PRACTICAL 12

Transfer Learning via Feature Extraction

ACTIVITY - Submission Required!

In this exercise, we will use the transfer learning concept as the feature extractor for the classification of handwritten Dzongkha digits. In this tutorial, we will use VGG16 networks to extract features from the custom handwritten digits.

VGG16 is a convolution neural network that is 16 layers deep. The pretrained network can classify images into 1000 object categories found in ImageNet. The network has learned rich feature representations for a wide range of images. The network has an image input size of 224 by 224 (RGC images).

NOTE: *Use the same **Dzo_MINST** dataset.*

1. Open a new Jupyter Notebook file.
2. Mount the Google Drive to load the **Dzo_MNIST** dataset.

```
...
```

3. Import the required libraries.

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from matplotlib import pyplot as plt
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
```

4. Create **train_datagen** and **val_datagen**. For **train_datagen**, use the following data augmentation:
    a. `resale = 1./255`
    b. `shear_range = 0.1`
    c. `rotation_range=10`
    d. `zoom_range=0.1`
    e. `validation_split=0.1`

```
.....
```

5. Create **train_generator** and **validation_generator**.

```
train_generator = train_datagen.flow_from_directory(
    path + "Dzo_MNIST",
```

```
    target_size = (128, 128),
    batch_size = 32,
    classes = ['0', '1', '2'],
    class_mode = 'categorical',
    color_mode = 'rgb',
    subset = 'training') # set as training data

validation_generator = val_datagen.flow_from_directory(
     path + "Dzo_MNIST", # same directory as training data
    target_size = (128, 128),
    batch_size=32,
    classes = ['0', '1', '2'],
    class_mode='categorical',
    color_mode = 'rgb',
    subset = 'validation') # set as validation data
```

6. Load the **VGG16** network

Load the VGG16 architecture with the pre-trained ImageNet weights leaving off the fully connected layers.

```
baseModel = VGG16(weights="imagenet", include_top=False, input_tensor =
Input(shape = (128, 128, 3)))
```

The original VGG16 network architecture is trained on ImageNet having an image size of 224 by 224 by 3, but, here we are changing it to 128 by 128 by 3. The VGG16 which is trained on RGB cannot be trained on grayscale (one channel), however, we can repeat the channel three times to make it three dimensions.

7. Freeze all the trainable layers.

```
for layer in baseModel.layers:
  layer.trainable = False
```

8. Construct the head of the model that will be placed on top of the base model.

```
headModel = baseModel.output
headModel
```

It will grab only the last layer before FC.

Expected Output:

```
<KerasTensor: shape=(None, 4, 4, 512) dtype=float32 (created by layer
'block5_pool')>
```

9. Now, flatten the previous layer.

```
headModel = Flatten(name="flatten")(headModel)
headModel
```

Expected Output:

```
<KerasTensor: shape=(None, 8192) dtype=float32 (created by layer
'flatten')>
```

10. Define the output layer.

```
headModel = Dense(3, activation="softmax")(headModel)
```

11. Now place the head FC model on top of the base model.

```
model = Model(inputs = baseModel.input, outputs = headModel)
model.summary()
```

```
Model: "model_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_17 (InputLayer)       [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808
```

```
 block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0

 flatten (Flatten)            (None, 8192)              0

 dense_10 (Dense)             (None, 3)                 24579

=================================================================
Total params: 14,739,267
Trainable params: 24,579
Non-trainable params: 14,714,688
```

12. You can check whether the trainable layers have been frozen or not.

```
for layer in baseModel.layers:
  print("{}: {}".format(layer, layer.trainable))
```

13. Compile the neural network using **.compile()** with **loss = 'categorical_crossentropy', optimizer='adam', metrics=['accuracy']**:

```
.....
.....
```

14. Fit the neural networks with **fit()** and provide the train and validation data generators, **epochs=5**, the **steps per epoch,** and the **validation steps**:

```
history = model.fit(
    train_generator,
    steps_per_epoch = len(train_generator),
    validation_data = validation_generator,
    validation_steps = len(validation_generator),
    epochs = 10,
)
```

Activity

1. Find train and validation accuracy.
2. Plot training and validation graphs (loss and accuracy).
3. Plot confusion matrix and classification report.

****