ДОДАТОК А

Лістинг програми дослідження моделей

```
import numpy as np
import cv2
from skimage.filters import gabor
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report, precision_score, recall_score, f1_score, accuracy_score
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model, load_model
import tensorflow as tf
import time
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, AveragePooling2D, concatenate, Input
import os

(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)

def color_histogram(image):
    chans = cv2.split(image)
    features = []
    for chan in chans:
        hist = cv2.calcHist([chan], [0], None, [256], [0, 256])
        features.extend(hist.flatten())
    return np.array(features).flatten()

def edge_detection(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray, 100, 200)
    return edges.flatten()

def gabor_features(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    filt_real, filt_imag = gabor(gray, frequency=0.6)
    return filt_real.flatten()

x_train_hist = np.array([color_histogram(img) for img in x_train])
x_val_hist = np.array([color_histogram(img) for img in x_val])
x_test_hist = np.array([color_histogram(img) for img in x_test])
```

```
x_train_gabor = np.array([gabor_features(img) for img in x_train])
x_val_gabor = np.array([gabor_features(img) for img in x_val])
x_test_gabor = np.array([gabor_features(img) for img in x_test])

x_train_edges = np.array([edge_detection(img) for img in x_train])
x_val_edges = np.array([edge_detection(img) for img in x_val])
x_test_edges = np.array([edge_detection(img) for img in x_test])

knn_hist = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
knn_hist.fit(x_train_hist, y_train.ravel())
knn_hist_training_time = time.time() - start_time

val_accuracy_knn_hist = knn_hist.score(x_val_hist, y_val.ravel())
print(f'Validation Accuracy (k-NN, histogram): {val_accuracy_knn_hist}')

start_time = time.time()
y_pred_knn_hist = knn_hist.predict(x_test_hist)
knn_hist_inference_time = time.time() - start_time

print(classification_report(y_test, y_pred_knn_hist, target_names=[str(i) for i in range(10)]))
precision_knn_hist = precision_score(y_test, y_pred_knn_hist, average='macro')
recall_knn_hist = recall_score(y_test, y_pred_knn_hist, average='macro')
f1_knn_hist = f1_score(y_test, y_pred_knn_hist, average='macro')
accuracy_knn_hist = accuracy_score(y_test, y_pred_knn_hist)

print(f'Precision (k-NN, histogram): {precision_knn_hist}')
print(f'Recall (k-NN, histogram): {recall_knn_hist}')
print(f'F1-Score (k-NN, histogram): {f1_knn_hist}')
print(f'Accuracy (k-NN, histogram): {accuracy_knn_hist}')
print(f'Training Time (k-NN, histogram): {knn_hist_training_time} seconds')
print(f'Inference Time (k-NN, histogram): {knn_hist_inference_time} seconds')

knn_gabor = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
knn_gabor.fit(x_train_gabor, y_train.ravel())
knn_gabor_training_time = time.time() - start_time

val_accuracy_knn_gabor = knn_gabor.score(x_val_gabor, y_val.ravel())
print(f'Validation Accuracy (k-NN, Gabor): {val_accuracy_knn_gabor}')

start_time = time.time()
y_pred_knn_gabor = knn_gabor.predict(x_test_gabor)
knn_gabor_inference_time = time.time() - start_time

print(classification_report(y_test, y_pred_knn_gabor, target_names=[str(i) for i in range(10)]))
precision_knn_gabor = precision_score(y_test, y_pred_knn_gabor, average='macro')
```

```
recall_knn_gabor = recall_score(y_test, y_pred_knn_gabor, average='macro')
f1_knn_gabor = f1_score(y_test, y_pred_knn_gabor, average='macro')
accuracy_knn_gabor = accuracy_score(y_test, y_pred_knn_gabor)

print(f'Precision (k-NN, Gabor): {precision_knn_gabor}')
print(f'Recall (k-NN, Gabor): {recall_knn_gabor}')
print(f'F1-Score (k-NN, Gabor): {f1_knn_gabor}')
print(f'Accuracy (k-NN, Gabor): {accuracy_knn_gabor}')
print(f'Training Time (k-NN, Gabor): {knn_gabor_training_time} seconds')
print(f'Inference Time (k-NN, Gabor): {knn_gabor_inference_time} seconds')

knn_edges = KNeighborsClassifier(n_neighbors=3)
start_time = time.time()
knn_edges.fit(x_train_edges, y_train.ravel())
knn_edges_training_time = time.time() - start_time

val_accuracy_knn_edges = knn_edges.score(x_val_edges, y_val.ravel())
print(f'Validation Accuracy (k-NN, edges): {val_accuracy_knn_edges}')

start_time = time.time()
y_pred_knn_edges = knn_edges.predict(x_test_edges)
knn_edges_inference_time = time.time() - start_time

print(classification_report(y_test, y_pred_knn_edges, target_names=[str(i) for i in range(10)]))
precision_knn_edges = precision_score(y_test, y_pred_knn_edges, average='macro')
recall_knn_edges = recall_score(y_test, y_pred_knn_edges, average='macro')
f1_knn_edges = f1_score(y_test, y_pred_knn_edges, average='macro')
accuracy_knn_edges = accuracy_score(y_test, y_pred_knn_edges)

print(f'Precision (k-NN, edges): {precision_knn_edges}')
print(f'Recall (k-NN, edges): {recall_knn_edges}')
print(f'F1-Score (k-NN, edges): {f1_knn_edges}')
print(f'Accuracy (k-NN, edges): {accuracy_knn_edges}')
print(f'Training Time (k-NN, edges): {knn_edges_training_time} seconds')
print(f'Inference Time (k-NN, edges): {knn_edges_inference_time} seconds')

kmeans_hist = KMeans(n_clusters=10, random_state=42)
start_time = time.time()
kmeans_hist.fit(x_train_hist)
kmeans_hist_training_time = time.time() - start_time

start_time = time.time()
y_pred_kmeans_hist = kmeans_hist.predict(x_test_hist)
kmeans_hist_inference_time = time.time() - start_time

cluster_labels_hist = np.zeros_like(y_test)
for i in range(10):
```

```
    mask = (y_pred_kmeans_hist == i)
    if mask.sum() > 0:
        cluster_labels_hist[mask] = np.bincount(y_test[mask].flatten()).argmax()

print(classification_report(y_test, cluster_labels_hist, target_names=[str(i) for i in range(10)]))
precision_kmeans_hist = precision_score(y_test, cluster_labels_hist, average='macro')
recall_kmeans_hist = recall_score(y_test, cluster_labels_hist, average='macro')
f1_kmeans_hist = f1_score(y_test, cluster_labels_hist, average='macro')
accuracy_kmeans_hist = accuracy_score(y_test, cluster_labels_hist)

print(f'Precision (k-means, histogram): {precision_kmeans_hist}')
print(f'Recall (k-means, histogram): {recall_kmeans_hist}')
print(f'F1-Score (k-means, histogram): {f1_kmeans_hist}')
print(f'Accuracy (k-means, histogram): {accuracy_kmeans_hist}')
print(f'Training Time (k-means, histogram): {kmeans_hist_training_time} seconds')
print(f'Inference Time (k-means, histogram): {kmeans_hist_inference_time} seconds')

kmeans_gabor = KMeans(n_clusters=10, random_state=42)
start_time = time.time()
kmeans_gabor.fit(x_train_gabor)
kmeans_gabor_training_time = time.time() - start_time

start_time = time.time()
y_pred_kmeans_gabor = kmeans_gabor.predict(x_test_gabor)
kmeans_gabor_inference_time = time.time() - start_time

cluster_labels_gabor = np.zeros_like(y_test)
for i in range(10):
    mask = (y_pred_kmeans_gabor == i)
    if mask.sum() > 0:
        cluster_labels_gabor[mask] = np.bincount(y_test

[mask].flatten()).argmax()

print(classification_report(y_test, cluster_labels_gabor, target_names=[str(i) for i in range(10)]))
precision_kmeans_gabor = precision_score(y_test, cluster_labels_gabor, average='macro')
recall_kmeans_gabor = recall_score(y_test, cluster_labels_gabor, average='macro')
f1_kmeans_gabor = f1_score(y_test, cluster_labels_gabor, average='macro')
accuracy_kmeans_gabor = accuracy_score(y_test, cluster_labels_gabor)

print(f'Precision (k-means, Gabor): {precision_kmeans_gabor}')
print(f'Recall (k-means, Gabor): {recall_kmeans_gabor}')
print(f'F1-Score (k-means, Gabor): {f1_kmeans_gabor}')
print(f'Accuracy (k-means, Gabor): {accuracy_kmeans_gabor}')
print(f'Training Time (k-means, Gabor): {kmeans_gabor_training_time} seconds')
print(f'Inference Time (k-means, Gabor): {kmeans_gabor_inference_time} seconds')
```

```
kmeans_edges = KMeans(n_clusters=10, random_state=42)
start_time = time.time()
kmeans_edges.fit(x_train_edges)
kmeans_edges_training_time = time.time() - start_time

start_time = time.time()
y_pred_kmeans_edges = kmeans_edges.predict(x_test_edges)
kmeans_edges_inference_time = time.time() - start_time

cluster_labels_edges = np.zeros_like(y_test)
for i in range(10):
    mask = (y_pred_kmeans_edges == i)
    if mask.sum() > 0:
        cluster_labels_edges[mask] = np.bincount(y_test[mask].flatten()).argmax()

print(classification_report(y_test, cluster_labels_edges, target_names=[str(i) for i in range(10)]))
precision_kmeans_edges = precision_score(y_test, cluster_labels_edges, average='macro')
recall_kmeans_edges = recall_score(y_test, cluster_labels_edges, average='macro')
f1_kmeans_edges = f1_score(y_test, cluster_labels_edges, average='macro')
accuracy_kmeans_edges = accuracy_score(y_test, cluster_labels_edges)

print(f'Precision (k-means, edges): {precision_kmeans_edges}')
print(f'Recall (k-means, edges): {recall_kmeans_edges}')
print(f'F1-Score (k-means, edges): {f1_kmeans_edges}')
print(f'Accuracy (k-means, edges): {accuracy_kmeans_edges}')
print(f'Training Time (k-means, edges): {kmeans_edges_training_time} seconds')
print(f'Inference Time (k-means, edges): {kmeans_edges_inference_time} seconds')

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

x = base_model.output
x = Flatten()(x)
predictions = Dense(10, activation='softmax')(x)

model_resnet = Model(inputs=base_model.input, outputs=predictions)
model_resnet.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

resnet_model_path = "resnet_model.h5"
if os.path.exists(resnet_model_path):
    model_resnet = load_model(resnet_model_path)
    resnet_training_time = 0
else:
    start_time = time.time()
    model_resnet.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size=32)
    resnet_training_time = time.time() - start_time
    model_resnet.save(resnet_model_path)
```

```
start_time = time.time()
y_pred_resnet = model_resnet.predict(x_test)
resnet_inference_time = time.time() - start_time
y_pred_resnet_classes = np.argmax(y_pred_resnet, axis=1)

print(classification_report(y_test, y_pred_resnet_classes, target_names=[str(i) for i in range(10)]))
precision_resnet = precision_score(y_test, y_pred_resnet_classes, average='macro')
recall_resnet = recall_score(y_test, y_pred_resnet_classes, average='macro')
f1_resnet = f1_score(y_test, y_pred_resnet_classes, average='macro')
accuracy_resnet = accuracy_score(y_test, y_pred_resnet_classes)

print(f'Precision (ResNet): {precision_resnet}')
print(f'Recall (ResNet): {recall_resnet}')
print(f'F1-Score (ResNet): {f1_resnet}')
print(f'Accuracy (ResNet): {accuracy_resnet}')
print(f'Training Time (ResNet): {resnet_training_time} seconds')
print(f'Inference Time (ResNet): {resnet_inference_time} seconds')

x_train_edges_cnn = x_train_edges.reshape(x_train_edges.shape[0], 32, 32, 1)
x_val_edges_cnn = x_val_edges.reshape(x_val_edges.shape[0], 32, 32, 1)
x_test_edges_cnn = x_test_edges.reshape(x_test_edges.shape[0], 32, 32, 1)

model_cnn = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model_cnn.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

cnn_model_path = "cnn_model.h5"
if os.path.exists(cnn_model_path):
    model_cnn = load_model(cnn_model_path)
    cnn_training_time = 0
else:
    start_time = time.time()
    model_cnn.fit(x_train_edges_cnn, y_train, validation_data=(x_val_edges_cnn, y_val), epochs=10,
batch_size=32)
    cnn_training_time = time.time() - start_time
    model_cnn.save(cnn_model_path)

start_time = time.time()
y_pred_cnn = model_cnn.predict(x_test_edges_cnn)
```

```
cnn_inference_time = time.time() - start_time
y_pred_cnn_classes = np.argmax(y_pred_cnn, axis=1)

print(classification_report(y_test, y_pred_cnn_classes, target_names=[str(i) for i in range(10)]))
precision_cnn = precision_score(y_test, y_pred_cnn_classes, average='macro')
recall_cnn = recall_score(y_test, y_pred_cnn_classes, average='macro')
f1_cnn = f1_score(y_test, y_pred_cnn_classes, average='macro')
accuracy_cnn = accuracy_score(y_test, y_pred_cnn_classes)

print(f'Precision (CNN on edges): {precision_cnn}')
print(f'Recall (CNN on edges): {recall_cnn}')
print(f'F1-Score (CNN on edges): {f1_cnn}')
print(f'Accuracy (CNN on edges): {accuracy_cnn}')
print(f'Training Time (CNN on edges): {cnn_training_time} seconds')
print(f'Inference Time (CNN on edges): {cnn_inference_time} seconds')

def create_alexnet_model(input_shape, num_classes):
    model = Sequential()
    model.add(Conv2D(96, (11, 11), strides=(4, 4), activation='relu', padding='same', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))
    model.add(Conv2D(256, (5, 5), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))
    model.add(Conv2D(384, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(384, (3, 3), padding='same', activation='relu'))
    model.add(Conv2D(256, (3, 3), padding='same', activation='relu'))
    model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same'))
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax'))
    return model

alexnet_model = create_alexnet_model((32, 32, 3), 10)
alexnet_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

alexnet_model_path = "alexnet_model.h5"
if os.path.exists(alexnet_model_path):
    alexnet_model = load_model(alexnet_model_path)
    alexnet_training_time = 0
else:
    start_time = time.time()
    alexnet_model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size=32)
    alexnet_training_time = time.time() - start_time
    alexnet_model.save(alexnet_model_path)
```

```
start

_time = time.time()
y_pred_alexnet = alexnet_model.predict(x_test)
alexnet_inference_time = time.time() - start_time
y_pred_alexnet_classes = np.argmax(y_pred_alexnet, axis=1)

print(classification_report(y_test, y_pred_alexnet_classes, target_names=[str(i) for i in range(10)]))
precision_alexnet = precision_score(y_test, y_pred_alexnet_classes, average='macro')
recall_alexnet = recall_score(y_test, y_pred_alexnet_classes, average='macro')
f1_alexnet = f1_score(y_test, y_pred_alexnet_classes, average='macro')
accuracy_alexnet = accuracy_score(y_test, y_pred_alexnet_classes)

print(f'Precision (AlexNet): {precision_alexnet}')
print(f'Recall (AlexNet): {recall_alexnet}')
print(f'F1-Score (AlexNet): {f1_alexnet}')
print(f'Accuracy (AlexNet): {accuracy_alexnet}')
print(f'Training Time (AlexNet): {alexnet_training_time} seconds')
print(f'Inference Time (AlexNet): {alexnet_inference_time} seconds')

def inception_module(x, filters):
    branch1 = Conv2D(filters[0], (1, 1), activation='relu', padding='same')(x)
    branch2 = Conv2D(filters[1], (1, 1), activation='relu', padding='same')(x)
    branch2 = Conv2D(filters[2], (3, 3), activation='relu', padding='same')(branch2)
    branch3 = Conv2D(filters[3], (1, 1), activation='relu', padding='same')(x)
    branch3 = Conv2D(filters[4], (5, 5), activation='relu', padding='same')(branch3)
    branch4 = MaxPooling2D((3, 3), strides=(1, 1), padding='same')(x)
    branch4 = Conv2D(filters[5], (1, 1), activation='relu', padding='same')(branch4)
    output = concatenate([branch1, branch2, branch3, branch4], axis=-1)
    return output

def create_inception_model(input_shape, num_classes):
    input_img = Input(shape=input_shape)
    x = Conv2D(64, (7, 7), activation='relu', padding='same')(input_img)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = Conv2D(192, (3, 3), activation='relu', padding='same')(x)
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = inception_module(x, [64, 96, 128, 16, 32, 32])
    x = inception_module(x, [128, 128, 192, 32, 96, 64])
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = inception_module(x, [192, 96, 208, 16, 48, 64])
    x = inception_module(x, [160, 112, 224, 24, 64, 64])
    x = inception_module(x, [128, 128, 256, 24, 64, 64])
    x = inception_module(x, [112, 144, 288, 32, 64, 64])
    x = inception_module(x, [256, 160, 320, 32, 128, 128])
    x = MaxPooling2D((3, 3), strides=(2, 2), padding='same')(x)
    x = inception_module(x, [256, 160, 320, 32, 128, 128])
```

```
    x = inception_module(x, [384, 192, 384, 48, 128, 128])
    x = AveragePooling2D((2, 2), strides=(1, 1))(x)
    x = Dropout(0.4)(x)
    x = Flatten()(x)
    x = Dense(num_classes, activation='softmax')(x)
    model = Model(inputs=input_img, outputs=x)
    return model

inception_model = create_inception_model((32, 32, 3), 10)
inception_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

inception_model_path = "inception_model.h5"
if os.path.exists(inception_model_path):
    inception_model = load_model(inception_model_path)
    inception_training_time = 0
else:
    start_time = time.time()
    inception_model.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=10, batch_size=32)
    inception_training_time = time.time() - start_time
    inception_model.save(inception_model_path)

start_time = time.time()
y_pred_inception = inception_model.predict(x_test)
inception_inference_time = time.time() - start_time
y_pred_inception_classes = np.argmax(y_pred_inception, axis=1)

print(classification_report(y_test, y_pred_inception_classes, target_names=[str(i) for i in range(10)]))
precision_inception = precision_score(y_test, y_pred_inception_classes, average='macro')
recall_inception = recall_score(y_test, y_pred_inception_classes, average='macro')
f1_inception = f1_score(y_test, y_pred_inception_classes, average='macro')
accuracy_inception = accuracy_score(y_test, y_pred_inception_classes)

print(f'Precision (Inception): {precision_inception}')
print(f'Recall (Inception): {recall_inception}')
print(f'F1-Score (Inception): {f1_inception}')
print(f'Accuracy (Inception): {accuracy_inception}')
print(f'Training Time (Inception): {inception_training_time} seconds')
print(f'Inference Time (Inception): {inception_inference_time} seconds')

x_train_combined = np.concatenate([x_train_hist, x_train_gabor, x_train_edges], axis=1)
x_val_combined = np.concatenate([x_val_hist, x_val_gabor, x_val_edges], axis=1)
x_test_combined = np.concatenate([x_test_hist, x_test_gabor, x_test_edges], axis=1)

input_shape = (x_train_combined.shape[1],)
inputs = Input(shape=input_shape)
x = Dense(512, activation='relu')(inputs)
x = Dropout(0.5)(x)
```

```
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(10, activation='softmax')(x)

combined_model = Model(inputs=inputs, outputs=outputs)
combined_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

combined_model_path = "combined_model.h5"
if os.path.exists(combined_model_path):
    combined_model = load_model(combined_model_path)
    combined_training_time = 0
else:
    start_time = time.time()
    combined_model.fit(x_train_combined, y_train, validation_data=(x_val_combined, y_val), epochs=10,
batch_size=32)
    combined_training_time = time.time() - start_time
    combined_model.save(combined_model_path)

start_time = time.time()
y_pred_combined = combined_model.predict(x_test_combined)
combined_inference_time = time.time() - start_time
y_pred_combined_classes = np.argmax(y_pred_combined, axis=1)

print(classification_report(y_test, y_pred_combined_classes, target_names=[str(i) for i in range(10)]))
precision_combined = precision_score(y_test, y_pred_combined_classes, average='macro')
recall_combined = recall_score(y_test, y_pred_combined_classes, average='macro')
f1_combined = f1_score(y_test, y_pred_combined_classes, average='macro')
accuracy_combined = accuracy_score(y_test, y_pred_combined_classes)

print(f'Precision (Combined): {precision_combined}')
print(f'Recall (Combined): {recall_combined}')
print(f'F1-Score (Combined): {f1_combined}')
print(f'Accuracy (Combined): {accuracy_combined}')
print(f'Training Time (Combined): {combined_training_time} seconds')
print(f'Inference Time (Combined): {combined_inference_time} seconds')

models = {
    "k-NN (histogram)": {
        "precision": precision_knn_hist,
        "recall": recall_knn_hist,
        "f1_score": f1_knn_hist,
        "accuracy": accuracy_knn_hist,
        "training_time": knn_hist_training_time,
        "inference_time": knn_hist_inference_time
    },
```

```
  "k-NN (Gabor)": {
     "precision": precision_knn_gabor,
     "recall": recall_knn_gabor,
     "f1_score": f1_knn_gabor,
     "accuracy": accuracy_knn_gabor,
     "training_time": knn_gabor_training_time,
     "inference_time": knn_gabor_inference_time
  },
  "k-NN (edges)":

{
     "precision": precision_knn_edges,
     "recall": recall_knn_edges,
     "f1_score": f1_knn_edges,
     "accuracy": accuracy_knn_edges,
     "training_time": knn_edges_training_time,
     "inference_time": knn_edges_inference_time
  },
  "k-means (histogram)": {
     "precision": precision_kmeans_hist,
     "recall": recall_kmeans_hist,
     "f1_score": f1_kmeans_hist,
     "accuracy": accuracy_kmeans_hist,
     "training_time": kmeans_hist_training_time,
     "inference_time": kmeans_hist_inference_time
  },
  "k-means (Gabor)": {
     "precision": precision_kmeans_gabor,
     "recall": recall_kmeans_gabor,
     "f1_score": f1_kmeans_gabor,
     "accuracy": accuracy_kmeans_gabor,
     "training_time": kmeans_gabor_training_time,
     "inference_time": kmeans_gabor_inference_time
  },
  "k-means (edges)": {
     "precision": precision_kmeans_edges,
     "recall": recall_kmeans_edges,
     "f1_score": f1_kmeans_edges,
     "accuracy": accuracy_kmeans_edges,
     "training_time": kmeans_edges_training_time,
     "inference_time": kmeans_edges_inference_time
  },
  "ResNet": {
     "precision": precision_resnet,
     "recall": recall_resnet,
     "f1_score": f1_resnet,
     "accuracy": accuracy_resnet,
```

```
        "training_time": resnet_training_time,
        "inference_time": resnet_inference_time
    },
    "CNN on edges": {
        "precision": precision_cnn,
        "recall": recall_cnn,
        "f1_score": f1_cnn,
        "accuracy": accuracy_cnn,
        "training_time": cnn_training_time,
        "inference_time": cnn_inference_time
    },
    "AlexNet": {
        "precision": precision_alexnet,
        "recall": recall_alexnet,
        "f1_score": f1_alexnet,
        "accuracy": accuracy_alexnet,
        "training_time": alexnet_training_time,
        "inference_time": alexnet_inference_time
    },
    "Inception": {
        "precision": precision_inception,
        "recall": recall_inception,
        "f1_score": f1_inception,
        "accuracy": accuracy_inception,
        "training_time": inception_training_time,
        "inference_time": inception_inference_time
    },
    "Combined Model": {
        "precision": precision_combined,
        "recall": recall_combined,
        "f1_score": f1_combined,
        "accuracy": accuracy_combined,
        "training_time": combined_training_time,
        "inference_time": combined_inference_time
    }
}

for model_name, metrics in models.items():
    print(f"\nModel: {model_name}")
    for metric_name, value in metrics.items():
        print(f"{metric_name}: {value}")
```

ДОДАТОК В

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	96 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Настенко С.В.			Дослідження моделей і алгоритмів пошуку зображень у сховищах великих даних	Шифр групи	Код напр./спец.
Перевірив		Смеляков К.С.				СШІм-22-1	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					