Transfer Learning

In this notebook, you will perform transfer learning to train CIFAR-10 dataset on ResNet50 model available in Keras.

Imports

```
[1] import os, re, time, json
    import PIL.Image, PIL.ImageFont, PIL.ImageDraw
    import numpy as np
    try:
      # %tensorflow_version only exists in Colab.
      %tensorflow_version 2.x
    except Exception:
      pass
    import tensorflow as tf
    from tensorflow.keras.applications.resnet50 import ResNet50
    from matplotlib import pyplot as plt
    import tensorflow_datasets as tfds

    print("Tensorflow version " + tf.__version__)
```

```
Tensorflow version 2.5.0
```

Parameters

- Define the batch size
- Define the class (category) names

```
[2] BATCH_SIZE = 32
    classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

Define some functions that will help you to create some visualizations. (These will be used later)

```
[3] #@title Visualization Utilities[RUN ME]
    #Matplotlib config
    plt.rc('image', cmap='gray')
    plt.rc('grid', linewidth=0)
    plt.rc('xtick', top=False, bottom=False, labelsize='large')
    plt.rc('ytick', left=False, right=False, labelsize='large')
    plt.rc('axes', facecolor='F8F8F8', titlesize="large", edgecolor='white')
    plt.rc('text', color='a8151a')
    plt.rc('figure', facecolor='F0F0F0')# Matplotlib fonts
    MATPLOTLIB_FONT_DIR = os.path.join(os.path.dirname(plt.__file__), "mpl-data/fonts/ttf")
    # utility to display a row of digits with their predictions
    def display_images(digits, predictions, labels, title):

      n = 10

      indexes = np.random.choice(len(predictions), size=n)
      n_digits = digits[indexes]
      n_predictions = predictions[indexes]
      n_predictions = n_predictions.reshape((n,))
      n_labels = labels[indexes]

      fig = plt.figure(figsize=(20, 4))
      plt.title(title)
      plt.yticks([])
      plt.xticks([])

      for i in range(10):
        ax = fig.add_subplot(1, 10, i+1)
        class_index = n_predictions[i]

        plt.xlabel(classes[class_index])
        plt.xticks([])
        plt.yticks([])
        plt.imshow(n_digits[i])

    # utility to display training and validation curves
    def plot_metrics(metric_name, title, ylim=5):
      plt.title(title)
      plt.ylim(0,ylim)
      plt.plot(history.history[metric_name],color='blue',label=metric_name)
      plt.plot(history.history['val_' + metric_name],color='green',label='val_' + metric_name)
```

Visualization Utilities[RUN ME]

Loading and Preprocessing Data

CIFAR-10 dataset has 32 x 32 RGB images belonging to 10 classes. You will load the dataset from Keras.

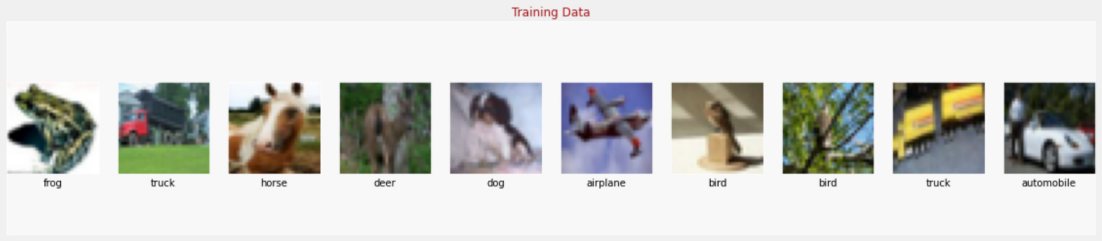
```
[4] (training_images, training_labels) , (validation_images, validation_labels) = tf.keras.datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
```

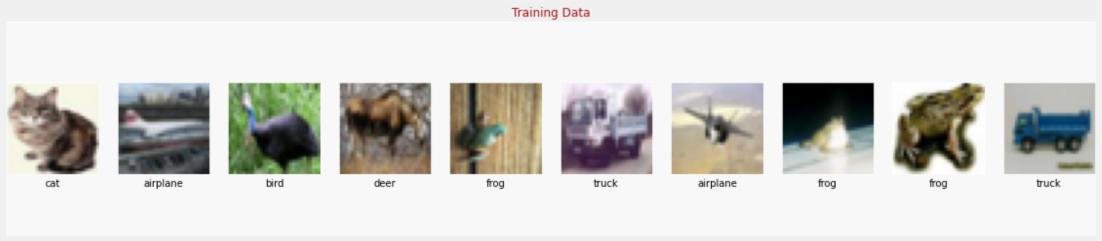
Visualize Dataset

Use the `display_image` to view some of the images and their class labels.

```
[5] display_images(training_images, training_labels, training_labels, "Training Data" )
```




```
[6] display_images(validation_images, validation_labels, validation_labels, "Training Data" )
```



Preprocess Dataset

Here, you'll perform normalization on images in training and validation set.

- You'll use the function preprocess_input from the ResNet50 model in Keras.

```
[7] def preprocess_image_input(input_images):
      input_images = input_images.astype('float32')
      output_ims = tf.keras.applications.resnet50.preprocess_input(input_images)
      return output_ims
```

```
[8] train_X = preprocess_image_input(training_images)
    valid_X = preprocess_image_input(validation_images)
```

Define the Network

You will be performing transfer learning on **ResNet50** available in Keras.

- You'll load pre-trained **imagenet weights** to the model.
- You'll choose to retain all layers of **ResNet50** along with the final classification layers.

```
[9] '''
    Feature Extraction is performed by ResNet50 pretrained on imagenet weights.
    Input size is 224 x 224.
    '''
    def feature_extractor(inputs):

      feature_extractor = tf.keras.applications.resnet.ResNet50(input_shape=(224, 224, 3),
                                                  include_top=False,
                                                  weights='imagenet')(inputs)
      return feature_extractor


    '''
    Defines final dense layers and subsequent softmax layer for classification.
    '''
    def classifier(inputs):
        x = tf.keras.layers.GlobalAveragePooling2D()(inputs)
        x = tf.keras.layers.Flatten()(x)
        x = tf.keras.layers.Dense(1024, activation="relu")(x)
        x = tf.keras.layers.Dense(512, activation="relu")(x)
        x = tf.keras.layers.Dense(10, activation="softmax", name="classification")(x)
        return x

    '''
    Since input image size is (32 x 32), first upsample the image by factor of (7x7) to transform it to (224 x 224)
    Connect the feature extraction and "classifier" layers to build the model.
    '''
    def final_model(inputs):

        resize = tf.keras.layers.UpSampling2D(size=(7,7))(inputs)

        resnet_feature_extractor = feature_extractor(resize)
        classification_output = classifier(resnet_feature_extractor)

        return classification_output

    '''
    Define the model and compile it.
    Use Stochastic Gradient Descent as the optimizer.
    Use Sparse Categorical CrossEntropy as the loss function.
    '''
    def define_compile_model():
      inputs = tf.keras.layers.Input(shape=(32,32,3))

      classification_output = final_model(inputs)
      model = tf.keras.Model(inputs=inputs, outputs = classification_output)

      model.compile(optimizer='SGD',
```

```
                  loss='sparse_categorical_crossentropy',
                  metrics = ['accuracy'])

    return model


model = define_compile_model()

model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94773248/94765736 [==============================] - 1s 0us/step
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 224, 224, 3)       0
_________________________________________________________________
resnet50 (Functional)        (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 1024)              2098176
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
classification (Dense)       (None, 10)                5130
=================================================================
Total params: 26,215,818
Trainable params: 26,162,698
Non-trainable params: 53,120
_________________________________________________________________
```

Train the model

```
[10] # this will take around 20 minutes to complete
     EPOCHS = 4
     history = model.fit(train_X, training_labels, epochs=EPOCHS, validation_data = (valid_X, validation_labels), batch_size=64)
```

```
Epoch 1/4
782/782 [==============================] - 529s 628ms/step - loss: 0.4005 - accuracy: 0.8692 - val_loss: 0.2145 - val_accuracy: 0.9266
Epoch 2/4
782/782 [==============================] - 494s 632ms/step - loss: 0.1039 - accuracy: 0.9664 - val_loss: 0.1739 - val_accuracy: 0.9396
Epoch 3/4
782/782 [==============================] - 494s 632ms/step - loss: 0.0368 - accuracy: 0.9897 - val_loss: 0.1609 - val_accuracy: 0.9494
Epoch 4/4
782/782 [==============================] - 494s 632ms/step - loss: 0.0146 - accuracy: 0.9967 - val_loss: 0.2903 - val_accuracy: 0.9297
```

Evaluate the Model

Calculate the loss and accuracy metrics using the model's `.evaluate` function.

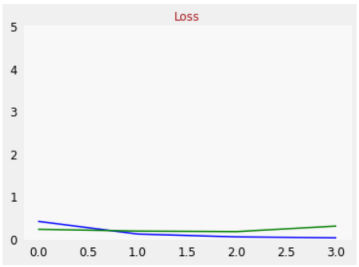
```
[11] loss, accuracy = model.evaluate(valid_X, validation_labels, batch_size=64)
```

```
157/157 [==============================] - 28s 176ms/step - loss: 0.2903 - accuracy: 0.9297
```

Plot Loss and Accuracy Curves

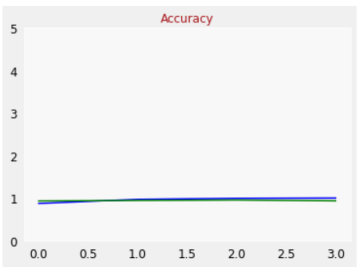
Plot the loss (in blue) and validation loss (in green).

```
[12] plot_metrics("loss", "Loss")
```



Plot the training accuracy (blue) as well as the validation accuracy (green).

```
[13] plot_metrics("accuracy", "Accuracy")
```



Visualize predictions

You can take a look at the predictions on the validation set.

```
probabilities = model.predict(valid_X, batch_size=64)
probabilities = np.argmax(probabilities, axis = 1)

display_images(validation_images, probabilities, validation_labels, "Bad predictions indicated in red.")
```

29s completed at 9:11 PM