ДОДАТОК А

Модель на основі попередньо навченої моделі MobileNet

```
import os
import datetime
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import classification_report
from PIL import Image
import random
import numpy as np
import torch

seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# gpu
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print(f"Using device: {device}")

# direction to save plots
plots_dir = 'plots'
os.makedirs(plots_dir, exist_ok=True)
print(f"Using directory: {plots_dir}")

# image transformation and data augmentation
transform = transforms.Compose([
   transforms.RandomResizedCrop(256, scale=(0.8, 1.0)),  #
random crop and resize
   transforms.RandomHorizontalFlip(p=0.5),  # 50% of images
   transforms.RandomRotation(15),  # random rotation ±15
degrees
   transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2, hue=0.1),  # brightness & color
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
])
```

```
# data
root_dir = 'maps-data'
dataset = datasets.ImageFolder(root=root_dir,
transform=transform)

# split
train_size = int(0.7 * len(dataset))
val_size = int(0.2 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_set, val_set, test_set = random_split(dataset,
[train_size, val_size, test_size])

# dataloaders
batch_size = 32
train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_set, batch_size=batch_size,
shuffle=False)

print(f"Train size: {len(train_loader)} batches")
print(f"Validation size: {len(val_loader)} batches")
print(f"Test size: {len(test_loader)} batches")

# MobileNet - load
model = models.mobilenet_v2(pretrained=True)

# classifier
model.classifier[1] = nn.Linear(model.last_channel, 1)  # 1
for binary classification
model = model.to(device)

# function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

def train_model(model, train_loader, val_loader, epochs=30):
   best_val_loss = float('inf')
   train_losses, val_losses = [], []
   train_accuracies, val_accuracies = [], []

   for epoch in range(epochs):
       model.train()
       total_loss, correct_train, total_train = 0, 0, 0
       for images, labels in train_loader:
           images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
           optimizer.zero_grad()
           outputs = model(images)
           loss = criterion(outputs, labels)
           loss.backward()
```

```
            optimizer.step()
            total_loss += loss.item()

            # train accuracy
            preds = (torch.sigmoid(outputs) > 0.5).int()
            correct_train += (preds == labels).sum().item()
            total_train += labels.size(0)

        train_losses.append(total_loss / len(train_loader))
        train_accuracies.append(correct_train / total_train)

        # validation
        model.eval()
        val_loss, correct_val, total_val = 0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                # validation accuracy
                preds = (torch.sigmoid(outputs) > 0.5).int()
                correct_val += (preds == labels).sum().item()
                total_val += labels.size(0)

        val_losses.append(val_loss / len(val_loader))
        val_accuracies.append(correct_val / total_val)

        print(f"Epoch {epoch+1}/{epochs}, Train Loss:
{train_losses[-1]:.4f}, "
              f"Train Acc: {train_accuracies[-1]:.4f}, "
              f"Val Loss: {val_losses[-1]:.4f}, Val Acc:
{val_accuracies[-1]:.4f}")

        # early stopping
        if val_losses[-1] < best_val_loss:
            best_val_loss = val_losses[-1]
            counter = 0
        else:
            counter += 1  # patience

        print(f"Early stopping patience counter: {counter}")

    return train_losses, val_losses, train_accuracies,
val_accuracies

train_losses, val_losses, train_accuracies, val_accuracies =
train_model(model, train_loader, val_loader)

# plots - loss
plt.plot(train_losses, label='Train Loss', color='silver')
```

```
plt.plot(val_losses, label='Validation Loss', color='tan')
plt.title('Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'loss_plot.png'))
plt.close()
print("Loss plot saved")
# plot - accuracy

plt.plot(train_accuracies, label='Train Accuracy',
color='silver')
plt.plot(val_accuracies, label='Validation Accuracy',
color='tan')
plt.title('Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'accuracy_plot.png'))
plt.close()
print("Accuracy plot saved")


def evaluate_model(model, test_loader):
   model.load_state_dict(torch.load('best_model.pth'))
   model.eval()
   true_labels, pred_labels = [], []
   with torch.no_grad():
       for images, labels in test_loader:
           images, labels = images.to(device),
labels.to(device)
           outputs = model(images)
           preds = (torch.sigmoid(outputs) > 0.5).int()
           true_labels.extend(labels.cpu().numpy())
           pred_labels.extend(preds.cpu().numpy())

   print("\nClassification Report:")
   print(classification_report(true_labels, pred_labels,
target_names=['European City', 'American City']))

evaluate_model(model, test_loader)
```

ДОДАТОК Б

Модель на основі попередньо навченої моделі EfficientNet

```
import os
import datetime
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from torch.utils.data import DataLoader, random_split,
WeightedRandomSampler
import matplotlib.pyplot as plt
import random
import numpy as np
from sklearn.metrics import classification_report

seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# gpu
device = torch.device("cuda" if torch.cuda.is_available()
else "cpu")
print(f"Using device: {device}")

# directories for plot
plots_dir = 'plots'
os.makedirs(plots_dir, exist_ok=True)

# transformation
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
])

# data
root_dir = 'maps-data-2'
dataset = datasets.ImageFolder(root=root_dir,
transform=transform)

# split data
```

```
train_size = int(0.7 * len(dataset))
val_size = int(0.2 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_set, val_set, test_set = random_split(dataset,
[train_size, val_size, test_size])

class_counts = [4419, 2869]
class_weights = 1.0 / torch.tensor(class_counts,
dtype=torch.float)

# weights
sample_weights = [class_weights[label] for _, label in
dataset.samples]

# sampler
sampler = WeightedRandomSampler(weights=sample_weights,
num_samples=len(sample_weights), replacement=True)

# dataloaders
batch_size = 10
train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_set, batch_size=batch_size,
shuffle=False)

# MobileNet
model = models.efficientnet_b0(pretrained=True)

# classifier
model.classifier[1] =
nn.Linear(model.classifier[1].in_features, 1)  # 1 for
binary classification
model = model.to(device)

# trainable and non-trainable parameters
total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters()
if p.requires_grad)
print(f"Total parameters: {total_params}")
print(f"Trainable parameters: {trainable_params}")
print(f"Percentage of trainable parameters:
{trainable_params/total_params*100:.2f}%")

# loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad,
model.parameters()), lr=0.0001)

def train_model(model, train_loader, val_loader, epochs=7):
```

```
    best_val_loss = float('inf')
    train_losses, val_losses = [], []
    train_accuracies, val_accuracies = [], []
    counter = 0  # early stop counter

    for epoch in range(epochs):
        model.train()
        total_loss, correct_train, total_train = 0, 0, 0
        for images, labels in train_loader:
            images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

            # train accuracy
            preds = (torch.sigmoid(outputs) > 0.5).int()
            correct_train += (preds == labels).sum().item()
            total_train += labels.size(0)

        train_losses.append(total_loss / len(train_loader))
        train_accuracies.append(correct_train / total_train)

        # validation
        model.eval()
        val_loss, correct_val, total_val = 0, 0, 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()

                # validation accuracy
                preds = (torch.sigmoid(outputs) > 0.5).int()
                correct_val += (preds == labels).sum().item()
                total_val += labels.size(0)

        val_losses.append(val_loss / len(val_loader))
        val_accuracies.append(correct_val / total_val)

        print(f"Epoch {epoch+1}/{epochs}, Train Loss:
{train_losses[-1]:.4f}, "
              f"Train Acc: {train_accuracies[-1]:.4f}, "
              f"Val Loss: {val_losses[-1]:.4f}, Val Acc:
{val_accuracies[-1]:.4f}")
```

```
        # early stopping
        if val_losses[-1] < best_val_loss:
            best_val_loss = val_losses[-1]
            torch.save(model.state_dict(), 'best_model.pth')
            counter = 0  # patience counter
        else:

            counter += 1
            if counter >= 3:  # stop if no improvement
                print("Early stopping triggered")
                break

    return train_losses, val_losses, train_accuracies,
val_accuracies

# train
train_losses, val_losses, train_accuracies, val_accuracies =
train_model(model, train_loader, val_loader)

# plot loss
plt.plot(train_losses, label='Train Loss', color='silver')
plt.plot(val_losses, label='Validation Loss', color='tan')
plt.title('Loss During Training')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'loss_plot.png'))
plt.close()
print("Loss plot saved")

# accuracy plot
plt.plot(train_accuracies, label='Train Accuracy',
color='silver')
plt.plot(val_accuracies, label='Validation Accuracy',
color='tan')
plt.title('Accuracy During Training')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'accuracy_plot.png'))
plt.close()
print("Accuracy plot saved")

def evaluate_model(model, test_loader):
    model.load_state_dict(torch.load('best_model.pth'))
    model.eval()
    true_labels, pred_labels = [], []
    with torch.no_grad():
        for images, labels in test_loader:
```

```
            images, labels = images.to(device),
labels.to(device)
            outputs = model(images)
            preds = (torch.sigmoid(outputs) > 0.5).int()
            true_labels.extend(labels.cpu().numpy())
            pred_labels.extend(preds.cpu().numpy())
    print("\nClassification Report:")
    print(classification_report(true_labels, pred_labels,
target_names=['0', '1']))
evaluate_model(model, test_loader)
```

ДОДАТОК В

Модель на основі попередньо навченої моделі ResNet

```
import os
import datetime
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from torch.utils.data import DataLoader, random_split
import matplotlib.pyplot as plt
import random
import numpy as np
from sklearn.metrics import classification_report
from torch.utils.data import WeightedRandomSampler
from PIL import Image

seed = 42
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# GPU
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print(f"Using device: {device}")

plots_dir = 'plots'
os.makedirs(plots_dir, exist_ok=True)
transform = transforms.Compose([
   transforms.Resize((256,256)),
   # transforms.RandomResizedCrop(256, scale=(0.8, 1.0)),
   transforms.RandomHorizontalFlip(p=0.5),
   transforms.RandomRotation(15), #degrees
   transforms.ColorJitter(brightness=0.2, contrast=0.2,
saturation=0.2, hue=0.1),
   transforms.ToTensor(),
   transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
])

# data
root_dir = 'maps-data'
# filter out big pictures

def remove_large_images(root_dir, max_size=(5000, 5000)):
```

```
    removed = 0
    for class_dir in os.listdir(root_dir):
        class_path = os.path.join(root_dir, class_dir)
        if not os.path.isdir(class_path):
            continue
        for img_name in os.listdir(class_path):
            img_path = os.path.join(class_path, img_name)
            try:
                with Image.open(img_path) as img:
                    if img.size[0] > max_size[0] or img.size[1]
> max_size[1]:
                        os.remove(img_path)
                        removed += 1
            except Exception as e:
                print(f"Error with image {img_path}: {e}")
                os.remove(img_path)  # corrupted pictures
                removed += 1
    print(f"Removed {removed} large/corrupted images.")

remove_large_images(root_dir, max_size=(5000, 5000))

dataset = datasets.ImageFolder(root=root_dir,
transform=transform)

# split
train_size = int(0.7 * len(dataset))
val_size = int(0.2 * len(dataset))
test_size = len(dataset) - train_size - val_size
train_set, val_set, test_set = random_split(dataset,
[train_size, val_size, test_size])

# weights
labels = [label for _, label in dataset.samples]
class_counts = np.bincount(labels)

print(f"Class Counts: {class_counts}") # [0, 1] => [European
City, American City], n of images in each class

class_weights = 1. / class_counts

print(f"Class Weights: {class_weights}")

sample_weights = [class_weights[label] for _, label in
dataset.samples]
sampler = WeightedRandomSampler(weights=sample_weights,
num_samples=len(sample_weights), replacement=True)

# weights
custom_class_weights = {0: 0.3, 1: 0.7}

sample_weights = [custom_class_weights[label] for _, label in
dataset.samples]
```

```
# WeightedRandomSampler
sampler = WeightedRandomSampler(weights=sample_weights,
num_samples=len(sample_weights), replacement=True)

# dataloaders
batch_size = 50
train_loader = DataLoader(train_set, batch_size=batch_size,
shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size,
shuffle=False)
test_loader = DataLoader(test_set, batch_size=batch_size,
shuffle=False)

def freeze_layers(model, freeze_percentage=0.3):
    all_params = list(model.parameters())
    total_params = len(all_params)
    freeze_layers_count = int(total_params * freeze_percentage)

    for i, param in enumerate(model.parameters()):
        param.requires_grad = i >= freeze_layers_count
    return model

# ResNet
model = models.resnet18(pretrained=True)

# freeze layers
model = freeze_layers(model, freeze_percentage=0.3)

# final layer for binary classification
num_features = model.fc.in_features
# model.fc = nn.Linear(num_features, 1)

model.fc = nn.Sequential(
    nn.Dropout(p=0.5),          # Dropout with 50% probability
    nn.Linear(num_features, 1) # Binary output
)
model = model.to(device)

# n trainable and non  parameters
total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters()
if p.requires_grad)
print(f"Total parameters: {total_params}")
print(f"Trainable parameters: {trainable_params}")
print(f"Percentage of trainable parameters:
{trainable_params/total_params*100:.2f}%")

# loss function and optimizer
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(filter(lambda p: p.requires_grad,
model.parameters()), lr=0.0001)
```

```
def train_model(model, train_loader, val_loader, optimizer,
criterion, device, epochs=5, early_patience=3):
   best_val_loss = float('inf')
   train_losses, val_losses = [], []
   train_accuracies, val_accuracies = [], []
   counter = 0  # early stopping counting

   for epoch in range(epochs):
       model.train()
       total_loss, correct_train, total_train = 0, 0, 0
       for images, labels in train_loader:
           images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
           optimizer.zero_grad()
           outputs = model(images)
           loss = criterion(outputs, labels)
           loss.backward()
           optimizer.step()
           total_loss += loss.item()

           # train accuracy
           preds = (torch.sigmoid(outputs) > 0.5).int()
           correct_train += (preds == labels).sum().item()
           total_train += labels.size(0)

       train_losses.append(total_loss / len(train_loader))
       train_accuracies.append(correct_train / total_train)

       # validation
       model.eval()
       val_loss, correct_val, total_val = 0, 0, 0
       with torch.no_grad():
           for images, labels in val_loader:
               images, labels = images.to(device),
labels.float().unsqueeze(1).to(device)
               outputs = model(images)
               loss = criterion(outputs, labels)
               val_loss += loss.item()

               # validation accuracy
               preds = (torch.sigmoid(outputs) > 0.5).int()
               correct_val += (preds == labels).sum().item()
               total_val += labels.size(0)

       val_losses.append(val_loss / len(val_loader))
       val_accuracies.append(correct_val / total_val)

       print(f"Epoch {epoch+1}/{epochs}, Train Loss:
{train_losses[-1]:.4f}, "
             f"Train Acc: {train_accuracies[-1]:.4f}, "
             f"Val Loss: {val_losses[-1]:.4f}, Val Acc:
{val_accuracies[-1]:.4f}")
```

```
        # FLOWING EARLY STOP____
        if val_losses[-1] < best_val_loss:
            best_val_loss = val_losses[-1]
            torch.save(model.state_dict(), 'best_model.pth')
            counter = 0  # Reset patience counter
        else:
            counter += 1
            if counter >= early_patience:
                print("Early stopping triggered")
                break
        # FLOWING EARLY STOP____
    return train_losses, val_losses, train_accuracies,
val_accuracies

# train model
train_losses, val_losses, train_accuracies, val_accuracies =
train_model(
    model=model,
    train_loader=train_loader,
    val_loader=val_loader,
    optimizer=optimizer,
    criterion=criterion,
    device=device,
    epochs=10,
    early_patience=3
)
# plot loss
plt.plot(train_losses, label='Train Loss', color='silver')
plt.plot(val_losses, label='Validation Loss', color='tan')
plt.title('Loss During Training')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'loss_plot.png'))
plt.close()
print("Loss plot saved")

# plot accuracy
plt.plot(train_accuracies, label='Train Accuracy',
color='silver')
plt.plot(val_accuracies, label='Validation Accuracy',
color='tan')
plt.title('Accuracy During Training')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.savefig(os.path.join(plots_dir, 'accuracy_plot.png'))
plt.close()
print("Accuracy plot saved")

def evaluate_model(model, test_loader):
```

```
    model.load_state_dict(torch.load('best_model.pth'))
    model.eval()
    true_labels, pred_labels = [], []
    with torch.no_grad():
        for images, labels in test_loader:
            images, labels = images.to(device),
labels.to(device)
            outputs = model(images)
            preds = (torch.sigmoid(outputs) > 0.5).int()
            true_labels.extend(labels.cpu().numpy())
            pred_labels.extend(preds.cpu().numpy())

    print("\nClassification Report:")
    print(classification_report(true_labels, pred_labels,
target_names=['europe', 'america']))

evaluate_model(model, test_loader)
```

ДОДАТОК Г

Таблиця результатів класифікації

Тлумачення таблиці:

s.n (serial number) – порядковий номер експерименту, наскрізний номер;

n – номер експерименту у межах однієї попередньо натренованої моделі;

task – номер завдання, яке вирішувались у рамках конкретного експерименту;

freezing – містить інформацію щодо використаної техніки замороження шарів моделі та, у разі використання, пропорції замороження;

batch size – розмір пакета;

data augmentation – застосування техніки доповнення даних, виражено бінарно;

epochs – кількісь епох, впродовж яких відбувалось навчання у конкреному експериментів;

lr – швидкісь навчання;

acc – результат класифікації, точність.

s.n.	n	**task**	**model**	**freezing**	**batch size**	**data augmentation**	**epochs**	**lr**	**acc**
1	1	1	MobileNetV2	None	10	FALSE	11	0.01	0.31
2	2	1	MobileNetV2	None	32	FALSE	11	0.01	0.31
3	3	1	MobileNetV2	None	10	FALSE	11	0.001	0.31
4	4	1	MobileNetV2	None	50	FALSE	11	0.001	0.31
5	5	1	MobileNetV2	None	32	TRUE	11	0.001	0.38
6	6	1	MobileNetV2	0.7	10	FALSE	11	0.001	0.77
7	7	1	MobileNetV2	0.5	10	FALSE	11	0.001	0.77
8	8	1	MobileNetV2	0.5	10	TRUE	11	0.001	0.62
9	9	1	MobileNetV2	0.3	10	TRUE	20	0.0001	0.92

10	10	1	MobileNetV2	0.7	10	TRUE	20	0.0001	0.92
11	11	1	MobileNetV2	0.7	10	TRUE	20	0.0001	0.92
12	1	1	EfficintNetB0	None	10	FALSE	11	0.001	0.46
13	2	1	EfficintNetB0	None	10	FALSE	11	0.0001	0.69
14	3	1	EfficintNetB0	None	50	FALSE	11	0.0001	0.85
15	4	1	EfficintNetB0	None	50	TRUE	11	0.0001	0.77
16	5	1	EfficintNetB0	None	10	TRUE	20	0.0001	0.77
17	6	1	EfficintNetB0	0.3	50	TRUE	11	0.0001	0.77
18	7	1	EfficintNetB0	0.55	50	TRUE	11	0.0001	0.77
19	8	1	EfficintNetB0	0.7	10	TRUE	11	0.0001	0.54
20	9	1	EfficintNetB0	0.7	50	TRUE	11	0.0001	0.69
21	9	1	EfficintNetB0	0.3	70	TRUE	11	0.001	0.38
22	1	1	RestNet18	0.3	10	TRUE	10	0.001	0.62
23	2	1	RestNet18	0.5	10	TRUE	10	0.0001	0.62
24	3	1	RestNet18	0.3	50	TRUE	10	0.0001	0.46
25	4	1	RestNet18	0.3	50	TRUE	10	0.0001	0.69
26	1	2	MobileNetV2	0.7	10	TRUE	20	0.0001	0.93
27	2	2	MobileNetV2	0.7	32	TRUE	5	0.01	0.93
28	3	2	MobileNetV2	0.7	50	FALSE	5	0.01	0.93
29	4	2	MobileNetV2	0.3	10	FALSE	5	0.0001	0.93
30	5	2	MobileNetV2	None	10	FALSE	5	0.0001	0.93
31	6	2	MobileNetV2	0.3	32	FALSE	5	0.0001	0.95
32	1	2	EfficintNetB0	0.3	70	TRUE	11	0.001	0.94
33	2	2	EfficintNetB0	0.3	10	TRUE	7	0.0001	0.94
34	3	2	EfficintNetB0	0.3	100	TRUE	7	0.001	0.94
35	4	2	EfficintNetB0	0.3	100	FALSE	10	0.001	0.96
36	5	2	EfficintNetB0	0.5/0.3	100	FALSE	10	0.001	0.95
37	6	2	EfficintNetB0	None	100	FALSE	10	0.001	0.96
38	1	2	RestNet18	0.3	10	FALSE	5	0.0001	0.95
39	2	2	RestNet18	0.3	10	FALSE	10	0.0001	0.95
40	3	2	RestNet18	0.3	10	FALSE	10	0.0001	0.95
41	4	2	RestNet18	0.5	50	FALSE	10	0.0001	0.96
42	5	2	RestNet18	0.7	100	TRUE	10	0.0001	0.93
43	6	2	RestNet18	None	50	TRUE	10	0.0001	0.94

44	7	2	RestNet18	None	50	TRUE	10	0.001	0.93
45	8	2	RestNet18	None	50	FALSE	10	0.0001	0.95
46	9	2	RestNet18	None	70	FALSE	10	0.01	0.87
47	10	2	RestNet18	None	10	TRUE	10	0.001	0.88

ДОДАТОК Д

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	72 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Терлецька О.О.			Розробка системи класифікації картографічних зображень на основі згорткової нейронної мережі	Шифр групи	Код напр./спец.
Перевірив		Бродецький Ф.А.				ІТШІпз-23-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Золотухін О.В.					