This is a sample Jupyter Notebook

Below is an example of a code cell. Put your cursor into the cell and press Shift+Enter to execute it and select the next one, or click 'Run Cell' button.

Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

To learn more about Jupyter Notebooks in PyCharm, see help. For an overview of PyCharm, go to Help -> Learn IDE features or refer to our documentation.

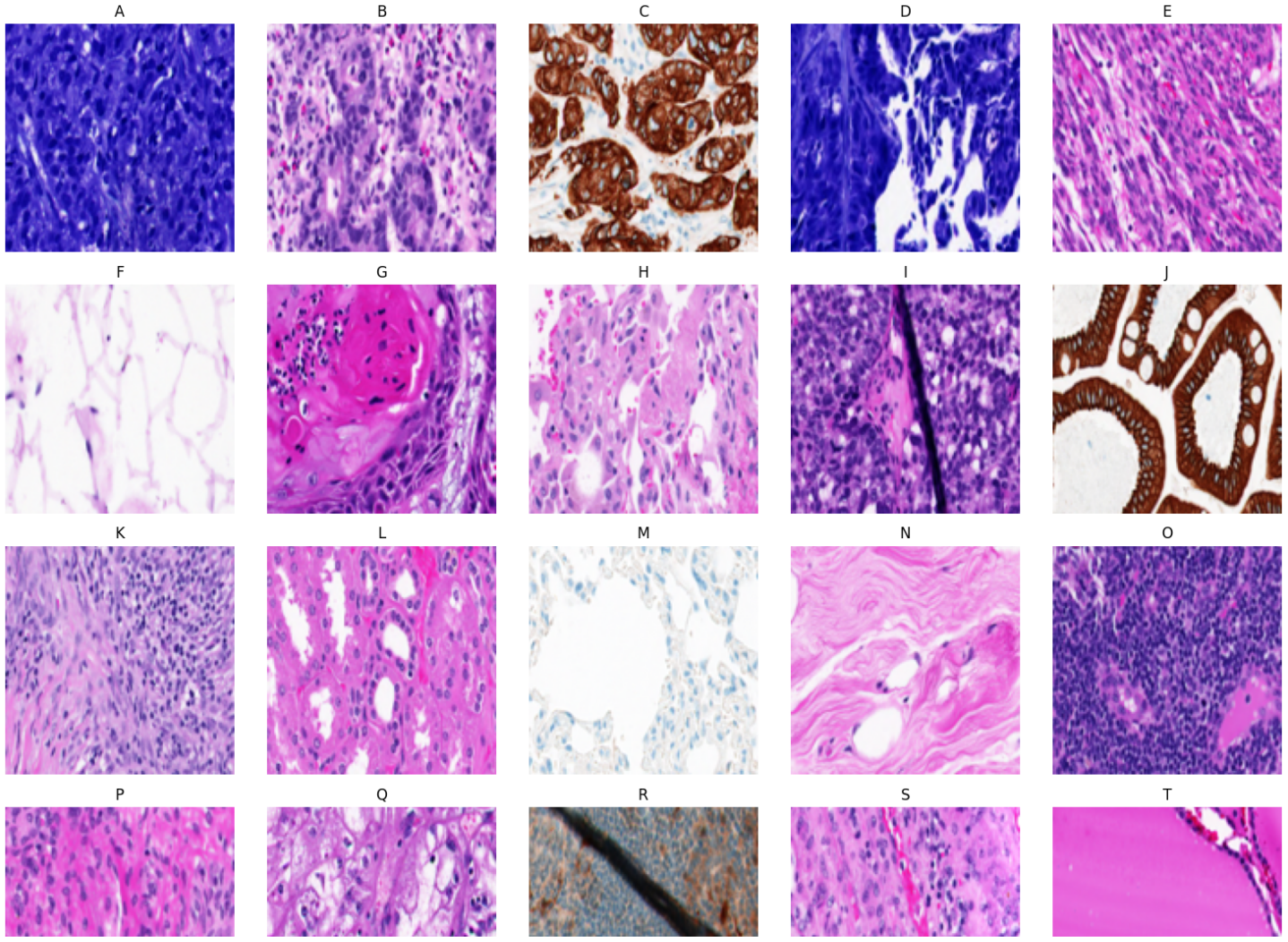
```
import os
import zipfile
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from PIL import Image
import tifffile
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.mixed_precision import set_global_policy
import json
from flask import Flask, request, render_template_string
import io
import base64
import shutil
from google.colab import userdata

set_global_policy('mixed_float16')

np.random.seed(42)
tf.random.set_seed(42)

print(tf.config.list_physical_devices('GPU'))

os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')

print("Step 1: Data Collection and Preprocessing")

dataset_slug = "ambarish/kimia-path-960"
os.system(f"kaggle datasets download -d {dataset_slug} -p ./data/shreyav/kimia-path-960 --unzip")
!pwd
dataset_path = './data/shreyav/kimia-path-960/'
output_path = './data/shreyav/kimia-path-960/KIMIA_Path_960_segregated/'
if not os.path.exists(output_path):
    os.makedirs(output_path)
'''
   This dataset has got 20 folders. Looking at the kaggle dataset ,
   each of these folders represent a specific type of tissue.
   Segregating them in to different folders.
'''

for filename in os.listdir(dataset_path):
    if filename.endswith('.tif'):
        issue_type = filename[:1]
        issue_type_folder = os.path.join(output_path, issue_type)
        if not os.path.exists(issue_type_folder):
            os.makedirs(issue_type_folder)
        shutil.move(os.path.join(dataset_path, filename), os.path.join(issue_type_folder, filename))

print("Dataset has been segregated into 20 folders.")

data_dir = output_path
classes = sorted(os.listdir(data_dir))
num_classes = len(classes)
print(f"Found {num_classes} classes: {classes}")

def load_and_preprocess_image(image_path, target_size=(128, 128)):
    img = tifffile.imread(image_path)
    if len(img.shape) == 3 and img.shape[2] == 3:
        img = Image.fromarray(img)
    else:
        img = Image.fromarray(img).convert('RGB')
    img = img.resize(target_size)
    img_array = np.array(img) / 255.0
    return img_array

image_paths = []
labels = []
for class_idx, class_name in enumerate(classes):
    class_dir = os.path.join(data_dir, class_name)
    for img_file in os.listdir(class_dir):
        if img_file.endswith('.tif'):
            image_paths.append(os.path.join(class_dir, img_file))
            labels.append(class_idx)
```

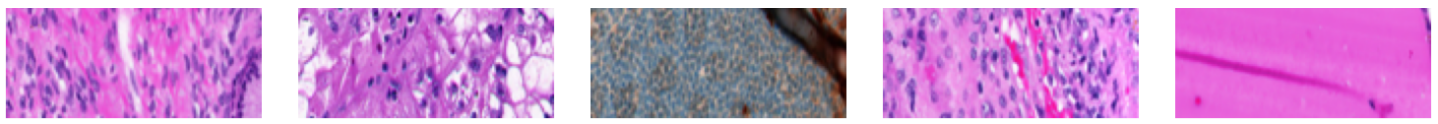
```
print(f"Total images collected: {len(image_paths)}")

train_paths, temp_paths, train_labels, temp_labels = train_test_split(
    image_paths, labels, test_size=0.3, stratify=labels, random_state=42
)
val_paths, test_paths, val_labels, test_labels = train_test_split(
    temp_paths, temp_labels, test_size=0.5, stratify=temp_labels, random_state=42
)

print(f"Train: {len(train_paths)}, Val: {len(val_paths)}, Test: {len(test_paths)}")

print("Step 2: Exploratory Data Analysis")

total_counts = [len(os.listdir(os.path.join(data_dir, c))) for c in classes]
train_counts = [int(0.7 * count) for count in total_counts]
val_counts = [int(0.15 * count) for count in total_counts]
test_counts = [count - t - v for count, t, v in zip(total_counts, train_counts, val_counts)]
class_dist = pd.DataFrame({
    'Class': classes,
    'Train_Count': train_counts,
    'Val_Count': val_counts,
    'Test_Count': test_counts
})
print("Class Distribution (approximate):\n", class_dist)

fig, axes = plt.subplots(4, 5, figsize=(15, 12))
for i, ax in enumerate(axes.flat):
    if i < num_classes:
        class_dir = os.path.join(data_dir, classes[i])
        sample_path = os.path.join(class_dir, os.listdir(class_dir)[0])
        sample_img = load_and_preprocess_image(sample_path)
        ax.imshow(sample_img)
        ax.set_title(classes[i])
        ax.axis('off')
plt.tight_layout()
plt.savefig('eda_samples.png')
plt.show()

plt.figure(figsize=(10, 6))
sns.barplot(data=class_dist.melt(id_vars='Class', value_vars=['Train_Count', 'Val_Count',
                                                              'Test_Count']),
                                                    x='Class', y='value', hue='variable')
plt.title('Class Distribution Across Splits')
plt.xticks(rotation=90)
plt.savefig('class_dist.png')
plt.show()

def data_generator(paths, labels, batch_size=8):
    indices = np.arange(len(paths))
    while True:
        np.random.shuffle(indices)
        for start in range(0, len(indices), batch_size):
            end = min(start + batch_size, len(indices))
            batch_indices = indices[start:end]
            batch_paths = [paths[i] for i in batch_indices]
            batch_labels = [labels[i] for i in batch_indices]
            batch_images = [load_and_preprocess_image(p) for p in batch_paths]
            yield np.array(batch_images), tf.keras.utils.to_categorical(batch_labels, num_classes)

train_gen = data_generator(train_paths, train_labels, batch_size=8)
val_gen = data_generator(val_paths, val_labels, batch_size=8)
test_gen = data_generator(test_paths, test_labels, batch_size=8)

print("Step 3: CNN Model Design and Implementation")

model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=(128, 128, 3)),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(32, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    BatchNormalization(),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax', dtype='float32')
])

model.summary()
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

print("Step 4: Model Training and Optimisation")

callbacks = [
    EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True),
    ModelCheckpoint('best_model.h5', monitor='val_accuracy', save_best_only=True),
    ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=1e-7)
]
```

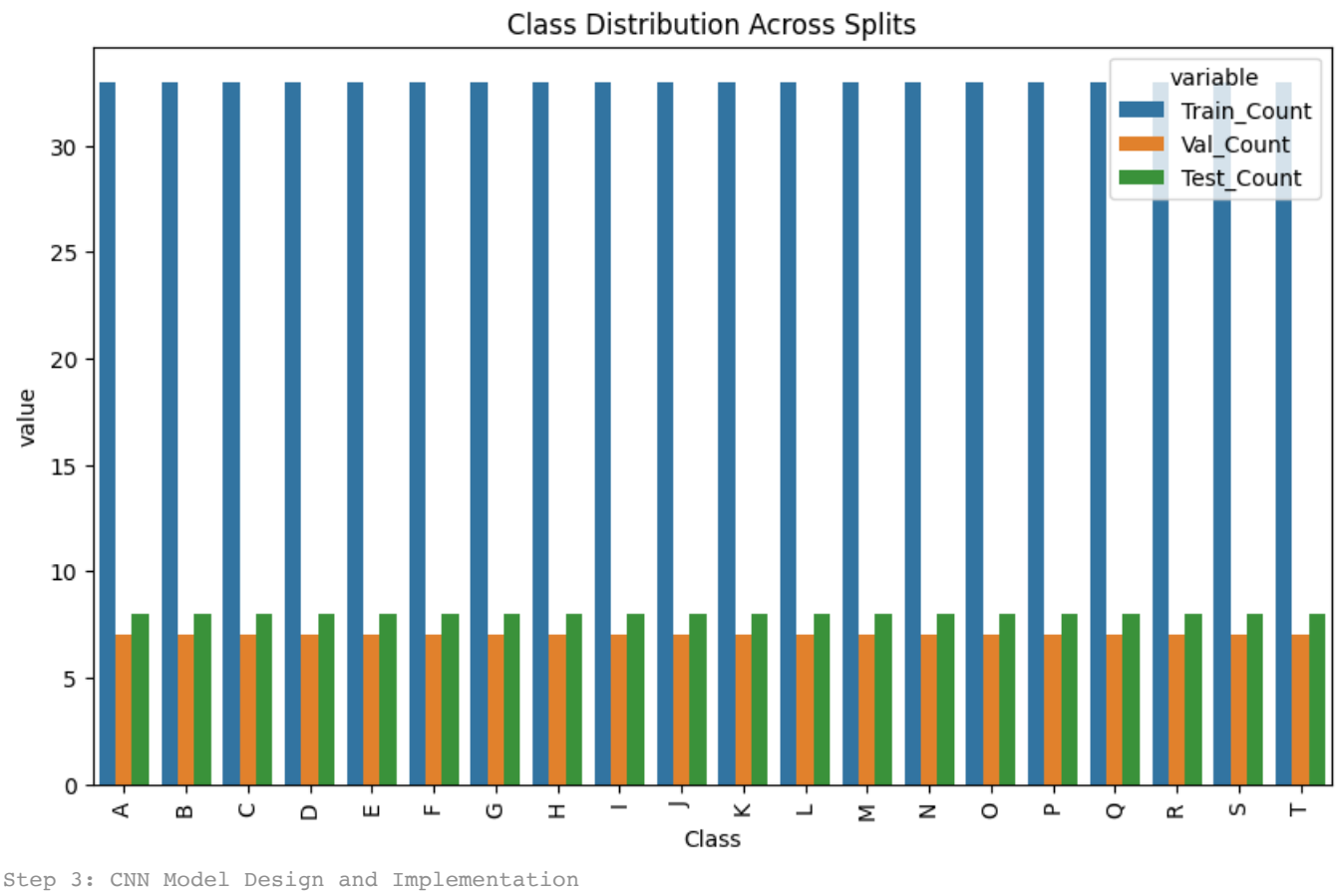
```
steps_per_epoch = len(train_paths) // 8
val_steps = len(val_paths) // 8
history = model.fit(
    train_gen, epochs=20, steps_per_epoch=steps_per_epoch,
    validation_data=val_gen, validation_steps=val_steps,
    callbacks=callbacks, verbose=1
)

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.legend()
plt.savefig('training_history.png')
plt.show()

history_dict = {k: v for k, v in history.history.items()}
with open('training_history.json', 'w') as f:
    json.dump(history_dict, f)

print("Step 5: Model Evaluation and Testing")

model.load_weights('best_model.h5')

test_steps = len(test_paths) // 8
test_loss, test_acc = model.evaluate(test_gen, steps=test_steps, verbose=0)
print(f"Test Accuracy: {test_acc:.4f}, Test Loss: {test_loss:.4f}")

test_preds = []
test_true = []
for _ in range(test_steps):
    imgs, lbls = next(test_gen)
    preds = model.predict(imgs)
    test_preds.extend(np.argmax(preds, axis=1))
    test_true.extend(np.argmax(lbls, axis=1))

report = classification_report(test_true, test_preds, target_names=classes, output_dict=True)
report_df = pd.DataFrame(report).transpose()
print("Classification Report:\n", report_df)
report_df.to_csv('classification_report.csv')

cm = confusion_matrix(test_true, test_preds)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', xticklabels=classes, yticklabels=classes, cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.xticks(rotation=90)
plt.yticks(rotation=0)
plt.savefig('confusion_matrix.png')
plt.show()

print("Step 6: Deployment and Final Report")

model.save('cancer_classifier_model.h5')

'''
deployment_html =
<!DOCTYPE html>
<html>
<head><title>Cancer Classifier</title></head>
<body>
    <h1>Upload .tif Image for Prediction</h1>
    <form method="post" enctype="multipart/form-data">
        <input type="file" name="file">
        <input type="submit" value="Predict">
    </form>
    {% if prediction %}
    <h2>Prediction: {{ prediction }}</h2>
    <img src="{{ image_b64 }}" width="300">
    {% endif %}
</body>
</html>


app = Flask(__name__)
app.template_folder = ''

@app.route('/', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        file = request.files['file']
        if file and file.filename.endswith('.tif'):
            img = tifffile.imread(file)
            if len(img.shape) == 2:
                img = np.stack((img,) * 3, axis=-1)
            img_pil = Image.fromarray(img).resize((128, 128))
            img_array = np.array(img_pil) / 255.0
            img_array = np.expand_dims(img_array, axis=0)
            pred = model.predict(img_array)[0]
```

```
            pred_class = classes[np.argmax(pred)]
            img_buffer = io.BytesIO()
            img_pil.save(img_buffer, format='PNG')
            img_b64 = base64.b64encode(img_buffer.getvalue()).decode()
            return render_template_string(deployment_html, prediction=pred_class, image_b64=img_b64)
    return render_template_string(deployment_html)

print("Deployment ready! Uncomment app.run() to start Flask server.")
'''
print("Model saved as 'cancer_classifier_model.h5'. Load with tf.keras.models.load_model() for inference.")

report_summary = {
    'Step': ['1. Data Collection & Preprocessing', '2. EDA', '3. CNN Design', '4. Training & Optimization',
    'Key Actions': [
        'Downloaded via Kaggle API; Loaded .tif with tifffile; Resized to 128x128; Normalized; Stratified sp
        'Class dist plots (approx); Sample viz (one per class load); Custom generator without augmentation.
        'Sequential CNN: 4 Conv blocks (16-64 filters) + BN + Pool + Dense(256) + Dropout + Softmax(20).',
        'Adam(0.001); 20 epochs; Callbacks: EarlyStop(p=10), Checkpoint, ReduceLR(f=0.2); Generator batches
        'Test acc/loss via generator; Batched predict; Class report CSV; Confusion heatmap; History plots/JS
        'Saved .h5 model'
    ],
    'Outputs Generated': [
        './data/kimia-path-960/, train/val/test paths (no arrays).',
        'eda_samples.png, class_dist.png.',
        'Model summary printed.',
        'best_model.h5, training_history.png, training_history.json.',
        'classification_report.csv, confusion_matrix.png.',
        'cancer_classifier_model.h5'
    ]
}
report_df = pd.DataFrame(report_summary)
print("\nFinal Report Summary:")
print(report_df.to_markdown(index=False))
report_df.to_csv('final_report.csv', index=False)
```

```
[]
Step 1: Data Collection and Preprocessing
/content
Dataset has been segregated into 20 folders.
Found 20 classes: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R'
Total images collected: 960
Train: 672, Val: 144, Test: 144
Step 2: Exploratory Data Analysis
Class Distribution (approximate):
    Class  Train_Count  Val_Count  Test_Count
0       A           33          7           8
1       B           33          7           8
2       C           33          7           8
3       D           33          7           8
4       E           33          7           8
5       F           33          7           8
6       G           33          7           8
7       H           33          7           8
8       I           33          7           8
9       J           33          7           8
10      K           33          7           8
11      L           33          7           8
12      M           33          7           8
13      N           33          7           8
14      O           33          7           8
15      P           33          7           8
16      Q           33          7           8
17      R           33          7           8
18      S           33          7           8
19      T           33          7           8
```







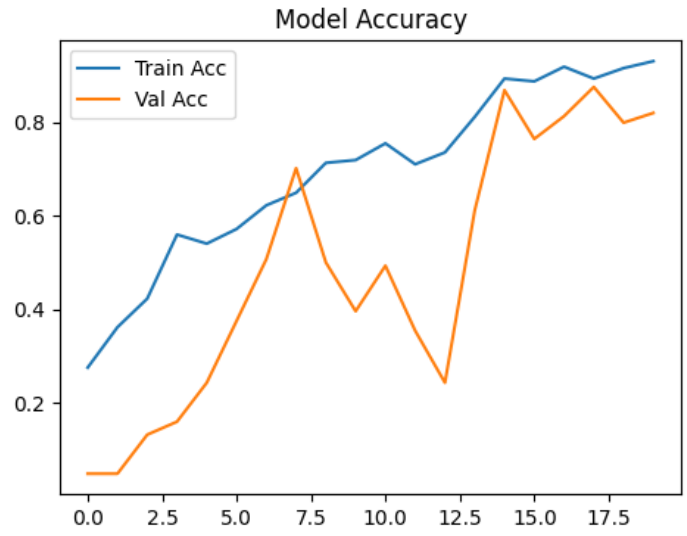

```
Step 3: CNN Model Design and Implementation
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

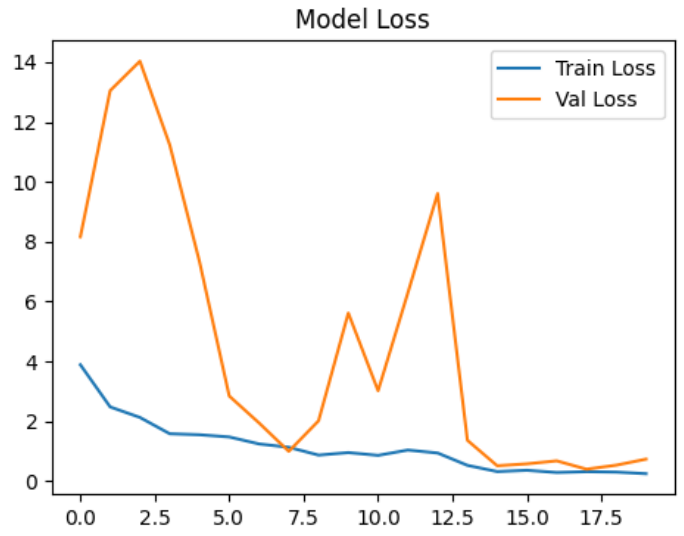
Model: "sequential"

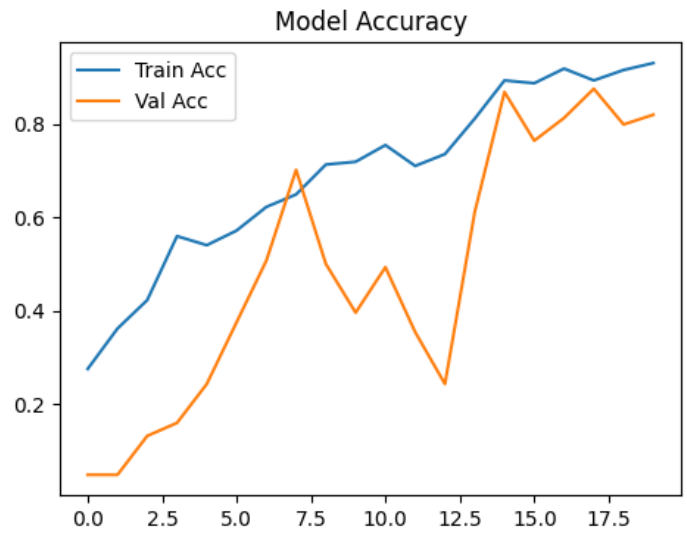
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 16)	448
batch_normalization (BatchNormalization)	(None, 126, 126, 16)	64
max_pooling2d (MaxPooling2D)	(None, 63, 63, 16)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	4,640
batch_normalization_1 (BatchNormalization)	(None, 61, 61, 32)	128
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	18,496
batch_normalization_2 (BatchNormalization)	(None, 28, 28, 64)	256
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 12, 12, 64)	256
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 256)	590,080
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 20)	5,140

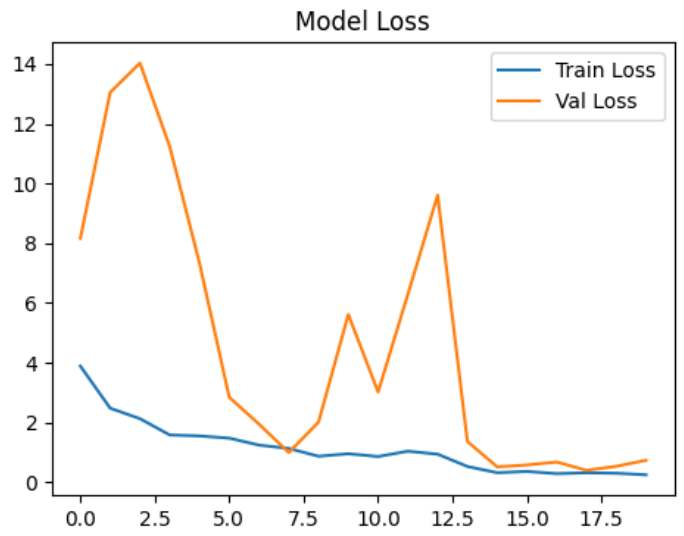
```
 Total params: 656,436 (2.50 MB)
 Trainable params: 656,084 (2.50 MB)
 Non-trainable params: 352 (1.38 KB)
Step 4: Model Training and Optimisation
Epoch 1/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.1944 - loss: 4.7773WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 327s 4s/step - accuracy: 0.1953 - loss: 4.7669 - val_accuracy: 0.0486 - val_loss
Epoch 2/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 320s 4s/step - accuracy: 0.3326 - loss: 2.5445 - val_accuracy: 0.0486 - val_loss
Epoch 3/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.4250 - loss: 2.1600WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 321s 4s/step - accuracy: 0.4249 - loss: 2.1596 - val_accuracy: 0.1319 - val_loss
Epoch 4/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.5691 - loss: 1.5380WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 319s 4s/step - accuracy: 0.5689 - loss: 1.5385 - val_accuracy: 0.1597 - val_loss
Epoch 5/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.5490 - loss: 1.5248WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.5489 - loss: 1.5251 - val_accuracy: 0.2431 - val_loss
Epoch 6/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.5954 - loss: 1.3392WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.5951 - loss: 1.3408 - val_accuracy: 0.3750 - val_loss
Epoch 7/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.5809 - loss: 1.3667WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.5814 - loss: 1.3652 - val_accuracy: 0.5069 - val_loss
Epoch 8/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.6552 - loss: 1.2601WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.6551 - loss: 1.2586 - val_accuracy: 0.7014 - val_loss
Epoch 9/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.7135 - loss: 0.9158 - val_accuracy: 0.5000 - val_loss
Epoch 10/20
```

```
84/84 ━━━━━━━━━━━━━━━━━━━━ 317s 4s/step - accuracy: 0.7183 - loss: 0.9507 - val_accuracy: 0.3958 - val_loss
Epoch 11/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.7657 - loss: 0.8564 - val_accuracy: 0.4931 - val_loss
Epoch 12/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 323s 4s/step - accuracy: 0.7501 - loss: 0.8838 - val_accuracy: 0.3542 - val_loss
Epoch 13/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 321s 4s/step - accuracy: 0.7457 - loss: 0.8493 - val_accuracy: 0.2431 - val_loss
Epoch 14/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.7765 - loss: 0.6385 - val_accuracy: 0.6111 - val_loss
Epoch 15/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.9094 - loss: 0.2987WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.9092 - loss: 0.2990 - val_accuracy: 0.8681 - val_loss
Epoch 16/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.8914 - loss: 0.3621 - val_accuracy: 0.7639 - val_loss
Epoch 17/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.9277 - loss: 0.2654 - val_accuracy: 0.8125 - val_loss
Epoch 18/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 0s 4s/step - accuracy: 0.8874 - loss: 0.3270WARNING:absl:You are saving your mod
84/84 ━━━━━━━━━━━━━━━━━━━━ 322s 4s/step - accuracy: 0.8875 - loss: 0.3269 - val_accuracy: 0.8750 - val_loss
Epoch 19/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 323s 4s/step - accuracy: 0.9379 - loss: 0.2531 - val_accuracy: 0.7986 - val_loss
84/84 ━━━━━━━━━━━━━━━━━━━━ 323s 4s/step - accuracy: 0.9379 - loss: 0.2531 - val_accuracy: 0.7986 - val_loss
Epoch 20/20
Epoch 20/20
84/84 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.9300 - loss: 0.2464 - val_accuracy: 0.8194 - val_loss
84/84 ━━━━━━━━━━━━━━━━━━━━ 318s 4s/step - accuracy: 0.9300 - loss: 0.2464 - val_accuracy: 0.8194 - val_loss
```



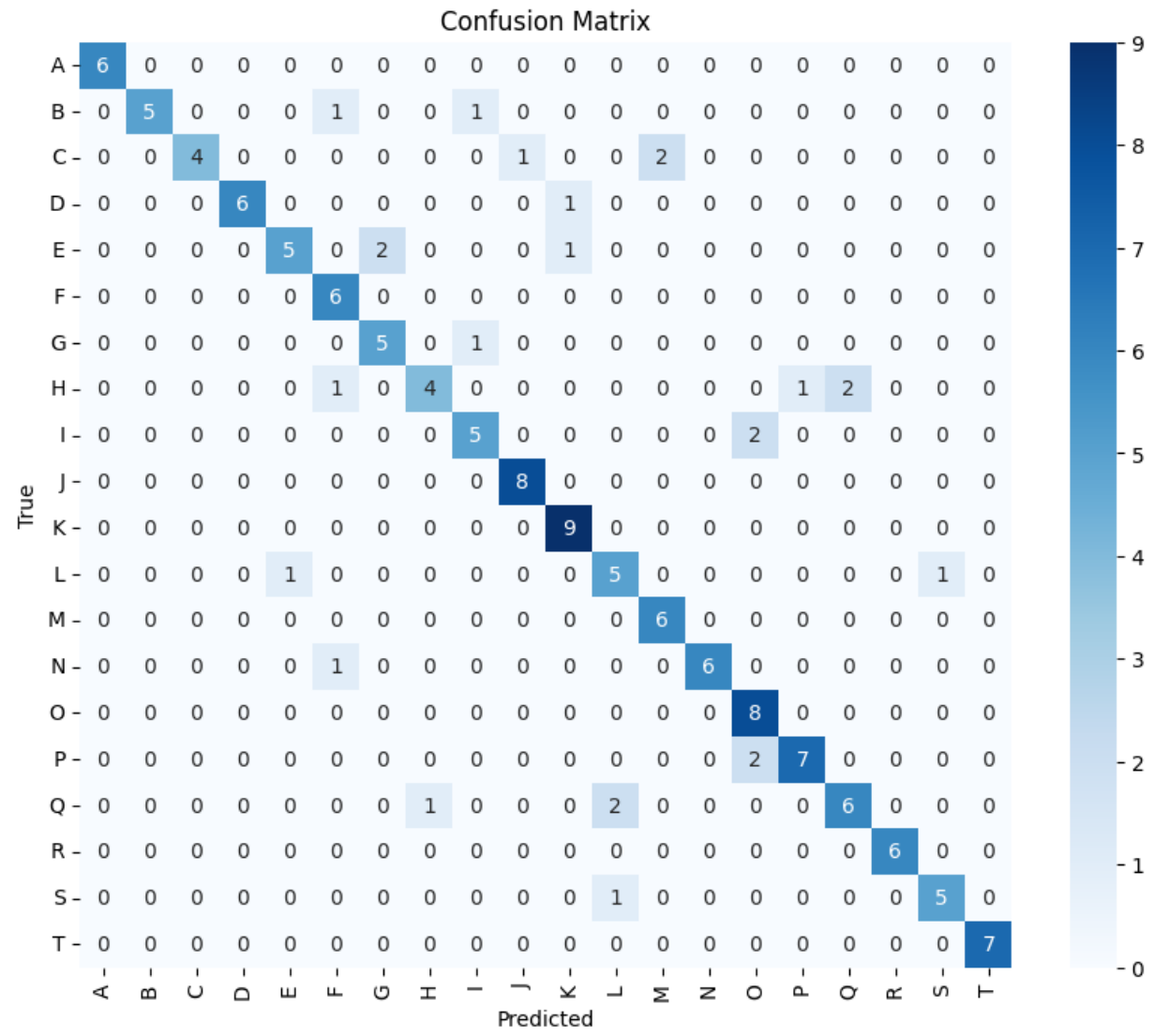


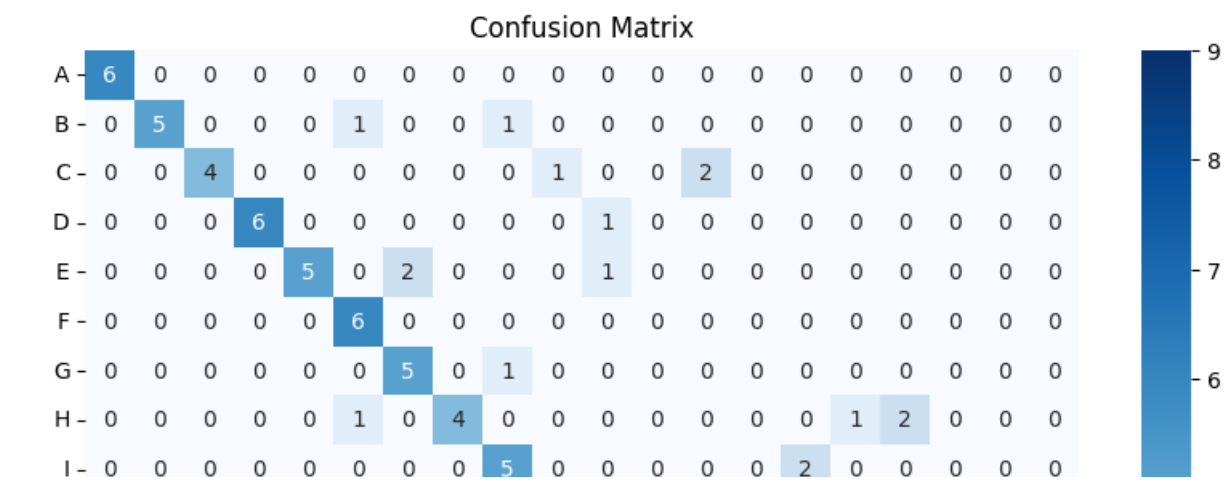


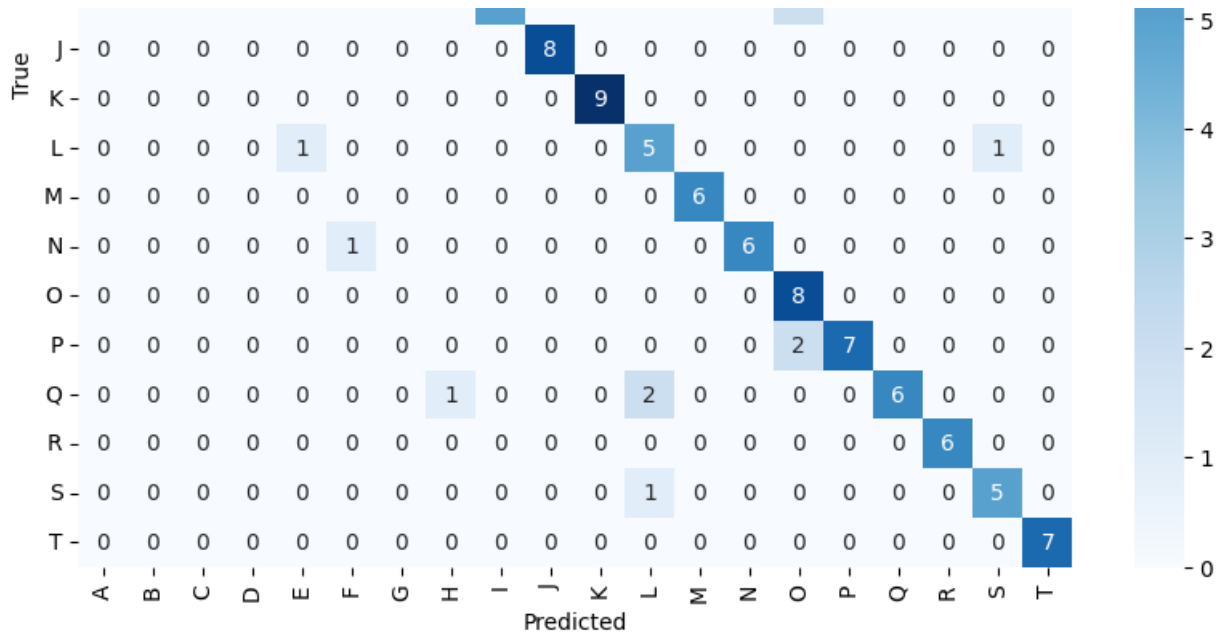



```
Step 5: Model Evaluation and Testing
Step 5: Model Evaluation and Testing
Test Accuracy: 0.8333, Test Loss: 0.5848
Test Accuracy: 0.8333, Test Loss: 0.5848
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 307ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 133ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 121ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 118ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 119ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 125ms/step
Classification Report:
```

```
                precision    recall  f1-score     support
A               1.000000  1.000000  1.000000    6.000000
B               1.000000  0.714286  0.833333    7.000000
C               1.000000  0.571429  0.727273    7.000000
D               1.000000  0.857143  0.923077    7.000000
E               0.833333  0.625000  0.714286    8.000000
F               0.666667  1.000000  0.800000    6.000000
G               0.714286  0.833333  0.769231    6.000000
H               0.800000  0.500000  0.615385    8.000000
I               0.714286  0.714286  0.714286    7.000000
J               0.888889  1.000000  0.941176    8.000000
K               0.818182  1.000000  0.900000    9.000000
L               0.625000  0.714286  0.666667    7.000000
M               0.750000  1.000000  0.857143    6.000000
N               1.000000  0.857143  0.923077    7.000000
O               0.666667  1.000000  0.800000    8.000000
P               0.875000  0.777778  0.823529    9.000000
Q               0.750000  0.666667  0.705882    9.000000
R               1.000000  1.000000  1.000000    6.000000
S               0.833333  0.833333  0.833333    6.000000
T               1.000000  1.000000  1.000000    7.000000
accuracy        0.826389  0.826389  0.826389    0.826389
macro avg       0.846782  0.833234  0.827384  144.000000
weighted avg    0.844864  0.826389  0.822942  144.000000
Classification Report:
                precision    recall  f1-score     support
A               1.000000  1.000000  1.000000    6.000000
B               1.000000  0.714286  0.833333    7.000000
C               1.000000  0.571429  0.727273    7.000000
D               1.000000  0.857143  0.923077    7.000000
E               0.833333  0.625000  0.714286    8.000000
F               0.666667  1.000000  0.800000    6.000000
G               0.714286  0.833333  0.769231    6.000000
H               0.800000  0.500000  0.615385    8.000000
I               0.714286  0.714286  0.714286    7.000000
J               0.888889  1.000000  0.941176    8.000000
K               0.818182  1.000000  0.900000    9.000000
L               0.625000  0.714286  0.666667    7.000000
M               0.750000  1.000000  0.857143    6.000000
N               1.000000  0.857143  0.923077    7.000000
O               0.666667  1.000000  0.800000    8.000000
P               0.875000  0.777778  0.823529    9.000000
Q               0.750000  0.666667  0.705882    9.000000
R               1.000000  1.000000  1.000000    6.000000
S               0.833333  0.833333  0.833333    6.000000
T               1.000000  1.000000  1.000000    7.000000
accuracy        0.826389  0.826389  0.826389    0.826389
macro avg       0.846782  0.833234  0.827384  144.000000
weighted avg    0.844864  0.826389  0.822942  144.000000
```








```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model
Step 6: Deployment and Final Report
Model saved as 'cancer_classifier_model.h5'. Load with tf.keras.models.load_model() for inference.

Final Report Summary:
| Step                                | Key Actions
|:------------------------------------|:-----------------------------------------------------------------
| 1. Data Collection & Preprocessing | Downloaded via Kaggle API; Loaded .tif with tifffile; Resized to 128
| 2. EDA                              | Class dist plots (approx); Sample viz (one per class load); Custom g
| 3. CNN Design                       | Sequential CNN: 4 Conv blocks (16-64 filters) + BN + Pool + Dense(25
| 4. Training & Optimization          | Adam(0.001); 20 epochs; Callbacks: EarlyStop(p=10), Checkpoint, Redu
| 5. Evaluation & Testing             | Test acc/loss via generator; Batched predict; Class report CSV; Confu
| 6. Deployment & Report              | Saved .h5 model
Step 6: Deployment and Final Report
Model saved as 'cancer_classifier_model.h5'. Load with tf.keras.models.load_model() for inference.

Final Report Summary:
| Step                                | Key Actions
|:------------------------------------|:-----------------------------------------------------------------
| 1. Data Collection & Preprocessing | Downloaded via Kaggle API; Loaded .tif with tifffile; Resized to 128
| 2. EDA                              | Class dist plots (approx); Sample viz (one per class load); Custom g
| 3. CNN Design                       | Sequential CNN: 4 Conv blocks (16-64 filters) + BN + Pool + Dense(25
| 4. Training & Optimization          | Adam(0.001); 20 epochs; Callbacks: EarlyStop(p=10), Checkpoint, Redu
| 5. Evaluation & Testing             | Test acc/loss via generator; Batched predict; Class report CSV; Confu
| 6. Deployment & Report              | Saved .h5 model
```