Tank classification from RGB images

Introduction:

Modern warfare is gradually becoming more and more reliant on unmanned weapon systems. However, new technologies create new kinds of threats, where, without human supervision, the hazard of friendly fire is more acute than ever before. This project aims to address this problem by developing a model able to differentiate between different tanks from often sparse image information, allowing the unmanned systems to reliably avoid eliminating enemy targets while safeguarding ally troops. Furthermore, not every kind of weapon is capable of engaging heavy armoured targets, and vice versa, not every target is worth shooting down with sophisticated and expensive weapon systems. That is why our project can be instrumental in autonomous decision-making on the modern battlefield. In this report, we will first formulate a machine learning problem and describe the dataset we will use during the analysis. Then, the choice of models and loss functions will be discussed. Subsequently, we will present the validation strategy and performance evaluation methods. Next, the results of the models will be shown, and the choice of the final model will be justified. Finally, we will summarise the findings of this project in a conclusion paragraph.

Problem formulation:

This project aims to classify military tanks by their model based on their RGB image. For this purpose, we use a CNN (convolutional neural network) model. The dataset consists of images (continuous data type) scraped from multiple publicly available datasets we found on universe.roboflow and kaggle(links in the appendix). It was subsequently filtered to delete unwanted images (either showing wrong tanks or not showing tanks at all), labelled and transformed into a unified format. In total, we operate on 7982 pictures, of which 3244 that fit our requirements are classified into 3 groups (labels) - American, Israeli, and Russian. Originally, images had various resolutions, so we unified them to 256x256. Therefore, we examine 65536 features (each RGB pixel of the image) and 3244 datapoints (RGB images). This formulates a supervised machine learning task where we label the tank models based on their RGB image.

Methods:

Dataset and feature selection:

We used 8 datasets to collect the data and got 7982 images. First, we unified the pictures' labelling using CSV files that were part of the former dataset (filename, vehicle name). Then, we discarded the tanks that do not fit our requirements, i.e. are neither from the USA nor Russia nor Israel. After this processing, we end up with 3244 labelled images, which serve as datapoints, of which 1752 are Israeli, 546 are American, and 946 are Russian. One of the

problems we encountered was the difference in image resolutions. We unified them to 256x256 using the Pillow Python library.

Choice of model:

Model number 1:
We will use the CNN (convoluted neural network) model because it is able to explore the spatial structure of the image and recognise image-specific patterns, like textures and edges, reducing the number of features. Because of the small dataset size, we opt for a transfer learning approach. We will use a backbone pretrained on ImageNet and then use fine-tuning instead of a feature extractor for more accuracy. We will unfreeze a large part of the network and train end-to-end with a low training rate to conserve pretrained weights. This way, we do not need to train the model to recognise basic image features (edges, textures, shapes), which reduces overfitting with small datasets. For optimiser, we choose AdamW because of fast convergence and no need for tuning.

Model number 2:
Random Forest was chosen as an alternative classifier to replace the CNN's final fully connected layer in order to evaluate how a non-parametric ensemble learner performs on deep visual embeddings. The CNN acts as a feature extractor, converting each tank image into a compact numerical representation, while the Random Forest classifies these embeddings based on non-linear decision boundaries. This hybrid design allows us to use the CNN's strength in feature learning and the Random Forest's resistance to overfitting on medium-sized datasets. It also provides a meaningful comparison between deep end-to-end training and classical machine learning approaches using learned features.

Loss function:

For the loss function (the same for both models), we choose cross-entropy. We opt for cross-entropy because of its simplicity and easy implementation. We were also considering testing label-smoothing as cross-entropy tends to be overconfident and weights as the dataset is not equally distributed; however, the results of the model were good enough (96% on validation, 94% test) without them. Nonetheless, we recognise that there might be a slight benefit in implementing those changes to the loss function. Other choices, like focal loss or ArcFace, were considered, but in this case, we do not see the need for them as the differences in tanks seem to be big enough for cross-entropy to suffice.

Validation process:

We split the data into three batches: 70% training, 15% validation, 15% testing, while maintaining the original split between classes in all sets. The test set remained unseen during model training and was used exclusively for final performance evaluation. We chose to split the data this way because 15% for validation and training is sufficient to check the model's accuracy while leaving enough data for the model not to be overfit. During the training, we will compute validation loss, accuracy and confusion matrix to check the performance of the model.

Performance Evaluation Measures:

The test set remained unseen during model training and was used exclusively for final performance evaluation. To assess the performance of both methods, we use 4 metrics: Accuracy, Precision, Recall and F1-score. Accuracy measures the correctness of all predictions. Precision measures the correctness of positive predictions. Recall measures how many actual positives the model correctly identified. The F1-score is a metric that combines precision and recall into a single value to give a balanced measure of a model's performance. We will use those methods for all classes of tanks.

Results:

Method 1: CNN
While using CNN, we obtained a very good result of 96.98% accuracy on the validation set and 94.16% accuracy on the training set. Other measures for different tank classes look as follows:

Validation set

	Precision	Recall	F1-score
American	0.96	0.89	0.92
Russian	0.95	0.98	0.97
Israeli	0.98	0.98	0.98

Test set

	Precision	Recall	F1-score
American	1.0	0.62	0.77
Russian	0.90	0.98	0.94
Israeli	0.96	0.99	0.97

We notice that the performance on the test set is slightly worse than on the validation set, which was highly expected and is a result of the model's overfitting to the validation set. We also notice that the model's worst performance is on American tanks, which was also expected, as the amount of data on those tanks is the lowest. The model performs exceptionally well on Israeli tanks, which is likely due to the large proportion of Israeli tanks in the dataset. Overall result of the model is, however, very satisfactory.

Method 2: Random Forest
While using Random Forest, we obtained an accuracy of 86.7%, which is a good result but significantly worse than the one given by the CNN. The testing measures look as follows:

	Precision	Recall	F1-score
American	0.72	0.38	0.49
Russian	0.80	0.99	0.88
Israeli	0.93	0.91	0.92

Those results show a good performance on Israeli and Russian tanks but a very bad performance on American tanks. This is most likely a result of too few American tanks in the dataset.

The results overwhelmingly show that CNN is more accurate in predicting tanks than Random Forest, especially the American ones, and therefore, we opt to choose CNN as the final model.

Conclusions:

This study has demonstrated the effectiveness of convolutional neural networks (CNNs) in classifying armoured vehicles by country of origin using images. By employing transfer learning and fine-tuning on a medium-sized, domain-specific dataset, the CNN achieved a validation accuracy of 96.98% and a test accuracy of 94.16%, thereby outperforming the hybrid CNN–Random Forest model, which achieved 86.7%. These findings show the CNN's superior capacity to extract and generalise spatial and textural representations pertinent to tank classification tasks, even in the presence of inter-class similarity and moderate data limitations.

Nevertheless, the study identified specific constraints. The classification of American tanks showed reduced accuracy, caused by the dataset imbalance and limited sample diversity. Furthermore, minor signs of overfitting were observed, indicating potential for enhanced generalisation through additional regularisation or improved augmentation techniques. Expanding the dataset to encompass a broader range of models and environmental conditions would also be beneficial for improving model robustness. Weighted Cross-Entropy can be implemented as a form of reducing the impacts of data imbalance.

In conclusion, the research validates CNN-based methodologies as a robust and scalable solution for military vehicle recognition tasks, capable of achieving high discriminative performance in complex visual domains. Future work may investigate the integration of advanced architectures such as Vision Transformers (ViTs) or hybrid CNN-transformer models to evaluate their efficacy in capturing long-range contextual dependencies within military assets.

AI statement:

AI was used only for research purposes, i.e. to dive deeper into strategies of training models, as it feels outside of the scope of this course.

Appendix:

1.https://universe.roboflow.com/mo-ewp9c/t90
2.https://universe.roboflow.com/abrams/abrams
3.https://www.kaggle.com/datasets/antoreepjana/military-tanks-dataset-images
4.https://universe.roboflow.com/garage-rjbkw/tanks-detection-2/browse?queryText=&pageSize=50&startingIndex=50&browseQuery=true
5.https://universe.roboflow.com/travma/all_in_one-etk0n/images/WuKPMFzRtINXccsR3PsB?queryText=&pageSize=50&startingIndex=0&browseQuery=true
6.https://universe.roboflow.com/makinerenme/tank-efm9r
7.https://universe.roboflow.com/set-of-armored-vehicle/seg_test-fycxw
8.https://universe.roboflow.com/idan-mknth/merkava/browse?queryText=&pageSize=50&startingIndex=50&browseQuery=true
9.https://docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html

This code merges the datasets and reshapes the images:

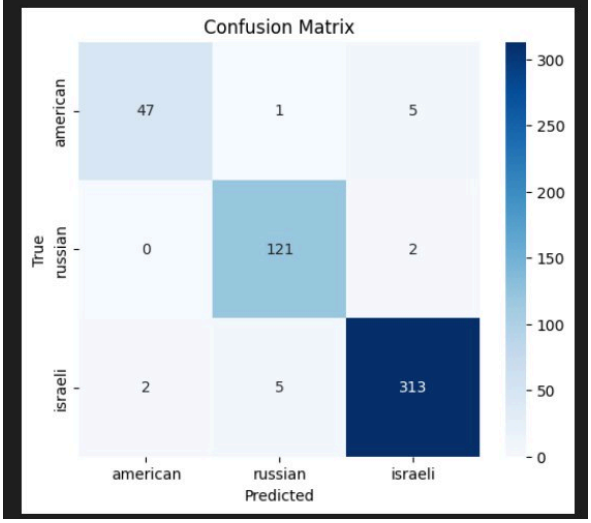
```python
import os
from PIL import Image
import pandas as pd
import random, shutil, re
from collections import Counter
source_dir = "."
target_dir = "data_combined"

splits = ["train", "test", "valid"]
exclude_folder = ["archive (1)","data_combined"]

for split in splits:
    split_dir = os.path.join(target_dir, split)
    os.makedirs(split_dir, exist_ok=True)
counter = 0
for class_name in os.listdir(source_dir):
    if class_name in exclude_folder or not os.path.isdir(class_name):
        continue

    class_path = os.path.join(source_dir, class_name)
    for split in splits:
        split_path = os.path.join(class_path, split)
        if not os.path.exists(split_path):
            continue
        label_file = None

        for f in os.listdir(split_path):
            if f.lower().endswith(".csv"):
                label_file = os.path.join(split_path, f)
                break

        if label_file:
            labe_pd = pd.read_csv(label_file)
        else:
            labe_pd = pd.DataFrame(columns=["filename", "class"])
        for root, dirs, files in os.walk(split_path):
            for file in files:
                if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".gif")):
                    src = os.path.join(root, file)
                    matching = labe_pd.loc[labe_pd["filename"] == file, "class"]
                    if not matching.empty:
                        new_name = matching.values[0]
                        ext = os.path.splitext(file)[1]
                        print(new_name)
                        new_filename = f"{new_name}{ext}"
                    else:
                        new_filename = file
                        print(f"File {file} not found in df")
                    counter += 1
                    dst = os.path.join(target_dir, split, f"{counter}_{new_filename}")
                    img = Image.open(src)
                    img = img.convert("RGB")
                    img = img.resize((256, 256))
                    img.save(dst)
```

This code moves russian tanks from val and test datasets to the train dataset to conserve the composition of Israeli, US and Russian tanks stays similar among all the datasets. It also counts the number of tanks in each class:

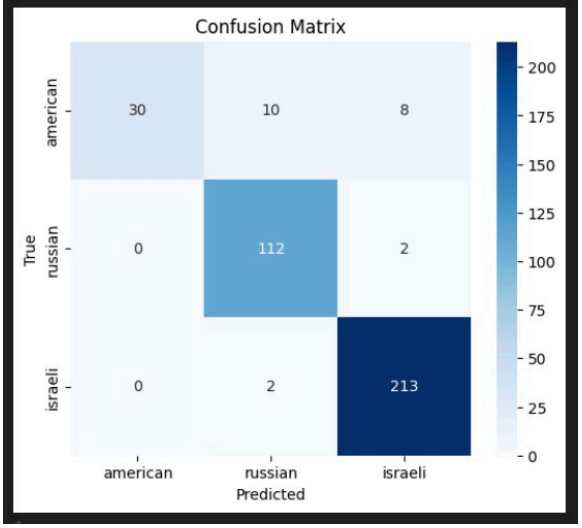
```python
import os
from PIL import Image
import pandas as pd
import random, shutil, re
from collections import Counter
source_dir = "."
target_dir = "data_combined"

splits = ["train", "test", "valid"]
exclude_folder = ["archive (1)","data_combined"]

for split in splits:
    split_dir = os.path.join(target_dir, split)
    os.makedirs(split_dir, exist_ok=True)
counter = 0
for class_name in os.listdir(source_dir):
    if class_name in exclude_folder or not os.path.isdir(class_name):
        continue

    class_path = os.path.join(source_dir, class_name)
    for split in splits:
        split_path = os.path.join(class_path, split)
        if not os.path.exists(split_path):
            continue
        label_file = None

        for f in os.listdir(split_path):
            if f.lower().endswith(".csv"):
                label_file = os.path.join(split_path, f)
                break

        if label_file:
            labe_pd = pd.read_csv(label_file)
        else:
            labe_pd = pd.DataFrame(columns=["filename", "class"])
        for root, dirs, files in os.walk(split_path):
            for file in files:
                if file.lower().endswith((".jpg", ".jpeg", ".png", ".bmp", ".gif")):
                    src = os.path.join(root, file)
                    matching = labe_pd.loc[labe_pd["filename"] == file, "class"]
                    if not matching.empty:
                        new_name = matching.values[0]
                        ext = os.path.splitext(file)[1]
                        print(new_name)
                        new_filename = f"{new_name}{ext}"
                    else:
                        new_filename = file
                        print(f"File {file} not found in df")
                    counter += 1
                    dst = os.path.join(target_dir, split, f"{counter}_{new_filename}")
                    img = Image.open(src)
                    img = img.convert("RGB")
                    img = img.resize((256, 256))
                    img.save(dst)
```

Performance of the CNN model:

```
Validation set performance:
Accuracy: 0.9698
Classification Report:
              precision    recall  f1-score   support

    american       0.96      0.89      0.92        53
     russian       0.95      0.98      0.97       123
     israeli       0.98      0.98      0.98       320

    accuracy                           0.97       496
   macro avg       0.96      0.95      0.96       496
weighted avg       0.97      0.97      0.97       496
```

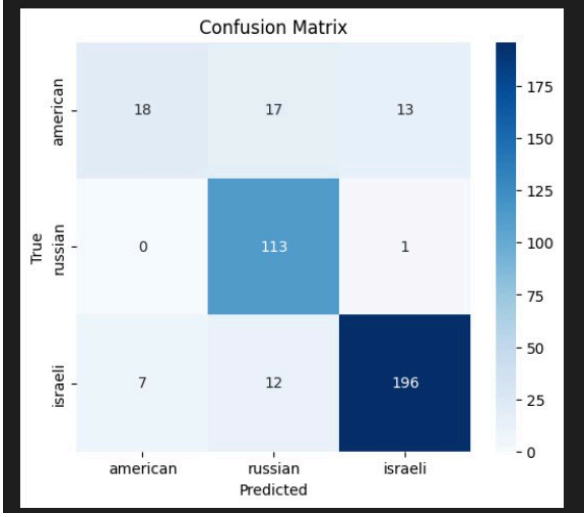



```
Test set performance:
Accuracy: 0.9416
Classification Report:
              precision    recall  f1-score   support

    american       1.00      0.62      0.77        48
     russian       0.90      0.98      0.94       114
     israeli       0.96      0.99      0.97       215

    accuracy                           0.94       377
   macro avg       0.95      0.87      0.89       377
weighted avg       0.95      0.94      0.94       377
```



Performance of the RandomForest:

```
Random Forest Test Accuracy: 86.74%

Classification Report:
              precision    recall  f1-score   support

    american       0.72      0.38      0.49        48
     russian       0.80      0.99      0.88       114
     israeli       0.93      0.91      0.92       215

    accuracy                           0.87       377
   macro avg       0.82      0.76      0.77       377
weighted avg       0.86      0.87      0.86       377
```



```python
import os
import torch
from torchvision.models import resnet50, ResNet50_Weights
from torchvision import transforms
from PIL import Image
import torch.nn as nn
import torch.optim as optim
import warnings
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from torchvision import models
import seaborn as sns
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
print(torch.__version__)
print(torch.version.cuda)
```

```python
model = resnet50(weights=ResNet50_Weights.DEFAULT)
num_classes = 3  #3 classes (american, russian, israeli)
model.fc = nn.Linear(model.fc.in_features, num_classes)

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

optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()
```

```python
image_dir = "data_combined/train"
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])
```

```python
images, labels = [], []
for fname in os.listdir(image_dir):
    if not fname.endswith(".jpg"):
        continue
    try:
        image = transform(Image.open(os.path.join(image_dir, fname)).convert("RGB"))
        images.append(image)
        label_str = fname.split("_")[0]
        if label_str == "american":
            label = 0
        elif label_str == "russian":
            label = 1
        else:
            label = 2
        labels.append(label)
    except Exception as e:
        print(f"Skipping {fname}: {e}")

images = torch.stack(images).to(device)
labels = torch.tensor(labels).to(device)
```

```
dataset = TensorDataset(images, labels)
batch_size = 16

loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
torch.cuda.empty_cache()
from tqdm import tqdm

model.train()
for epoch in range(5):
    batch_iter = tqdm(loader, desc=f"Epoch {epoch+1}", leave=False)
    for batch_images, batch_labels in batch_iter:
        optimizer.zero_grad()
        outputs = model(batch_images)
        loss = criterion(outputs, batch_labels)
        loss.backward()
        optimizer.step()
        batch_iter.set_postfix(loss=loss.item())
    print(f"Epoch {epoch+1}\n")
```

```
import os
import torch
from torchvision.models import resnet50, ResNet50_Weights
from torchvision import transforms
from PIL import Image
import torch.nn as nn
import torch.optim as optim
import warnings
from torch.utils.data import TensorDataset, DataLoader
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
from torchvision import models
import seaborn as sns
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
print(torch.__version__)
print(torch.version.cuda)
```

```
model = resnet50(weights=ResNet50_Weights.DEFAULT)
num_classes = 3  #3 classes (american, russian, israeli)
model.fc = nn.Linear(model.fc.in_features, num_classes)

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

optimizer = optim.AdamW(model.parameters(), lr=1e-4, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()
```

```
image_dir = "data_combined/train"
transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])
```

```python
def load_images_and_labels(image_dir, transform):
    images, labels = [], []
    for fname in os.listdir(image_dir):
        if not fname.endswith(".jpg"):
            continue
        try:
            image = transform(Image.open(os.path.join(image_dir, fname)).convert("RGB"))
            images.append(image)
            label_str = fname.split("_")[0]
            if label_str == "american":
                label = 0
            elif label_str == "russian":
                label = 1
            else:
                label = 2
            labels.append(label)
        except Exception as e:
            print(f"Skipping {fname}: {e}")
    images = torch.stack(images)
    labels = torch.tensor(labels)
    return images, labels
images, labels = load_images_and_labels(image_dir, transform)
images = images.to(device)
labels = labels.to(device)
```

```python
dataset = TensorDataset(images, labels)
batch_size = 16

loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
torch.cuda.empty_cache()
from tqdm import tqdm

model.train()
for epoch in range(5):
    batch_iter = tqdm(loader, desc=f"Epoch {epoch+1}", leave=False)
    for batch_images, batch_labels in batch_iter:
        optimizer.zero_grad()
        outputs = model(batch_images)
        loss = criterion(outputs, batch_labels)
        loss.backward()
        optimizer.step()
        batch_iter.set_postfix(loss=loss.item())
    print(f"Epoch {epoch+1}\n")
```

```python
def evaluate(model, images, labels, device, class_names=["american", "russian", "israeli"]):
    model.eval()
    with torch.no_grad():
        outputs = model(images)
        preds = outputs.argmax(dim=1)
        accuracy = (preds == labels).float().mean().item()
    print(f"Accuracy: {accuracy:.4f}")
    cm = confusion_matrix(labels.cpu(), preds.cpu())
    print("Classification Report:\n", classification_report(labels.cpu().numpy(), preds.cpu().numpy(), target_names=class_names))

    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=class_names, yticklabels=class_names)
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix")
    plt.show()

val_images, val_labels = load_images_and_labels("data_combined/valid", transform)
test_images, test_labels = load_images_and_labels("data_combined/test", transform)

val_images, val_labels = val_images.to(device), val_labels.to(device)
test_images, test_labels = test_images.to(device), test_labels.to(device)

print("Validation set performance:")
evaluate(model, val_images, val_labels, device)

print("Test set performance:")
evaluate(model, test_images, test_labels, device)
```

```python
image_path = "merkava_1.jpg"

model.eval()

transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.ToTensor(),
])


image = Image.open(image_path).convert("RGB")
image_tensor = transform(image).unsqueeze(0).to(device)

with torch.no_grad():
    outputs = model(image_tensor)
    pred = outputs.argmax(dim=1).item()

classes = ["american", "russian", "israeli"]
print(f"Predicted class: {classes[pred]}")
```

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

resnet = models.resnet18(pretrained=True)
resnet = nn.Sequential(*list(resnet.children())[:-1])
resnet.to(device)
resnet.eval()

def extract_features(image_tensors):
    feats = []
    with torch.no_grad():
        for img in image_tensors:
            if img.ndim == 3:
                img = img.unsqueeze(0)
            img = img.to(device)
            f = resnet(img).cpu().numpy().flatten()
            feats.append(f)
    return np.array(feats)

X_train = extract_features(images)
X_val   = extract_features(val_images)
X_test  = extract_features(test_images)

y_train = labels.cpu().numpy() if torch.is_tensor(labels) else np.array(labels)
y_val   = val_labels.cpu().numpy() if torch.is_tensor(val_labels) else np.array(val_labels)
y_test  = test_labels.cpu().numpy() if torch.is_tensor(test_labels) else np.array(test_labels)

rf = RandomForestClassifier(n_estimators=200, n_jobs=-1, random_state=42, criterion='entropy')
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"Random Forest Test Accuracy: {acc*100:.2f}%\n")

cm = confusion_matrix(y_test, y_pred)
class_names = ["american", "russian", "israeli"]

print("Classification Report:\n", classification_report(y_test, y_pred, target_names=class_names))
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```

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

resnet = models.resnet50(pretrained=True)
resnet = nn.Sequential(*list(resnet.children())[:-1])
resnet.to(device)
resnet.eval()

def extract_features(image_tensors):
    feats = []
    with torch.no_grad():
        for img in image_tensors:
            if img.ndim == 3:
                img = img.unsqueeze(0)
            img = img.to(device)
            f = resnet(img).cpu().numpy().flatten()
            feats.append(f)
    return np.array(feats)

X_train = extract_features(images)
X_val   = extract_features(val_images)
X_test  = extract_features(test_images)

y_train = labels.cpu().numpy() if torch.is_tensor(labels) else np.array(labels)
y_val   = val_labels.cpu().numpy() if torch.is_tensor(val_labels) else np.array(val_labels)
y_test  = test_labels.cpu().numpy() if torch.is_tensor(test_labels) else np.array(test_labels)

rf = RandomForestClassifier(n_estimators=200, n_jobs=-1, random_state=42, criterion='entropy')
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)
acc = accuracy_score(y_test, y_pred)

print(f"Random Forest Test Accuracy: {acc*100:.2f}%\n")

cm = confusion_matrix(y_test, y_pred)
class_names = ["american", "russian", "israeli"]

print("Classification Report:\n", classification_report(y_test, y_pred, target_names=class_names))
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()
```