Model Optimization and Tuning Phase Template

Date	12 July 2024
Team ID	SWTID1720067113
Project Title	Dog Breed Identification using Transfer Learning
Maximum Marks	10 Marks

Model Optimization and Tuning Phase

The Model Optimization and Tuning Phase involves refining neural network models for peak performance. It includes optimized model code, fine-tuning hyperparameters, comparing performance metrics, and justifying the final model selection for enhanced predictive accuracy and efficiency.

Hyperparameter Tuning Documentation (8 Marks):

Model	Tuned Hyperparameters

EfficientNetB7	1. Rescale: `1./255`
	  o Normalizes the image pixel values to the range [0, 1].
	2. Shear Range: `0.2`
	  o Randomly applies shearing transformations to the images.
	3. Zoom Range: `0.2`
	  o Randomly zooms inside images.
	4. Horizontal Flip: `True`
	  o Randomly flips half of the images horizontally.
	5. Target Size: `(224, 224)`
	  o The size to which all images are resized.
	6. Batch Size: `32`
	  o The number of images to be used in each batch for training.
	7. Optimizer: `Adam`
	  o Optimizer used to minimize the loss function.
	8. Loss Function: `categorical_crossentropy`
	  o Loss function used for multi-class classification problems.
	9. Epochs: `15`
	  o The number of times the entire dataset is passed through the network during training.

10. **Activation**: `softmax`
    - Activation function used in the output layer for multi-class classification.

```
[ ] import os

    def make_dir(x):
        if os.path.exists(x) == False:
            os.makedirs(x)
    base_dir = './subset'
    make_dir(base_dir)
```

```
[ ] train_dir = os.path.join(base_dir, 'train')
    make_dir(train_dir)
```

```
    breeds = labels.breed.unique()
    for breed in breeds:
        _ = os.path.join(train_dir, breed)
        make_dir(_)

        images = labels[labels.breed == breed]['id']
        for image in images:
            source = os.path.join(dataset_dir, f'{image}.jpg')
            destination = os.path.join(train_dir, breed, f'{image}.jpg')
            shutil.copyfile(source, destination)
```

```
[ ] from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
[ ] train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
```

```
[ ] selected_classes = []
    for breed in labels.breed.unique():
        selected_classes.append(breed)
```

```
[ ] datagen = ImageDataGenerator()

    generator = datagen.flow_from_directory(
        'subset/train',
        target_size=(224, 224),
        batch_size=32,
        class_mode='categorical',
        shuffle=False,
        classes=selected_classes
    )
```

```
    Found 10222 images belonging to 120 classes.
```

```
[ ] test_datagen = ImageDataGenerator(rescale=1./255)
```

vgg19

1. Learning Rate: Defined by the `Adam` optimizer with default settings.
2. Batch Size: 32 (used in `ImageDataGenerator`).
3. Epochs: 15 (defined in the `model.fit` function).
4. Image Size: 224x224 pixels (used in `ImageDataGenerator` and `load_img`).
5. Pre-trained Model: VGG19 (imported and used as base model).

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.activations import softmax
```

```
[ ] from keras import activations
```

```
[ ] from tensorflow.keras.applications.vgg19 import VGG19
```

```
[ ] from tensorflow.keras.layers import Dense, Flatten
```

```
[ ] from tensorflow.keras.models import Model
```

```
[ ] from tensorflow.keras.optimizers import Adam
```

```
[ ] Image_size=[224,224]
    sol=VGG19(input_shape=Image_size + [3], weights='imagenet', include_top = False)
    for i in sol.layers:
        i.trainable = False
    y=Flatten()(sol.output)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop
80134624/80134624 [==============================] - 5s 0us/step
```

```
[ ] final = Dense(120, activation='softmax')(y)
```

```
model = Model(inputs=sol.input, outputs=final)
```

```
[ ] model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['Accuracy'])
```

```
[ ] for data, labels in generator:
        print('Data shape:', data.shape)
        print('Labels shape:', labels.shape)
        break  # Only need to check one batch
```

```
Data shape: (32, 224, 224, 3)
Labels shape: (32, 120)
```

```
# Fit the model
model.fit(generator, epochs=15)
```

```
Epoch 1/15
320/320 [==============================] - 65s 176ms/step - loss: 294.5328 - Accuracy: 0.0252
Epoch 2/15
320/320 [==============================] - 56s 174ms/step - loss: 124.2027 - Accuracy: 0.3150
Epoch 3/15
320/320 [==============================] - 57s 178ms/step - loss: 26.9172 - Accuracy: 0.7512
Epoch 4/15
320/320 [==============================] - 55s 172ms/step - loss: 9.4837 - Accuracy: 0.8945
Epoch 5/15
320/320 [==============================] - 56s 173ms/step - loss: 7.1624 - Accuracy: 0.9158
Epoch 6/15
320/320 [==============================] - 57s 178ms/step - loss: 3.5580 - Accuracy: 0.9541
Epoch 7/15
320/320 [==============================] - 57s 177ms/step - loss: 21.3413 - Accuracy: 0.8532
Epoch 8/15
320/320 [==============================] - 55s 172ms/step - loss: 18.8485 - Accuracy: 0.8492
Epoch 9/15
320/320 [==============================] - 57s 178ms/step - loss: 37.6908 - Accuracy: 0.8043
Epoch 10/15
320/320 [==============================] - 55s 172ms/step - loss: 20.9604 - Accuracy: 0.8862
```

<table>
<tr><td></td><td>

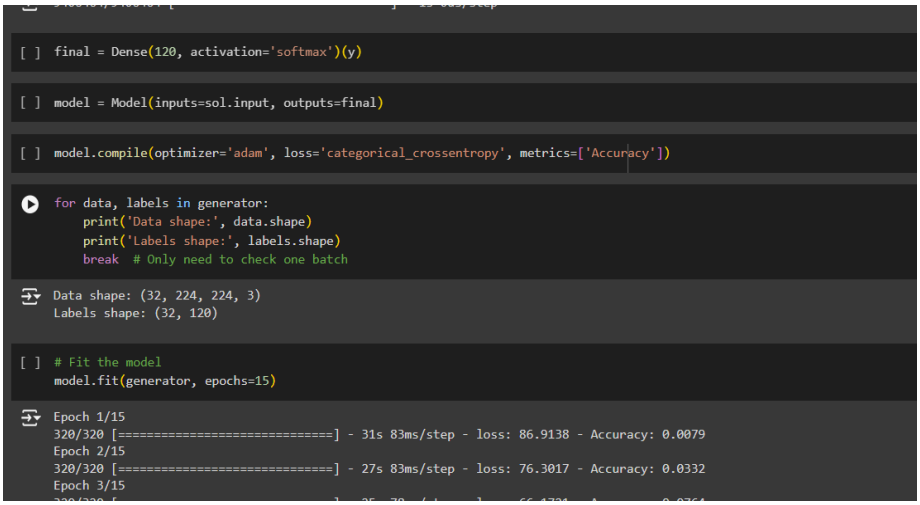
```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] model.save('/content/drive/MyDrive/Kaggle/models/dogbreed_vgg19.h5')

    /usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your m
      saving_api.save_model(

[ ] !cp dogbreed.h5 /content/drive/MyDrive/Kaggle/models
```

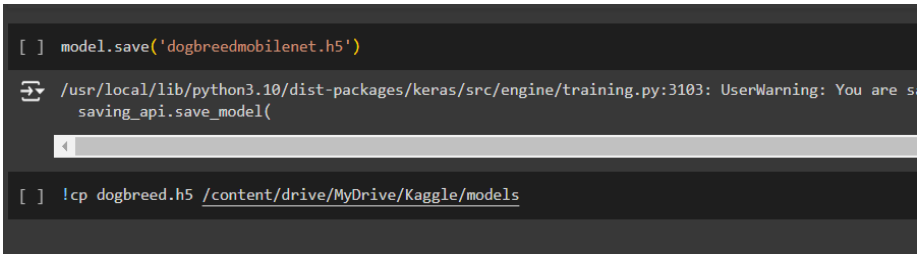
</td></tr>
<tr><td>MobileNet</td><td>

1. Image Size: The target size of the input images used for the VGG19 model.
2. Pre-trained Weights: Specifies that the model uses pre-trained weights from ImageNet.
3. Trainable Layers: Sets all layers of VGG19 to be non-trainable to use the pre-trained features.
4. Output Layer Activation: The activation function used in the final output layer, set to 'softmax' for multi-class classification.

```
[ ] import tensorflow as tf
    from tensorflow import keras
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.activations import softmax

[ ] from keras import activations

[ ] from tensorflow.keras.applications import MobileNetV2

[ ] from tensorflow.keras.layers import Dense, Flatten

[ ] from tensorflow.keras.models import Model

[ ] from tensorflow.keras.optimizers import Adam

    Image_size=[224,224]
    sol=MobileNetV2(input_shape=Image_size + [3], weights='imagenet', include_top = False)
    for i in sol.layers:
        i.trainable = False
    y=Flatten()(sol.output)

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weig
    9406464/9406464 [==============================] - 1s 0us/step

[ ] final = Dense(120, activation='softmax')(y)

[ ] model = Model(inputs=sol.input, outputs=final)
```

</td></tr>
</table>





Final Model Selection Justification (2 Marks):

Final Model	Reasoning
EfficientNetB7	• EfficientNetB7 is known for its superior performance on a variety of image classification benchmarks. It achieves high accuracy with relatively fewer parameters compared to other models.

	• The EfficientNet architecture scales efficiently in terms of depth, width, and resolution, allowing it to perform well on large datasets while maintaining computational efficiency. • EfficientNetB7 comes pre-trained on ImageNet, providing a strong feature extraction capability which is beneficial for fine-tuning on the dog breed identification dataset. • Despite being a deep model, EfficientNetB7 is optimized for both speed and memory usage, making it practical for real-world applications where computational resources might be limited. • **We chose the model because for specific images nearly every prediction was correct for this model which was not much accurate in other models.**