Deploying Your Model

Now that we have a well trained model, it's time to use it. In this exercise, we'll expose new images to our model and detect the correct letters of the sign language alphabet. Let's get started!

Objectives

- Load an already-trained model from disk
- Reformat images for a model trained on images of a different format
- Perform inference with new images, never seen by the trained model and evaluate its performance

Loading the Model

Now that we're in a new notebook, let's load the saved model that we trained. Our save from the previous exercise created a folder called "asl_model". We can load the model by selecting the same folder.

In [ ]:

```
from tensorflow import keras

model = keras.models.load_model('asl_model')
```

If you'd like to make sure everything looks intact, you can see the summary of the model again.

In [ ]:

```
model.summary()
```

Preparing an Image for the Model

It's now time to use the model to make predictions on new images that it's never seen before. This is also called inference. We've given you a set of images in the data/asl_images folder. Try opening it using the left navigation and explore the images.

You'll notice that the images we have are much higher resolution than the images in our dataset. They are also in color. Remember that our images in the dataset were 28x28 pixels and grayscale. It's important to keep in mind that whenever you make predictions with a model, the input must match the shape of the data that the model was trained on. For this model, the training dataset was of the shape: (27455, 28, 28, 1). This corresponded to 27455 images of 28 by 28 pixels each with one color channel (grayscale).

Showing the Images

When we use our model to make predictions on new images, it will be useful to show the image as well. We can use the matplotlib library to do this.

In [ ]:

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

def show_image(image_path):
    image = mpimg.imread(image_path)
    plt.imshow(image, cmap='gray')
```

In [ ]:

```
show_image('data/asl_images/b.png')
```

Scaling the Images

The images in our dataset were 28x28 pixels and grayscale. We need to make sure to pass the same size and grayscale images into our method for prediction. There are a few ways to edit images with Python, but Keras has a built-in utility that works well.

In [ ]:

```
from tensorflow.keras.preprocessing import image as image_utils

def load_and_scale_image(image_path):
    image = image_utils.load_img(image_path, color_mode="grayscale", target_size=(28,28))
    return image
```

In [ ]:

```
image = load_and_scale_image('data/asl_images/b.png')
plt.imshow(image, cmap='gray')
```

Preparing the Image for Prediction

Now that we have a 28x28 pixel grayscale image, we're close to being ready to pass it into our model for prediction. First we need to reshape our image to match the shape of the dataset the model was trained on. Before we can reshape, we need to convert our image into a more rudimentary format. We'll do this with a keras utility called image_to_array.

In [ ]:

```
image = image_utils.img_to_array(image)
```

Now we can reshape our image to get it ready for prediction.

In [ ]:

```
# This reshape corresponds to 1 image of 28x28 pixels with one color channel
image = image.reshape(1,28,28,1)
```

Finally, we should remember to normalize our data (making all values between 0-1), as we did with our training dataset:

In [ ]:

```
image = image / 255
```

Making Predictions

Okay, now we're ready to predict! This is done by passing our pre-processed image into the model's predict method.

In [ ]:

```
prediction = model.predict(image)
print(prediction)
```

Understanding the Prediction

The predictions are in the format of a 24 length array. Though it looks a bit different, this is the same format as our "binarized" categorical arrays from y_train and y_test. Each element of the array is a probability between 0 and 1, representing the confidence for each category. Let's make it a little more readable. We can start by finding which element of the array represents the highest probability. This can be done easily with the numpy library and the argmax (https://numpy.org/doc/stable/reference/generated/numpy.argmax.html) function.

In [ ]:

```
import numpy as np
np.argmax(prediction)
```

Each element of the prediction array represents a possible letter in the sign language alphabet. Remember that j and z are not options because they involve moving the hand, and we're only dealing with still photos. Let's create a mapping between the index of the predictions array, and the corresponding letter.

In [ ]:

```
# Alphabet does not contain j or z because they require movement
alphabet = "abcdefghiklmnopqrstuvwxy"
```

We can now pass in our prediction index to find the corresponding letter.

In [ ]:

```
alphabet[np.argmax(prediction)]
```

Exercise: Put it all Together

Let's put everything in a function so that we can make predictions just from the image file. Implement it in the function below using the functions and steps above. If you need help, you can reveal the solution by clicking the three dots below.

In [ ]:

```
def predict_letter(file_path):
    # Show image
    FIXME
    # Load and scale image
    image = FIXME
    # Convert to array
    image = FIXME
    # Reshape image
    image = FIXME
    # Normalize image
    image = FIXME
    # Make prediction
    prediction = FIXME
    # Convert prediction to letter
    predicted_letter = FIXME
    # Return prediction
    return predicted_letter
```

Solution

Click on the '...' below to view the solution.

```
def predict_letter(file_path):
    show_image(file_path)
    image = load_and_scale_image(file_path)
    image = image_utils.img_to_array(image)
    image = image.reshape(1,28,28,1)
    image = image/255
    prediction = model.predict(image)
    # convert prediction to letter
    predicted_letter = alphabet[np.argmax(prediction)]
    return predicted_letter
```

In [ ]:

```
predict_letter("data/asl_images/b.png")
```

Let's also use the function with the 'a' letter in the asl_images datset:

In [ ]:

```
predict_letter("data/asl_images/a.png")
```

Summary

Great work on these exercises! You've gone through the full process of training a highly accurate model from scratch, and then using the model to make new and valuable predictions. If you have some time, we encourage you to take pictures with your webcam, upload them by dropping them into the data/asl_images folder, and test out the model on them. For Mac you can use Photo Booth. For windows you can select the Camera app from your start screen. We hope you try it. It's a good opportunity to learn some sign language! For instance, try out the letters of your name.

We can imagine how this model could be used in an application to teach someone sign language, or even help someone who cannot speak interact with a computer. If you're comfortable with web development, models can even be used in the browser with a library called TensorFlow.js (https://www.tensorflow.org/js).

Clear the Memory

Before moving on, please execute the following cell to clear up the GPU memory.

In [ ]:

```
import IPython
app = IPython.Application.instance()
app.kernel.do_shutdown(True)
```

Next

We hope you've enjoyed these exercises. In the next sections we will learn how to take advantage of deep learning when we don't have a robust dataset available. See you there! To learn more about inference on the edge, check out this nice paper (http://web.eecs.umich.edu/~mosharaf/Readings/FB-ML-Edge.pdf) on the topic.

Now that we're familiar building your own models and have some understanding of how they work, we will turn our attention to the very powerful technique of using pre-trained models to expedite your work.