Lab 6.3 กลุ่มบาบูนตึงตึง

1. Load the MobileNet Model: โค้ดนี้โหลดโมเดล MobileNet ที่ถูกฝึกอบรมล่วงหน้า (pre-trained) ด้วยชุดข้อมูล imagenet และกำหนดรูปภาพเป้าหมายขนาด (224, 224, 3) สำหรับการนำเข้ารูปภาพในโมเดล.

```
# Load the MobileNet model
base_model = MobileNet(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

2. Add New Layers: เพิ่มเลเยอร์บน MobileNet ที่ถูกโหลด เริ่มต้นด้วย Global Average Pooling Layer และตามด้วย Dense Layer สองชั้น. ชั้นสุดท้ายเป็นชั้น Dense ที่มี 3 โหลดสำหรับการทำนายคลาส.

```
# Add new layers
x = base_model.output

# Global Average Pooling Layer
x = GlobalAveragePooling2D()(x)

# Add Dense layers
x = Dense(1024, activation='relu')(x)
x = Dense(512, activation='relu')(x)  # Adjusted to 512 nodes
preds = Dense(3, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=preds)
```

3. Freeze and Unfreeze Layers: จากนั้น, โค้ดแยกชั้นของโมเดลออกเป็นกลุ่ม และระบุว่าชั้นใดให้ฝึกอบรมและชั้นใดไม่ต้องฝึกอบรม ในกรณีนี้, 86 ชั้นแรกไม่สามารถฝึกอบรม, และชั้น 86 ถึง 130 สามารถฝึกอบรม และชั้นที่ 130 และต่อจากนั้นก็สามารถฝึกอบรม.

```python
# Freeze layers
for layer in model.layers[:86]:
    layer.trainable = False

# Unfreeze layers for the second set
for layer in model.layers[86:130]:
    layer.trainable = True

# Unfreeze layers for the third set
for layer in model.layers[130:]:
    layer.trainable = True
```

4. Compile the Model: โค้ดคอมไพล์โมเดลด้วย optimizer Adam, ฟังก์ชัน 'categorical_crossentropy' สำหรับงานการจำแนกหลายคลาส, และ metrics ในกรณีนี้เป็นความแม่นยำ.

```python
# Create Optimizer
opts = Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy', optimizer=opts, metrics=['accuracy'])
eps = 50
step_size_train = train_generator.n // train_generator.batch_size
step_size_val = val_generator.n // val_generator.batch_size

# Check step size train = step size val
if step_size_train != step_size_val:
    print("Warning: step_size_train is not equal to step_size_val.")
    new_batch_size = val_generator.n // step_size_train
    val_generator = datagen.flow_from_directory(
        './Validate/',
        target_size=(224, 224),
        color_mode='rgb',
        batch_size=new_batch_size,
        class_mode='categorical',
        seed=seed_val,
        shuffle=True
    )
    step_size_val = val_generator.n // val_generator.batch_size
    print(f"Adjusted batch size to {new_batch_size} to make step_size_train equal to step_size_val.")
```

5. Data Preprocessing and Image Data Generators: สร้าง ImageDataGenerator สำหรับการประมวลผลรูปภาพโดยปรับเปลี่ยนคุณสมบัติของรูปภาพ เช่น การหมุน, การซูม, การเลื่อนตำแหน่ง, เงื่อนไข, การพลิกภาพ, และประมวลผลภาพอื่น ๆ. จากนั้น, สร้าง ImageDataGenerator สำหรับชุดข้อมูลการฝึกอบรมและการทดสอบ.

```python
# Create DataGenerator objects
datagen = ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    horizontal_flip=True,
    preprocessing_function=preprocess_input,
    fill_mode="nearest",
)
```

6. Train the Model: โค้ดนี้ใช้ fit_generator เพื่อฝึกโมเดล โดยระบุจำนวนรอบการฝึกอบรม (epochs) และแสดงผลลัพธ์ขณะฝึกอบรม การฝึกอบรมถูกดำเนินการด้วยชุดข้อมูลการฝึกอบรมและการทดสอบ.

```python
eps = 100

history = model.fit_generator(generator=train_generator,
                              steps_per_epoch=step_size_train,
                              validation_data=val_generator,
                              validation_steps=step_size_val,
                              epochs=eps,
                              verbose=1)
```

7. Performance Visualization: ในส่วนนี้, โค้ดนี้ใช้ Matplotlib เพื่อแสดงกราฟความแม่นยำและความสูญเสียในชุดข้อมูลการฝึกอบรมและการทดสอบ โดยทั้งความแม่นยำและความสูญเสียถูกพล็อตเทียบกัน.

```python
# Performance Visualization
epochs = range(1, 100 + 1)

# View Accuracy (Training, Validation)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(epochs, history.history["accuracy"], label="Train_acc")
plt.plot(epochs, history.history["val_accuracy"], label="Validate_acc")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Training and Validation Accuracy")
plt.legend()

# View Loss (Training, Validation)
plt.subplot(1, 2, 2)
plt.plot(epochs, history.history['loss'], label="Train_loss")
plt.plot(epochs, history.history['val_loss'], label="Validate_loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.title("Training and Validation Loss")
plt.legend()

plt.tight_layout()
plt.show()
```

8. Test the Model: ในส่วนนี้, โค้ดสร้าง ImageDataGenerator สำหรับชุดข้อมูลการทดสอบและใช้โมเดลเพื่อทำนายคลาสของภาพในชุดข้อมูลการทดสอบ ผลลัพธ์ที่ทำนายถูกเก็บไว้ใน y_pred.

```python
# Initial test generator
testPath = './Test/'
test_generator = datagen.flow_from_directory(
    testPath,
    class_mode="categorical",
    target_size=(224, 224),
    color_mode="rgb",
    shuffle=False,
    batch_size=1
)

# Get class id for y_real_class
y_true = test_generator.classes

# Predict images according to test_generator
preds = model.predict_generator(test_generator)
print(preds.shape)
print(preds)

# Get predicted class labels (argmax along axis 1)
y_pred = np.argmax(preds, axis=1)
print(y_true)
print(y_pred)
```

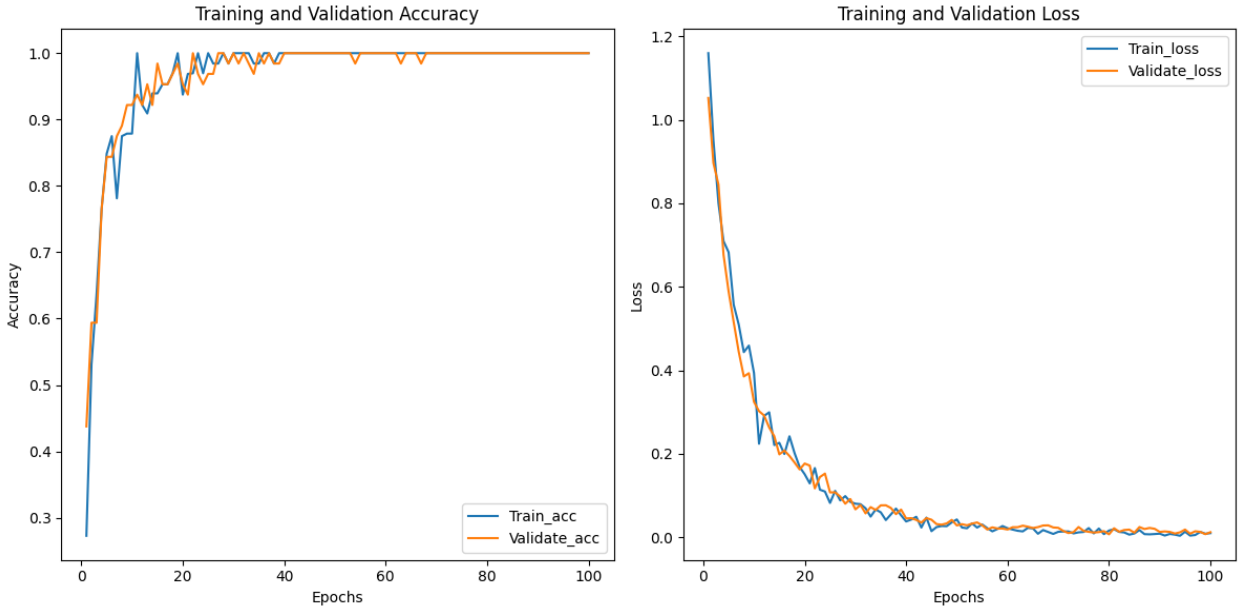
9. Confusion Matrix and Classification Report: โค้ดนี้คำนวณและแสดง Confusion Matrix และ Classification Report

```python
# Calculate confusion matrix and classification report
confusion = confusion_matrix(y_true, y_pred)
classification_rep = classification_report(y_true, y_pred)

print("Confusion Matrix:")
print(confusion)
print("\nClassification Report:")
print(classification_rep)
y_pred = np.argmax(preds, axis=1)
print(test_generator.classes)
print(y_pred)

# Calculate confusion matrix, classification report between y_true and df_class
print(confusion_matrix(y_true, y_pred))
print(classification_report(y_true, y_pred))
```

ผลลัพธ์

```
(15, 3)
[[9.35132384e-01 5.93701787e-02 5.49741695e-03]
 [9.83673036e-01 1.01632038e-02 6.16384856e-03]
 [9.86213386e-01 1.36608677e-02 1.25721504e-04]
 [8.79792690e-01 4.20874171e-02 7.81198964e-02]
 [7.06118415e-04 9.99036431e-01 2.57523556e-04]
 [4.56959824e-04 9.97283816e-01 2.25915620e-03]
 [7.24284025e-03 9.88642037e-01 4.11515217e-03]
 [1.17053673e-01 8.46150875e-01 3.67954373e-02]
 [5.16739237e-05 9.99840260e-01 1.07986285e-04]
 [7.86798191e-04 9.96764064e-01 2.44911900e-03]
 [8.23914725e-03 2.51253678e-05 9.91735756e-01]
 [5.54107167e-02 1.49482945e-02 9.29641008e-01]
 [2.27867067e-02 1.10278297e-02 9.66185510e-01]
 [2.54614592e-01 8.22370313e-03 7.37161696e-01]
 [5.52889484e-04 1.85267755e-03 9.97594416e-01]]
[0 0 0 0 1 1 1 1 1 1 2 2 2 2 2]
[0 0 0 0 1 1 1 1 1 1 2 2 2 2 2]
Confusion Matrix:
[[4 0 0]
 [0 6 0]
 [0 0 5]]
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15

[0 0 0 0 1 1 1 1 1 1 2 2 2 2 2]
[0 0 0 0 1 1 1 1 1 1 2 2 2 2 2]
[[4 0 0]
 [0 6 0]
 [0 0 5]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         4
           1       1.00      1.00      1.00         6
           2       1.00      1.00      1.00         5

    accuracy                           1.00        15
   macro avg       1.00      1.00      1.00        15
weighted avg       1.00      1.00      1.00        15
```


Training and Validation Accuracy
Accuracy
Epochs
Train_acc
Validate_acc
Training and Validation Loss
Loss
Epochs
Train_loss
Validate_loss