Theo_Crandall_ML_Final_Project

May 2, 2023

Assignment: ML Final Project

Name: Theo Crandall

Description: A CNN that predicts the number drawn on a canvas.

[ ]:
```
# Import the necessary libraries
from scipy.io import loadmat
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Conv2D, MaxPooling2D,␣
↪Flatten, Dense
```

[ ]:
```
# Load the data
data = loadmat("mnist-original.mat")
X = data["data"].T
y = data["label"].T

# Reshape the data
X = X.reshape(-1, 28, 28, 1)
```

[ ]:
```
# Split the data into training and testing sets
test_percent = 0.2
test_size = int(test_percent * len(X))
X_train, X_test = X[:-test_size], X[-test_size:]
y_train, y_test = y[:-test_size], y[-test_size:]
```

Create the model

- The first layer is a convolutional layer with 32 filters, a kernel size of 3x3, and a ReLU activation function.
- The second layer is a max pooling layer with a pool size of 2x2. This reduces the image size.
- The third layer is a flattening layer. This flattens the image into a 1D array.
- The fourth layer is a dense layer with 128 neurons and a ReLU activation function.
- The fifth layer is a dense layer with 10 neurons and a softmax activation function. This is the output layer.

[ ]:
```
model = Sequential(
    [
        Conv2D(32, kernel_size=(3, 3), activation="relu",␣
↪input_shape=(28, 28, 1)),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation="relu"),
        Dense(10, activation="softmax"),
    ]
)
```

[ ]:
```
# Compile the model
model.compile(
    optimizer="adam", loss="sparse_categorical_crossentropy",␣
↪metrics=["accuracy"]
)
```

[ ]:
```
# Train the model for 10 epochs
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=2)
```

```
Epoch 1/10
1750/1750 - 28s - loss: 0.4279 - accuracy: 0.9377
Epoch 2/10
1750/1750 - 24s - loss: 0.0738 - accuracy: 0.9778
Epoch 3/10
1750/1750 - 24s - loss: 0.0519 - accuracy: 0.9837
Epoch 4/10
1750/1750 - 19s - loss: 0.0428 - accuracy: 0.9871
Epoch 5/10
1750/1750 - 25s - loss: 0.0342 - accuracy: 0.9892
Epoch 6/10
1750/1750 - 21s - loss: 0.0298 - accuracy: 0.9905
Epoch 7/10
1750/1750 - 23s - loss: 0.0199 - accuracy: 0.9939
Epoch 8/10
1750/1750 - 20s - loss: 0.0208 - accuracy: 0.9934
Epoch 9/10
1750/1750 - 16s - loss: 0.0179 - accuracy: 0.9947
Epoch 10/10
1750/1750 - 17s - loss: 0.0186 - accuracy: 0.9947
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x2953806a0>
```

[ ]:
```
# Evaluate the model
accuracy = model.evaluate(X_test, y_test, verbose=2)
```

```
438/438 - 3s - loss: 0.2525 - accuracy: 0.9640
```

[ ]:
```
# Save the model
model.save(f"models/mnist_model - {accuracy[1]:.4f}.h5")
```

Running the Model

Uses tkinter to create a drawing app that predicts the number drawn.

[ ]:
```python
# Import the necessary libraries
import tkinter as tk
from PIL import Image, ImageDraw
import numpy as np
from tensorflow.python.keras.models import load_model
```

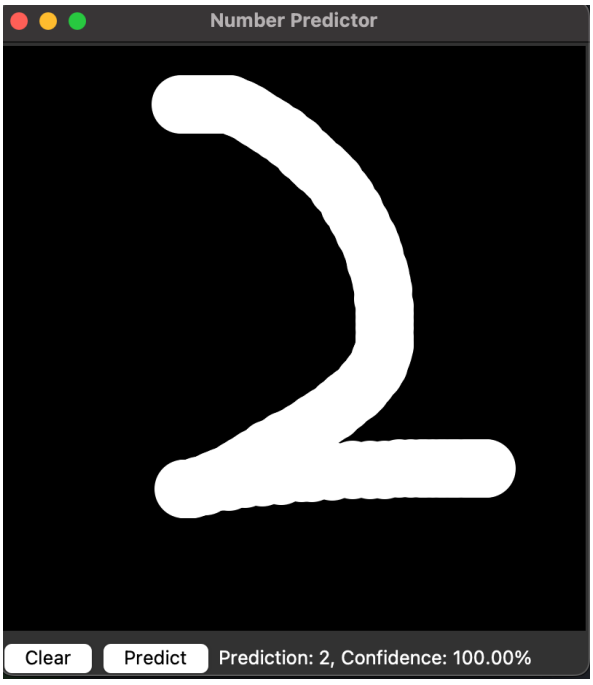
[ ]:
```python
# Select the model to use
model_name = "mnist_model - 0.9728"
```

[ ]:
```python
class App:
    def __init__(self, master: tk.Tk) -> None:
        """A drawing app that predicts the number drawn.

        Args:
            master (tk.Tk): The root window.
        """
        self.master = master
        master.title("Number Predictor")

        # Create the widgets
        self.create_canvas()
        self.create_buttons()
        self.create_prediction_box()

        # Load the model
        self.model = load_model(f"models/{model_name}.h5")

    def create_canvas(self) -> None:
        """Creates the canvas."""
        self.canvas = tk.Canvas(self.master, width=400, height=400,␣
↪bg="black")
        self.canvas.pack()
        self.img = Image.new("RGB", (400, 400), "black")
        self.imgDraw = ImageDraw.Draw(self.img)

        # Set up canvas bindings
        self.canvas.bind("<B1-Motion>", self.draw)

    def create_buttons(self) -> None:
        """Creates the clear and predict buttons."""
        self.clear_button = tk.Button(
            self.master, text="Clear", command=self.clear_canvas
        )
        self.clear_button.pack(side="left")

        self.predict_button = tk.Button(
            self.master, text="Predict", command=self.predict
        )
        self.predict_button.pack(side="left")
```

```
    def create_prediction_box(self) -> None:
        """Creates a box to display the prediction in."""
        self.prediction_box = tk.Label(self.master, text="Prediction:␣
↪None")
        self.prediction_box.pack(side="left")

    def update_prediction_text(self, num: int, confidence: float) -> None:
        """Updates the prediction text.

        Args:
            num (int): The number predicted.
            confidence (float): The confidence of the prediction.
        """
        self.prediction_box[
            "text"
        ] = f"Prediction: {num}, Confidence: {confidence * 100:.2f}%"

    def clear_canvas(self) -> None:
        """Clears the canvas."""
        self.canvas.delete("all")
        self.img = Image.new("RGB", (400, 400), "black")
        self.imgDraw = ImageDraw.Draw(self.img)

    def draw(self, event: tk.Event) -> None:
        """Draws a circle on the canvas.

        Args:
            event (tk.Event): The event that triggered the function.
        """
        r = 20
        x, y = event.x, event.y
        self.canvas.create_oval(x - r, y - r, x + r, y + r, fill="white",␣
↪outline="")
        self.imgDraw.ellipse((x - r, y - r, x + r, y + r), fill="white")

    def img_from_canvas(self) -> Image:
        """Creates an image from the canvas and performs the following␣
↪operations:
            - Grayscale
            - Resize to 28x28

        Returns:
            Image: The processed image.
        """
        self.img = self.img.convert("L")
        self.img = self.img.resize((28, 28))
        return self.img

    def img_to_array(self, img: Image) -> np.ndarray:
        """Converts an image to a numpy array. Using uint8 encoding.

        Args:
```

```
            img (Image): The image to convert.

        Returns:
            np.ndarray: The image as a numpy array.
        """
        return np.asarray(img, dtype=np.uint8).reshape(1, 28, 28, 1)

    def predict(self) -> None:
        """Predicts the number drawn on the canvas and displays it."""
        img = self.img_from_canvas()
        img_arr = self.img_to_array(img)
        prediction = self.model.predict(img_arr)
        self.update_prediction_text(np.argmax(prediction), np.
↪max(prediction))
```

[ ]:
```
# Run the app
root = tk.Tk()
app = App(root)
root.mainloop()
```



Sources

Using TensorFlow on M-series Macs: https://developer.apple.com/metal/tensorflow-plugin/
Kaggle Dataset: https://www.kaggle.com/datasets/avnishnish/mnist-original
GitHub Copilot: https://github.com/features/copilot
Tkinter docs: https://tkdocs.com/
TensorFlow docs: https://www.tensorflow.org/overview