Functions to Implement

1. View Images (for Inspection of Dataset)

Purpose: Quickly scan your raw images to make sure labels look correct, there are no weird aspect-ratio issues, color channels are right, etc.

Implementation:

```
from PIL import Image
import matplotlib.pyplot as plt
import random

def view_samples(image_paths, labels, n=16):
    picks = random.sample(list(zip(image_paths, labels)), n)
    cols = 4; rows = n // cols
    fig, axs = plt.subplots(rows, cols, figsize=(12,12))
    for ax, (path, label) in zip(axs.flat, picks):
        img = Image.open(path)
        ax.imshow(img)
        ax.set_title(label)
        ax.axis('off')
    plt.show()
```

---

2. View Model Architecture (as Block Diagrams)

Purpose: Get a high-level block diagram of every layer (and tensor shape) to sanity-check your network design.

Implementation (Keras example):

```
from tensorflow.keras.utils import plot_model

# Writes a file 'architecture.png' showing each layer and its output shape
plot_model(model, to_file='architecture.png', show_shapes=True)
```

(Requires `pydot` and Graphviz installed.)

---

3. Plot Loss vs. Epoch (Train & Validation)

Purpose: Visualize training stability, overfitting/underfitting, convergence speed.

Implementation:

```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

---

4. Plot Accuracy vs. Epoch (Train & Validation)

Purpose: See how quickly accuracy rises, whether it plateaus, and if validation accuracy diverges from training.

Implementation:

```
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

---

5. Save and Load Model Parameters (Weights)

Purpose: Persist your trained weights so you don't have to retrain every time, and to deploy or resume training later.

Implementation:

- **Keras:**

```
# Save
model.save_weights('my_model_weights.h5')

# Load
model.load_weights('my_model_weights.h5')
```

- **PyTorch:**

```
# Save
torch.save(model.state_dict(), 'model.pt')

# Load
model.load_state_dict(torch.load('model.pt'))
```

---

6. Display Test-Set Samples with Predictions

Purpose: Spot-check how your model is doing on unseen data—are there systematic misclassifications or weird artifacts?

Implementation:

```
import matplotlib.pyplot as plt

def show_predictions(model, test_loader, class_names, n=9):
    imgs, labels = next(iter(test_loader))
    preds = model.predict(imgs)  # or model(imgs) in PyTorch
    top_preds = preds.argmax(axis=1)

    fig, axs = plt.subplots(3, 3, figsize=(9,9))
    for i, ax in enumerate(axs.flat):
        ax.imshow(imgs[i])
        title = f"P: {class_names[top_preds[i]]}\nT: {class_names[labels[i]]}"
        ax.set_title(title)
        ax.axis('off')
    plt.show()
```