1. Import Necessary Libraries

```
import pandas as pd
import numpy as np
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

2. Load and Preprocess Your Data

```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving nzmsa-2024.zip to nzmsa-2024.zip
```

```
import zipfile
# Replace 'your_zip_file.zip' with the name of your uploaded zip file
with zipfile.ZipFile('nzmsa-2024.zip', 'r') as zip_ref:
    zip_ref.extractall('nzmsa-2024 (1)')
```

```
extracted_files = os.listdir('nzmsa-2024 (1)')
print(extracted_files)
```

```
['__MACOSX', 'nzmsa-2024']
```

```
# Load the CSV files
train_df = pd.read_csv('nzmsa-2024 (1)/nzmsa-2024/train.csv')
sample_submission_df = pd.read_csv('nzmsa-2024 (1)/nzmsa-2024/sample_submission.csv')

# Specify directories for train and test images
train_images_dir = 'nzmsa-2024 (1)/nzmsa-2024/cifar10_images/train'
test_images_dir = 'nzmsa-2024 (1)/nzmsa-2024/cifar10_images/test'

# Function to load images from directory based on the dataframe ids
def load_images_from_dir(image_ids, directory, image_size=(32, 32)):
    images = []
    for image_id in image_ids:
        image_path = os.path.join(directory, f'image_{image_id}.png')
        image = load_img(image_path, target_size=image_size)
        image = img_to_array(image)
        images.append(image)
    return np.array(images)

# Load train and test images
X_train = load_images_from_dir(train_df['id'], train_images_dir)
y_train = to_categorical(train_df['label'], num_classes=10)


X_test = load_images_from_dir(sample_submission_df['id'], test_images_dir)


# Normalize the pixel values to improve the model's performance:
X_train = X_train / 255.0
X_test = X_test / 255.0
```

3. Build the Model

```
from tensorflow.keras.layers import BatchNormalization
# Build a CNN model
model = Sequential([
   Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Dropout(0.25),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam


model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

4. Train the Model

```
history = model.fit(X_train, y_train,
                    validation_split=0.2,
                    epochs=30,
                    batch_size=32)
```

```
Epoch 2/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 141ms/step - accuracy: 0.3363 - loss: 1.8046 - val_accuracy: 0.4320 - val_loss:
Epoch 3/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 141ms/step - accuracy: 0.4142 - loss: 1.6043 - val_accuracy: 0.4716 - val_loss:
Epoch 4/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 204s 143ms/step - accuracy: 0.4442 - loss: 1.5339 - val_accuracy: 0.4864 - val_loss:
Epoch 5/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 178s 142ms/step - accuracy: 0.4633 - loss: 1.4807 - val_accuracy: 0.5172 - val_loss:
Epoch 6/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 201s 142ms/step - accuracy: 0.4840 - loss: 1.4368 - val_accuracy: 0.5095 - val_loss:
Epoch 7/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 205s 144ms/step - accuracy: 0.4917 - loss: 1.4063 - val_accuracy: 0.5258 - val_loss:
Epoch 8/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 199s 142ms/step - accuracy: 0.5038 - loss: 1.3834 - val_accuracy: 0.5168 - val_loss:
Epoch 9/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 202s 142ms/step - accuracy: 0.5113 - loss: 1.3624 - val_accuracy: 0.5652 - val_loss:
Epoch 10/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 201s 141ms/step - accuracy: 0.5244 - loss: 1.3317 - val_accuracy: 0.5194 - val_loss:
Epoch 11/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 202s 141ms/step - accuracy: 0.5299 - loss: 1.3129 - val_accuracy: 0.5632 - val_loss:
Epoch 12/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 204s 143ms/step - accuracy: 0.5355 - loss: 1.2975 - val_accuracy: 0.5766 - val_loss:
Epoch 13/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 179s 143ms/step - accuracy: 0.5437 - loss: 1.2771 - val_accuracy: 0.5674 - val_loss:
Epoch 14/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 142ms/step - accuracy: 0.5494 - loss: 1.2643 - val_accuracy: 0.5672 - val_loss:
Epoch 15/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 204s 143ms/step - accuracy: 0.5484 - loss: 1.2553 - val_accuracy: 0.5800 - val_loss:
Epoch 16/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 200s 141ms/step - accuracy: 0.5518 - loss: 1.2484 - val_accuracy: 0.5731 - val_loss:
Epoch 17/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 201s 141ms/step - accuracy: 0.5619 - loss: 1.2250 - val_accuracy: 0.5895 - val_loss:
Epoch 18/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 203s 142ms/step - accuracy: 0.5691 - loss: 1.2082 - val_accuracy: 0.5855 - val_loss:
Epoch 19/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 141ms/step - accuracy: 0.5674 - loss: 1.2070 - val_accuracy: 0.5807 - val_loss:
Epoch 20/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 141ms/step - accuracy: 0.5777 - loss: 1.1923 - val_accuracy: 0.5932 - val_loss:
Epoch 21/30
```

```
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 176s 141ms/step - accuracy: 0.5888 - loss: 1.1645 - val_accuracy: 0.5982 - val_loss:
Epoch 23/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 206s 144ms/step - accuracy: 0.5839 - loss: 1.1639 - val_accuracy: 0.6022 - val_loss:
Epoch 24/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 179s 143ms/step - accuracy: 0.5873 - loss: 1.1589 - val_accuracy: 0.5966 - val_loss:
Epoch 25/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 202s 143ms/step - accuracy: 0.5932 - loss: 1.1530 - val_accuracy: 0.6098 - val_loss:
Epoch 26/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 202s 143ms/step - accuracy: 0.6024 - loss: 1.1152 - val_accuracy: 0.6199 - val_loss:
Epoch 27/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 200s 142ms/step - accuracy: 0.5962 - loss: 1.1251 - val_accuracy: 0.6137 - val_loss:
Epoch 28/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 201s 141ms/step - accuracy: 0.6018 - loss: 1.1231 - val_accuracy: 0.5872 - val_loss:
Epoch 29/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 177s 141ms/step - accuracy: 0.6089 - loss: 1.1091 - val_accuracy: 0.6098 - val_loss:
Epoch 30/30
1250/1250 ━━━━━━━━━━━━━━━━━━━━ 203s 142ms/step - accuracy: 0.6062 - loss: 1.0973 - val_accuracy: 0.6226 - val_loss:
```

5. Making Predictions

```
# CNN predictions
cnn_predictions = model.predict(X_test)
cnn_pred_labels = np.argmax(cnn_predictions, axis=1)


# Save predictions to CSV (e.g., for submission)
submission_df = pd.DataFrame({
    'id': sample_submission_df['id'],
    'label': cnn_pred_labels
})
submission_df.to_csv('cnn_predictions1.csv', index=False)
```

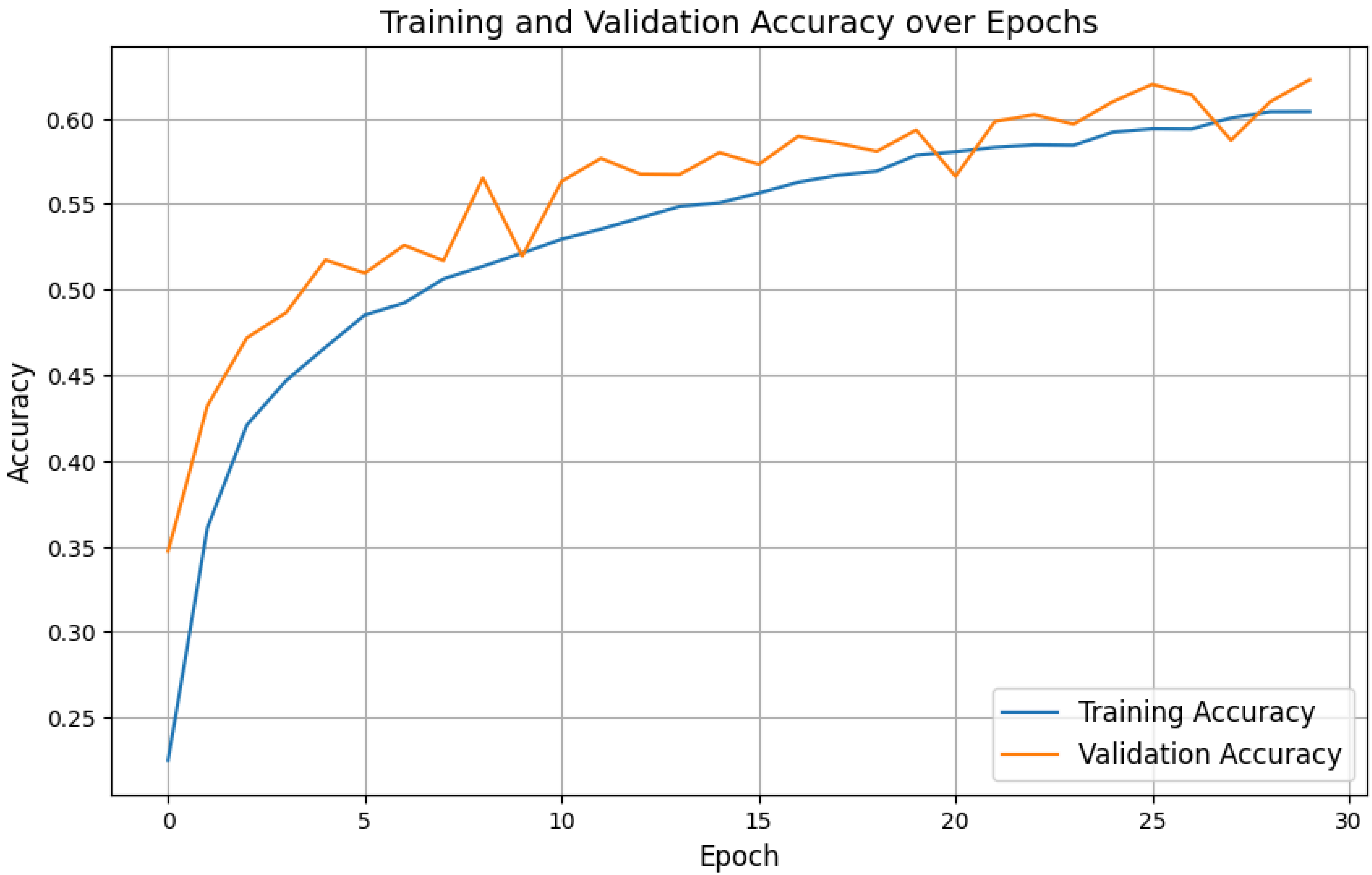
```
157/157 ━━━━━━━━━━━━━━━━━━━━ 12s 74ms/step
```

```
from google.colab import files

files.download('cnn_predictions1.csv')
```

6. Visualize Results

```
# Plotting Training and Validation Accuracy
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.title('Training and Validation Accuracy over Epochs', fontsize=14)
plt.legend(loc='lower right', fontsize=12)
plt.grid(True)
plt.show()
```



7. Summary

In this project, I built a Convolutional Neural Network (CNN) to classify images from the CIFAR-10 dataset. The model architecture includes:

Convolutional Layers: Two Conv2D layers with 64 and 128 filters, each followed by MaxPooling and Dropout layers to reduce overfitting.

Flatten Layer: Converts the 2D feature maps to a 1D vector.

Dense Layers: A Dense layer with 256 neurons and ReLU activation, followed by Dropout.

Output Layer: A Dense layer with 10 neurons and softmax activation for multi-class classification.

The model was compiled with the Adam optimizer and categorical cross-entropy loss, tracking accuracy during training. This setup is designed to effectively learn and classify the CIFAR-10 images, leveraging the CNN's ability to capture spatial features.