39_Sanskruti Nijai

EXPERIMENT NO 06

Design and implement a CNN model for digit recognition application.

```
[ ] import pandas as pd
```

```
[ ] train_data_dir  = 'train.csv'
    test_data_dir  = 'test.csv'
```

```
[ ] train_data = pd.read_csv(train_data_dir)
    test_data = pd.read_csv(test_data_dir)
```

```
[ ] train_data.head()
```

Now we want to see some images in the train file

```
[ ] import numpy as np
    import matplotlib.pyplot as plt
```

Randomly select 5 rows

```
[ ] num_samples = 5
    selected_rows = train_data.sample(num_samples)
```

Extract the labels and pixel values

```
[ ] labels = selected_rows['label']
    pixels = selected_rows.drop(columns=['label'])
```

Reshape pixel values into 28x28 images

```
[ ] images = pixels.values.reshape(-1, 28, 28)
```

Create a figure to display the images

```
[ ] plt.figure(figsize=(12, 5))

    <Figure size 1200x500 with 0 Axes>
    <Figure size 1200x500 with 0 Axes>
```

Plot the images

```
for i in range(num_samples):
    plt.subplot(1, num_samples, i + 1)
    plt.imshow(images[i], cmap='copper')
    plt.title(f"Label: {labels.iloc[i]}")
    plt.axis('off')
plt.show()
```

Now we want to Normalize and create model

```
[ ] import pandas as pd
    import numpy as np
    import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from sklearn.model_selection import train_test_split
```

Load our CSV files again

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')
```

Extract the pixel values and labels from the data

```
train_pixels = train_df.drop(columns=['label']).values
train_labels = train_df['label'].values
```

Normalize the pixel values to the range [0, 1]

```
train_pixels = train_pixels / 255.0
```

Reshape the pixel values to the appropriate shape for image data

```
train_images = train_pixels.reshape(-1, 28, 28, 1)
```

Split the data into training and validation sets

```
train_images, valid_images, train_labels, valid_labels = train_test_split(
    train_images,
    train_labels,
    test_size=0.2,
    random_state=42
)
```

Set the seed

```
tf.random.set_seed(42)
```

Create an ImageDataGenerator for data augmentation

```
train_datagen = ImageDataGenerator(rescale=1./255)
valid_datagen = ImageDataGenerator(rescale=1./255)
```

Create data generators from the preprocessed data

```
train_data = train_datagen.flow(
    x=train_images,
    y=train_labels,
    batch_size=32,
    seed=42
)
valid_data = valid_datagen.flow(
    x=valid_images,
    y=valid_labels,
    batch_size=32,
    seed=42
)
```

Create CNN model

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(filters=10,
                         kernel_size=3,
                         activation="relu",
                         input_shape=(28, 28, 1)),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(pool_size=2,
                            padding="valid"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.Conv2D(10, 3, activation="relu"),
  tf.keras.layers.MaxPool2D(2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(10, activation="softmax")
])
```

Compile the model

```
[ ] model.compile(loss="sparse_categorical_crossentropy",
                  optimizer=tf.keras.optimizers.Adam(),
                  metrics=["accuracy"])
```

Fit the model

```
[ ] history = model.fit(train_data,
                        epochs=5,
                        steps_per_epoch=len(train_data),
                        validation_data=valid_data,
                        validation_steps=len(valid_data))
```

```
Epoch 1/5
1050/1050 [==============================] - 34s 31ms/step - loss: 2.3017 - accuracy: 0.1118 - val_loss: 2.3011 - val_accuracy: 0.1082
Epoch 2/5
1050/1050 [==============================] - 30s 29ms/step - loss: 2.3014 - accuracy: 0.1124 - val_loss: 2.3008 - val_accuracy: 0.1082
Epoch 3/5
1050/1050 [==============================] - 30s 29ms/step - loss: 2.3015 - accuracy: 0.1124 - val_loss: 2.3011 - val_accuracy: 0.1082
Epoch 4/5
1050/1050 [==============================] - 32s 30ms/step - loss: 2.3014 - accuracy: 0.1124 - val_loss: 2.3010 - val_accuracy: 0.1082
Epoch 5/5
1050/1050 [==============================] - 32s 30ms/step - loss: 2.3015 - accuracy: 0.1124 - val_loss: 2.3012 - val_accuracy: 0.1082
```

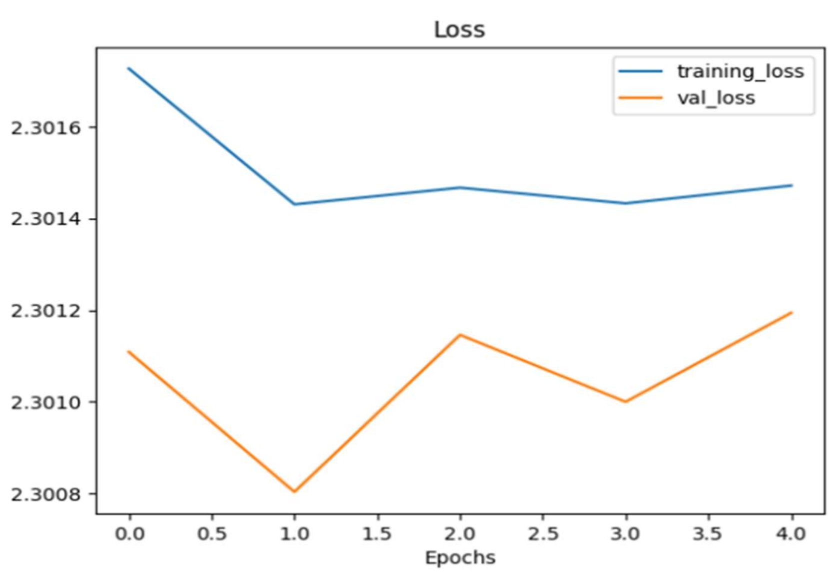
Now we want to plot training_loss, val_loss, training_accuracy and val_accuracy

```
import matplotlib.pyplot as plt

def plot_loss_curves(history):
  """
  Returns separate loss curves for training and validation metrics.
  """
  loss = history.history['loss']
  val_loss = history.history['val_loss']
```

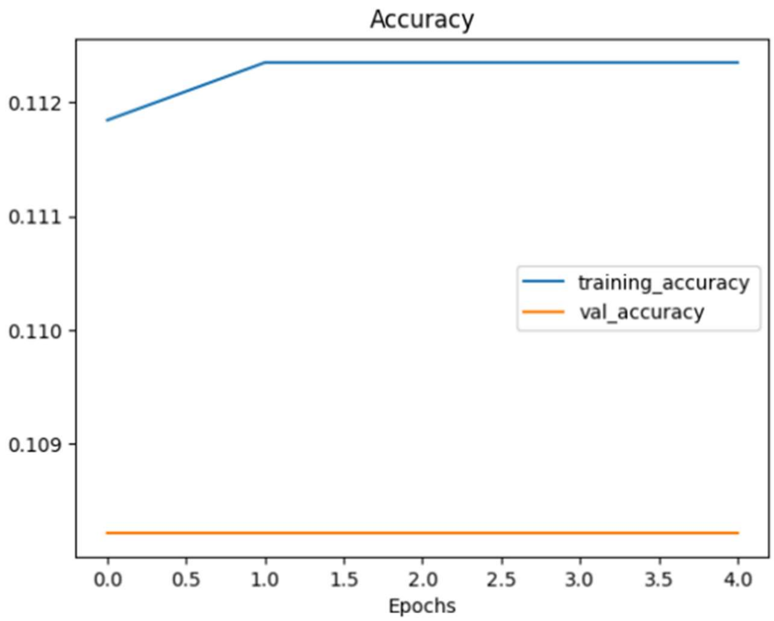
```
[ ]   accuracy = history.history['accuracy']
      val_accuracy = history.history['val_accuracy']

      epochs = range(len(history.history['loss']))

      # Plot loss
      plt.plot(epochs, loss, label='training_loss')
      plt.plot(epochs, val_loss, label='val_loss')
      plt.title('Loss')
      plt.xlabel('Epochs')
      plt.legend()

      # Plot accuracy
      plt.figure()
      plt.plot(epochs, accuracy, label='training_accuracy')
      plt.plot(epochs, val_accuracy, label='val_accuracy')
      plt.title('Accuracy')
      plt.xlabel('Epochs')
      plt.legend();
```

Check out the loss curves of model

```
plot_loss_curves(history)
```



[ ]



Now we want to predict on test data

Preprocess the test data

```
[ ]  test_pixels = test_df.values / 255.0  # Normalize the pixel values
     test_images = test_pixels.reshape(-1, 28, 28, 1)
```

Use the trained model to make predictions

```
[ ]  predictions = model.predict(test_images)

     875/875 [==============================] - 10s 11ms/step
```

Convert the predicted probabilities to class labels

```
[ ]  predicted_labels = np.argmax(predictions, axis=1)
```

Convert the predicted labels to a DataFrame with 'ImageId' and 'Label' columns

```
[ ]  image_ids = range(1, len(predicted_labels) + 1)
     submission_df = pd.DataFrame({'ImageId': image_ids, 'Label': predicted_labels})
```

Save the DataFrame to a CSV file

```
[ ]  submission_df.to_csv('submission.csv', index=False)
```

Because of we don't have the actual labels for the test data, we won't be able to calculate traditional evaluation metrics like accuracy, precision, recall, or F1-score, as these metrics require a ground truth for comparison.

However, we can still get an idea of how well your model is performing on the test data by doing ***Visual Inspection***

Visual Inspection: Take a look at some of the predicted labels and corresponding images to get a qualitative sense of the model's performance. We can use matplotlib to display the images and predicted labels.

```python
import matplotlib.pyplot as plt

# Display some random test images with their predicted labels
num_samples_to_display = 10
random_indices = np.random.choice(len(predicted_labels), num_samples_to_display, replace=False)

for i, idx in enumerate(random_indices):
    plt.subplot(2, 5, i + 1)
    plt.imshow(test_images[idx].reshape(28, 28), cmap='summer')
    plt.title(f'Predicted: {predicted_labels[idx]}')
    plt.axis('off')

plt.tight_layout()
plt.show()
```