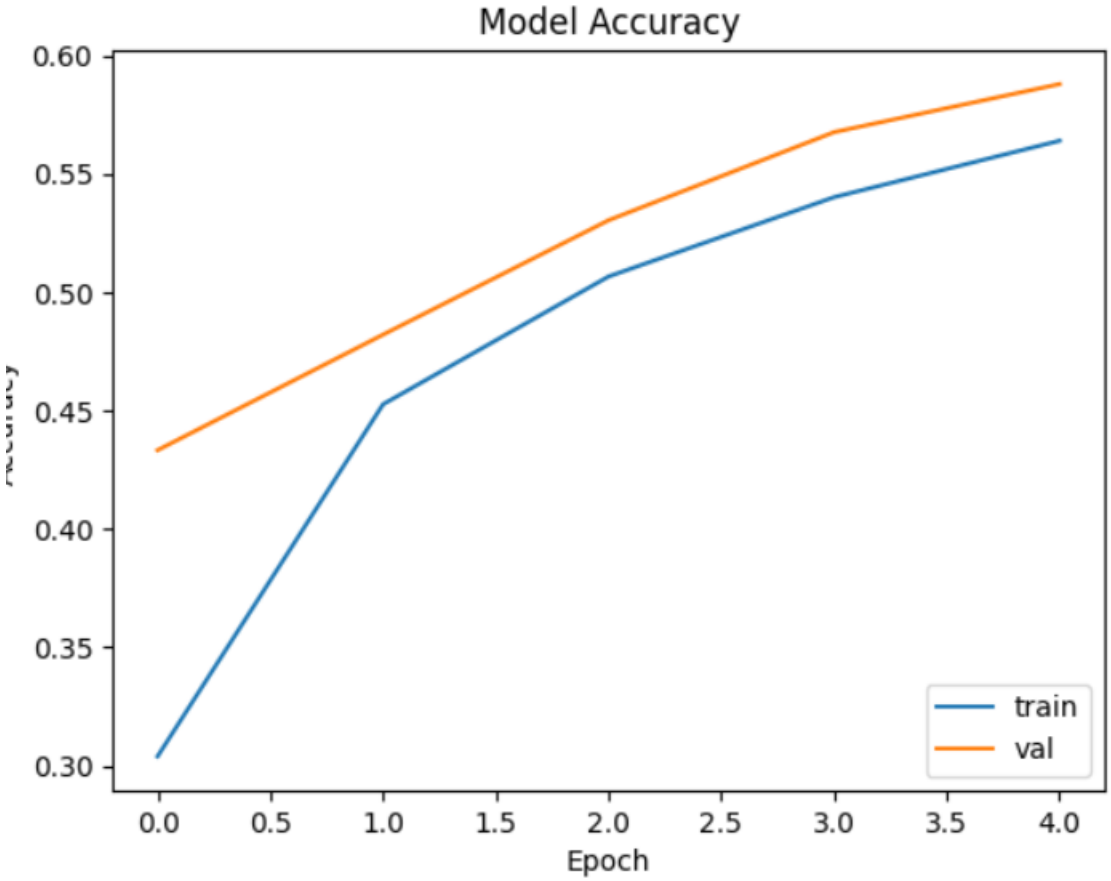
   - The `title`, `ylabel`, and `xlabel` functions set the title and labels for the graph.

   - The line `plt.legend(['train', 'val'], loc='lower right')` adds a legend to the graph, indicating which line corresponds to training accuracy and which one corresponds to validation accuracy.

   - Finally, `plt.show()` displays the graph.

Output:


Model Loss
train
val
Loss
Epoch
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Model Accuracy
train
val
Epoch
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0