National University
Of Computer and Emerging Sciences

Department of Computer Science

Name: Tazmeen Afroz **Date:** 23 Nov 2025
Semester: 7th
Roll No : 22P-9252

Computer Vision

Lab 8: Introduction to Deep Learning with TensorFlow – Implementing a Simple CNN on MNIST

Lab Tasks – Implementing Popular CNN Architectures in TensorFlow

Kaggle Notebook Link : code-link

Best Test Accuracy Achieved : 88.28%

Model: (pretrained Resnet50 (imagenet weights))

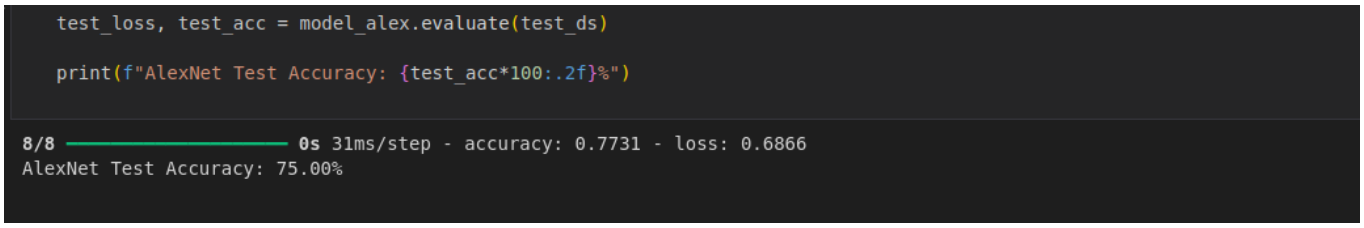
Task 1.1, implement **AlexNet** using TensorFlow's Keras API by defining each layer manually in a Sequential or Functional model. Use the provided image of the AlexNet architecture as a reference for the correct number of filters, kernel sizes, and fully connected layers. Build and train the model on your custom image classification dataset.

Code:

```
#  ALEXNET
model_alex = models.Sequential([
    layers.Conv2D(96, (11, 11), strides=(4, 4), padding='valid', activation='relu',
input_shape=(*IMG_SIZE, 3)),
   layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),

   layers.Conv2D(256, (5, 5), strides=(1, 1), padding='same', activation='relu'),
   layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),

   layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
   layers.Conv2D(384, (3, 3), padding='same', activation='relu'),
    layers.Conv2D(256, (3, 3), padding='same', activation='relu'),
    layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)),

    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(num_classes, activation='softmax')
])

model_alex.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

history_alex = model_alex.fit(train_ds, validation_data=val_ds, epochs=30,
callbacks=callbacks)
```

Results:

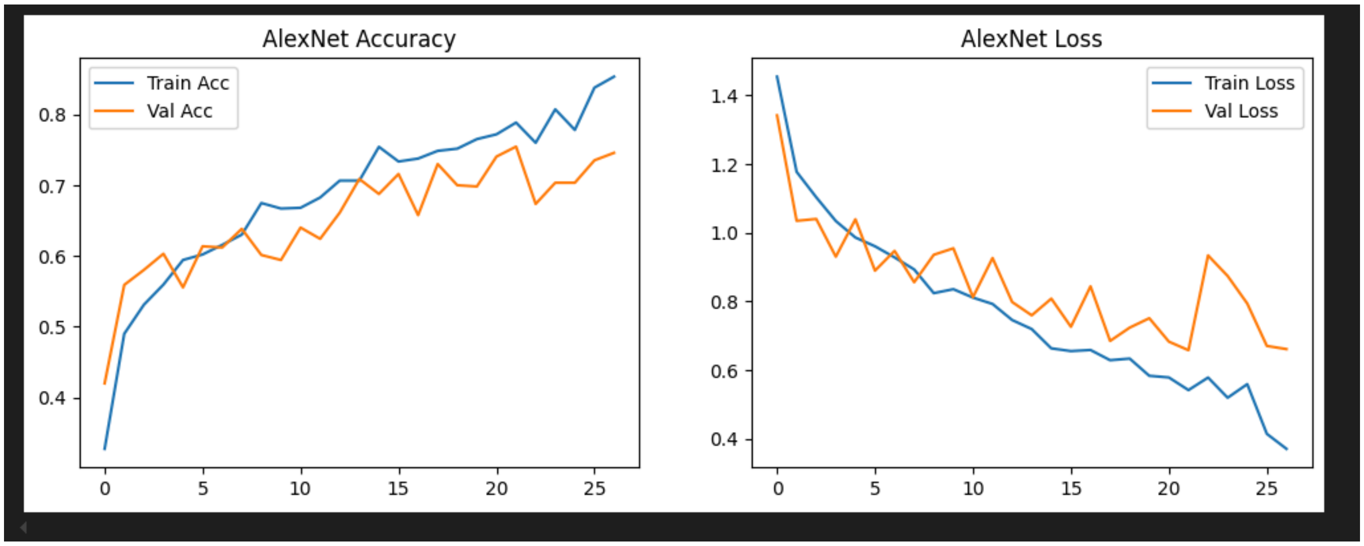
Test Accuracy: 75%



```
test_loss, test_acc = model_alex.evaluate(test_ds)

print(f"AlexNet Test Accuracy: {test_acc*100:.2f}%")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.7731 - loss: 0.6866
AlexNet Test Accuracy: 75.00%
```

Training and Validation Accuracy and loss over 30 epochs:



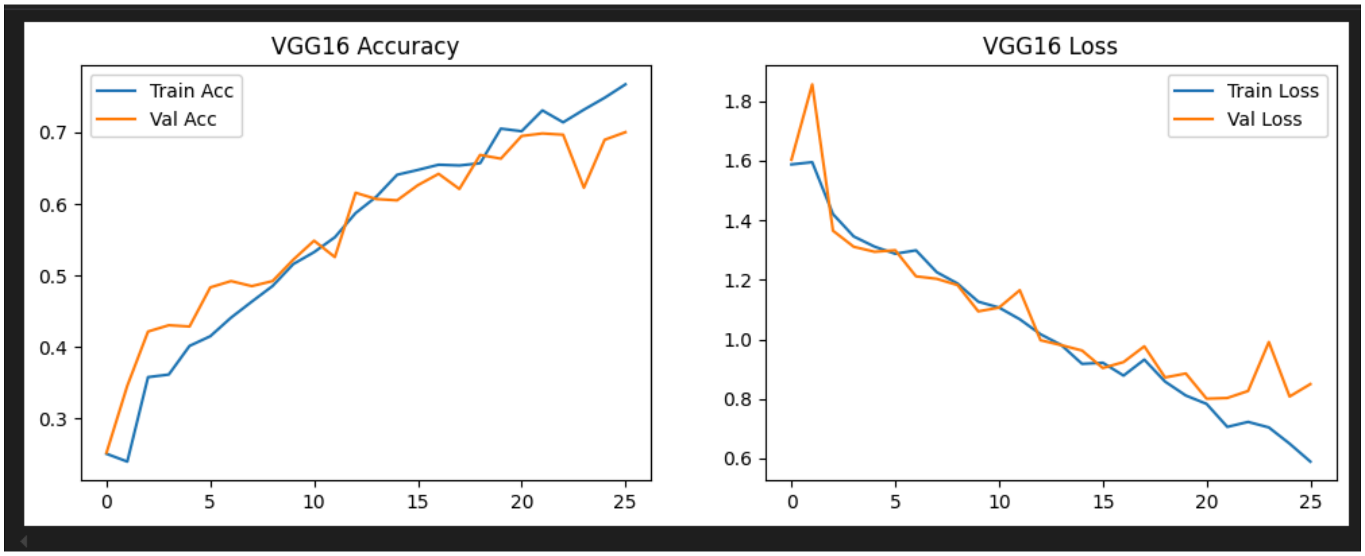
In **Task 1.2**, implement **VGG-16** by stacking convolutional, pooling, and dense layers exactly as shown in the VGG-16 architecture diagram. Train the custom-built model on your dataset, then load the pretrained `VGG16` model from `tensorflow.keras.applications`, fine-tune its final layers, and compare both models in terms of accuracy, loss, and training time.

Code:

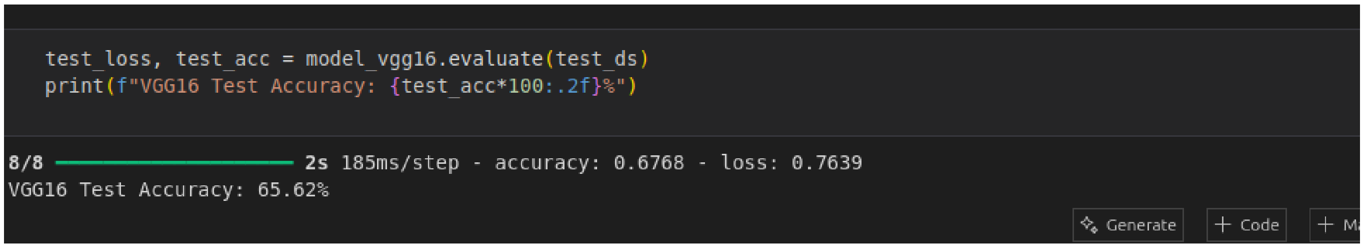
```
# ---------- VGG16 ----------
model_vgg16 = models.Sequential([
   layers.Input(shape=(*IMG_SIZE, 3)),

   # Block 1
   layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
   layers.MaxPooling2D((2, 2)),

   # Block 2
   layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
   layers.MaxPooling2D((2, 2)),

   # Block 3
   layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
   layers.MaxPooling2D((2, 2)),

   # Block 4
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.MaxPooling2D((2, 2)),

   # Block 5
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
   layers.MaxPooling2D((2, 2)),

   # Classifier
   layers.Flatten(),
   layers.Dense(4096, activation='relu'),
   layers.Dense(4096, activation='relu'),
   layers.Dense(num_classes, activation='softmax')
])

model_vgg16.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

history_vgg16 = model_vgg16.fit(train_ds, validation_data=val_ds, epochs=30,
callbacks=callbacks)
```

Results:

Training and Validation Accuracy and loss over 30 epochs:



Test Accuracy: 65.5%



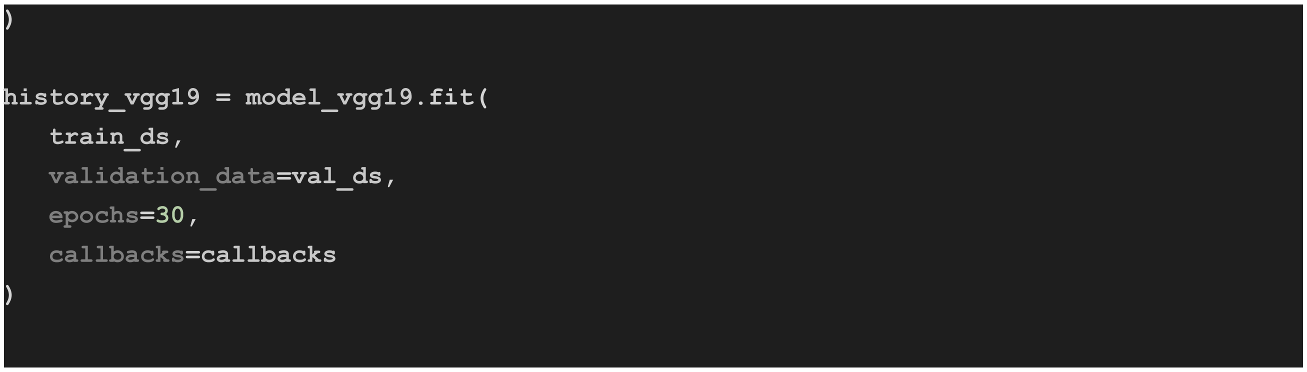
```
test_loss, test_acc = model_vgg16.evaluate(test_ds)
print(f"VGG16 Test Accuracy: {test_acc*100:.2f}%")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 185ms/step - accuracy: 0.6768 - loss: 0.7639
VGG16 Test Accuracy: 65.62%
```

In **Task 1.3**, create **VGG-19** in the same way, referring to the provided diagram for layer counts and structure. Train your custom VGG-19 model on the dataset and then use TensorFlow's pretrained `VGG19` model for comparison. Discuss the performance difference caused by the deeper architecture.

Code:

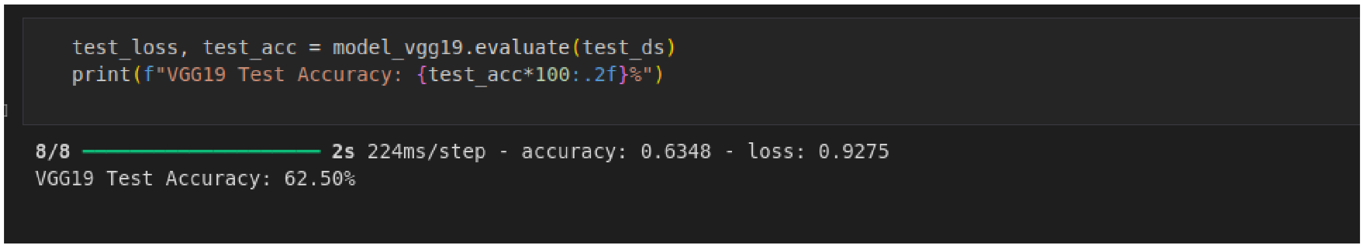
```
model_vgg19 = models.Sequential([
    layers.Input(shape=(*IMG_SIZE, 3)),

    # Block 1
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    # Block 2
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    # Block 3
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(256, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    # Block 4
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    # Block 5
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.Conv2D(512, (3, 3), activation='relu', padding='same'),
    layers.MaxPooling2D((2, 2)),

    # Classifier
    layers.Flatten(),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(4096, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

optimizer = optimizers.Adam(learning_rate=1e-5)

model_vgg19.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
```



```
)

history_vgg19 = model_vgg19.fit(
    train_ds,
    validation_data=val_ds,
    epochs=30,
    callbacks=callbacks
)
```

Results:

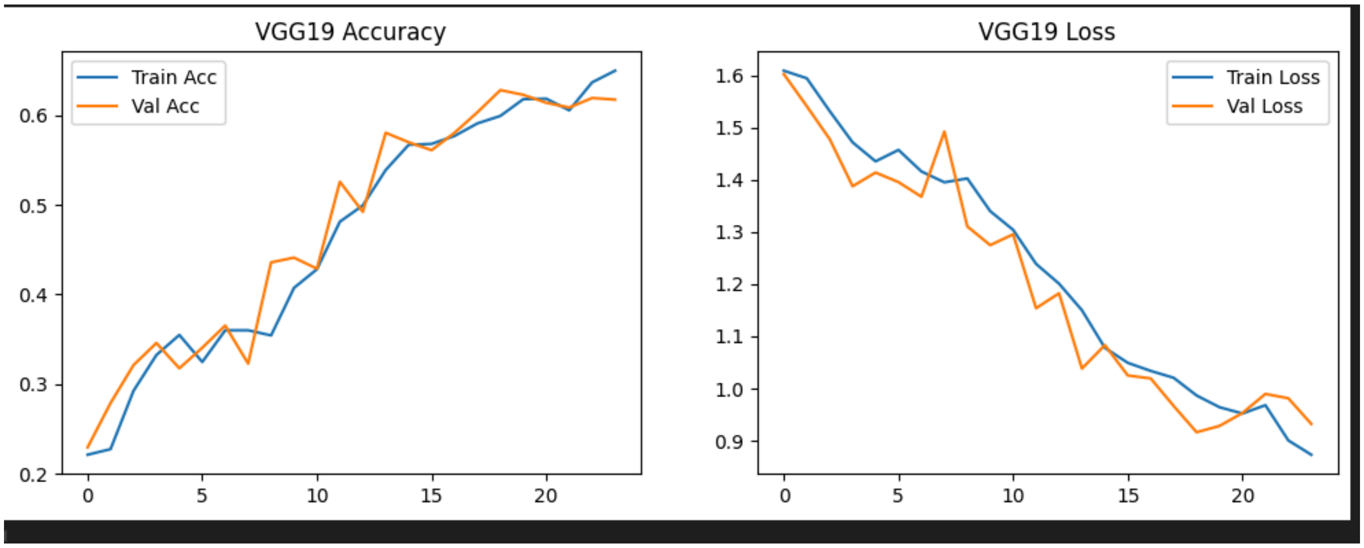
Test Accuracy: 62%



```
test_loss, test_acc = model_vgg19.evaluate(test_ds)
print(f"VGG19 Test Accuracy: {test_acc*100:.2f}%")
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 2s 224ms/step - accuracy: 0.6348 - loss: 0.9275
VGG19 Test Accuracy: 62.50%
```

Training and Validation Accuracy and loss over 30 epochs:



In **Task 1.4**, implement **GoogLeNet (Inception v1)** by defining the Inception modules as shown in the reference image. Construct the network by concatenating convolutional paths with different filter sizes within each module, and train it using the same dataset. Compare its feature extraction performance with the previous VGG models.

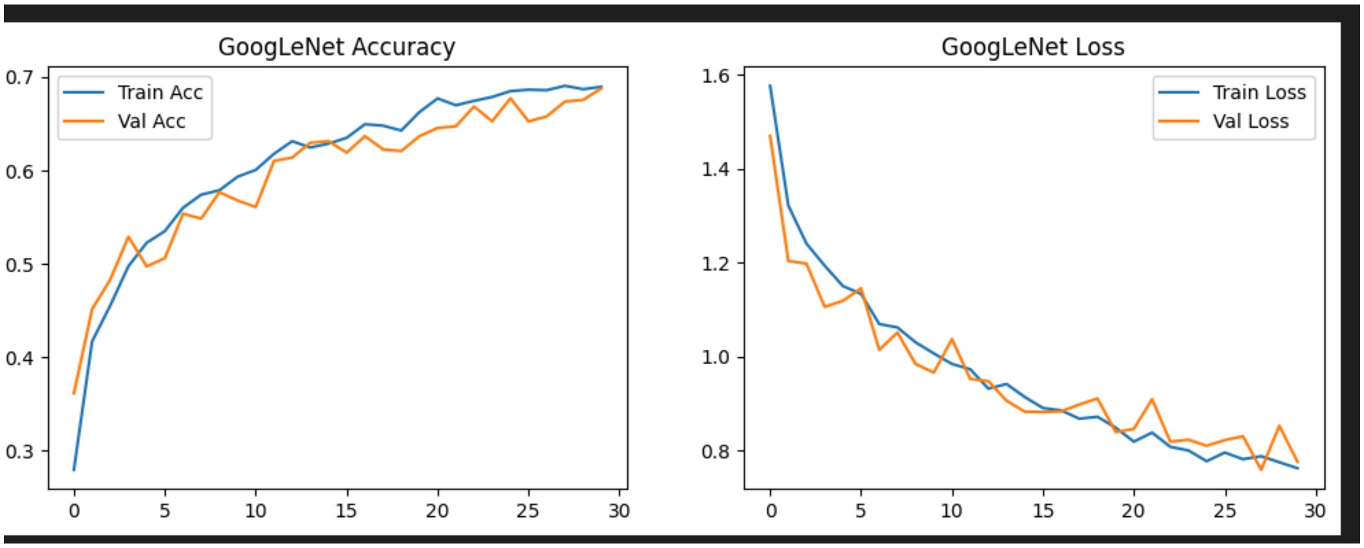
Code:

```
# ---------- TASK 1.4: GOOGLENET (Inception v1) ----------

def inception_module(x, f1, f3_r, f3, f5_r, f5, proj):
    # Path 1: 1x1 Conv
   path1 = layers.Conv2D(f1, (1, 1), padding='same', activation='relu')(x)

   # Path 2: 1x1 Conv -> 3x3 Conv
   path2 = layers.Conv2D(f3_r, (1, 1), padding='same', activation='relu')(x)
            path2 = layers.Conv2D(f3, (3, 3), padding='same',
activation='relu')(path2)

   # Path 3: 1x1 Conv -> 5x5 Conv
   path3 = layers.Conv2D(f5_r, (1, 1), padding='same', activation='relu')(x)
            path3 = layers.Conv2D(f5, (5, 5), padding='same',
activation='relu')(path3)

   # Path 4: 3x3 MaxPool -> 1x1 Conv
   path4 = layers.MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
           path4 = layers.Conv2D(proj, (1, 1), padding='same',
activation='relu')(path4)

   # Concatenate all paths
   return layers.Concatenate(axis=-1)([path1, path2, path3, path4])

def build_googlenet(input_shape, num_classes):
   inputs = layers.Input(shape=input_shape)
```

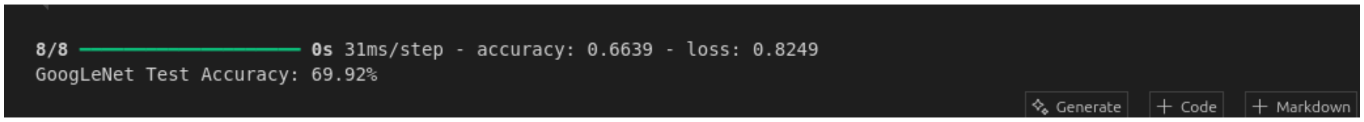
```python
    # Initial Conv + MaxPool
    x = layers.Conv2D(64, (7, 7), strides=(2, 2), padding='same', activation='relu')(inputs)
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = layers.Conv2D(64, (1, 1), padding='same', activation='relu')(x)
    x = layers.Conv2D(192, (3, 3), padding='same', activation='relu')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    # Inception Blocks

    x = inception_module(x, 64, 96, 128, 16, 32, 32)    # Inception 3a
    x = inception_module(x, 128, 128, 192, 32, 96, 64) # Inception 3b
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    x = inception_module(x, 192, 96, 208, 16, 48, 64)  # Inception 4a

    # Output layers
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dropout(0.4)(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    return models.Model(inputs, outputs, name="GoogLeNet_Mini")

model_googlenet = build_googlenet((*IMG_SIZE, 3), num_classes)

model_googlenet.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                        loss='sparse_categorical_crossentropy',
                        metrics=['accuracy'])

history_googlenet = model_googlenet.fit(train_ds,
                                        validation_data=val_ds,
                                        epochs=30,
                                        callbacks=callbacks)
```

```
# %%
plot_metrics(history_googlenet, "GoogLeNet")

test_loss, test_acc = model_googlenet.evaluate(test_ds)
print(f"GoogLeNet Test Accuracy: {test_acc*100:.2f}%")
```

Results:
Training and Validation Accuracy and loss over 30 epochs:



Test Accuracy: 69.9%



```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 0.6639 - loss: 0.8249
GoogLeNet Test Accuracy: 69.92%
```

In **Task 1.5**, implement **ResNet** using TensorFlow by building residual blocks that include skip (shortcut) connections between layers, following the structure shown in the provided diagram. Train your custom ResNet model and then use the pretrained `ResNet50` model from `tensorflow.keras.applications` for evaluation. Analyze how the use of residual connections helps overcome the vanishing gradient problem and improves performance for deeper networks.

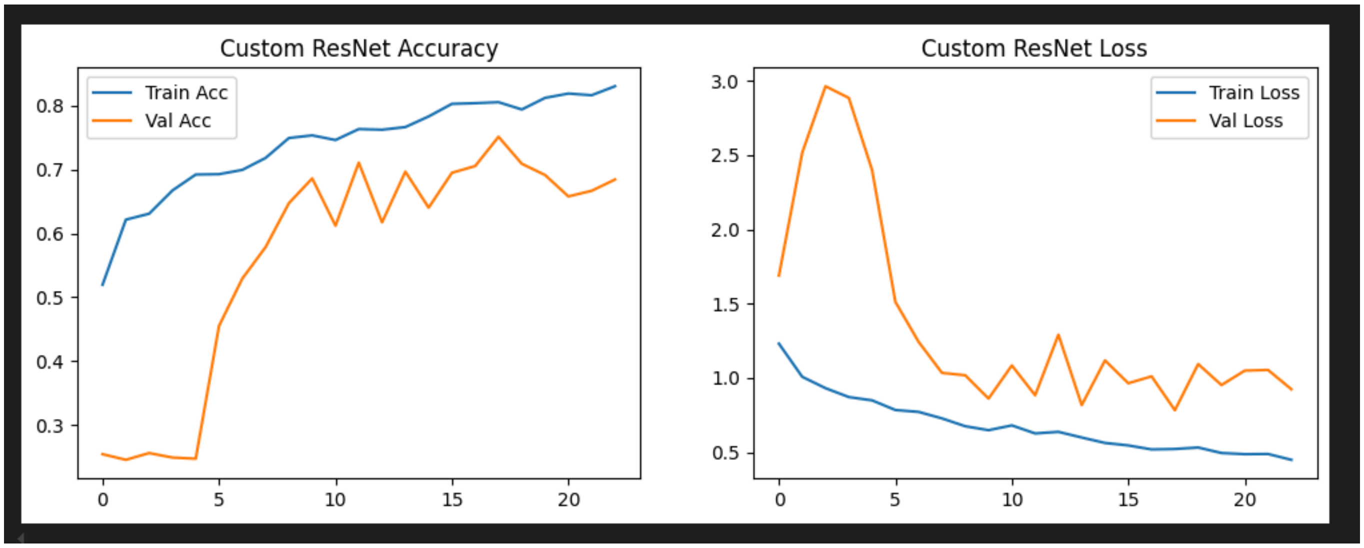
Custom ResNet:
Code:

```
# %%
# ---------- TASK 1.5: Custom ResNet ----------

def residual_block(x, filters, stride=1):
   shortcut = x

   # First Conv
   x = layers.Conv2D(filters, (3, 3), strides=stride, padding='same')(x)
   x = layers.BatchNormalization()(x)
   x = layers.Activation('relu')(x)

   # Second Conv
   x = layers.Conv2D(filters, (3, 3), strides=1, padding='same')(x)
   x = layers.BatchNormalization()(x)

   # Shortcut path:

   if stride != 1 or shortcut.shape[-1] != filters:
       shortcut = layers.Conv2D(filters, (1, 1), strides=stride,
padding='same')(shortcut)
       shortcut = layers.BatchNormalization()(shortcut)

   # Add the shortcut to the main path
   x = layers.Add()([x, shortcut])
   x = layers.Activation('relu')(x)
   return x

def build_custom_resnet(input_shape, num_classes):
   inputs = layers.Input(shape=input_shape)

   # Initial Conv
   x = layers.Conv2D(64, (7, 7), strides=(2, 2), padding='same')(inputs)
   x = layers.BatchNormalization()(x)
```

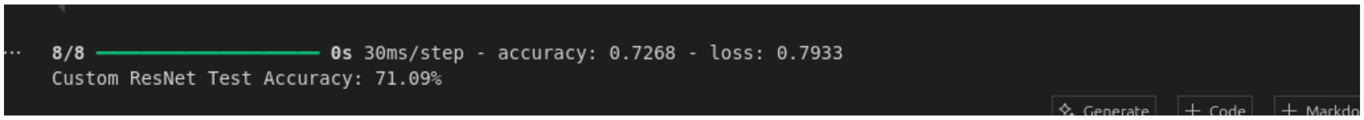
```
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)

    # Residual Blocks
    x = residual_block(x, 64)
    x = residual_block(x, 64)

    x = residual_block(x, 128, stride=2) # Downsample
    x = residual_block(x, 128)

    x = residual_block(x, 256, stride=2) # Downsample
    x = residual_block(x, 256)

    # Classifier
    x = layers.GlobalAveragePooling2D()(x)
    outputs = layers.Dense(num_classes, activation='softmax')(x)

    return models.Model(inputs, outputs, name="Custom_ResNet")

model_resnet_custom = build_custom_resnet((*IMG_SIZE, 3), num_classes)

model_resnet_custom.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                            loss='sparse_categorical_crossentropy',
                            metrics=['accuracy'])

history_resnet_custom = model_resnet_custom.fit(train_ds,
                                                validation_data=val_ds,
                                                epochs=30,
                                                callbacks=callbacks)

# %%
plot_metrics(history_resnet_custom, "Custom ResNet")

test_loss, test_acc = model_resnet_custom.evaluate(test_ds)
print(f"Custom ResNet Test Accuracy: {test_acc*100:.2f}%")
```

Results:

Training and Validation Accuracy and loss over 30 epochs:



Test Accuracy: 71%



```
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.7268 - loss: 0.7933
Custom ResNet Test Accuracy: 71.09%
```

Pretrained ResNet50:

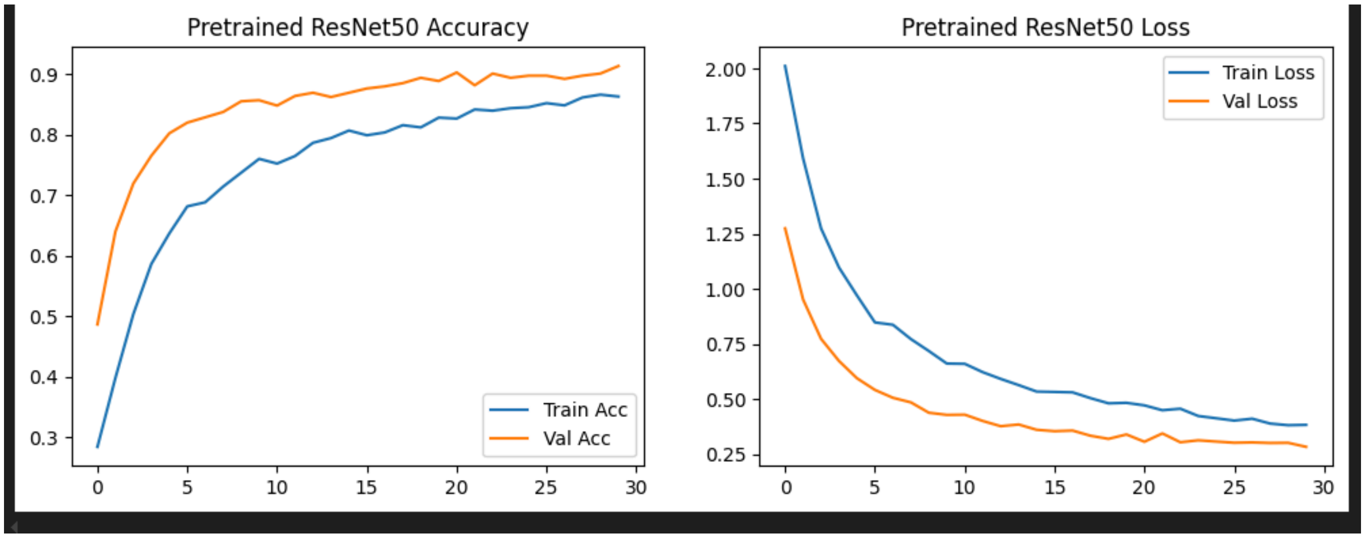
Code:

```
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(*IMG_SIZE, 3))
base_model.trainable = False  # Freeze weights

inputs = layers.Input(shape=(*IMG_SIZE, 3))

x = base_model(inputs, training=False)
x = layers.GlobalAveragePooling2D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(num_classes, activation='softmax')(x)
```

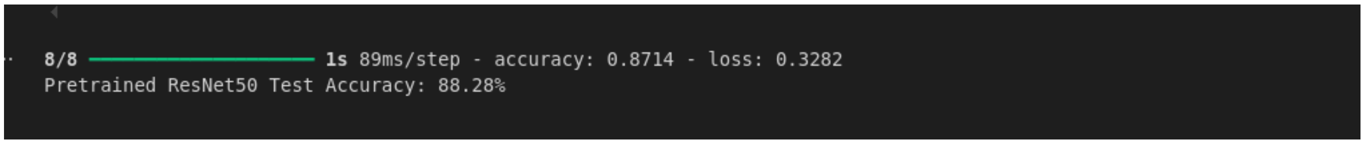
```
model_resnet50 = models.Model(inputs, outputs, name="Pretrained_ResNet50")

# 4. TRAIN
model_resnet50.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                       loss='sparse_categorical_crossentropy',
                       metrics=['accuracy'])
```

Results:

Training and Validation Accuracy and loss over 30 epochs:



Test Accuracy: 88%



```
8/8 ━━━━━━━━━━━━━━━━━━━━ 1s 89ms/step - accuracy: 0.8714 - loss: 0.3282
Pretrained ResNet50 Test Accuracy: 88.28%
```