In [1]:

```
#pip install numpy==1.26.4
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

In [3]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [4]:

```
import json
import cv2
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
```

In [5]:

```
#pip install --upgrade matplotlib
```

In [6]:

```
df = pd.read_csv('master_doodle_dataframe.csv')
#df = df.sample(n=1000, random_state=42)  # Sample the dataset
# List of words you want to keep
words_to_keep = ['apple', 'cat', 'dog', 'bird', 'hammer']

# Filter the DataFrame
filtered_df = df[df['word'].isin(words_to_keep)]

# Verify the filtering
print(filtered_df['word'].value_counts())  # Check the distribution of clas
df = filtered_df

print(df.shape)
df.head()
```

```
word
dog       3000
hammer    3000
cat       3000
apple     3000
bird      3000
Name: count, dtype: int64
(15000, 6)
```

Out[6]:

	countrycode	drawing	key_id	recognized	word	imag
417000	US	[[[187, 194, 193, 176, 171, 137, 102, 83, 73, ...	5736696387731456	True	dog	data/dog/57366963877314
417001	US	[[[33, 33, 35, 54, 53, 45, 49, 73, 73, 89, 87,...	4507372158451712	True	dog	data/dog/45073721584517
417002	US	[[[67, 61, 51, 38, 25, 11, 4, 0, 0, 3, 17, 30,...	5607059879886848	True	dog	data/dog/56070598798868
417003	US	[[[0, 8, 39, 99, 160, 180, 217, 235, 240, 254,...	6175971466018816	True	dog	data/dog/61759714660188
417004	SA	[[[162, 167, 189, 198, 206, 215, 219, 220, 217...	6286201801670656	True	dog	data/dog/62862018016706

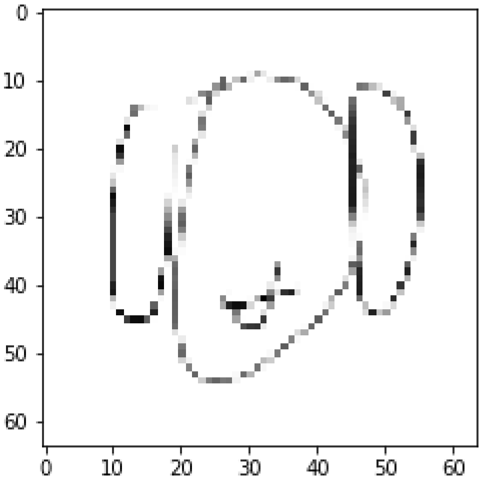
In [7]:

```
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image from the 'image_path' column
def load_images(image_paths, base_path='C:/Users/sweth/Swetha S U II MSc It
    base_path = os.path.expanduser(base_path)  # Expands '~' to the home di
    images = []
    for file_path in image_paths:
        full_path = os.path.join(base_path, file_path)  # Combine base_path

        # Print full path for debugging
        print(f"Trying to load image from: {full_path}")

        # Check if the file exists
        if os.path.exists(full_path):
            img = cv2.imread(full_path, cv2.IMREAD_GRAYSCALE)  # Load image
            if img is not None:
                img = cv2.resize(img, target_size)  # Resize the image to 6
                img = img / 255.0  # Normalize pixel values to [0, 1]
            else:
                print(f"Failed to load image from: {full_path}, returning b
                img = np.zeros(target_size)  # Return a blank image if fail
        else:
            print(f"File does not exist: {full_path}, returning blank image
            img = np.zeros(target_size)  # If the image cannot be loaded, r

        images.append(img)

    return np.array(images)

# Apply the function to the entire 'image_path' column
image_array = load_images(df['image_path'].values)

# Display an example image using plt.imshow
if len(image_array) > 10:  # Check if the image_array has enough elements
    plt.imshow(image_array[10], cmap='gray')
    plt.show()
else:
    print("Image array does not contain enough images.")
```

```
Trying to load image from: C:/Users/sweth/Swetha S U II MSc It/doodle/d
ata/dog/5736696387731456.png
File does not exist: C:/Users/sweth/Swetha S U II MSc It/doodle/data/do
g/5736696387731456.png, returning blank image.
Trying to load image from: C:/Users/sweth/Swetha S U II MSc It/doodle/d
ata/dog/4507372158451712.png
File does not exist: C:/Users/sweth/Swetha S U II MSc It/doodle/data/do
g/4507372158451712.png, returning blank image.
Trying to load image from: C:/Users/sweth/Swetha S U II MSc It/doodle/d
ata/dog/5607059879886848.png
File does not exist: C:/Users/sweth/Swetha S U II MSc It/doodle/data/do
g/5607059879886848.png, returning blank image.
Trying to load image from: C:/Users/sweth/Swetha S U II MSc It/doodle/d
ata/dog/6175971466018816.png
File does not exist: C:/Users/sweth/Swetha S U II MSc It/doodle/data/do
g/6175971466018816.png, returning blank image.
Trying to load image from: C:/Users/sweth/Swetha S U II MSc It/doodle/d
ata/dog/6286201801670656.png
File does not exist: C:/Users/sweth/Swetha S U II MSc It/doodle/data/do
```

In [8]:

```
df['image_path'] = df['image_path'].str.replace('data/', '', regex=False)

# Verify the change
#print(df['image_path'].head())

# Load image from the 'image_path' column
def load_images(image_paths, base_path='C:/Users/sweth/Swetha S U II MSc It
    images = []
    for file_path in image_paths:
        full_path = base_path + file_path
        img = cv2.imread(full_path, cv2.IMREAD_GRAYSCALE)  # Load as graysc
        if img is not None:
            img = cv2.resize(img, target_size)  # Resize the image to 64x64
            img = img / 255.0  # Normalize pixel values to [0, 1]
        else:
            img = np.zeros(target_size)  # If the image cannot be loaded, r
        images.append(img)
    return np.array(images)

# Apply the function to the entire 'image_path' column
image_array = load_images(df['image_path'].values)

# Display an example image using plt.imshow
plt.imshow(image_array[10], cmap='gray')
plt.show()
```



In [9]:

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
df['label'] = encoder.fit_transform(df['word'])
```

In [10]:

```
X = np.stack(image_array)  # Stack all image arrays into a single numpy arr

y = df['label'].values  # Labels
X.dtype
```

Out[10]:

```
dtype('float64')
```

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ra
```

In [12]:
```
from tensorflow.keras.utils import to_categorical

num_classes = len(words_to_keep)
y_train = to_categorical(y_train, num_classes=num_classes)
y_test = to_categorical(y_test, num_classes=num_classes)
```

In [20]:
```
y_train
```

Out[20]:
```
array([[1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0.]])
```

In [21]:

```
X_train
```

Out[21]:

```
array([[[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]],

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]],

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]],

       ...,

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]],

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]],

       [[1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        ...,
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.],
        [1., 1., 1., ..., 1., 1., 1.]]])
```

In [13]:

```
from tensorflow.keras import layers, models

model = models.Sequential()

# First Convolutional Layer
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64,
model.add(layers.MaxPooling2D((2, 2)))

# Second Convolutional Layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Third Convolutional Layer
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

# Flatten the layers before passing to the fully connected layers
model.add(layers.Flatten())

# Fully connected layer with 64 units
model.add(layers.Dense(64, activation='relu'))

# Output layer with softmax activation for multi-class classification
model.add(layers.Dense(num_classes, activation='softmax'))  # num_classes i

# Model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	
conv2d (Conv2D)	(None, 62, 62, 32)	
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	
conv2d_1 (Conv2D)	(None, 29, 29, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	
conv2d_2 (Conv2D)	(None, 12, 12, 64)	
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 64)	
flatten (Flatten)	(None, 2304)	
dense (Dense)	(None, 64)	
dense_1 (Dense)	(None, 5)	

Total params: 203,589 (795.27 KB)

Trainable params: 203,589 (795.27 KB)

Non-trainable params: 0 (0.00 B)

In [14]:
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [15]:
```
history = model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y
```

```
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 23s 51ms/step - accuracy: 0.5286 - loss: 1.10
19 - val_accuracy: 0.8210 - val_loss: 0.4566
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 49ms/step - accuracy: 0.8278 - loss: 0.46
61 - val_accuracy: 0.8707 - val_loss: 0.3563
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8755 - loss: 0.33
75 - val_accuracy: 0.8750 - val_loss: 0.3275
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.8907 - loss: 0.29
69 - val_accuracy: 0.9010 - val_loss: 0.2858
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.9145 - loss: 0.23
51 - val_accuracy: 0.9037 - val_loss: 0.2770
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 49ms/step - accuracy: 0.9148 - loss: 0.22
57 - val_accuracy: 0.9033 - val_loss: 0.2736
Epoch 7/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.9337 - loss: 0.18
38 - val_accuracy: 0.9070 - val_loss: 0.2815
Epoch 8/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 49ms/step - accuracy: 0.9415 - loss: 0.15
73 - val_accuracy: 0.9093 - val_loss: 0.2743
Epoch 9/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.9476 - loss: 0.13
71 - val_accuracy: 0.9033 - val_loss: 0.2907
Epoch 10/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 18s 48ms/step - accuracy: 0.9560 - loss: 0.11
84 - val_accuracy: 0.9097 - val_loss: 0.2943
```

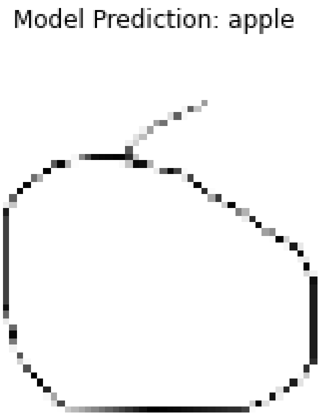
In [16]:
```
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test accuracy:', test_acc)
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 19ms/step - accuracy: 0.9087 - loss: 0.3022
Test accuracy: 0.9096666574478149
```

In [29]:

```
# Predict on the test set
predictions = model.predict(X_test)

# Convert predictions back to label form (from one-hot encoding)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = np.argmax(y_test, axis=1)

predicted_words = encoder.inverse_transform(predicted_labels)
true_words = encoder.inverse_transform(true_labels)

# Select an index to display
index = 2500  # We can Change this index to display a different image

# Reshape the image if necessary (assuming grayscale images of size 64x64)
plt.imshow(X_test[index].reshape(64, 64), cmap='gray')

# Set the title with the model's prediction
plt.title(f"Model Prediction: {predicted_words[index]}")
plt.axis('off')  # Hide axes for better visualization
plt.show()

# Print the true label and predicted label for verification
print(f"True Label: {true_words[index]}, Predicted Label: {predicted_words[
```

```
94/94 ━━━━━━━━━━━━━━━━━━━━ 2s 17ms/step
```



```
True Label: apple, Predicted Label: apple
```

In [18]:

```
model.save("doodle.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()`
or `keras.saving.save_model(model)`. This file format is considered legac
y. We recommend using instead the native Keras format, e.g. `model.save('m
y_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
```