CS156 (Introduction to AI), Fall 2022

Homework 8 submission

Roster Name: Preet LNU

Student ID: 014755741

Email address: preet.lnu@sjsu.edu

References and sources

List all your references and sources here. This includes all sites/discussion boards/blogs/posts/etc. where you grabbed some code examples.

Solution

Load libraries and set random number generator seed

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import os
import matplotlib.pyplot as plt
from skimage import io
import numpy as np

from zipfile import ZipFile


np.random.seed(42)
```

Code the solution

```
with ZipFile('/content/homework8_input_data.zip', 'r') as zipObj:
  zipObj.extractall('/content')
```

```
image_size = (180, 180)
batch_size = 32

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/flowers/training",
    validation_split=0.2,
    subset="training",
    seed=42,

    labels='inferred',
    label_mode='categorical',

    image_size=image_size,
    batch_size=batch_size,
)
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/flowers/training",
    validation_split=0.2,
    subset="validation",
    seed=42,

    labels='inferred',
    label_mode='categorical',

    image_size=image_size,
    batch_size=batch_size,
)
test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/flowers/test",
    seed=42,

    labels='inferred',
    label_mode='categorical',

    image_size=image_size,
    batch_size=1,
)
```

```
Found 3456 files belonging to 5 classes.
Using 2765 files for training.
Found 3456 files belonging to 5 classes.
Using 691 files for validation.
Found 861 files belonging to 5 classes.
```

```
data_augmentation = keras.Sequential(
    [
        layers.experimental.preprocessing.RandomFlip("horizontal"),
        layers.experimental.preprocessing.RandomRotation(0.1),
    ]
)
```

```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")
```

```
train_ds = train_ds.prefetch(buffer_size=32)
val_ds = val_ds.prefetch(buffer_size=32)

def make_model(input_shape, num_classes):
    inputs = keras.Input(shape=input_shape)
    # Image augmentation block
    x = data_augmentation(inputs)

    # Entry block
    x = layers.experimental.preprocessing.Rescaling(1.0 / 255)(x)
```

```
    x = layers.Conv2D(32, 3, strides=2, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.Conv2D(64, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    previous_block_activation = x  # Set aside residual

    for size in [128, 256, 512, 728]:
        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.Activation("relu")(x)
        x = layers.SeparableConv2D(size, 3, padding="same")(x)
        x = layers.BatchNormalization()(x)

        x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

        # Project residual
        residual = layers.Conv2D(size, 1, strides=2, padding="same")(
            previous_block_activation
        )
        x = layers.add([x, residual])  # Add back residual
        previous_block_activation = x  # Set aside next residual

    x = layers.SeparableConv2D(1024, 3, padding="same")(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)

    x = layers.GlobalAveragePooling2D()(x)
    if num_classes == 2:
        activation = "sigmoid"
        units = 1
    else:
        activation = "softmax"
        units = num_classes

    x = layers.Dropout(0.5)(x)
    outputs = layers.Dense(units, activation=activation)(x)
    return keras.Model(inputs, outputs)


model = make_model(input_shape=image_size + (3,), num_classes=5)
#keras.utils.plot_model(model, show_shapes=True)
model.summary()
```

```
 separable_conv2d_5 (SeparableC  (None, 23, 23, 512)  267264     ['activation_7[
 onv2D)
```

```
 batch_normalization_7 (BatchNo  (None, 23, 23, 512)  2048        ['separable_con
 rmalization)

 max_pooling2d_2 (MaxPooling2D)  (None, 12, 12, 512)  0           ['batch_normali

 conv2d_4 (Conv2D)              (None, 12, 12, 512)  131584      ['add_1[0][0]']

 add_2 (Add)                    (None, 12, 12, 512)  0           ['max_pooling2d
                                                                  'conv2d_4[0][0

 activation_8 (Activation)      (None, 12, 12, 512)  0           ['add_2[0][0]']

 separable_conv2d_6 (SeparableC  (None, 12, 12, 728)  378072      ['activation_8[
 onv2D)

 batch_normalization_8 (BatchNo  (None, 12, 12, 728)  2912        ['separable_con
 rmalization)

 activation_9 (Activation)      (None, 12, 12, 728)  0           ['batch_normali

 separable_conv2d_7 (SeparableC  (None, 12, 12, 728)  537264      ['activation_9[
 onv2D)

 batch_normalization_9 (BatchNo  (None, 12, 12, 728)  2912        ['separable_con
 rmalization)

 max_pooling2d_3 (MaxPooling2D)  (None, 6, 6, 728)   0           ['batch_normali

 conv2d_5 (Conv2D)              (None, 6, 6, 728)    373464      ['add_2[0][0]']

 add_3 (Add)                    (None, 6, 6, 728)    0           ['max_pooling2d
                                                                  'conv2d_5[0][0

 separable_conv2d_8 (SeparableC  (None, 6, 6, 1024)  753048      ['add_3[0][0]']
 onv2D)

 batch_normalization_10 (BatchN  (None, 6, 6, 1024)  4096        ['separable_con
 ormalization)

 activation_10 (Activation)     (None, 6, 6, 1024)   0           ['batch_normali

 global_average_pooling2d (Glob  (None, 1024)        0           ['activation_10
 alAveragePooling2D)

 dropout (Dropout)              (None, 1024)         0           ['global_averag
                                                                 ]

 dense (Dense)                  (None, 5)            5125        ['dropout[0][0]

================================================================================
Total params: 2,786,749
Trainable params: 2,778,013
Non-trainable params: 8,736
________________________________________________________________________________
```

```
epochs = 20

callbacks = [
    keras.callbacks.ModelCheckpoint("save_at_{epoch}.h5"),
]
model.compile(
    optimizer=keras.optimizers.Adam(1e-3),
    loss="categorical_crossentropy",
    metrics=["accuracy"],
)
model.fit(
    train_ds, epochs=epochs, callbacks=callbacks, validation_data=val_ds,
)
```

```
Epoch 1/20
87/87 [==============================] - 818s 9s/step - loss: 1.2242 - accuracy:
Epoch 2/20
87/87 [==============================] - 806s 9s/step - loss: 1.0021 - accuracy:
Epoch 3/20
87/87 [==============================] - 808s 9s/step - loss: 0.8992 - accuracy:
Epoch 4/20
87/87 [==============================] - 811s 9s/step - loss: 0.8134 - accuracy:
Epoch 5/20
87/87 [==============================] - 809s 9s/step - loss: 0.7778 - accuracy:
Epoch 6/20
87/87 [==============================] - 821s 9s/step - loss: 0.7100 - accuracy:
Epoch 7/20
87/87 [==============================] - 818s 9s/step - loss: 0.6534 - accuracy:
Epoch 8/20
87/87 [==============================] - 811s 9s/step - loss: 0.6602 - accuracy:
Epoch 9/20
87/87 [==============================] - 811s 9s/step - loss: 0.6017 - accuracy:
Epoch 10/20
87/87 [==============================] - 813s 9s/step - loss: 0.5778 - accuracy:
Epoch 11/20
87/87 [==============================] - 811s 9s/step - loss: 0.5652 - accuracy:
Epoch 12/20
87/87 [==============================] - 811s 9s/step - loss: 0.5201 - accuracy:
Epoch 13/20
87/87 [==============================] - 810s 9s/step - loss: 0.5102 - accuracy:
Epoch 14/20
87/87 [==============================] - 797s 9s/step - loss: 0.4841 - accuracy:
Epoch 15/20
87/87 [==============================] - 774s 9s/step - loss: 0.4753 - accuracy:
Epoch 16/20
87/87 [==============================] - 807s 9s/step - loss: 0.4548 - accuracy:
Epoch 17/20
87/87 [==============================] - 807s 9s/step - loss: 0.4222 - accuracy:
Epoch 18/20
87/87 [==============================] - 808s 9s/step - loss: 0.4424 - accuracy:
Epoch 19/20
87/87 [==============================] - 817s 9s/step - loss: 0.4041 - accuracy:
Epoch 20/20
87/87 [==============================] - 808s 9s/step - loss: 0.4102 - accuracy:
<keras.callbacks.History at 0x7f6021cf4810>
```

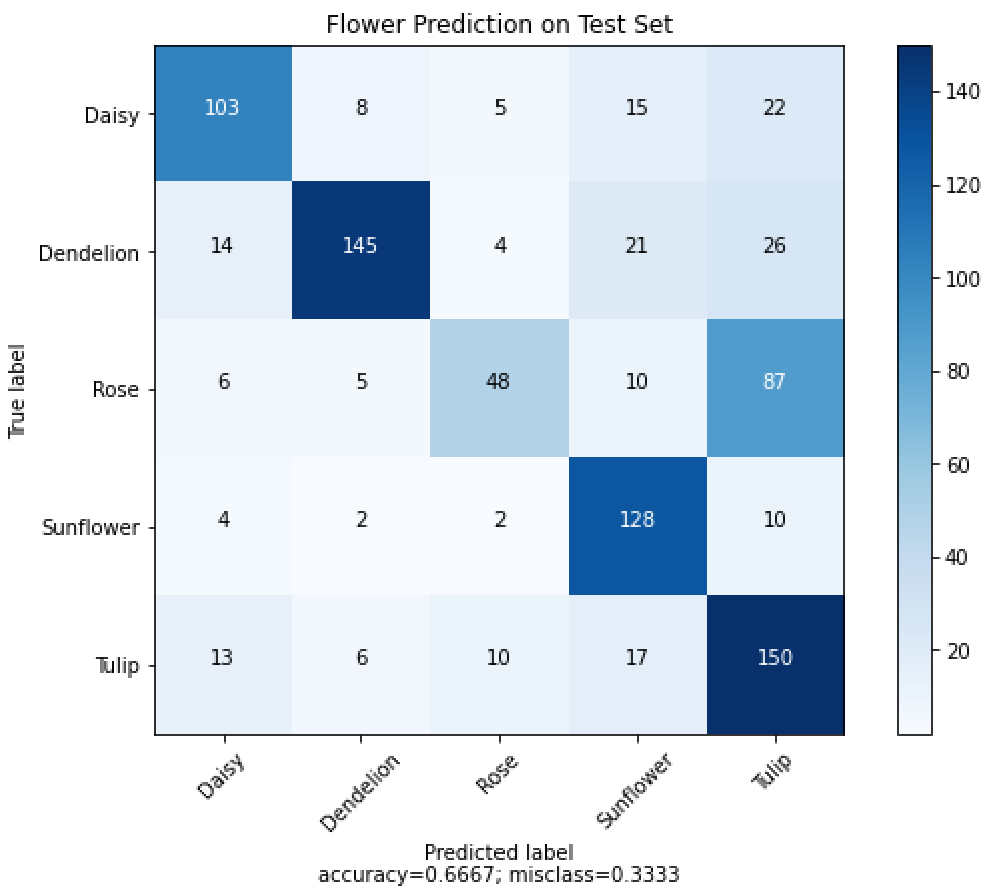
```
true_labels = []
predicted_labels = []
#x = image, y = label
for x, y in test_ds:
    pred = model.predict(x)
    true_labels.append(np.where(y == 1.)[1][0])
    predicted_labels.append(np.where(pred == np.amax(pred))[1][0])
```

```
1/1 [==============================] - 1s 600ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 103ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 133ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 90ms/step
1/1 [==============================] - 0s 91ms/step
```

```
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 81ms/step
1/1 [==============================] - 0s 82ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 85ms/step
1/1 [==============================] - 0s 86ms/step
```

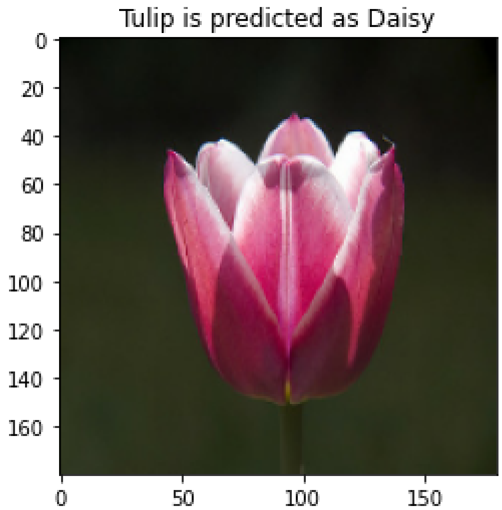
```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):

    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
```

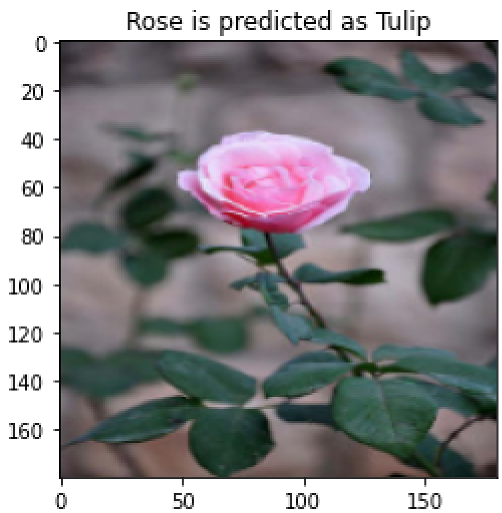
```
                         color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy,
    plt.show()


plot_confusion_matrix(cm = tf.math.confusion_matrix(labels=true_labels, predictions=p
                      normalize    = False,
                      target_names = ['Daisy', 'Dendelion', 'Rose', 'Sunflower', 'Tul
                      title        = "Flower Prediction on Test Set")
```




```
def target_translator (input_number) :

    if (input_number == 0) :
        return 'Daisy'

    elif (input_number == 1) :
        return 'Dendelion'

    elif (input_number == 2) :
        return 'Rose'

    elif (input_number == 3) :
```

```
            return 'Sunflower'

        elif (input_number == 4) :
            return 'Tulip'


    breaker = 0


    for counter in range (100) :
        if (true_labels[counter] != predicted_labels[counter]) :

            plt.title(target_translator(true_labels[counter]) + ' is predicted as ' + tar

            plt.imshow(images[counter].numpy().astype("uint8"))

            plt.show()

            breaker = breaker + 1

        if (breaker > 2) :
            break
```





✓ 0s completed at 10:11 PM