Summarize of the project

Number of Iterations (Epochs):

There are 3 iterations/ epochs. It determines how many times the entire training dataset is processed by the model during training.

One epoch consists of one forward pass (computing predictions), one backward pass (computing gradients), and one weight update. In each epoch, the model learns from the entire dataset.

I increase the No of epochs from 1 to 3 in this model and the accuracy score incresed from 0.49 to 0.57. I noticed that when I increased the epoch no, the loss for each epoch decreased.

Learning rate:

I started with increasing the learning rate from 0.0005 to 0.01 because it was too small and would have resulted in a slow convergence rate. The learning rate of 0.01 was too high and caused the model to overshoot the optimal solution. I then adjusted the learning rate to 0.001 and gradually increased it to 0.002. This balance between overshooting and convergence speed worked well in this case.

Optimzie method:

I use Stochastic Gradient Descent (SGD) as the optimize method because it has a faster convergence.

This generally helps me updating the weights of the model.

My Findings:

A Pipeline for Image Classification

In my image classification task, I've employed a well-established pipeline to achieve accurate results. Here are the key steps I've followed:

1. Data Preprocessing:
   - First, I load and transform the image data from raw vectors into tensors. This transformation is crucial for compatibility with deep learning models.
   - To achieve this, I utilize data loaders and a composition of transformers. These transformers include operations like horizontal flipping, normalization, and conversion to tensors.
   - For instance, horizontal flipping helps diversify the training data, while normalization ensures the input values are within a consistent range.

```
transform = transforms.Compose([
    transforms.RandomHorizontalFlip(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    transforms.ToTensor()
])
```

2. Model Selection and Hyperparameters:
   - Next, I select a Convolutional Neural Network (CNN) model that suits my specific task. I've noticed that increasing the number of output channels in the model architecture often leads to improved performance.(Need to learn this parameters thing further as well, the formua behind this??)
   - Setting hyperparameters like learning rates, batch sizes, and model depth plays a crucial role in model training. Experimentation helps identify the best values.

```
model = CNN(num_output_channels=64, ...)
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

3. Loss Function and Optimization:
   - I define a loss function. Cross-entropy loss is my choiceof use in image classification tasks.
   - For optimization, I employ the Stochastic Gradient Descent (SGD) method. SGD is known for its speedy convergence and adaptability to large datasets.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

4. Training:
   - With the setup complete, I specify the number of epochs for training. Increasing the number of epochs typically leads to improved accuracy on the training data, although it's essential to monitor for overfitting.

```
num_epochs = 3
for epoch in range(num_epochs):
    train_model(...)
```

5. Testing:
   - Finally, I employ the trained model to make predictions on the test data, evaluating its performance on unseen samples.
6. result.the model reaches the accuracy of prediciting of 0.57