```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision.datasets import FashionMNIST
import time
import numpy as np
import matplotlib.pyplot as plt
```

VisionTransformer for image classification tasks. It initializes the model's layers, parameters, and processes input images to produce class predictions.

```
# Define Vision Transformer class
class VisionTransformer(nn.Module):
    def __init__(self, img_size=28, patch_size=7, embed_dim=128, num_heads=4, num_layers=4, num_classes=10):
        super(VisionTransformer, self).__init__()
        self.patch_size = patch_size
        self.embed_dim = embed_dim
        self.num_patches = (img_size // patch_size) ** 2
        self.patch_embedding = nn.Conv2d(1, embed_dim, kernel_size=patch_size, stride=patch_size)
        self.positional_embedding = nn.Parameter(torch.randn(1, self.num_patches + 1, embed_dim))
        self.transformer_encoder = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model=embed_dim, nhead=num_heads),
            num_layers=num_layers
        )
        self.classification_head = nn.Linear(embed_dim, num_classes)

    def forward(self, x):
        x = self.patch_embedding(x)
        x = x.flatten(2).transpose(1, 2)
        x = torch.cat((x, self.positional_embedding.expand(x.shape[0], -1, -1)), dim=1)
        x = self.transformer_encoder(x)
        x = x.mean(dim=1)
        x = self.classification_head(x)
        return x
```

The NanoGPTVision model class is designed for vision tasks. It consists of a vision transformer and a linear layer for processing textual data. During the forward pass, image data is processed through the vision transformer and text data through the linear layer.

```
# Define NanoGPT Model class for vision tasks
class NanoGPTVision(nn.Module):
    def __init__(self, img_size=28, patch_size=7, embed_dim=128, num_heads=4, num_layers=4, num_classes=10, vocab_size=50000):
        super(NanoGPTVision, self).__init__()
        self.vision_transformer = VisionTransformer(img_size, patch_size, embed_dim, num_heads, num_layers, embed_dim)
        self.text_model = nn.Linear(1, embed_dim)

    def forward(self, images, tokens):
        vision_output = self.vision_transformer(images)

        # Reshape tokens to match the expected shape for matrix multiplication
        tokens = tokens.view(tokens.size(0), 1)

        # Convert tokens to float data type before passing to the linear layer
        tokens = tokens.float()

        text_output = self.text_model(tokens)
        combined_output = vision_output + text_output  # Combine vision and text features
        return combined_output
```

Prepares the FashionMNIST dataset for training and testing by defining transformations, loading datasets, creating DataLoader objects, and setting the device for computation to GPU if available, otherwise to CPU.

```
# Data transformations and loaders
transform = transforms.Compose([
    transforms.ToTensor(),
])

train_dataset = FashionMNIST(root='./data', train=True, transform=transform, download=True)
test_dataset = FashionMNIST(root='./data', train=False, transform=transform, download=True)

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

batch_size = 32
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/train-
100%|██████████| 26421880/26421880 [00:01<00:00, 14785402.50it/s]
Extracting ./data/FashionMNIST/raw/train-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/train-
100%|██████████| 29515/29515 [00:00<00:00, 268016.94it/s]
Extracting ./data/FashionMNIST/raw/train-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw/t10k-im
100%|██████████| 4422102/4422102 [00:00<00:00, 5100398.54it/s]
Extracting ./data/FashionMNIST/raw/t10k-images-idx3-ubyte.gz to ./data/FashionMNIST/raw

Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
Downloading http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz to ./data/FashionMNIST/raw/t10k-la
100%|██████████| 5148/5148 [00:00<00:00, 5807497.85it/s]Extracting ./data/FashionMNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/Fash
```

Initializes the NanoGPTVision model, loss function, optimizer, and learning rate scheduler for training.

```
# Model, loss, optimizer, and scheduler
model = NanoGPTVision().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/transformer.py:286: UserWarning: enable_nested_tensor is True, but self.us
  warnings.warn(f"enable_nested_tensor is True, but self.use_nested_tensor is False because {why_not_sparsity_fast_path}")
```

Trains a model, evaluates it on a test set after each epoch, and generates visualizations of a subset of test images with predicted labels.

```
# Training loop
total_training_time = 0.0
num_epochs = 10
for epoch in range(num_epochs):
    start_time = time.time()
    model.train()
    for images, labels in train_loader:
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images, torch.zeros(images.shape[0], dtype=torch.long).to(device))

        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    scheduler.step()

    epoch_time = time.time() - start_time
    total_training_time += epoch_time

    # Evaluation on test set
    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images, torch.zeros(images.shape[0], dtype=torch.long).to(device))
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        print(f'Epoch [{epoch+1}/{num_epochs}], Test Accuracy: {100 * correct / total:.2f}%, Epoch Time: {epoch_time:.2f} seconds')

print(f'Total Training Time: {total_training_time:.2f} seconds')

# Final testing
model.eval()
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images, torch.zeros(images.shape[0], dtype=torch.long).to(device))
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

        # Plot images
        fig, axs = plt.subplots(4, 4, figsize=(12, 12))
        for i in range(4):
            for j in range(4):
                idx = i * 4 + j
                img = images[idx].cpu().numpy().squeeze()
                axs[i, j].imshow(img, cmap='gray')
                axs[i, j].set_title(f'Label: {class_names[labels[idx].item()]}\nPredicted: {class_names[predicted[idx].item()]}')
                axs[i, j].axis('off')
        plt.show()

    print(f'Test Accuracy: {100 * correct / total:.2f}%')
```

```
Epoch [1/10], Test Accuracy: 79.40%, Epoch Time: 27.01 seconds
Epoch [2/10], Test Accuracy: 83.04%, Epoch Time: 26.11 seconds
Epoch [3/10], Test Accuracy: 83.86%, Epoch Time: 25.60 seconds
Epoch [4/10], Test Accuracy: 85.85%, Epoch Time: 25.48 seconds
Epoch [5/10], Test Accuracy: 86.20%, Epoch Time: 25.61 seconds
Epoch [6/10], Test Accuracy: 86.51%, Epoch Time: 25.71 seconds
Epoch [7/10], Test Accuracy: 86.62%, Epoch Time: 25.44 seconds
Epoch [8/10], Test Accuracy: 86.53%, Epoch Time: 26.14 seconds
Epoch [9/10], Test Accuracy: 86.55%, Epoch Time: 25.37 seconds
Epoch [10/10], Test Accuracy: 86.52%, Epoch Time: 25.63 seconds
Total Training Time: 258.09 seconds
```