# MNIST Classification with Softmax Regression and FCNN

This notebook implements a Softmax regression model and a Fully Connected Neural Network to classify handwritten digits from the MNIST dataset.

## Dataset Overview

The MNIST dataset contains 70,000 grayscale images of handwritten digits (0-9), split into 60,000 training and 10,000 test images. Each image is 28x28 pixels. The data is concatenated and split into train:validate:test with ratio 0.6:0.2:0.2 respectively.

# Data Loading and Preprocessing

```python
import torch
import numpy as np
import pandas as pd
from PIL import Image
import codecs
import matplotlib.pyplot as plt
import math

import kagglehub

# Download latest version
path = kagglehub.dataset_download("hojjatk/mnist-dataset")

print("Path to dataset files:", path)

# copy the downloaded dataset to the current directory
import shutil
shutil.copytree(path, "dataset", dirs_exist_ok=True)
```

```
Path to dataset files: /kaggle/input/mnist-dataset
```

```
'dataset'
```

## Data Reading Utilities

- Custom functions to read MNIST's binary file formats
- `get_int()`: Converts binary data to integers
- `read_label_file()`: Extracts label data
- `read_image_file()`: Extracts image data with proper reshaping

```python
def get_int(b):
    return int(codecs.encode(b, 'hex'), 16)

def read_label_file(path):
    with open(path, 'rb') as f:
        data = f.read()
        assert get_int(data[:4]) == 2049
        length = get_int(data[4:8])
        parsed = np.frombuffer(data, dtype=np.uint8, offset=8)
        return np.array(parsed, dtype=np.uint8).reshape(length)


def read_image_file(path):
    with open(path, 'rb') as f:
        data = f.read()
        assert get_int(data[:4]) == 2051
        length = get_int(data[4:8])
        num_rows = get_int(data[8:12])
        num_cols = get_int(data[12:16])
        parsed = np.frombuffer(data, dtype=np.uint8, offset=16)
        return np.array(parsed,
dtype=np.uint8).reshape((length,num_rows,num_cols))
```

## Data Loading

```python
train_images = read_image_file('dataset/train-images.idx3-ubyte')
train_labels = read_label_file('dataset/train-labels.idx1-ubyte')
test_images = read_image_file('dataset/t10k-images.idx3-ubyte')
test_labels = read_label_file('dataset/t10k-labels.idx1-ubyte')

print(train_images.shape, train_labels.shape)
```

```
(60000, 28, 28) (60000,)
```

```python
# concat train and test
images = np.concatenate([train_images, test_images])
labels = np.concatenate([train_labels, test_labels])
```

## Train-Validation-Test Split

- Uses scikit-learn's train_test_split
- Creates training set, validation set, test set

```python
from sklearn.model_selection import train_test_split
train_test_images, val_images, train_test_labels, val_labels =
train_test_split(images, labels, test_size=0.2, random_state=42)
train_images, test_images, train_labels, test_labels =
train_test_split(train_test_images, train_test_labels, test_size=0.2,
random_state=42)

print(train_images.shape, train_labels.shape)
```

```
(44800, 28, 28) (44800,)
```

# Data Preprocessing

## Custom Dataset Class

- Create a PyTorch Dataset class for MNIST
- Implement `__init__`, `__getitem__`, and `__len__` methods
- Add image transformations

```
from torch.utils.data import DataLoader, Dataset
from torchvision.transforms import Compose, ToTensor, Normalize

class MNISTDataset(Dataset):
    def __init__(self, images, labels, image_transform:Compose =
None):
        self.images = images
        self.labels = labels
        self.image_transform = image_transform

    def __getitem__(self, index: int):
        image, label = self.images[index], self.labels[index]
        image = Image.fromarray(image, mode='L')
        if self.image_transform is not None:
            image = self.image_transform(image)

        return image, label

    def __len__(self):
        return len(self.images)
```

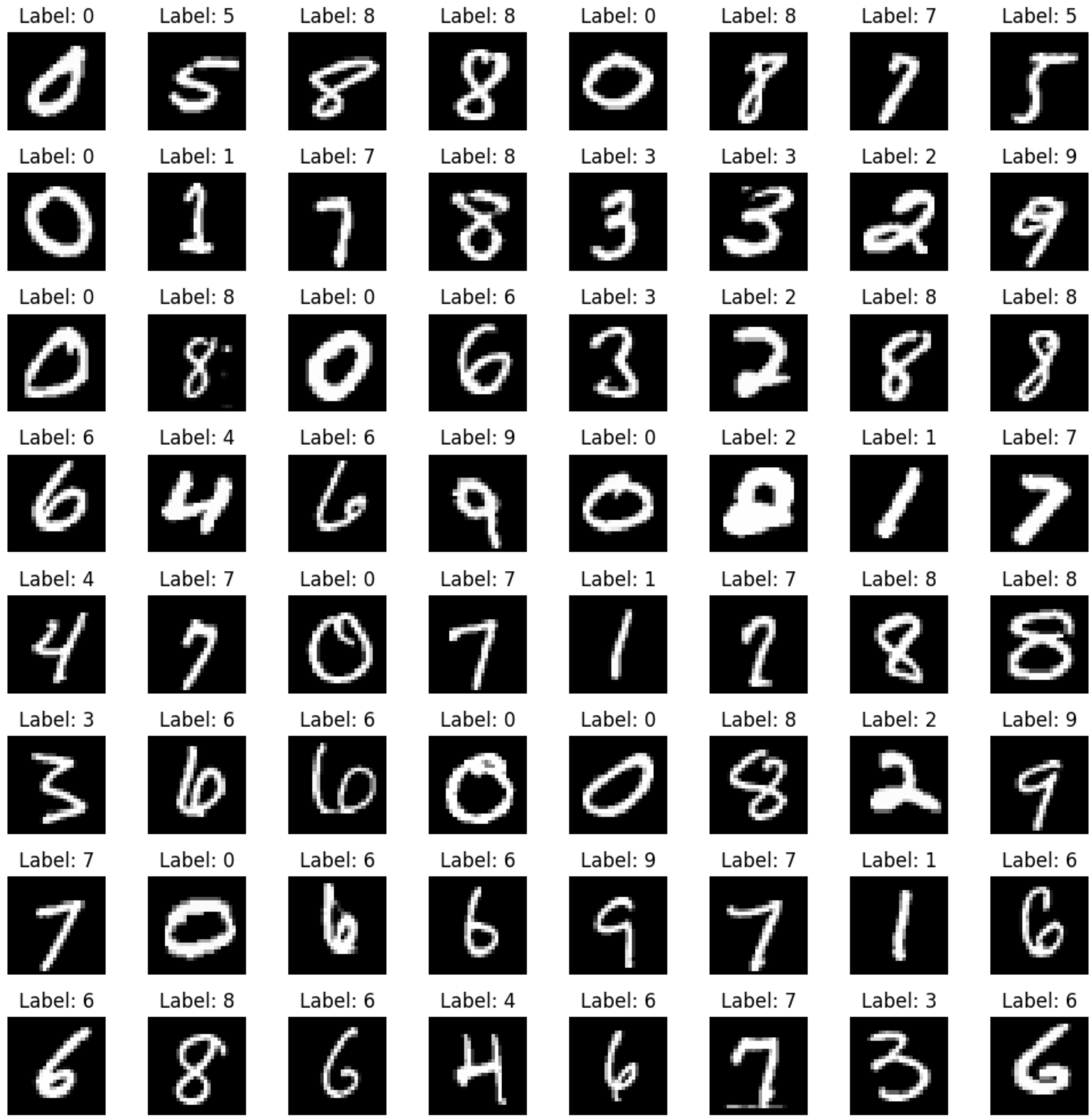
## Image Transformations

- Define transformation pipelines using Compose
- Include:
  - ToTensor conversion
  - Normalization
  - Optional augmentations for training RandomAffine,GaussianBlur,etc.

```
image_transform = Compose([
    ToTensor(),
    Normalize((0.1307,), (0.3081,))
])

from torchvision.transforms import (
    RandomAffine,
    GaussianBlur,
    ElasticTransform
)
```

```python
image_augmentation_transform = Compose([
    ToTensor(),
    RandomAffine(
        degrees=7.5,
        translate=(0.1, 0.1),
        scale=(0.9, 1.1),
        shear=5,
    ),
    GaussianBlur(kernel_size=3, sigma=(0.1, 0.2)),
    ElasticTransform(alpha=37.0, sigma=5.0),
    Normalize((0.1307,), (0.3081,))
])

train_dataset = MNISTDataset(train_images, train_labels,
image_transform)
train_augmented_dataset = MNISTDataset(train_images, train_labels,
image_augmentation_transform)
val_dataset = MNISTDataset(val_images, val_labels, image_transform)
test_dataset = MNISTDataset(test_images, test_labels, image_transform)

train_loader = DataLoader(train_dataset, batch_size=64, num_workers=4,
prefetch_factor=2, shuffle=True)
train_augmented_loader = DataLoader(train_augmented_dataset,
batch_size=64, num_workers=4, prefetch_factor=2, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64, num_workers=4,
prefetch_factor=2)
test_loader = DataLoader(test_dataset, batch_size=64, num_workers=4,
prefetch_factor=2)

def visualize_batch(loader):
    batch_images, batch_labels = next(iter(loader))
    print(batch_images.size(), batch_labels.size())
    batch_size = batch_images.size(0)
    cols = math.ceil(math.sqrt(batch_size))
    rows = math.ceil(batch_size / cols)
    # Create a grid of images
    fig, axes = plt.subplots(rows, cols, figsize=(10, 10))
    for i, (image, label) in enumerate(zip(batch_images,
batch_labels)):
        ax = axes[i//rows, i%cols]
        # Convert tensor back to image for display
        ax.imshow(image.squeeze(), cmap='gray')
        ax.axis('off')
        ax.set_title(f'Label: {label.item()}')
    plt.tight_layout()
    plt.show()

visualize_batch(train_loader)
```

```
torch.Size([64, 1, 28, 28]) torch.Size([64])
```




# Training

## Training Configuration

- Create a dataclass for experiment configuration
- Include hyperparameters:
  - Learning rates
  - Batch sizes
  - Dropout rates

- Optimization algorithms
- Scheduling strategies

```
from dataclasses import dataclass, field
from typing import List

@dataclass
class TrainingConfig:
    # Model parameters
    input_size: int = 28 * 28
    output_size: int = 10
    hidden_sizes: List[int] = field(default_factory=lambda: [])  # Empty by default for softmax
    dropout_rate: float = 0.0

    # Training parameters
    batch_size: int = 128
    num_epochs: int = 50
    learning_rate: float = 0.01
    weight_decay: float = 0.0

    # Optimizer parameters
    optimizer: str = "sgd"  # "sgd", "adam"

    # LR scheduler parameters
    scheduler_type: str = "none"  # "none", "cosine", "warmup_cosine"
    warmup_epochs: int = 5
    min_lr: float = 1e-5

    # Early stopping parameters
    early_stopping: bool = True
    patience: int = 10
    loss_delta: float = 0.02

    # DataLoader parameters
    num_workers: int = 4
    prefetch_factor: int = 2
    use_augmentation: bool = False

    # Device
    device: str = "cuda" if torch.cuda.is_available() else "cpu"

    # Experiment name
    experiment_name: str = "mnist_exp"
    project_name: str = "mnist-exp"
```

# Model Architectures

- Implement two model types:
  - Softmax Regression

    - Simple linear model
    - No hidden layers
  - Deep Neural Network based on Deep Big Simple Neural Nets Excel on Hand-written Digit Recognition (Dan Claudiu Ciresan, et al.)
    - Multiple configurable hidden layers
    - Batch normalization
    - Dropout
    - Kaiming He initialization
    - ReLU activation

```
class SoftmaxRegression(torch.nn.Module):
    def __init__(self, input_size, output_size, dropout_rate=0.0):
        super(SoftmaxRegression, self).__init__()
        self.input_size = input_size
        self.output_size = output_size
        self.dropout = torch.nn.Dropout(dropout_rate)
        self.linear = torch.nn.Linear(input_size, output_size)

    def forward(self, x):
        x = x.view(-1, self.input_size) # remove the channel
dimension, flatten the image
        x = self.dropout(x)
        return self.linear(x)

class DeepBigSimpleNet(torch.nn.Module):
    def __init__(self, input_size=784, hidden_sizes=[2500, 2000,
1500], num_classes=10, dropout_rate=0.5):
        super(DeepBigSimpleNet, self).__init__()
        self.input_size = input_size
        self.hidden_sizes = hidden_sizes
        self.num_classes = num_classes

        # Create layers list
        layers = []

        # Input layer
        layers.append(torch.nn.Linear(input_size, hidden_sizes[0]))
        layers.append(torch.nn.BatchNorm1d(hidden_sizes[0]))
        layers.append(torch.nn.ReLU())
        layers.append(torch.nn.Dropout(dropout_rate))

        # Hidden layers with decreasing sizes
        for i in range(len(hidden_sizes) - 1):
            layers.append(torch.nn.Linear(hidden_sizes[i],
hidden_sizes[i+1]))
            layers.append(torch.nn.BatchNorm1d(hidden_sizes[i+1]))
            layers.append(torch.nn.ReLU())
            layers.append(torch.nn.Dropout(dropout_rate))
```

```
        # Output layer
        layers.append(torch.nn.Linear(hidden_sizes[-1], num_classes))

        self.model = torch.nn.Sequential(*layers)
        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, torch.nn.Linear):
                torch.nn.init.kaiming_normal_(m.weight, mode='fan_in',
nonlinearity='relu')
                if m.bias is not None:
                    torch.nn.init.constant_(m.bias, 0)
            elif isinstance(m, torch.nn.BatchNorm1d):
                torch.nn.init.constant_(m.weight, 1)
                torch.nn.init.constant_(m.bias, 0)

    def forward(self, x):
        x = x.view(-1, self.input_size)  # Flatten the input
        return self.model(x)

from torch.nn import CrossEntropyLoss
import torch.nn.functional as F
import torch.optim as optim
from torch.optim.lr_scheduler import LinearLR, CosineAnnealingLR,
SequentialLR
from tqdm import tqdm
from tabulate import tabulate
import wandb
from time import time
```

# Logging and Visualization

1. Wandb integration for experiment tracking
    - Log and visualize:
        - Hyperparameters
        - Metrics
        - Visualizations
        - Model performance
        - Misclassifications
        - Confusion matrix
        - Gradient and weights histograms
2. Create experiment tracker
    - Log and visualize:
        - Training curves
        - Loss progression
        - Accuracy metrics

- Confusion matrices

```
wandb.login(key="d8ff0fac98c036a4ac0587814c4fd1a2e60f2512")
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: ahmedayman4a77. Use `wandb login --
relogin` to force relogin
wandb: WARNING If you're specifying your api key in code, ensure this
code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment
variable, or running `wandb login` from the command line.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
```

```
True
```

```
import wandb
import torch
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

class WandBLogger:
    def __init__(self, config):
        self.run = wandb.init(
            project=config.project_name,
            name=config.experiment_name,
            config=config.__dict__,
        )

    def log_model_graph(self, model, loss):
        wandb.watch(model, criterion=loss, log="all", log_freq=100)

    def log_batch_predictions(self, images, labels, predictions,
step):
        # Log sample predictions
        fig, ax = plt.subplots(4, 4, figsize=(10, 10))
        for idx in range(16):
            i, j = idx//4, idx%4
            ax[i,j].imshow(images[idx].squeeze(), cmap='gray')
            ax[i,j].set_title(f'Pred: {predictions[idx]}\nTrue:
{labels[idx]}')
            ax[i,j].axis('off')
        wandb.log({"predictions": wandb.Image(fig)}, commit=False)
        plt.close()

    def log_confusion_matrix(self, true_labels, predictions, step):
        cm = confusion_matrix(true_labels, predictions)
        fig = plt.figure(figsize=(10, 10))
        sns.heatmap(cm, annot=True, fmt='d')
        wandb.log({"confusion_matrix": wandb.Image(fig)},
```

```
commit=False)
        plt.close()

    def log_grad_flow(self, named_parameters):
        ave_grads = []
        layers = []
        for n, p in named_parameters:
            if p.requires_grad and p.grad is not None:
                layers.append(n)
                ave_grads.append(p.grad.abs().mean().item())
        fig = plt.figure(figsize=(10, 5))
        plt.plot(ave_grads, alpha=0.3, color="b")
        plt.title("Gradient Flow")
        wandb.log({"grad_flow": wandb.Image(fig)}, commit=False)
        plt.close()

    def log_misclassified(self, images, labels, predictions, step):
        mask = predictions != labels
        if not mask.any():
            return

        misclassified_images = images[mask][:16]
        misclassified_labels = labels[mask][:16]
        misclassified_preds = predictions[mask][:16]

        fig, ax = plt.subplots(4, 4, figsize=(10, 10))
        for idx in range(min(16, len(misclassified_images))):
            i, j = idx//4, idx%4
            ax[i,j].imshow(misclassified_images[idx].squeeze(),
cmap='gray')
            ax[i,j].set_title(f'Pred: {misclassified_preds[idx]}\
nTrue: {misclassified_labels[idx]}')
            ax[i,j].axis('off')
        wandb.log({"misclassified": wandb.Image(fig)}, commit=False)
        plt.close()

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Dict, List

class ExperimentTracker:
    def __init__(self):
        self.experiment_metrics = {}  # Store per-epoch metrics
        self.experiment_summaries = []  # Store final results

    def log_epoch(self, experiment_name: str, epoch_metrics: Dict):
        if experiment_name not in self.experiment_metrics:
            self.experiment_metrics[experiment_name] = []
        self.experiment_metrics[experiment_name].append(epoch_metrics)
```

```
    def log_summary(self, experiment_name: str, summary_metrics: Dict):
        summary_metrics['experiment_name'] = experiment_name
        self.experiment_summaries.append(summary_metrics)

    def get_experiment_df(self, experiment_name: str) -> pd.DataFrame:
        return pd.DataFrame(self.experiment_metrics[experiment_name])

    def get_all_summaries(self) -> pd.DataFrame:
        return pd.DataFrame(self.experiment_summaries)

    def plot_training_curves(self, metric: str = 'loss'):
        plt.figure(figsize=(12, 6))
        for exp_name, metrics in self.experiment_metrics.items():
            df = pd.DataFrame(metrics)
            plt.plot(df['epoch'], df[f'train_{metric}'], label=f'{exp_name} (train)')
            plt.plot(df['epoch'], df[f'val_{metric}'], label=f'{exp_name} (val)')
        plt.xlabel('Epoch')
        plt.ylabel(metric.capitalize())
        plt.title(f'Training and Validation {metric.capitalize()} Curves')
        plt.legend()
        plt.grid(True)
        return plt.gcf()

    def plot_confusion_matrix(self, experiment_name: str):
        summary_df = self.get_all_summaries()
        cm = summary_df[summary_df['experiment_name'] == experiment_name]['confusion_matrix'].values[0]
        plt.figure(figsize=(10, 10))
        sns.heatmap(cm, annot=True, fmt='d')
        plt.title(f'Confusion Matrix for {experiment_name}')
        return plt.gcf()

    def plot_experiment_comparison(self, metric: str):
        summary_df = self.get_all_summaries()
        plt.figure(figsize=(10, 6))
        sns.barplot(data=summary_df, x='experiment_name', y=metric)
        plt.xticks(rotation=45)
        plt.title(f'Comparison of {metric} across experiments')
        return plt.gcf()
```

## Early Stopping

- Monitor validation loss
- Stop training if loss does not improve after a certain number of epochs

```
class EarlyStopper:
    def __init__(self, patience=1, min_delta=0):
        self.patience = patience
        self.min_delta = min_delta
        self.counter = 0
        self.min_validation_loss = float('inf')

    def __call__(self, validation_loss):
        if validation_loss < self.min_validation_loss:
            self.min_validation_loss = validation_loss
            self.counter = 0
        elif validation_loss > (self.min_validation_loss + 
self.min_delta):
            self.counter += 1
            if self.counter >= self.patience:
                return True
        return False
```

## Evaluator

Track:

- Accuracy
- Inference time
- Confusion matrix

```
from sklearn.metrics import precision_recall_fscore_support, 
confusion_matrix

class Evaluator:
    def __init__(self, model, test_loader, device):
        self.model = model
        self.test_loader = test_loader
        self.device = device

    def __call__(self):
        self.model.eval()
        all_preds = []
        all_labels = []
        inference_times = []

        with torch.no_grad():
            for images, labels in self.test_loader:
                images = images.to(self.device)
                start_time = time()
                outputs = self.model(images)
                inference_times.append(time() - start_time)

                preds = outputs.argmax(dim=1)
```

```
                all_preds.extend(preds.cpu().numpy())
                all_labels.extend(labels.numpy())

        metrics = self.compute_metrics(all_labels, all_preds,
inference_times)
        return metrics

    def compute_metrics(self, labels, preds, times):

        return {
            'accuracy': (np.array(labels) == np.array(preds)).mean(),
            'confusion_matrix': confusion_matrix(labels, preds),
            'avg_inference_time': np.mean(times),
            'std_inference_time': np.std(times)
        }
```

## Training Infrastructure

Develop training manager class that includes:

- Optimizer selection
- Learning rate scheduling
- Early stopping
- Logging mechanisms
- Training loop
- Validation loop
- Wandb integration

```
class TrainingManager:
    def __init__(self, config: TrainingConfig, tracker:
ExperimentTracker):
        self.config = config
        self.logger = WandBLogger(config)
        self.tracker = tracker
        if self.config.early_stopping:
            self.early_stopper =
EarlyStopper(patience=self.config.patience,
min_delta=self.config.loss_delta)
        else:
            self.early_stopper = None

    def get_scheduler(self, optimizer):
        if self.config.scheduler_type == "none":
            return None
        elif self.config.scheduler_type == "cosine":
            return torch.optim.lr_scheduler.CosineAnnealingLR(
                optimizer,
                T_max=self.config.num_epochs,
```

```
                eta_min=self.config.min_lr
            )
        elif self.config.scheduler_type == "warmup_cosine":
            warmup_scheduler = LinearLR(optimizer,
total_iters=self.config.warmup_epochs)
            main_scheduler = CosineAnnealingLR(optimizer,
T_max=self.config.num_epochs - self.config.warmup_epochs,
eta_min=self.config.min_lr)
            scheduler = SequentialLR(optimizer,
schedulers=[warmup_scheduler, main_scheduler],
milestones=[self.config.warmup_epochs])
            return scheduler

    def train(self, model, train_loader, val_loader, test_loader):
        model = model.to(self.config.device)
        if self.config.optimizer == "sgd":
            optimizer = torch.optim.SGD(
                model.parameters(),
                lr=self.config.learning_rate,
                weight_decay=self.config.weight_decay
            )
        elif self.config.optimizer == "adam":
            optimizer = torch.optim.Adam(
                model.parameters(),
                lr=self.config.learning_rate,
                weight_decay=self.config.weight_decay
            )
        else:
            raise ValueError("Invalid optimizer type")

        scheduler = self.get_scheduler(optimizer)
        criterion = CrossEntropyLoss()
        self.logger.log_model_graph(model, criterion)


        start_time = time()
        for epoch in range(self.config.num_epochs):
            epoch_start = time()
            # Training
            model.train()
            train_loss = 0
            correct = 0
            total = 0

            for batch_idx, (inputs, targets) in
enumerate(train_loader):
                inputs, targets = inputs.to(self.config.device),
targets.to(self.config.device)
```

```
                optimizer.zero_grad()
                outputs = model(inputs)
                loss = criterion(outputs, targets)
                loss.backward()
                optimizer.step()

                train_loss += loss.item()
                _, predicted = outputs.max(1)
                total += targets.size(0)
                correct += predicted.eq(targets).sum().item()

            train_acc = 100. * correct / total
            train_loss = train_loss / len(train_loader)

            # Validation
            val_loss, val_acc = self.validate(model, val_loader,
criterion, epoch)

            # Update scheduler
            if scheduler is not None:
                scheduler.step()

            epoch_time = time() - epoch_start

            epoch_metrics = {
                "epoch": epoch,
                "train_loss": train_loss,
                "train_acc": train_acc,
                "val_loss": val_loss,
                "val_acc": val_acc,
                "learning_rate": optimizer.param_groups[0]['lr'],
                'epoch_time': epoch_time,
            }
            # Log metrics
            self.tracker.log_epoch(self.config.experiment_name,
epoch_metrics)
            wandb.log(epoch_metrics)


            if self.early_stopper is not None and
self.early_stopper(val_loss):
                print(f"Early stopping triggered at epoch {epoch}")
                break

        total_time = time() - start_time

        # Final evaluation
        evaluate = Evaluator(model, test_loader, self.config.device)
        test_metrics = evaluate()
```

```
        test_metrics['total_time'] = total_time
        self.tracker.log_summary(self.config.experiment_name,
test_metrics)


        fig = plt.figure(figsize=(10, 10))
        sns.heatmap(test_metrics['confusion_matrix'], annot=True,
fmt='d')
        # Log final metrics
        wandb.log({
            'total_training_time': total_time,
            'test_accuracy': test_metrics['accuracy'],
            'avg_inference_time': test_metrics['avg_inference_time'],
            'test_confusion_matrix': wandb.Image(fig)
        })

        wandb.finish()
        return test_metrics

    def validate(self, model, val_loader, criterion, epoch):
        model.eval()
        val_loss = 0
        correct = 0
        total = 0
        val_images, val_labels = [], []
        val_preds = []

        with torch.no_grad():
            for inputs, targets in val_loader:
                inputs, targets = inputs.to(self.config.device),
targets.to(self.config.device)
                outputs = model(inputs)
                loss = criterion(outputs, targets)

                val_loss += loss.item()
                predictions = outputs.argmax(dim=1)
                total += targets.size(0)
                correct += predictions.eq(targets).sum().item()
                val_images.extend(inputs.cpu())
                val_labels.extend(targets.cpu())
                val_preds.extend(predictions.cpu())

        val_images = torch.stack(val_images)
        val_labels = torch.tensor(val_labels)
        val_preds = torch.tensor(val_preds)

        # Log validation artifacts
        self.logger.log_batch_predictions(
            val_images[:16],
```

```
            val_labels[:16],
            val_preds[:16],
            epoch
        )
        self.logger.log_confusion_matrix(
            val_labels,
            val_preds,
            epoch
        )
        self.logger.log_misclassified(
            val_images,
            val_labels,
            val_preds,
            epoch
        )
        self.logger.log_grad_flow(model.named_parameters())

        return val_loss / len(val_loader), 100. * correct / total
```

## Experiment Runner

Create function to run multiple experiments with different configurations of:

- Model architectures
- Hyperparameters
- Optimization strategies
- Scheduling strategies
- Regularization

```
experiments = [
    TrainingConfig(learning_rate=1.0,
experiment_name="softmax_lr_1.0", early_stopping=False),
    TrainingConfig(learning_rate=0.01,
experiment_name="softmax_lr_0.01"),
    TrainingConfig(learning_rate=0.00001,
experiment_name="softmax_lr_0.00001"),
    TrainingConfig(
        learning_rate=0.01,
        scheduler_type="warmup_cosine",
        warmup_epochs=5,
        experiment_name="softmax_warmup_cosine"
    ),
    TrainingConfig(batch_size=1, experiment_name="softmax_batch_1"),
    TrainingConfig(batch_size=32, experiment_name="softmax_batch_32"),
    TrainingConfig(batch_size=256,
experiment_name="softmax_batch_256"),
    TrainingConfig(
        learning_rate=0.01,
        weight_decay=0.01,
```

```
        dropout_rate=0.1,
        batch_size=256,
        experiment_name="softmax_with_regularization"
    ),
     # DeepBigSimpleNet experiments
    TrainingConfig(
        input_size=28 * 28,
        output_size=10,
        hidden_sizes=[40, 15],
        dropout_rate=0.1,
        learning_rate=0.01,
        weight_decay=1e-4,
        batch_size=128,
        scheduler_type="warmup_cosine",
        experiment_name="deep_net_nano"
    ),
    TrainingConfig(
        input_size=28 * 28,
        output_size=10,
        hidden_sizes=[1500, 1000, 500],
        dropout_rate=0.1,
        learning_rate=0.01,
        weight_decay=1e-4,
        batch_size=128,
        scheduler_type="warmup_cosine",
        optimizer="adam",
        use_augmentation=True,
        experiment_name="deep_net_medium"
    ),
    TrainingConfig(
        input_size=28 * 28,
        output_size=10,
        hidden_sizes=[2500, 2000, 1500, 1000, 500],
        dropout_rate=0.1,
        learning_rate=0.01,
        weight_decay=1e-4,
        batch_size=128,
        scheduler_type="warmup_cosine",
        optimizer="adam",
        use_augmentation=True,
        experiment_name="deep_net_large"
    )
]

def run_experiments(experiments, tracker=None):
    if tracker is None:
        tracker = ExperimentTracker()

    for config in experiments:
        print(f"Running experiment: {config.experiment_name}")
```

```
        chose_train_dataset = train_augmented_dataset if
config.use_augmentation else train_dataset
        # Create data loaders
        train_loader = DataLoader(
            chose_train_dataset,
            batch_size=config.batch_size,
            num_workers=config.num_workers,
            prefetch_factor=config.prefetch_factor,
            shuffle=True
        )
        val_loader = DataLoader(
            val_dataset,
            batch_size=config.batch_size,
            num_workers=config.num_workers,
            prefetch_factor=config.prefetch_factor
        )
        test_loader = DataLoader(
            test_dataset,
            batch_size=config.batch_size,
            num_workers=config.num_workers,
            prefetch_factor=config.prefetch_factor
        )

        # Create appropriate model based on config
        if len(config.hidden_sizes) > 0:
            # Use DeepBigSimpleNet if hidden_sizes are specified
            model = DeepBigSimpleNet(
                input_size=config.input_size,
                hidden_sizes=config.hidden_sizes,
                num_classes=config.output_size,
                dropout_rate=config.dropout_rate
            )
        else:
            # Use SoftmaxRegression for basic experiments
            model = SoftmaxRegression(
                input_size=config.input_size,
                output_size=config.output_size,
                dropout_rate=config.dropout_rate
            )

        trainer = TrainingManager(config, tracker)
        trainer.train(model, train_loader, val_loader, test_loader)
```

## Running Experiments

All experiments are run using the experiment runner function. The results are logged and visualized using Wandb. You can view the results here.

```
tracker = ExperimentTracker()
run_experiments(experiments, tracker)
```

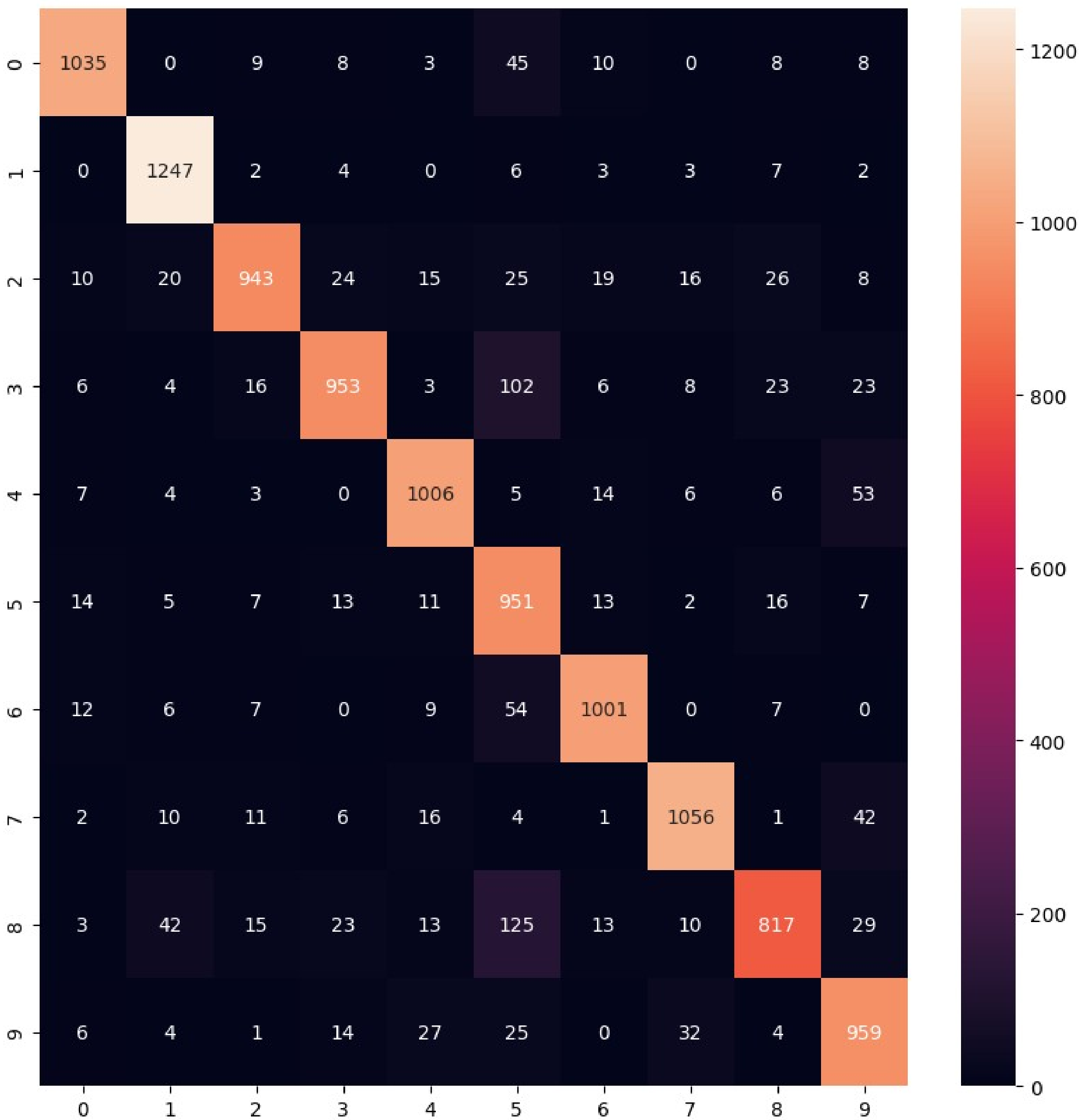
```
Running experiment: softmax_lr_1.0

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: softmax_lr_0.01

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: softmax_lr_0.00001

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: softmax_warmup_cosine

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/torch/optim/
lr_scheduler.py:232: UserWarning: The epoch parameter in
`scheduler.step()` was not necessary and is being deprecated where
possible. Please use `scheduler.step()` to step the scheduler. During
the deprecation, if epoch is different from None, the closed form is
used instead of the new chainable form, where available. Please open
an issue if you are unable to replicate your use case:
https://github.com/pytorch/pytorch/issues/new/choose.
  warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: softmax_batch_1

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Early stopping triggered at epoch 10

<IPython.core.display.HTML object>
```

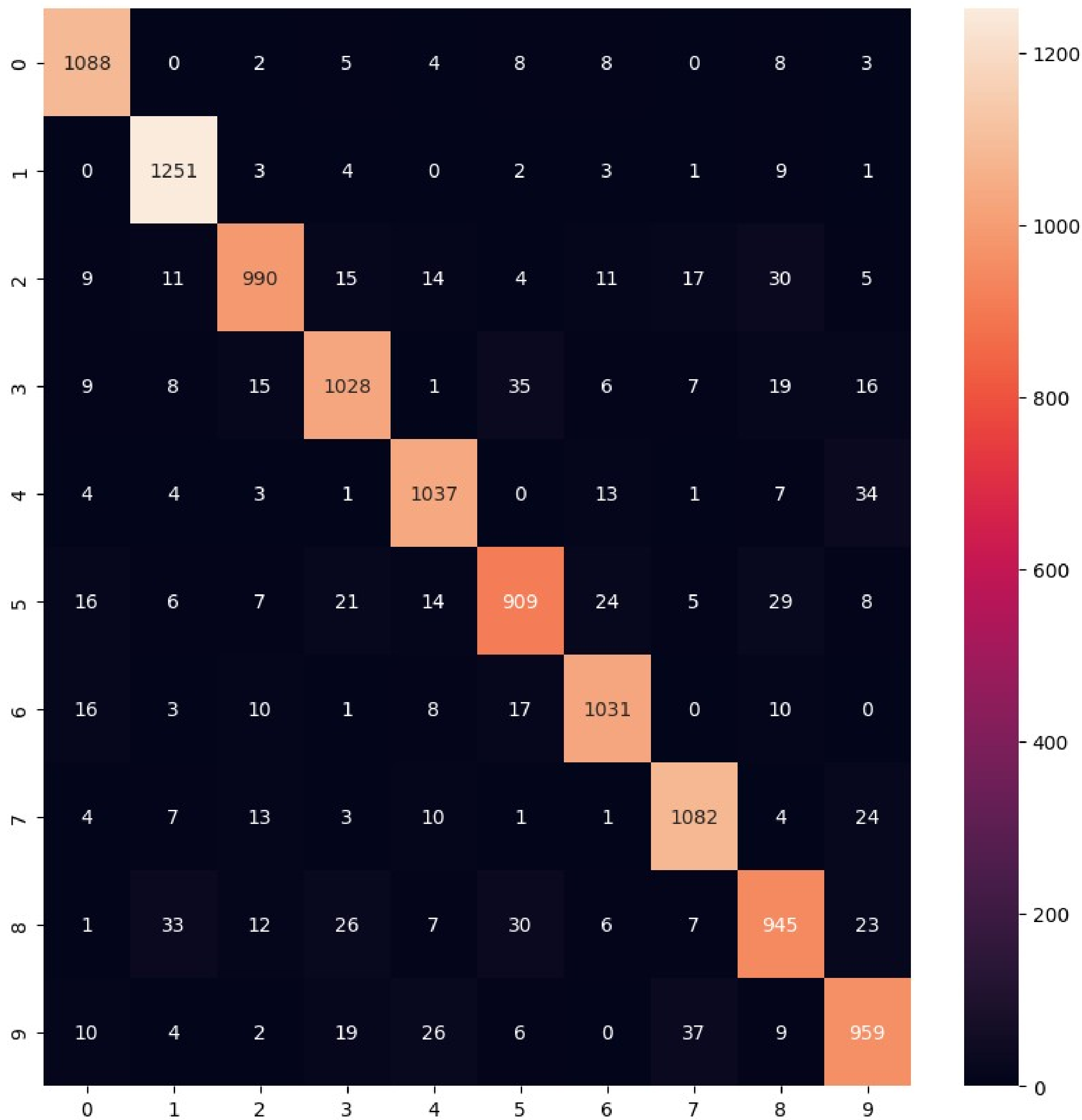
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Running experiment: softmax_batch_32
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Running experiment: softmax_batch_256
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
Running experiment: softmax_with_regularization
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```

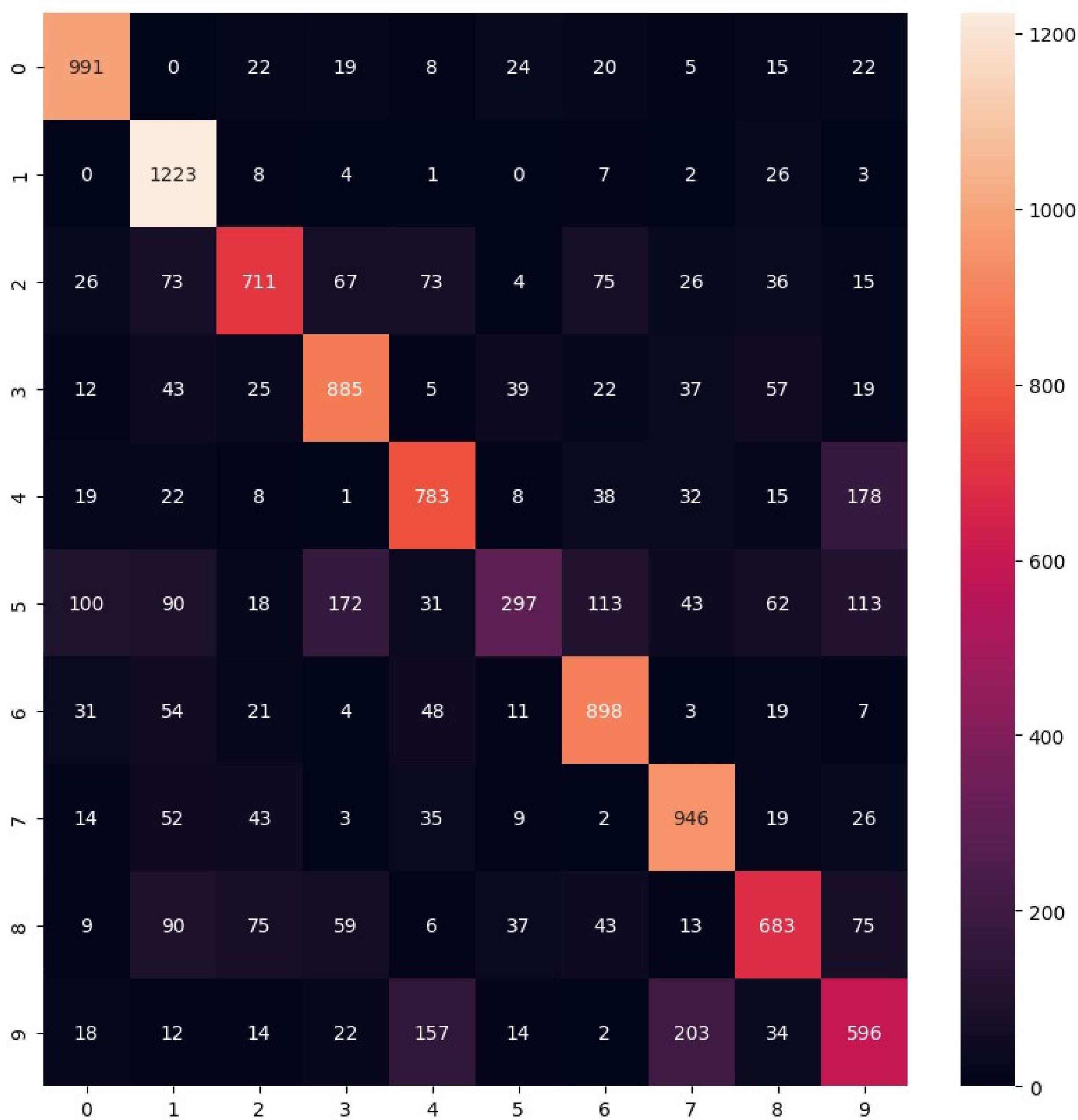
```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: deep_net_nano

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/torch/optim/
lr_scheduler.py:232: UserWarning: The epoch parameter in
`scheduler.step()` was not necessary and is being deprecated where
possible. Please use `scheduler.step()` to step the scheduler. During
the deprecation, if epoch is different from None, the closed form is
used instead of the new chainable form, where available. Please open
an issue if you are unable to replicate your use case:
https://github.com/pytorch/pytorch/issues/new/choose.
  warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: deep_net_medium

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/torch/optim/
lr_scheduler.py:232: UserWarning: The epoch parameter in
```

```
`scheduler.step()` was not necessary and is being deprecated where
possible. Please use `scheduler.step()` to step the scheduler. During
the deprecation, if epoch is different from None, the closed form is
used instead of the new chainable form, where available. Please open
an issue if you are unable to replicate your use case:
https://github.com/pytorch/pytorch/issues/new/choose.
  warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Running experiment: deep_net_large

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

/usr/local/lib/python3.10/dist-packages/torch/optim/
lr_scheduler.py:232: UserWarning: The epoch parameter in
`scheduler.step()` was not necessary and is being deprecated where
possible. Please use `scheduler.step()` to step the scheduler. During
the deprecation, if epoch is different from None, the closed form is
used instead of the new chainable form, where available. Please open
an issue if you are unable to replicate your use case:
https://github.com/pytorch/pytorch/issues/new/choose.
  warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

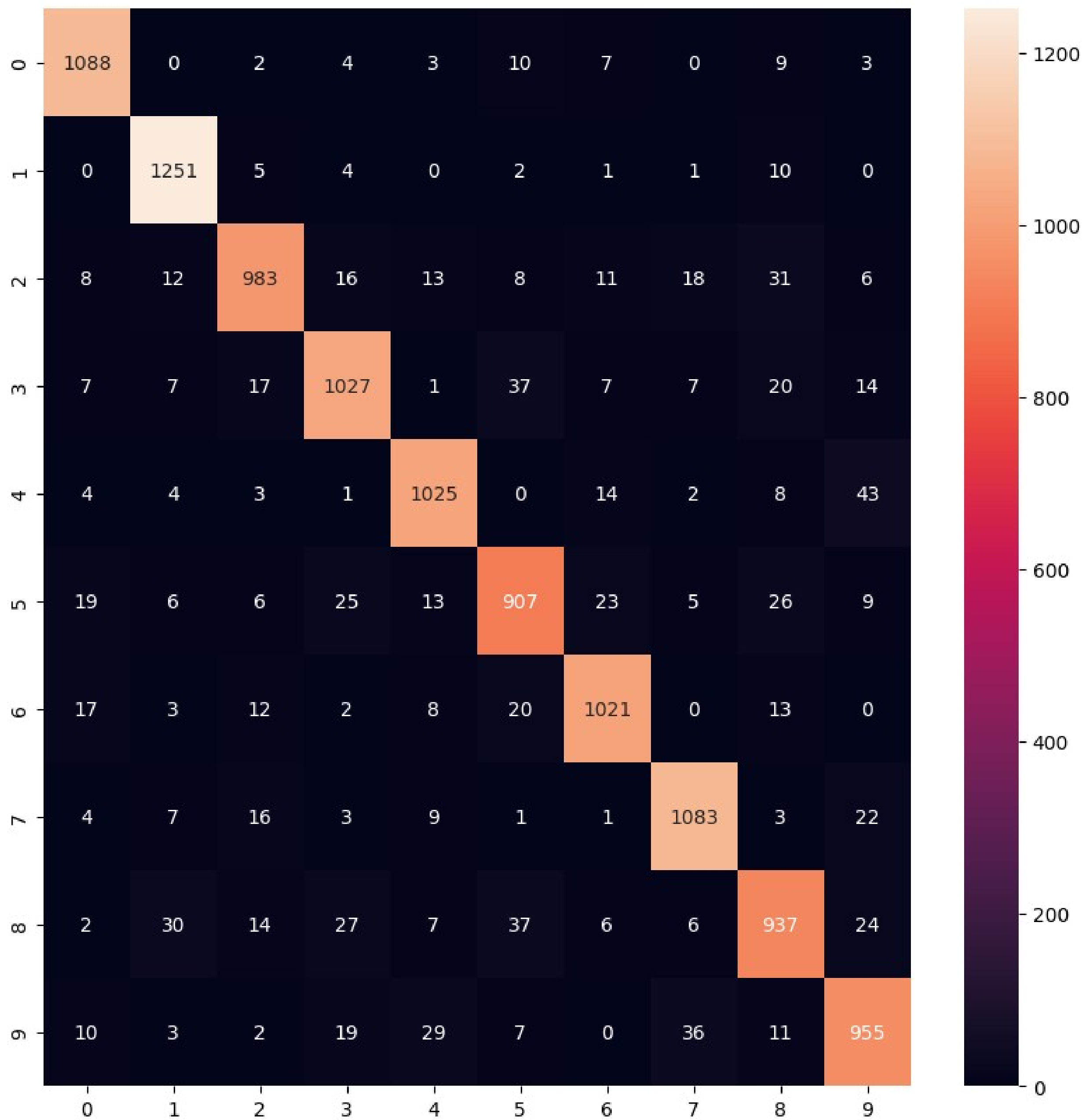


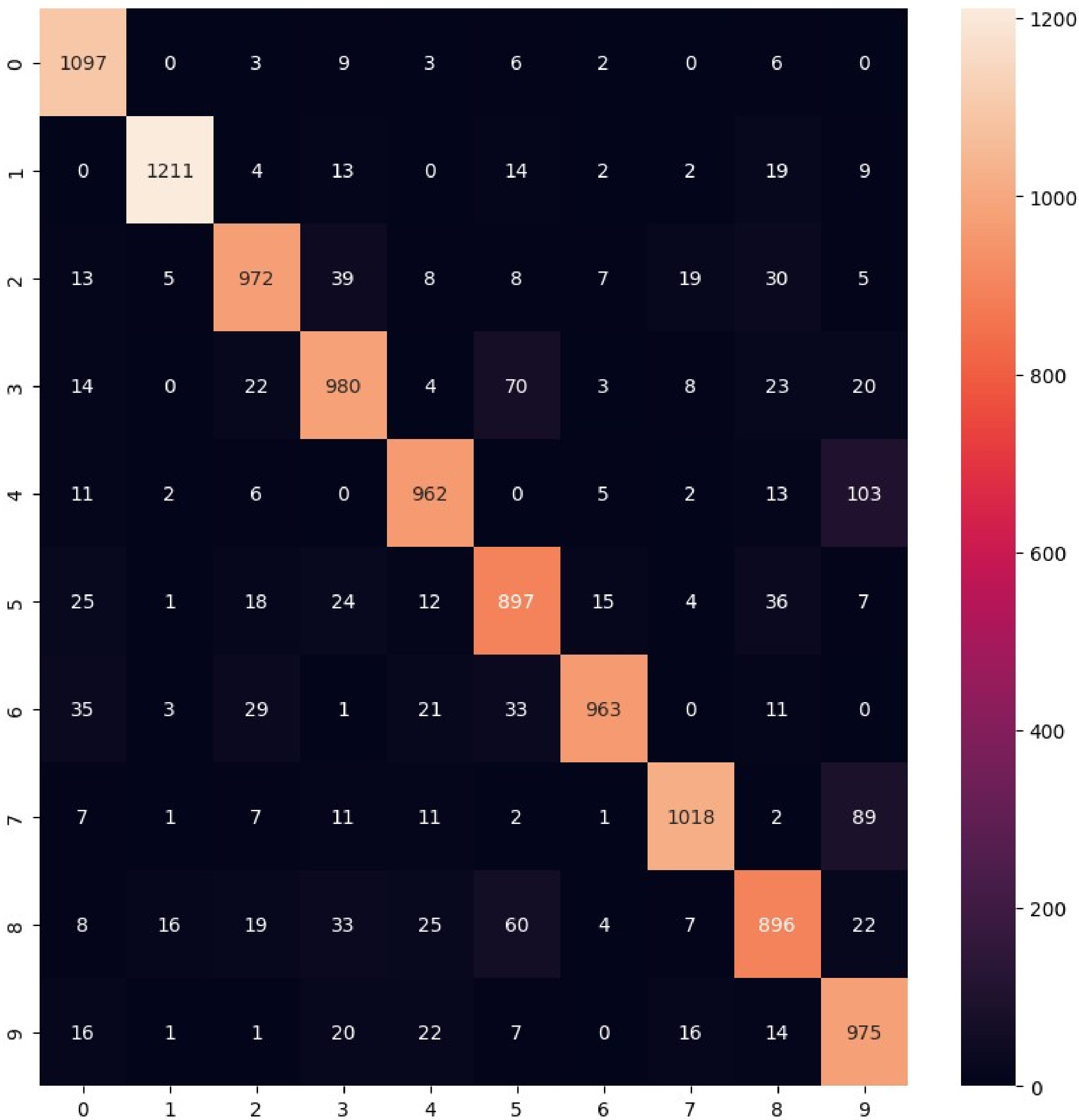
1035 0 9 8 3 45 10 0 8 8
0 1247 2 4 0 6 3 3 7 2
10 20 943 24 15 25 19 16 26 8
6 4 16 953 3 102 6 8 23 23
7 4 3 0 1006 5 14 6 6 53
14 5 7 13 11 951 13 2 16 7
12 6 7 0 9 54 1001 0 7 0
2 10 11 6 16 4 1 1056 1 42
3 42 15 23 13 125 13 10 817 29
6 4 1 14 27 25 0 32 4 959
0 1 2 3 4 5 6 7 8 9
0 200 400 600 800 1000 1200

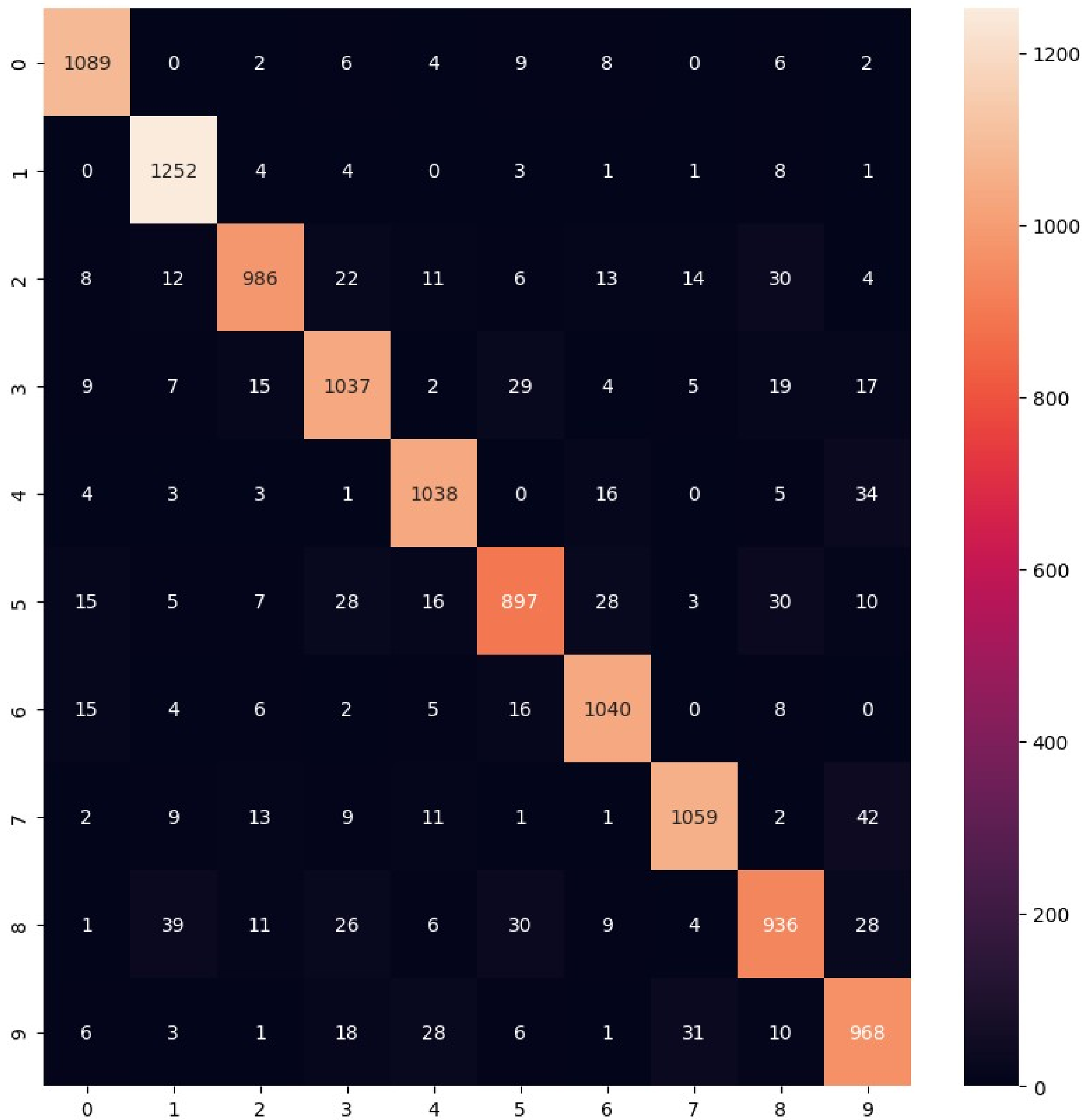

1088 0 2 5 4 8 8 0 8 3
0 1251 3 4 0 2 3 1 9 1
9 11 990 15 14 4 11 17 30 5
9 8 15 1028 1 35 6 7 19 16
4 4 3 1 1037 0 13 1 7 34
16 6 7 21 14 909 24 5 29 8
16 3 10 1 8 17 1031 0 10 0
4 7 13 3 10 1 1 1082 4 24
1 33 12 26 7 30 6 7 945 23
10 4 2 19 26 6 0 37 9 959
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0



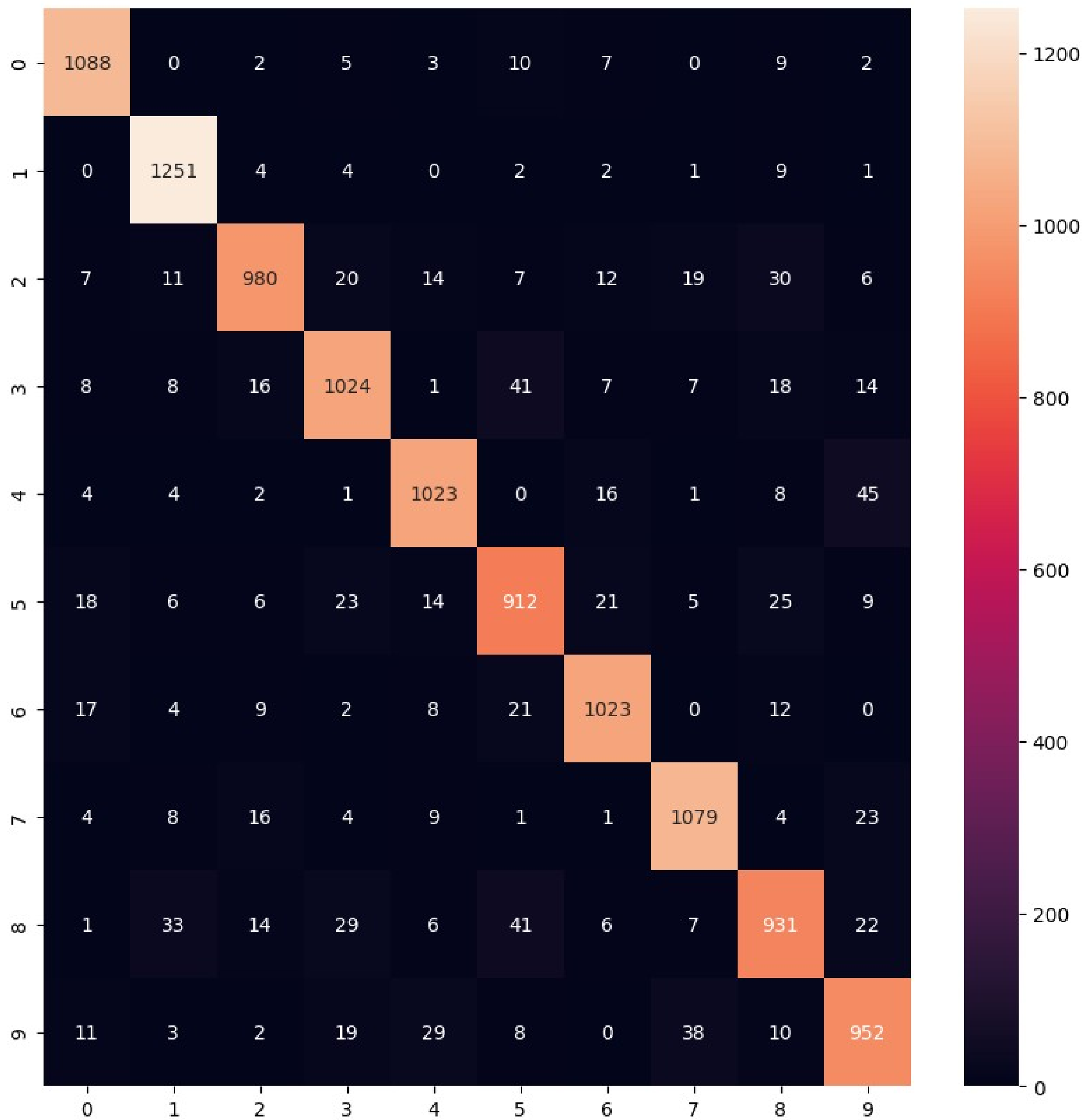
0
1
2
3
4
5
6
7
8
9
1200
1000
800
600
400
200
0
991 0 22 19 8 24 20 5 15 22
0 1223 8 4 1 0 7 2 26 3
26 73 711 67 73 4 75 26 36 15
12 43 25 885 5 39 22 37 57 19
19 22 8 1 783 8 38 32 15 178
100 90 18 172 31 297 113 43 62 113
31 54 21 4 48 11 898 3 19 7
14 52 43 3 35 9 2 946 19 26
9 90 75 59 6 37 43 13 683 75
18 12 14 22 157 14 2 203 34 596


1088 0 2 4 3 10 7 0 9 3
0 1251 5 4 0 2 1 1 10 0
8 12 983 16 13 8 11 18 31 6
7 7 17 1027 1 37 7 7 20 14
4 4 3 1 1025 0 14 2 8 43
19 6 6 25 13 907 23 5 26 9
17 3 12 2 8 20 1021 0 13 0
4 7 16 3 9 1 1 1083 3 22
2 30 14 27 7 37 6 6 937 24
10 3 2 19 29 7 0 36 11 955
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0

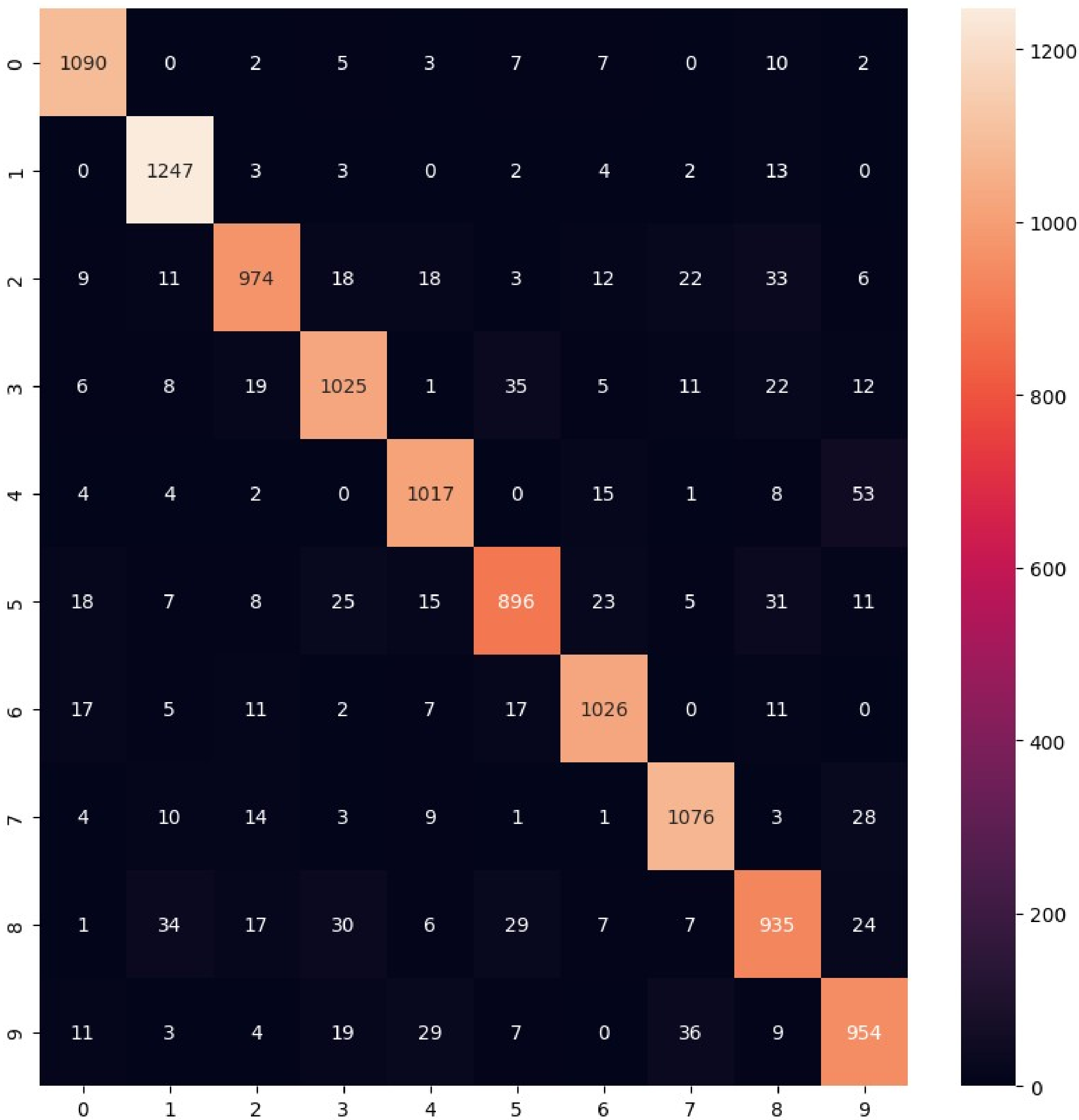

0 1 2 3 4 5 6 7 8 9
0: 1097 0 3 9 3 6 2 0 6 0
1: 0 1211 4 13 0 14 2 2 19 9
2: 13 5 972 39 8 8 7 19 30 5
3: 14 0 22 980 4 70 3 8 23 20
4: 11 2 6 0 962 0 5 2 13 103
5: 25 1 18 24 12 897 15 4 36 7
6: 35 3 29 1 21 33 963 0 11 0
7: 7 1 7 11 11 2 1 1018 2 89
8: 8 16 19 33 25 60 4 7 896 22
9: 16 1 1 20 22 7 0 16 14 975
1200
1000
800
600
400
200
0

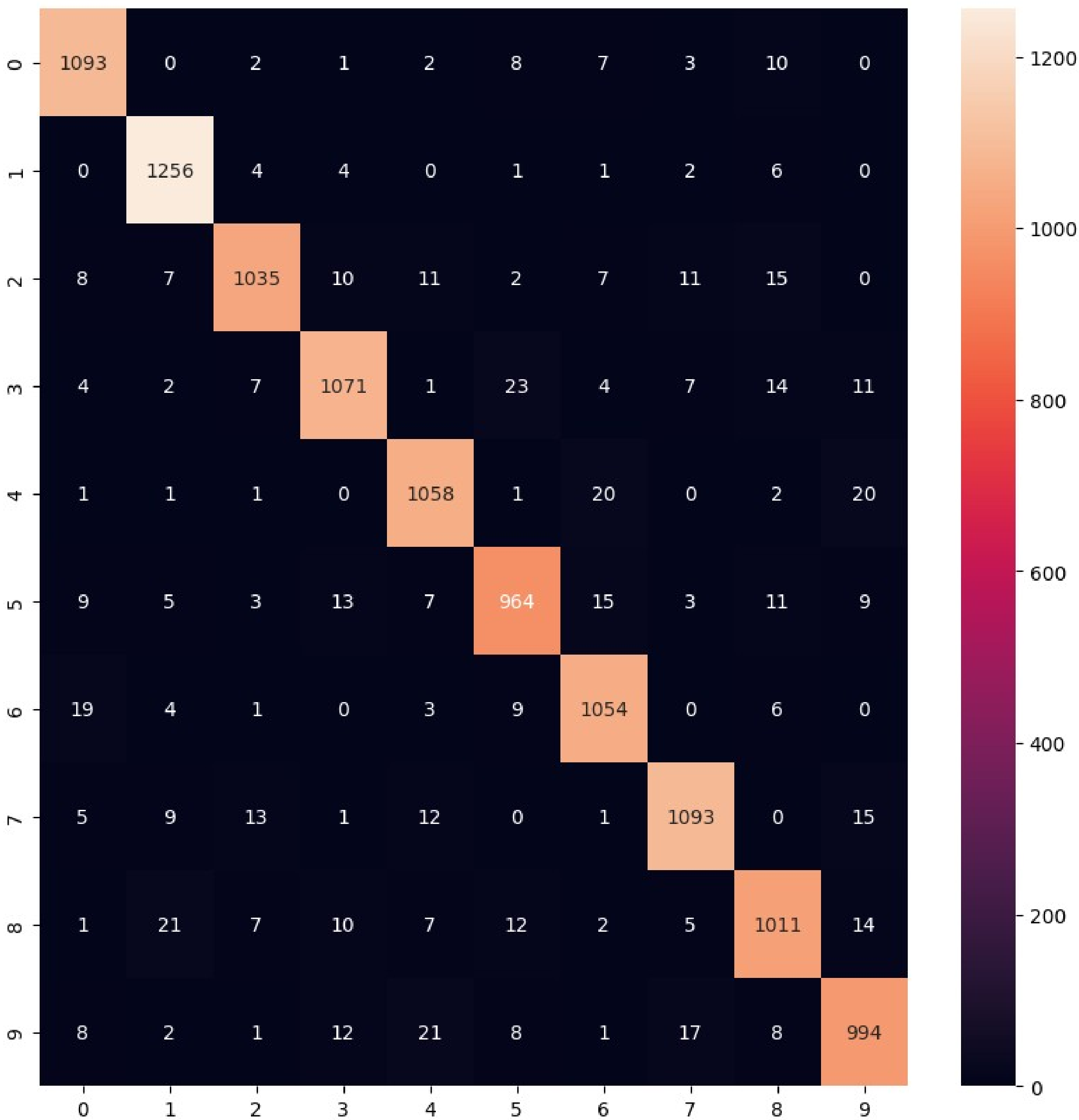


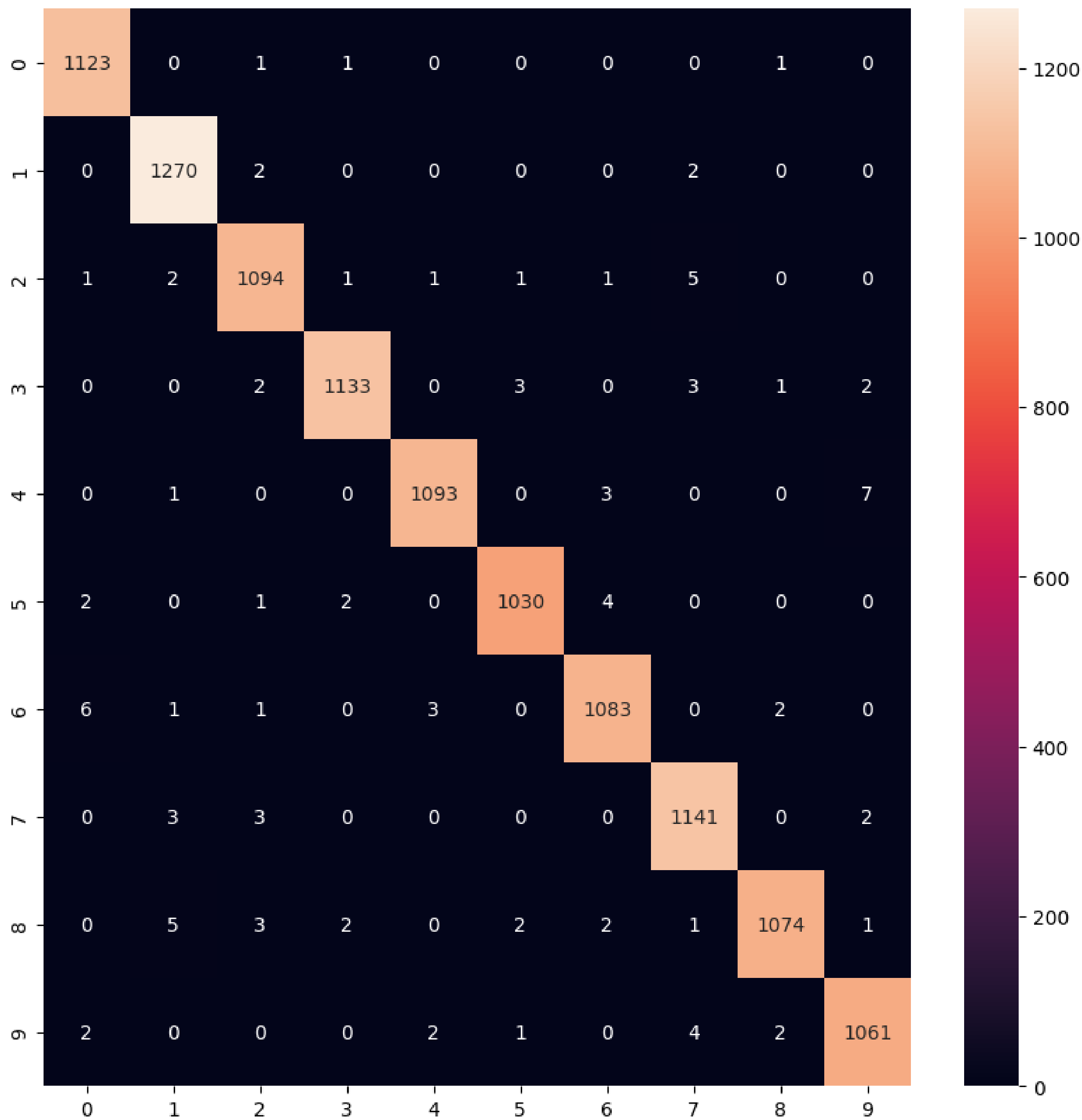
1089 0 2 6 4 9 8 0 6 2
0 1252 4 4 0 3 1 1 8 1
8 12 986 22 11 6 13 14 30 4
9 7 15 1037 2 29 4 5 19 17
4 3 3 1 1038 0 16 0 5 34
15 5 7 28 16 897 28 3 30 10
15 4 6 2 5 16 1040 0 8 0
2 9 13 9 11 1 1 1059 2 42
1 39 11 26 6 30 9 4 936 28
6 3 1 18 28 6 1 31 10 968
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0

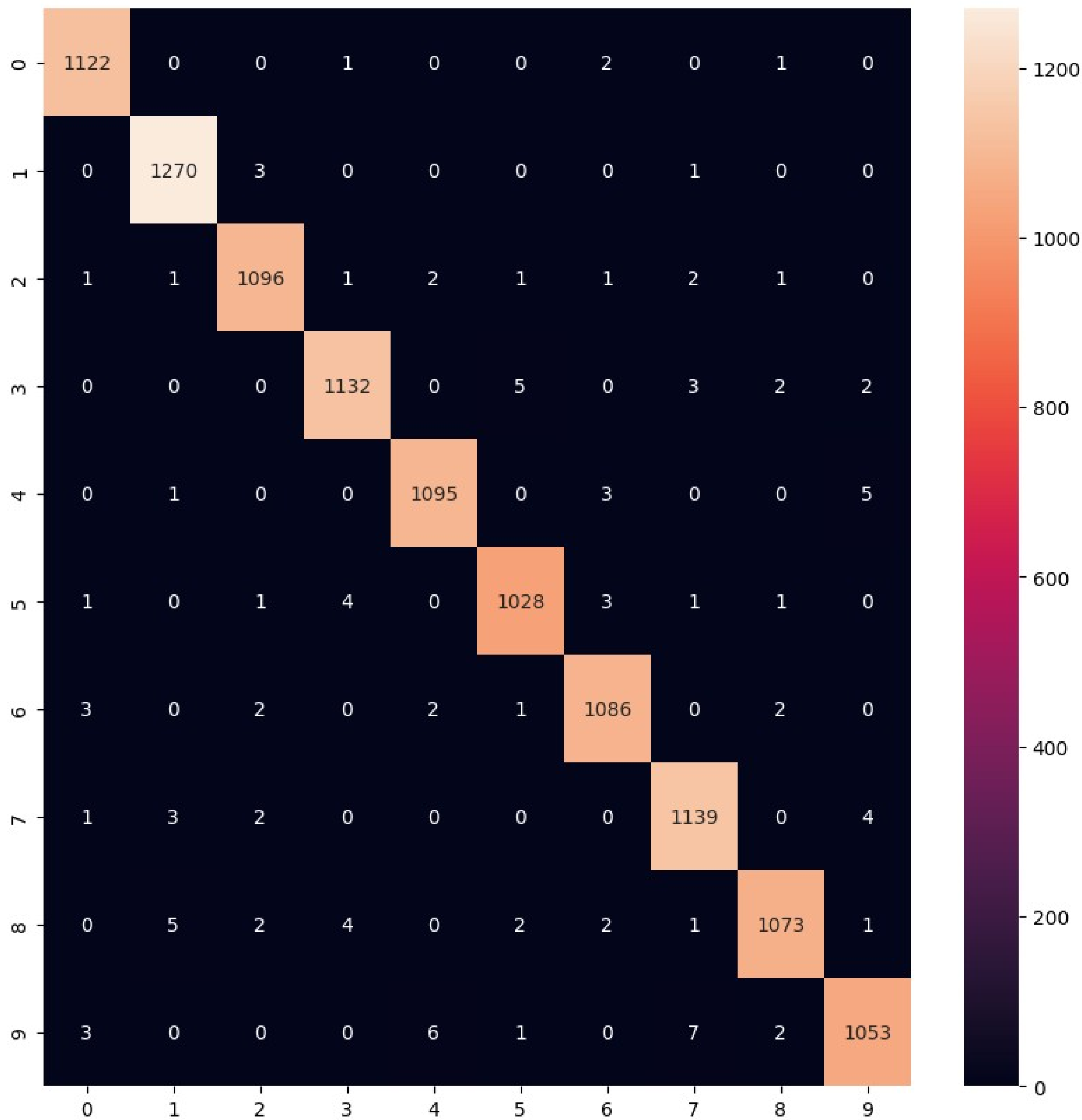


1088 0 2 5 3 10 7 0 9 2
0 1251 4 4 0 2 2 1 9 1
7 11 980 20 14 7 12 19 30 6
8 8 16 1024 1 41 7 7 18 14
4 4 2 1 1023 0 16 1 8 45
18 6 6 23 14 912 21 5 25 9
17 4 9 2 8 21 1023 0 12 0
4 8 16 4 9 1 1 1079 4 23
1 33 14 29 6 41 6 7 931 22
11 3 2 19 29 8 0 38 10 952
0 1 2 3 4 5 6 7 8 9
0 200 400 600 800 1000 1200


0 1 2 3 4 5 6 7 8 9
0 1090 0 2 5 3 7 7 0 10 2
1 0 1247 3 3 0 2 4 2 13 0
2 9 11 974 18 18 3 12 22 33 6
3 6 8 19 1025 1 35 5 11 22 12
4 4 4 2 0 1017 0 15 1 8 53
5 18 7 8 25 15 896 23 5 31 11
6 17 5 11 2 7 17 1026 0 11 0
7 4 10 14 3 9 1 1 1076 3 28
8 1 34 17 30 6 29 7 7 935 24
9 11 3 4 19 29 7 0 36 9 954
1200
1000
800
600
400
200
0


0 1093 0 2 1 2 8 7 3 10 0
1 0 1256 4 4 0 1 1 2 6 0
2 8 7 1035 10 11 2 7 11 15 0
3 4 2 7 1071 1 23 4 7 14 11
4 1 1 1 0 1058 1 20 0 2 20
5 9 5 3 13 7 964 15 3 11 9
6 19 4 1 0 3 9 1054 0 6 0
7 5 9 13 1 12 0 1 1093 0 15
8 1 21 7 10 7 12 2 5 1011 14
9 8 2 1 12 21 8 1 17 8 994
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0


0 1 2 3 4 5 6 7 8 9
1123 0 1 1 0 0 0 0 1 0
0 1270 2 0 0 0 0 2 0 0
1 2 1094 1 1 1 1 5 0 0
0 0 2 1133 0 3 0 3 1 2
0 1 0 0 1093 0 3 0 0 7
2 0 1 2 0 1030 4 0 0 0
6 1 1 0 3 0 1083 0 2 0
0 3 3 0 0 0 0 1141 0 2
0 5 3 2 0 2 2 1 1074 1
2 0 0 0 2 1 0 4 2 1061
1200
1000
800
600
400
200
0




# Evaluation

There are some evaluation insights in the notebook, but for a detailed evaluation, please refer to the Wandb dashboard.

```
# Display summary table
summary_df = tracker.get_all_summaries()
# remove confusion matrix from summary
display(summary_df.drop(columns=['confusion_matrix']).style.highlight_
max(subset=['accuracy']))
```

```
<pandas.io.formats.style.Styler at 0x7bdd5de2a4d0>
```

# Model Comparison

## Accuracy Analysis

- Best performing models:
    a. `deep_net_medium`: 99.13% accuracy
    b. `deep_net_large`: 99.05% accuracy
    c. `deep_net_nano`: 94.90% accuracy
- Softmax regression models range between 71-92% accuracy

## Learning Rate Impact

- `softmax_lr_0.01`: Best softmax regression model (92.14% accuracy)
- `softmax_lr_1.0`: Moderate performance (89.00% accuracy)
    - Indicates learning rate is too high
- `softmax_lr_0.00001`: Poor performance (71.54% accuracy)
    - Indicates learning rate is crucial for model convergence
    - Too low learning rate leads to slow convergence

## Batch Size Experiments

- `softmax_batch_32`: Optimal batch size for softmax (91.98% accuracy)
- `softmax_batch_1`: Lowest performance (89.03% accuracy) with much longer training time.
- `softmax_batch_256`: Slightly lower performance (91.63% accuracy)

## Deep Network Insights

- Deeper networks consistently outperform softmax regression
- `deep_net_medium` shows marginal improvement over `deep_net_large`
- Increasing network complexity doesn't always guarantee better performance

## Inference Time Analysis

- Softmax models: ~0.0002-0.0003 seconds per inference
- Deep networks: ~0.0007-0.0015 seconds per inference
- Trade-off between accuracy and computational complexity

## Training Time Observations

- Softmax models: 389-463 seconds
- Deep networks: 503-2887 seconds
- Deep networks require more training time due to increased complexity

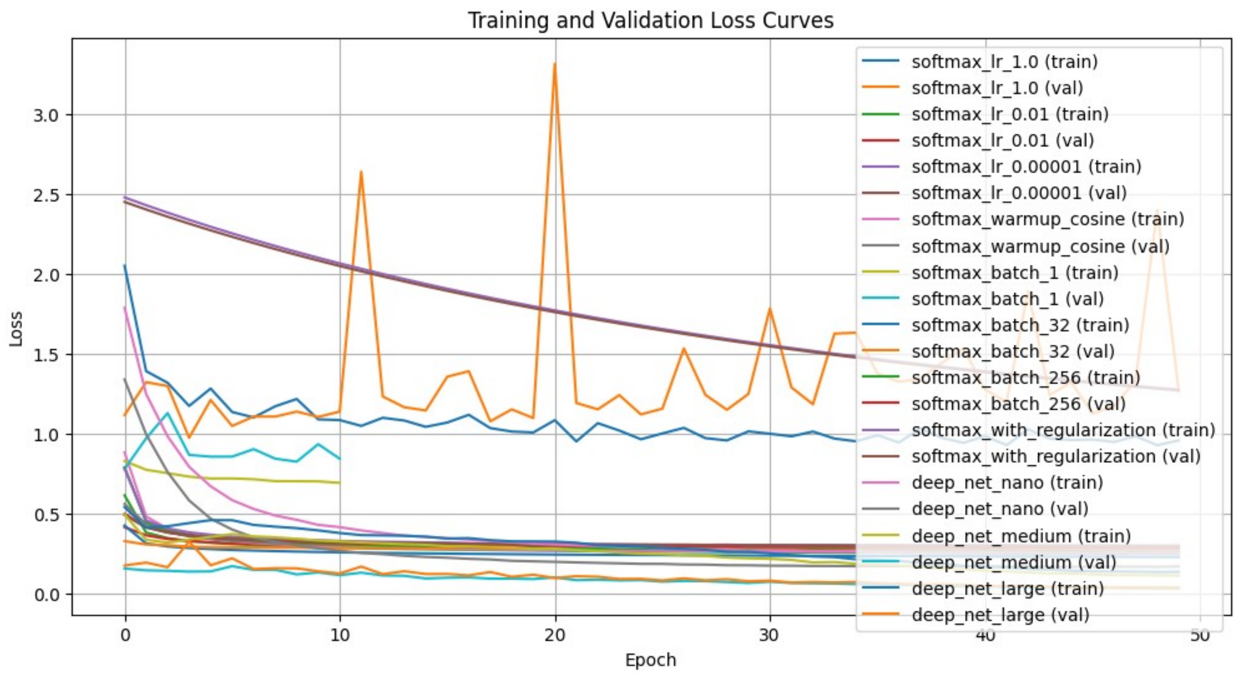
## Regularization Impact

- `softmax_with_regularization`: Slight performance reduction
- Suggests careful hyperparameter tuning needed

## Key Takeaways

1. Deep networks superior for MNIST
2. Learning rate critical for model performance
3. Batch size influences training speed and accuracy
4. Computational complexity increases with network depth

```python
# Plot training curves
fig = tracker.plot_training_curves('loss')
plt.show()
```




```python
fig = tracker.plot_training_curves('acc')
plt.show()
```


# Analysis of Training and Validation Curves

## Key Observations

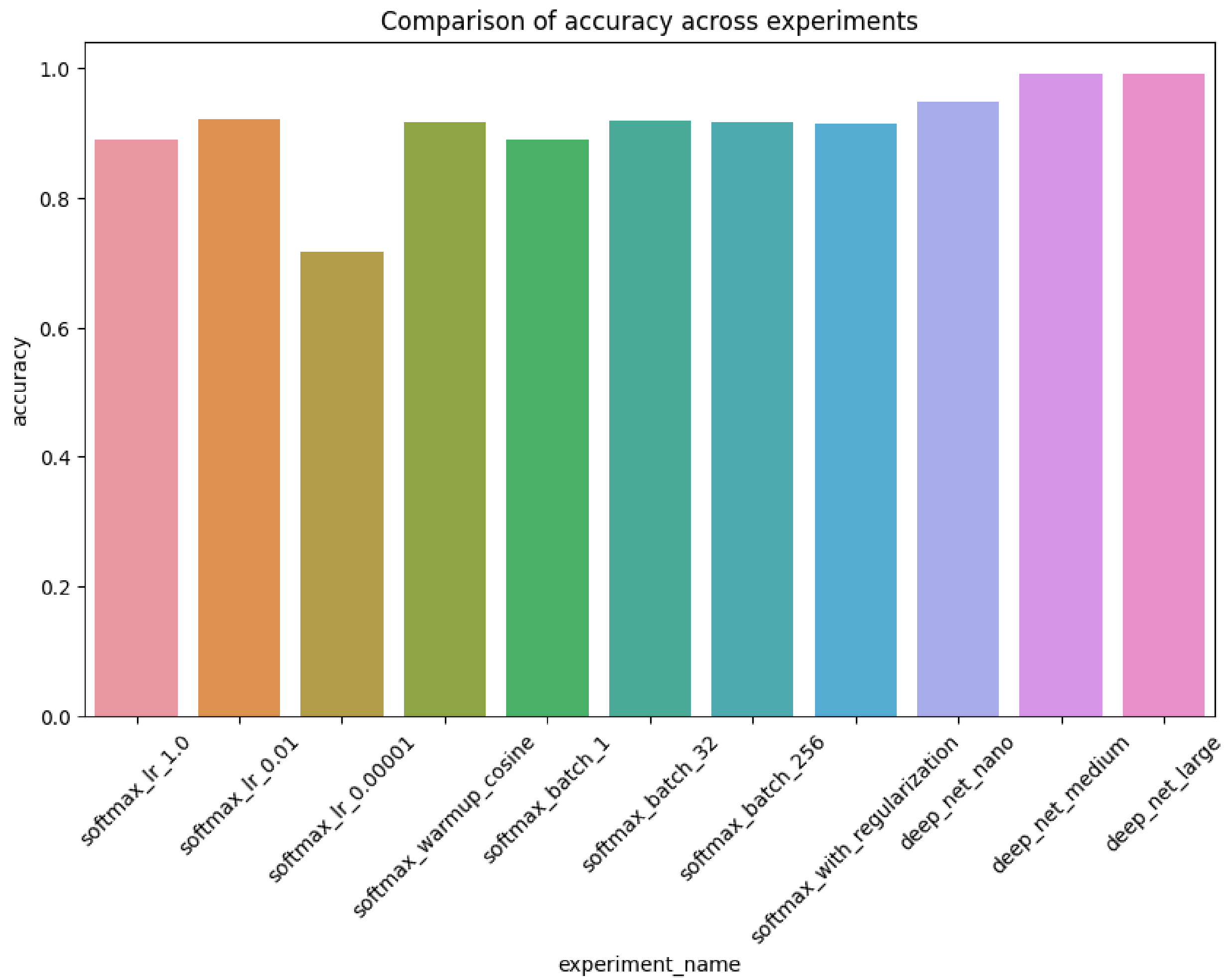
1. Softmax Models:
   - `softmax_lr_1.0`:
     - Exhibits unstable training behavior with spikes in loss values.
     - Indicates that a high learning rate prevents smooth convergence.
   - `softmax_lr_0.01`:
     - Best performance among softmax models, achieving the lowest validation loss.
     - Shows smoother training and validation curves, suggesting good learning rate.
   - `softmax_lr_0.00001`:
     - High and slow-decreasing loss values throughout training.
     - Reflects that a learning rate too small leads to slow or incomplete convergence.
   - Batch Size Impact:
     - Smaller batches (e.g., `softmax_batch_1`) result in noisier curves due to less accurate gradient estimates.
     - Larger batches (e.g., `softmax_batch_256`) stabilize the curves but sacrifice slight accuracy.
2. Deep Networks:
   - Models like `deep_net_medium` and `deep_net_large` consistently show lower training and validation losses compared to softmax models.
   - Network Size vs. Convergence:

    - Smaller networks like `deep_net_nano` converge faster but reach lower accuracy values.
    - Larger networks take longer to converge but achieve better accuracy.
3. Regularization:
   - Regularization models show slightly elevated loss values compared to unregularized models, indicating the penalty imposed by regularization to prevent overfitting.
4. Cosine Warmup Learning Rate:
   - The `softmax_warmup_cosine` curve demonstrates a gradual reduction in loss, indicating effective scheduling for stable convergence.

## Insights from the Curves

1. Learning Rate:
   - Proper tuning is critical, as seen in the superior performance of `softmax_lr_0.01` compared to other learning rate configurations.
   - Extremely low or high learning rates lead to poor convergence and unstable training.
2. Model Complexity:
   - Deep networks outperform softmax models in terms of achieving lower loss and higher accuracy.
   - Increasing network complexity doesn't always yield significant performance improvement (e.g., `deep_net_medium` vs. `deep_net_large`).
3. Batch Size:
   - Optimal batch sizes balance stability and accuracy, as evidenced by the superior performance of `softmax_batch_32`.
4. Overfitting Indicators:
   - Narrow gaps between training and validation loss curves indicate better generalization, as seen in `deep_net_medium`.
5. Stability and Convergence:
   - Models with smoother loss curves (e.g., `deep_net_medium` and `softmax_lr_0.01`) are more reliable and generalize better than those with fluctuating loss patterns.

```
# Compare final metrics on test set
fig = tracker.plot_experiment_comparison('accuracy')
plt.show()
```




```python
# Export results
summary_df.to_csv('experiment_results.csv')

# Detailed view of specific experiment
exp_df = tracker.get_experiment_df('deep_net_medium')
display(exp_df)
```

```
    epoch  train_loss  train_acc  val_loss    val_acc  
learning_rate  \
0       0    0.497291  84.109375  0.159649  94.942857       0.004667

1       1    0.338992  89.111607  0.148122  95.471429       0.006000

2       2    0.321408  89.743304  0.145222  95.571429       0.007333

3       3    0.340370  89.200893  0.140176  95.735714       0.008667

4       4    0.356799  88.883929  0.141500  95.450000       0.010000

5       5    0.374468  88.024554  0.174082  94.557143       0.009988
```

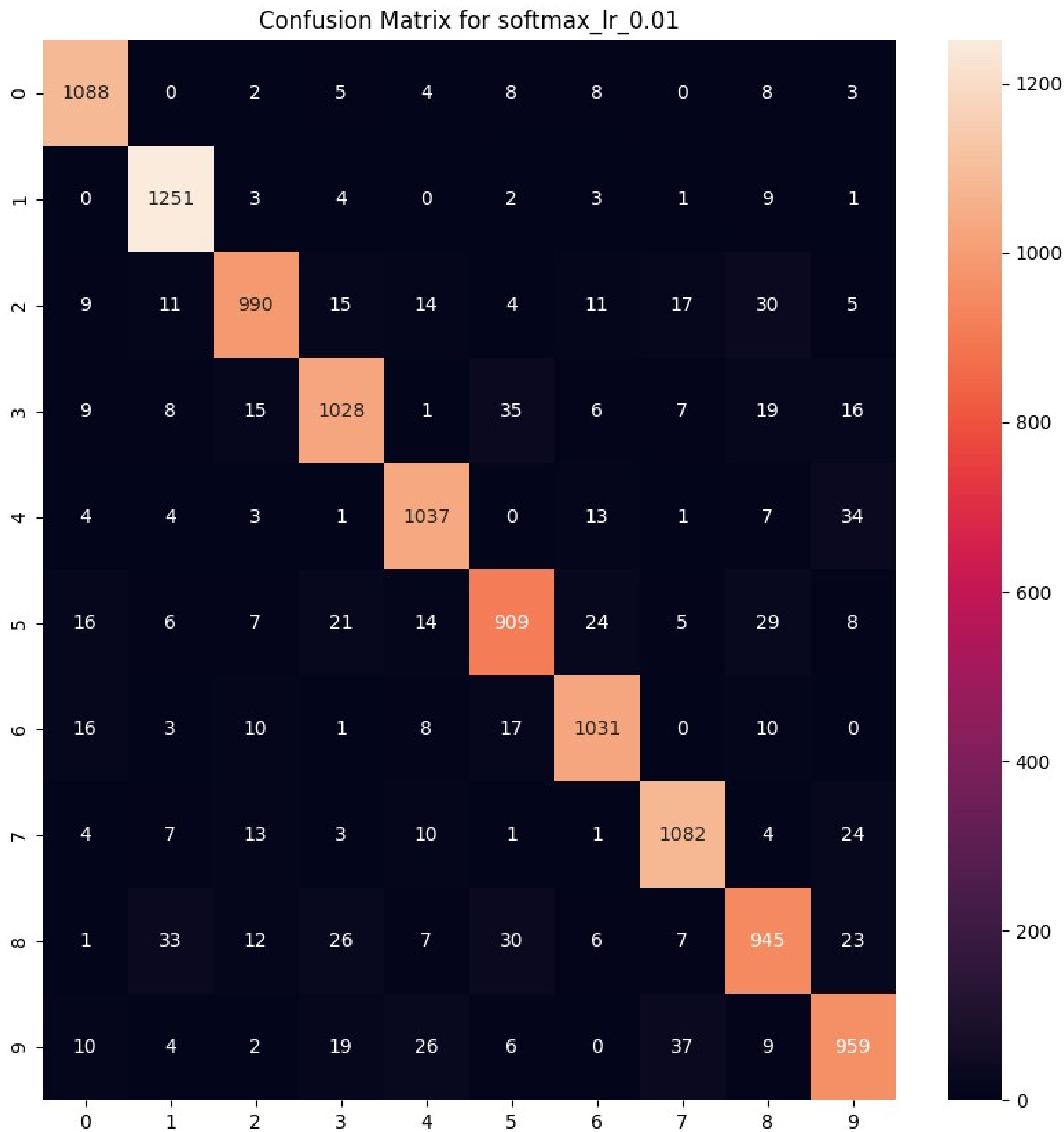
| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 6 | 0.360194 | 88.705357 | 0.151320 | 95.435714 | 0.009951 |
| 7 | 7 | 0.354372 | 88.816964 | 0.151848 | 95.357143 | 0.009891 |
| 8 | 8 | 0.344741 | 89.189732 | 0.122903 | 96.364286 | 0.009807 |
| 9 | 9 | 0.334645 | 89.392857 | 0.134888 | 96.164286 | 0.009699 |
| 10 | 10 | 0.328464 | 89.738839 | 0.118059 | 96.371429 | 0.009568 |
| 11 | 11 | 0.322392 | 89.819196 | 0.133623 | 95.692857 | 0.009415 |
| 12 | 12 | 0.316809 | 90.008929 | 0.116012 | 96.471429 | 0.009241 |
| 13 | 13 | 0.311356 | 90.098214 | 0.114108 | 96.392857 | 0.009046 |
| 14 | 14 | 0.307336 | 90.133929 | 0.097346 | 97.064286 | 0.008831 |
| 15 | 15 | 0.298610 | 90.589286 | 0.101906 | 96.671429 | 0.008598 |
| 16 | 16 | 0.297787 | 90.720982 | 0.103948 | 96.678571 | 0.008347 |
| 17 | 17 | 0.293910 | 90.794643 | 0.096233 | 96.900000 | 0.008080 |
| 18 | 18 | 0.291423 | 90.948661 | 0.096302 | 96.964286 | 0.007798 |
| 19 | 19 | 0.282541 | 91.131696 | 0.093726 | 97.007143 | 0.007502 |
| 20 | 20 | 0.284431 | 91.033482 | 0.103723 | 96.657143 | 0.007195 |
| 21 | 21 | 0.274697 | 91.321429 | 0.086767 | 97.221429 | 0.006876 |
| 22 | 22 | 0.266992 | 91.658482 | 0.089871 | 97.214286 | 0.006549 |
| 23 | 23 | 0.258531 | 91.805804 | 0.084989 | 97.342857 | 0.006213 |
| 24 | 24 | 0.252172 | 92.109375 | 0.086919 | 97.364286 | 0.005872 |
| 25 | 25 | 0.248957 | 92.162946 | 0.077618 | 97.550000 | 0.005527 |
| 26 | 26 | 0.243560 | 92.412946 | 0.081955 | 97.621429 | 0.005179 |
| 27 | 27 | 0.238646 | 92.497768 | 0.080512 | 97.642857 | 0.004831 |
| 28 | 28 | 0.228051 | 92.863839 | 0.075118 | 97.642857 | 0.004483 |
| 29 | 29 | 0.223390 | 92.979911 | 0.069270 | 97.814286 | 0.004138 |
| 30 | 30 | 0.219947 | 93.198661 | 0.077390 | 97.514286 | 0.003797 |
| 31 | 31 | 0.213256 | 93.270089 | 0.068730 | 97.850000 | 0.003461 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 32 | 32 | 0.197659 | 93.662946 | 0.067091 | 97.964286 | 0.003134 |
| 33 | 33 | 0.199889 | 93.622768 | 0.065623 | 97.942857 | 0.002815 |
| 34 | 34 | 0.188858 | 94.082589 | 0.062283 | 97.928571 | 0.002508 |
| 35 | 35 | 0.184824 | 94.250000 | 0.058486 | 98.178571 | 0.002212 |
| 36 | 36 | 0.178929 | 94.415179 | 0.056118 | 98.242857 | 0.001930 |
| 37 | 37 | 0.169910 | 94.756696 | 0.055078 | 98.307143 | 0.001663 |
| 38 | 38 | 0.158018 | 95.064732 | 0.050002 | 98.442857 | 0.001412 |
| 39 | 39 | 0.153615 | 95.169643 | 0.047878 | 98.478571 | 0.001179 |
| 40 | 40 | 0.145442 | 95.466518 | 0.047226 | 98.557143 | 0.000964 |
| 41 | 41 | 0.141253 | 95.564732 | 0.043872 | 98.600000 | 0.000769 |
| 42 | 42 | 0.134877 | 95.665179 | 0.041059 | 98.728571 | 0.000595 |
| 43 | 43 | 0.128707 | 95.924107 | 0.040305 | 98.700000 | 0.000442 |
| 44 | 44 | 0.128374 | 95.988839 | 0.040662 | 98.735714 | 0.000311 |
| 45 | 45 | 0.121996 | 96.247768 | 0.039315 | 98.771429 | 0.000203 |
| 46 | 46 | 0.118953 | 96.287946 | 0.037799 | 98.771429 | 0.000119 |
| 47 | 47 | 0.117970 | 96.319196 | 0.036895 | 98.850000 | 0.000059 |
| 48 | 48 | 0.117851 | 96.350446 | 0.036186 | 98.878571 | 0.000022 |
| 49 | 49 | 0.115109 | 96.274554 | 0.036407 | 98.871429 | 0.000010 |

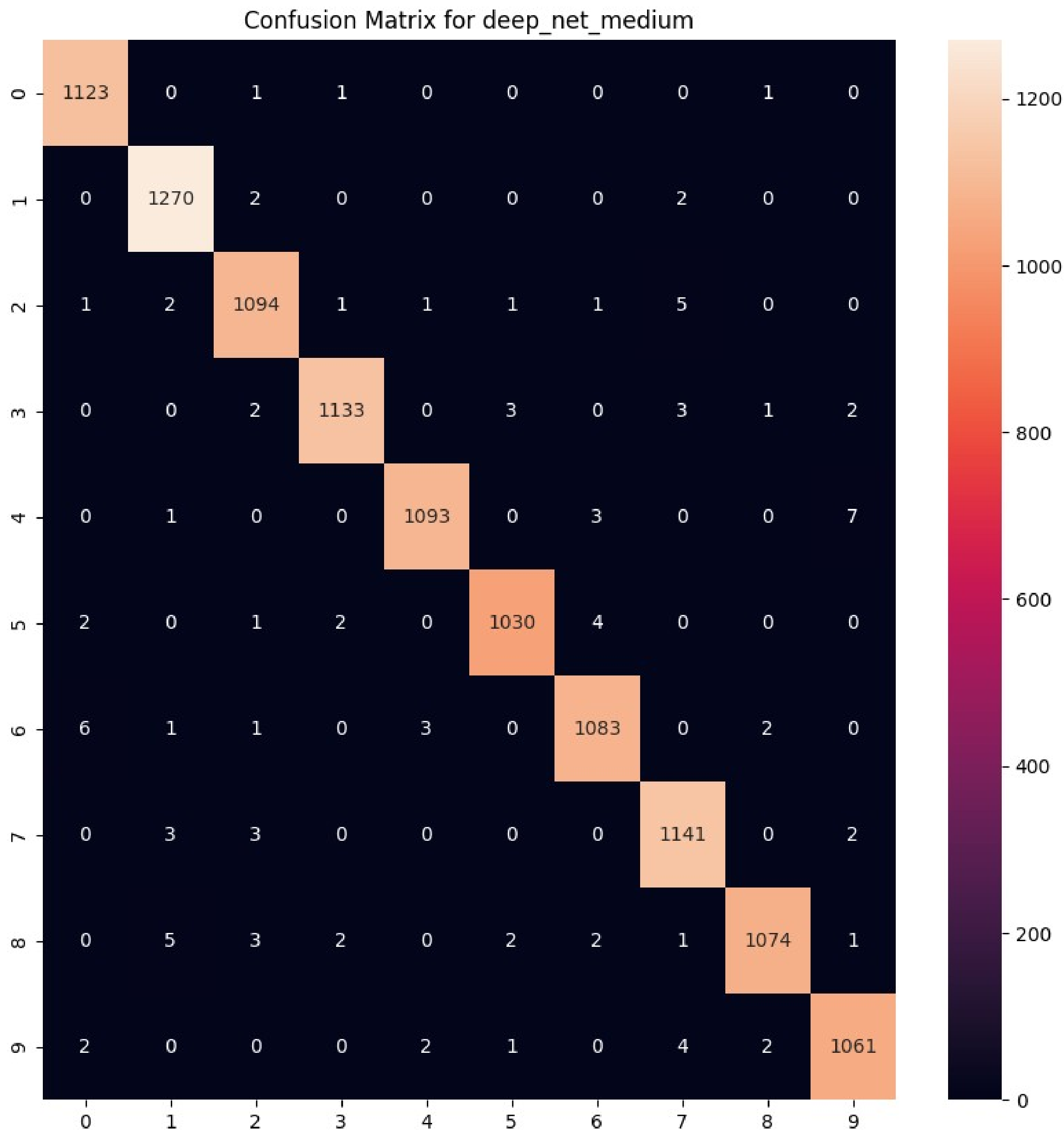
| | epoch_time |
|---|---|
| 0 | 55.996205 |
| 1 | 55.830900 |
| 2 | 55.088057 |
| 3 | 55.864079 |
| 4 | 54.722889 |
| 5 | 55.230920 |
| 6 | 55.298661 |
| 7 | 55.741643 |
| 8 | 55.441713 |
| 9 | 54.673023 |
| 10 | 55.307901 |
| 11 | 55.262882 |

| | |
|---|---|
| 12 | 54.352537 |
| 13 | 56.216399 |
| 14 | 57.066223 |
| 15 | 56.571239 |
| 16 | 56.457256 |
| 17 | 55.229942 |
| 18 | 55.152944 |
| 19 | 55.387287 |
| 20 | 57.551653 |
| 21 | 56.896703 |
| 22 | 57.808463 |
| 23 | 55.984190 |
| 24 | 56.250254 |
| 25 | 56.417300 |
| 26 | 56.342187 |
| 27 | 56.036708 |
| 28 | 55.699501 |
| 29 | 56.235218 |
| 30 | 55.848260 |
| 31 | 55.152828 |
| 32 | 55.881460 |
| 33 | 55.737539 |
| 34 | 55.819153 |
| 35 | 55.711973 |
| 36 | 56.376323 |
| 37 | 56.225123 |
| 38 | 55.842007 |
| 39 | 55.884803 |
| 40 | 55.952831 |
| 41 | 55.170945 |
| 42 | 55.267149 |
| 43 | 56.017612 |
| 44 | 72.692998 |
| 45 | 56.677200 |
| 46 | 57.785577 |
| 47 | 56.317838 |
| 48 | 55.671846 |
| 49 | 56.070641 |

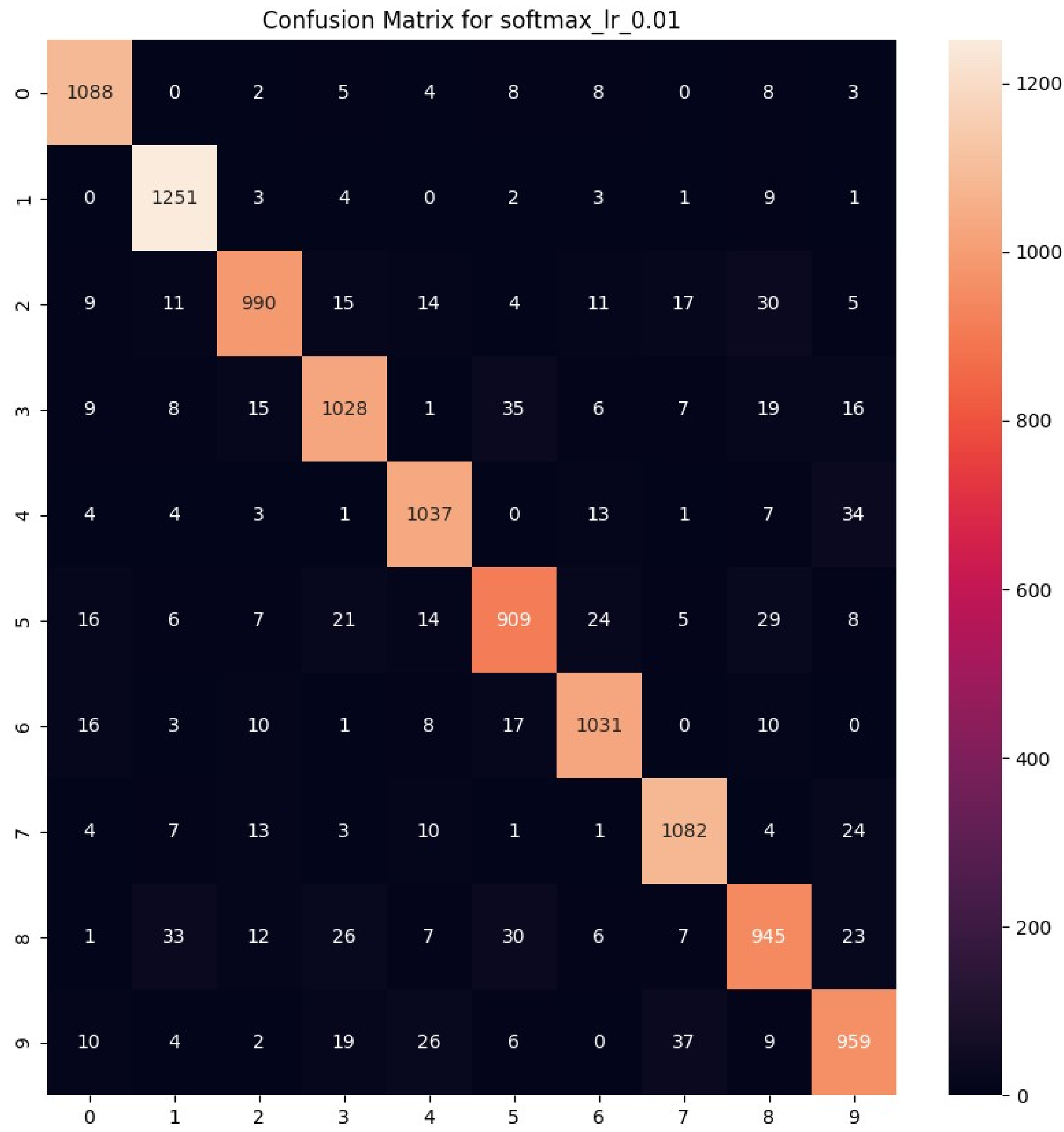
```
tracker.plot_confusion_matrix('deep_net_medium')
tracker.plot_confusion_matrix('softmax_lr_0.01')
```



Confusion Matrix for softmax_lr_0.01
1088 0 2 5 4 8 8 0 8 3
0 1251 3 4 0 2 3 1 9 1
9 11 990 15 14 4 11 17 30 5
9 8 15 1028 1 35 6 7 19 16
4 4 3 1 1037 0 13 1 7 34
16 6 7 21 14 909 24 5 29 8
16 3 10 1 8 17 1031 0 10 0
4 7 13 3 10 1 1 1082 4 24
1 33 12 26 7 30 6 7 945 23
10 4 2 19 26 6 0 37 9 959
0 1 2 3 4 5 6 7 8 9
1200
1000
800
600
400
200
0



Confusion Matrix for deep_net_medium




## Confusion Matrix Analysis

- The confusion matrix for the softmax regression model with the learning rate of 0.01 shows that the model is performing well, with relatively low misclassifications. However, it still struggles with certain digits, especially those that are visually similar e.g., 3vs5 or 4vs9.

- The confusion matrix of the deep neural network reveals an exceptionally high accuracy rate, with very few misclassifications. Most of the confusion happens between digits that are similar, like 4 vs 9, but these are minimal.

The deep neural network shows that it can effectively differentiate between a wide range of digit variations, which is reflected in the low false positive and false negative rates.

## Misclassification Analysis on Deep Neural Network

We can see from the sample of the miscalssified images that the deep neural network is struggling with digits that are not clearly written or are ambiguous. This is expected as even humans can struggle with these images.