Task I: Understanding the Provided Python Scripts

We will analyze the Python scripts step by step, focusing on the following:

1. **Network Architecture**
2. **Optimizer**
3. **Loss Function**
4. **TensorBoard Logging**

---

1. Network Architecture

Original Model (`model.py`)

The provided model is a **fully connected feedforward neural network (FFNN)**:

Code Breakdown

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()  # Flattens input image into a 1D
vector
        self.ffnn = nn.Sequential(
            nn.Linear(128 * 128 * 3, 2048),  # Input Layer
            nn.ReLU(),  # Activation Function
            nn.Linear(2048, 2048),  # Hidden Layer
            nn.ReLU(),  # Activation Function
            nn.Linear(2048, 37),  # Output Layer
        )
```

Layer Analysis

Layer	Type	Description
Flatten	`nn.Flatten()`	Converts image from `(3, 128, 128)` to `(49152,)` (3 × 128 × 128)
Fully Connected (Dense) Layer 1	`nn.Linear(49152, 2048)`	First dense layer with 2048 neurons
Activation	`nn.ReLU()`	Non-linearity to learn complex patterns
Fully Connected (Dense) Layer 2	`nn.Linear(2048, 2048)`	Second dense layer with 2048 neurons
Activation	`nn.ReLU()`	Non-linearity applied again
Output Layer	`nn.Linear(2048, 37)`	Final layer with 37 output neurons (for 37 breeds)

Key Observations

- This model **does not use Convolutional Layers** (CNN), which are more efficient for image tasks.
- Instead, it **flattens images into a vector**, losing spatial information.
- **Dense layers alone** require too many parameters, making it inefficient for image classification.

---

2. Optimizer

```
optimizer = torch.optim.SGD(model.parameters(), lr=LEARNING_RATE)
```

Optimizer	Type	Description
SGD (Stochastic Gradient Descent)	`torch.optim.SGD`	Updates weights using gradients with a fixed learning rate.

Why use SGD?

✅ Simple and effective for small datasets.
❌ Slower convergence than adaptive optimizers (e.g., Adam).
❌ Might require **learning rate scheduling** for better performance.

---

3. Loss Function

```
loss_fn = nn.CrossEntropyLoss()
```

Loss Function	Type	Description
CrossEntropyLoss	`nn.CrossEntropyLoss()`	Measures classification error between predicted logits and actual labels.

Why is `CrossEntropyLoss` used?

✅ Suitable for **multi-class classification** (37 breeds).
✅ Works well with **raw logits** (before softmax).
❌ For binary classification (dogs vs. cats), `BinaryCrossEntropyLoss` would be better.

---

4. TensorBoard Logging

TensorBoard is used for tracking loss and accuracy.

Logging Setup (`train.py`):

```
from torch.utils.tensorboard.writer import SummaryWriter

writer = SummaryWriter()
```

Logging Metrics During Training:

```
writer.add_scalar("Loss/Train", train_loss, step)
writer.add_scalar("Accuracy/Train", correct, step)
```

Logging During Evaluation:

```
writer.add_scalar("Loss/Test", test_loss, step)
writer.add_scalar("Accuracy/Test", correct, step)
```

Benefit	Description
Tracks loss & accuracy	Helps visualize training progress
Identifies overfitting	Can compare train & validation loss
Optimizes hyperparameters	Helps in tuning learning rates