UHAR_Expirement_noise_robustness

October 4, 2025

1 noise robustness expirement

[ ]:
```
import os
import time
import json
import logging
import torch
import timm
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from io import BytesIO

from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from PIL import Image, ImageFilter, ImageOps
from tqdm.auto import tqdm
```

[ ]:
```
# unzip a file
!unzip "/content/archive (4).zip" -d "/content/uhar"
```

[ ]:
```
CONFIG = {
    "data_path": "/content/uhar/data/data",
    "checkpoints_dir": "/content/drive/MyDrive/uhar/checkpoints/",
    "robustness_results_dir": "/content/drive/MyDrive/uhar/robustness_results/",

    "num_classes": 40,
    "batch_size": 64,
    "image_size": 224,

    "models_to_evaluate": [
        "resnet18",
        "resnet50",
        "mobilenetv2_100",
        "efficientnet_b0",
        "vit_tiny_patch16_224",
        "swin_tiny_patch4_window7_224",
        "deit_tiny_distilled_patch16_224"
    ],

    "corruptions": {
        "gaussian_noise": {
            "func": "add_gaussian_noise",
            "severities": [5, 10, 20, 40, 60]
        },
        "salt_and_pepper": {
            "func": "add_salt_and_pepper",
            "severities": [0.005, 0.01, 0.03, 0.06, 0.1]
        },
        "jpeg_compression": {
            "func": "jpeg_compress",
            "severities": [95, 75, 50, 30, 10]
        },
        "motion_blur": {
            "func": "motion_blur",
            "severities": [2, 4, 8, 15, 25]
        },
        "occlusion": {
            "func": "occlude_patch",
            "severities": [0.05, 0.1, 0.2, 0.3, 0.4]
        },
        "rotation": {
            "func": "apply_rotation",
            "severities": [5, 15, 30, 45, 60]
        }
    }
}
```

[ ]:
```
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

os.makedirs(CONFIG["robustness_results_dir"], exist_ok=True)

logging.basicConfig(level=logging.INFO,
                    format='%(asctime)s - %(levelname)s - %(message)s',
                    handlers=[
                        logging.FileHandler("robustness_experiment.log"),
                        logging.StreamHandler()
                    ])

logging.info(f"Using device: {DEVICE}")
```

adds Gaussian noise to a PIL Image.

```
[ ]: def add_gaussian_noise(img, sigma):
         """"""
         arr = np.array(img).astype(np.float32)
         noise = np.random.normal(0, sigma, arr.shape)
         arr = np.clip(arr + noise, 0, 255).astype(np.uint8)
         return Image.fromarray(arr)
```

Adds salt and pepper noise to a PIL Image.

```
[ ]: def add_salt_and_pepper(img, prob):
         """"""
         arr = np.array(img).copy()
         h, w = arr.shape[:2]
         num_pixels = int(h * w * prob)
         for _ in range(num_pixels):
             y, x = np.random.randint(0, h), np.random.randint(0, w)
             if arr.ndim == 2: # Grayscale
                 arr[y, x] = 0 if random.random() < 0.5 else 255
             else: # RGB
                 arr[y, x, :] = 0 if random.random() < 0.5 else 255
         return Image.fromarray(arr)
```

Applies JPEG compression to a PIL Image.

```
[ ]: def jpeg_compress(img, quality):
         """"""
         buffer = BytesIO()
         img.save(buffer, format='JPEG', quality=int(quality))
         buffer.seek(0)
         return Image.open(buffer).convert(img.mode)
```

Applies a simple motion-like blur using GaussianBlur for simplicity

```
[ ]: def motion_blur(img, radius):

         return img.filter(ImageFilter.GaussianBlur(radius=radius))
```

Occludes a random patch of the image.

```
[ ]: def occlude_patch(img, frac):

         arr = np.array(img).copy()
         h, w = arr.shape[:2]
         patch_h = int(h * frac**0.5)
         patch_w = int(w * frac**0.5)
         y = np.random.randint(0, max(1, h - patch_h))
         x = np.random.randint(0, max(1, w - patch_w))
         mean_color = int(arr.mean())
```

```
    if arr.ndim == 2:
        arr[y:y+patch_h, x:x+patch_w] = mean_color
    else:
        arr[y:y+patch_h, x:x+patch_w, :] = mean_color
    return Image.fromarray(arr)
```

[ ]:
```
def apply_rotation(img, degrees):
    """Applies rotation to a PIL Image."""
    return img.rotate(degrees, resample=Image.BICUBIC, fillcolor=int(np.
 ↪mean(img)))
```

[ ]:
```
CORRUPTION_FUNCTIONS = {
    "add_gaussian_noise": add_gaussian_noise,
    "add_salt_and_pepper": add_salt_and_pepper,
    "jpeg_compress": jpeg_compress,
    "motion_blur": motion_blur,
    "occlude_patch": occlude_patch,
    "apply_rotation": apply_rotation,
}
```

[ ]:
```
def create_test_dataset_pil(config):
    """
    Creates a test dataset that returns PIL images and handles class name␣
 ↪inconsistencies.
    """
    data_transform = transforms.Compose([
        transforms.Resize((config["image_size"], config["image_size"])),
        transforms.Grayscale(num_output_channels=3),
    ])

    train_dir = os.path.join(config["data_path"], "characters_train_set")
    test_dir = os.path.join(config["data_path"], "characters_test_set")

    if not os.path.exists(train_dir) or not os.path.exists(test_dir):
        logging.error("Training or test directory not found.")
        return None, []

    train_dataset_for_mapping = datasets.ImageFolder(train_dir)
    class_to_idx_lower = {cls.lower(): idx for cls, idx in␣
 ↪train_dataset_for_mapping.class_to_idx.items()}
    class_names = train_dataset_for_mapping.classes

    test_dataset = datasets.ImageFolder(test_dir, transform=data_transform,␣
 ↪loader=lambda x: Image.open(x))

    valid_imgs = []
    for path, _ in test_dataset.imgs:
        class_name_from_folder = os.path.basename(os.path.dirname(path)).lower()
        correct_idx = class_to_idx_lower.get(class_name_from_folder)
        if correct_idx is not None:
            valid_imgs.append((path, correct_idx))
        else:
            logging.warning(f"Class '{os.path.basename(os.path.dirname(path))}'␣
↪from test set not in training set. Skipping.")

    test_dataset.imgs = valid_imgs
    test_dataset.samples = valid_imgs
    test_dataset.targets = [s[1] for s in valid_imgs]

    logging.info(f"Test dataset created with {len(test_dataset)} PIL images.")
    return test_dataset, class_names

test_dataset_pil, class_names = create_test_dataset_pil(CONFIG)
```

```
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
```

```
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
```

```
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
```

```
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
WARNING:root:Class 'Twaa' from test set not in training set. Skipping.
```

[ ]:
```
def create_collate_fn(transform, corruption_fn=None, severity=None):
    """

    factory function to create a collate_fn for the DataLoader.
    this handles applying corruptions and the final tensor transform.

    """
    def collate_fn(batch):
        images, labels = zip(*batch)

        if corruption_fn and severity is not None:
            images = [corruption_fn(img, severity) for img in images]

        images = [transform(img) for img in images]

        images_tensor = torch.stack(images)
        labels_tensor = torch.tensor(labels)

        return images_tensor, labels_tensor
    return collate_fn


def evaluate_robustness(model, device, test_dataset, corruption_name,␣
 ↪corruption_fn, severities, config):
    """
    evaluates a model's accuracy on a clean dataset and on multiple severities␣
 ↪of a corruption.
    returns a dictionary with the results.
    """
    model.eval()
    results = {"corruption": corruption_name, "severities": {}}

    post_corruption_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225])
    ])

    logging.info("Evaluating clean accuracy...")
    clean_collate_fn = create_collate_fn(transform=post_corruption_transform)
    clean_loader = DataLoader(test_dataset, batch_size=config["batch_size"],␣
↪shuffle=False, collate_fn=clean_collate_fn, num_workers=2)

    correct, total = 0, 0
    with torch.no_grad():
        for images, labels in tqdm(clean_loader, desc="Clean Eval"):
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

    clean_acc = correct / total
    results["clean_accuracy"] = clean_acc
    logging.info(f"Clean Accuracy: {clean_acc:.4f}")

    for sev in tqdm(severities, desc=f"Severities for {corruption_name}"):
        logging.info(f"Evaluating {corruption_name} at severity {sev}...")

        corrupted_collate_fn =␣
↪create_collate_fn(transform=post_corruption_transform,␣
↪corruption_fn=corruption_fn, severity=sev)
        corrupted_loader = DataLoader(test_dataset,␣
↪batch_size=config["batch_size"], shuffle=False,␣
↪collate_fn=corrupted_collate_fn, num_workers=2)

        correct, total = 0, 0
        with torch.no_grad():
            for images, labels in corrupted_loader:
                images, labels = images.to(device), labels.to(device)

                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        acc = correct / total
        results["severities"][sev] = acc
        logging.info(f"  > Accuracy at severity {sev}: {acc:.4f}")

    return results

def calculate_robustness_auc(severities, accuracies):
    """calculates the normalized area under the accuracy-severity curve."""
    sorted_pairs = sorted(zip(severities, accuracies))
    s_sorted, a_sorted = zip(*sorted_pairs)

    if len(s_sorted) > 1 and s_sorted[-1] - s_sorted[0] > 0:
        return np.trapz(a_sorted, x=s_sorted) / (s_sorted[-1] - s_sorted[0])
    return 0.0
```

[ ]:
```
model_to_run_now = "deit_tiny_distilled_patch16_224"
corruption_to_run_now = "rotation" # gaussian_noise, salt_and_pepper,␣
 ↪jpeg_compression, motion_blur, occlusion,  rotation,


def run_single_robustness_experiment(model_name, corruption_name, config,␣
 ↪test_dataset):
    logging.info(f"\n{'='*60}\nStarting Robustness Experiment for MODEL:␣
 ↪{model_name} | CORRUPTION: {corruption_name}\n{'='*60}")

    model_checkpoint_path =  f"{model_name}_best_model.pth"
    if not os.path.exists(model_checkpoint_path):
        logging.error(f"Checkpoint not found for model '{model_name}' at␣
 ↪{model_checkpoint_path}. Please run the first experiment first.")
        return

    model = timm.create_model(model_name, pretrained=False,␣
 ↪num_classes=config["num_classes"]).to(DEVICE)
    model.load_state_dict(torch.load(model_checkpoint_path,␣
 ↪map_location=DEVICE))
    logging.info(f"Successfully loaded checkpoint for {model_name}.")

    corruption_details = config["corruptions"].get(corruption_name)
    if not corruption_details:
        logging.error(f"Corruption '{corruption_name}' not defined in CONFIG.")
        return

    corruption_fn = CORRUPTION_FUNCTIONS[corruption_details["func"]]
    severities = corruption_details["severities"]

    robustness_results = evaluate_robustness(model, DEVICE, test_dataset,␣
 ↪corruption_name, corruption_fn, severities, config)

    accuracies = list(robustness_results["severities"].values())
    auc = calculate_robustness_auc(severities, accuracies)
    robustness_results["robustness_auc"] = auc
    logging.info(f"Robustness AUC for {corruption_name}: {auc:.4f}")

    final_results = {
        "model": model_name,
        **robustness_results
    }

    output_filename = os.path.join(config["robustness_results_dir"],␣
↪f"{model_name}_{corruption_name}_robustness.json")
    with open(output_filename, 'w') as f:
        json.dump(final_results, f, indent=4)

    logging.info(f"Results saved to {output_filename}")
    logging.info(f"Finished experiment for {model_name} on {corruption_name}.
↪\n")


if test_dataset_pil:
    run_single_robustness_experiment(model_to_run_now, corruption_to_run_now,␣
↪CONFIG, test_dataset_pil)
else:
    logging.error("Test dataset could not be loaded. Aborting experiment.")
```

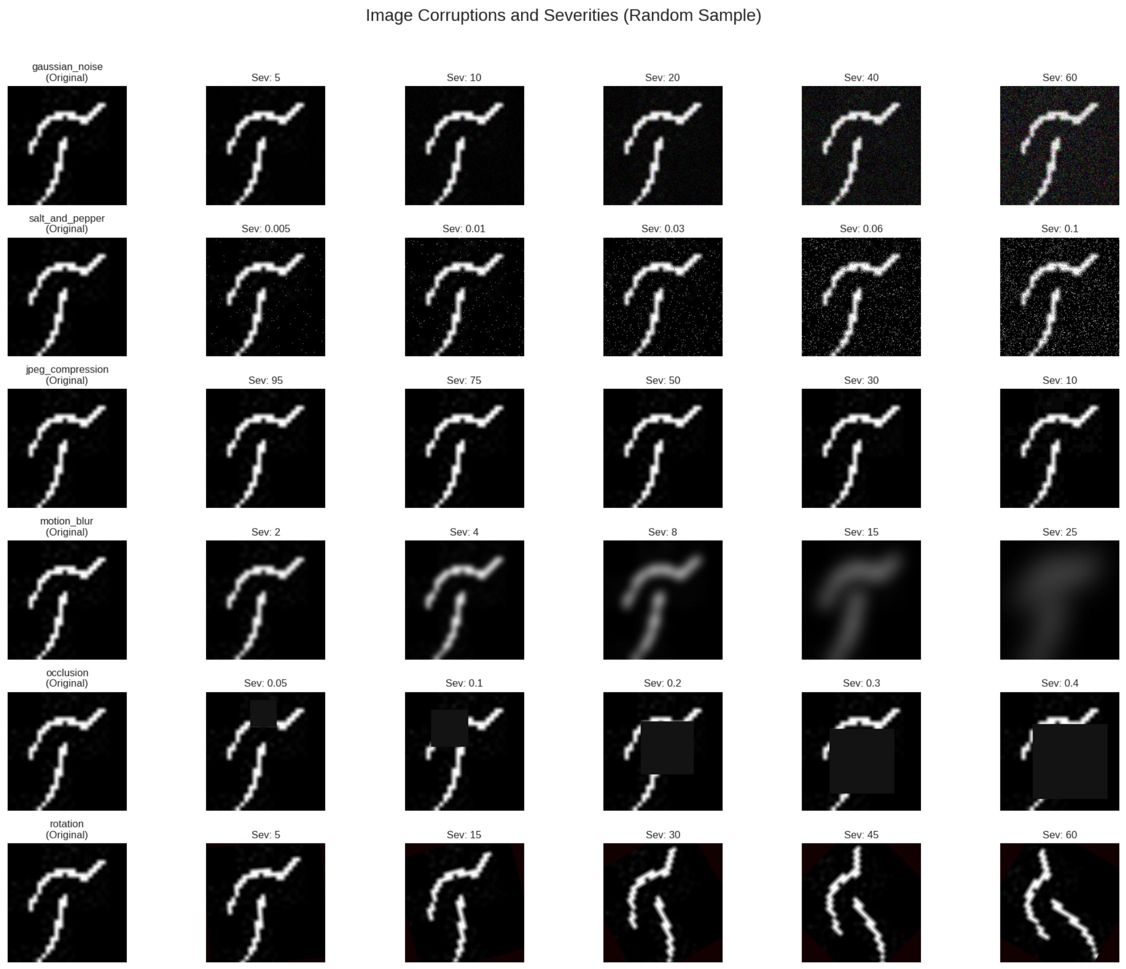
```
Clean Eval:   0%|          | 0/75 [00:00<?, ?it/s]
```

```
Severities for rotation:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
/tmp/ipython-input-3805177312.py:83: DeprecationWarning: `trapz` is deprecated.
Use `trapezoid` instead, or one of the numerical integration functions in
`scipy.integrate`.
  return np.trapz(a_sorted, x=s_sorted) / (s_sorted[-1] - s_sorted[0])
```

```
[ ]: def visualize_corruptions(dataset, config):
         if not dataset:
             logging.warning("dataset not loaded, cannot visualize.")
             return

         # get a sample image
         sample_img, _ = random.choice(dataset.samples)
         sample_img = Image.open(sample_img).convert("RGB")
         sample_img = transforms.Resize((CONFIG["image_size"],␣
     ↪CONFIG["image_size"]))(sample_img)

         corruption_names = list(config["corruptions"].keys())
         num_corruptions = len(corruption_names)
         num_severities =␣
     ↪len(config["corruptions"][corruption_names[0]]["severities"])

         fig, axes = plt.subplots(num_corruptions, num_severities + 1, figsize=(20,␣
     ↪16))
```

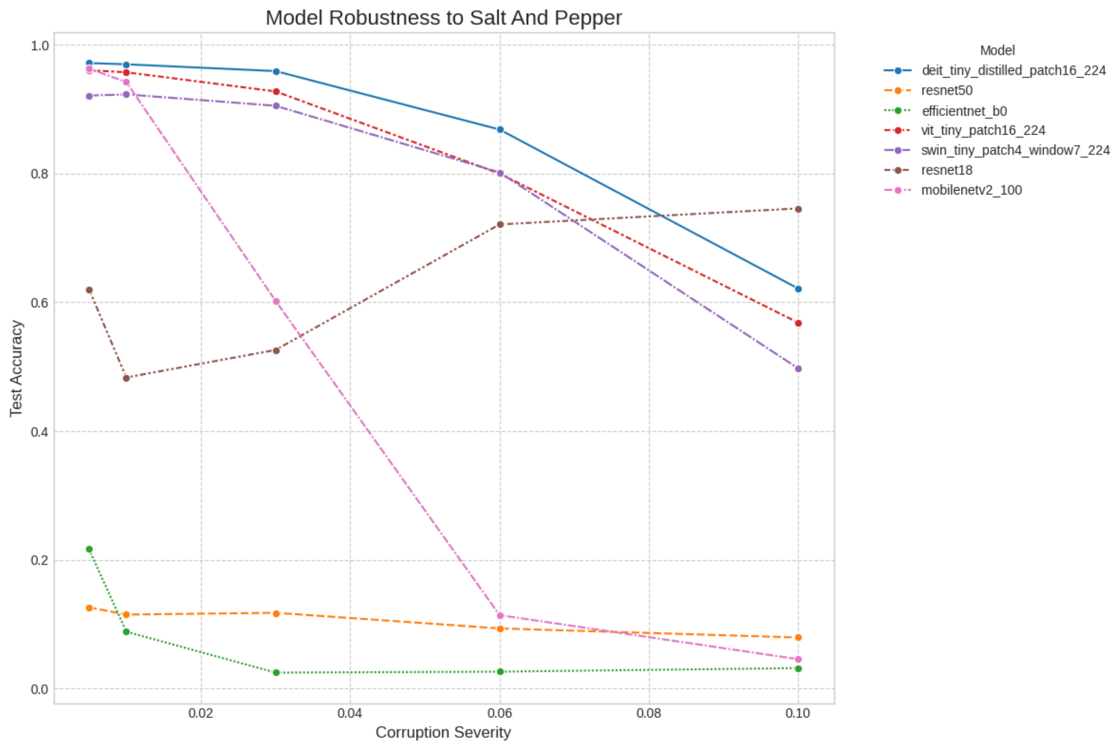
```
    fig.suptitle("Image Corruptions and Severities (Random Sample)",␣
↪fontsize=20)

    for i, corr_name in enumerate(corruption_names):
        # get the originall image
        axes[i, 0].imshow(sample_img, cmap='gray')
        axes[i, 0].set_title(f"{corr_name}\n(Original)")
        axes[i, 0].axis('off')

        corr_details = config["corruptions"][corr_name]
        corr_fn = CORRUPTION_FUNCTIONS[corr_details["func"]]

        for j, sev in enumerate(corr_details["severities"]):
            corrupted_img = corr_fn(sample_img, sev)
            axes[i, j + 1].imshow(corrupted_img, cmap='gray')
            axes[i, j + 1].set_title(f"Sev: {sev}")
            axes[i, j + 1].axis('off')

    plt.tight_layout(rect=[0, 0, 1, 0.96])
    plt.savefig(os.path.join(CONFIG["robustness_results_dir"],␣
↪"corruption_examples.png"), dpi=300)
    plt.show()

if test_dataset_pil:
    visualize_corruptions(test_dataset_pil, CONFIG)
```

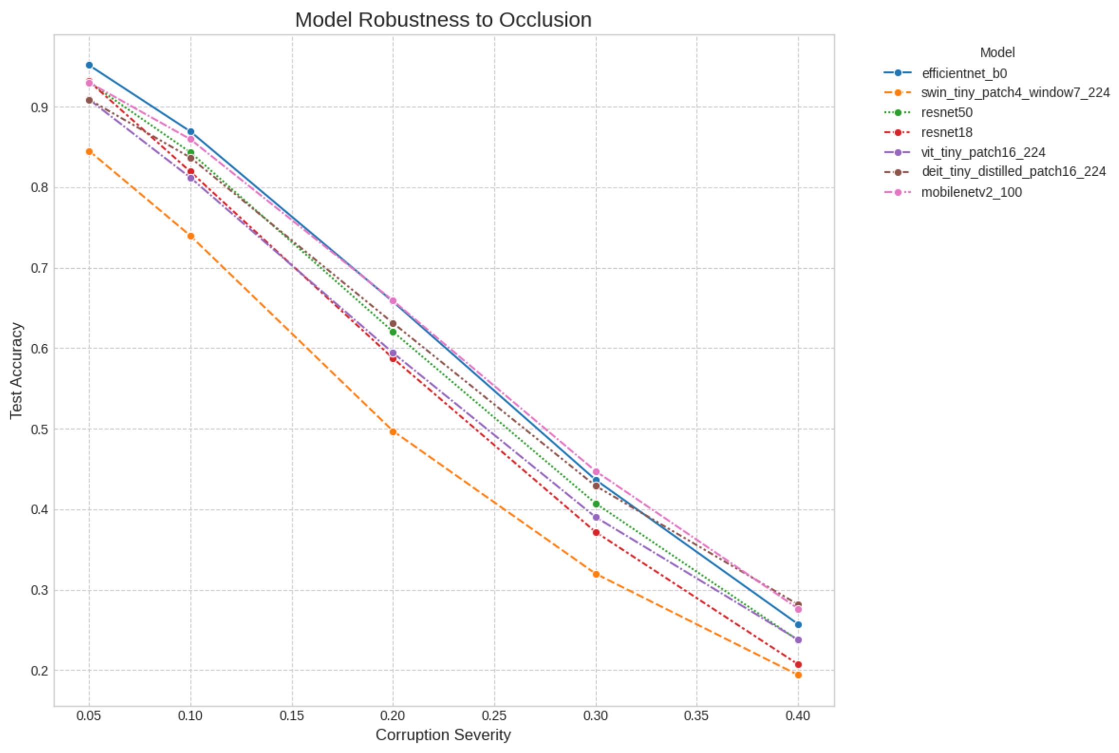


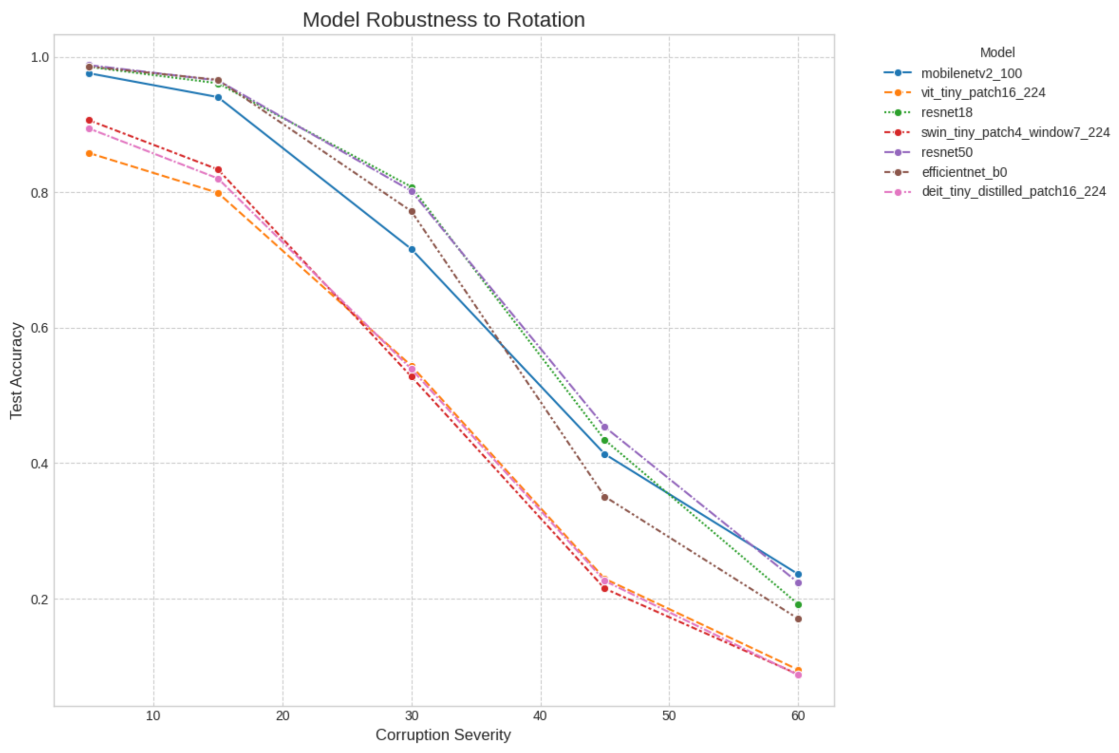
[ ]:
```
import glob
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

def summarize_all_results(config):
    result_files = glob.glob(os.path.join(config["robustness_results_dir"],␣
 ↪"*_robustness.json"))

    if not result_files:
        logging.warning("No robustness result files found. Nothing to summarize.
 ↪")
        return

    all_data = [json.load(open(f)) for f in result_files]
    auc_summary = [{"Model": d["model"], "Corruption": d["corruption"],␣
↪"Robustness AUC": d["robustness_auc"]} for d in all_data]
    auc_df = pd.DataFrame(auc_summary)

    auc_pivot = auc_df.pivot(index="Model", columns="Corruption",␣
↪values="Robustness AUC")

    model_order = [m for m in config["models_to_evaluate"] if m in auc_pivot.
↪index]
    auc_pivot = auc_pivot.loc[model_order]

    logging.info("\n\n" + "="*80)
    logging.info("********************** Robustness AUC Summary␣
↪**********************")
    logging.info("="*80 + "\n")
    print(auc_pivot)
    auc_pivot.to_csv(os.path.join(config["robustness_results_dir"],␣
↪"robustness_auc_summary.csv"))

    curve_data = []
    for d in all_data:
        for sev, acc in d["severities"].items():
            curve_data.append({
                "Model": d["model"],
                "Corruption": d["corruption"],
                "Severity": float(sev),
                "Accuracy": acc,
                "Clean Accuracy": d["clean_accuracy"]
            })
    curve_df = pd.DataFrame(curve_data)

    logging.info("\nGenerating Accuracy vs. Severity plots...")
    corruption_types = curve_df["Corruption"].unique()

    for corruption in corruption_types:
        plt.style.use('seaborn-v0_8-whitegrid')
        plt.figure(figsize=(12, 8))

        subset_df = curve_df[curve_df["Corruption"] == corruption]

        sns.lineplot(data=subset_df, x="Severity", y="Accuracy", hue="Model",␣
↪marker="o", style="Model")

        plt.title(f"Model Robustness to {corruption.replace('_', ' ').
↪title()}", fontsize=16)
        plt.xlabel("Corruption Severity", fontsize=12)
        plt.ylabel("Test Accuracy", fontsize=12)
        plt.legend(title="Model", bbox_to_anchor=(1.05, 1), loc='upper left')
        plt.grid(True, which='both', linestyle='--')
        plt.tight_layout()
        plt.savefig(os.path.join(config["robustness_results_dir"],␣
↪f"acc_vs_severity_{corruption}.png"), dpi=300)
        plt.show()

    logging.info("Generating Robustness AUC bar chart...")
    plt.style.use('seaborn-v0_8-whitegrid')
    plt.figure(figsize=(18, 10))

    sns.barplot(data=auc_df, x="Corruption", y="Robustness AUC", hue="Model",␣
↪hue_order=model_order)

    plt.title("Robustness AUC Comparison Across Corruptions", fontsize=18)
    plt.xlabel("Corruption Type", fontsize=14)
    plt.ylabel("Normalized Robustness AUC", fontsize=14)
    plt.xticks(rotation=45)
    plt.legend(title="Model", bbox_to_anchor=(1.02, 1), loc='upper left')
    plt.tight_layout()
    plt.savefig(os.path.join(config["robustness_results_dir"],␣
↪"robustness_auc_barchart.png"), dpi=300)
    plt.show()

    logging.info("Generating Robustness AUC heatmap...")
    plt.figure(figsize=(14, 10))
    sns.heatmap(auc_pivot, annot=True, fmt=".3f", cmap="viridis", linewidths=.5)
    plt.title("Robustness AUC Heatmap (Higher is Better)", fontsize=18)
    plt.xlabel("Corruption Type", fontsize=14)
    plt.ylabel("Model", fontsize=14)
    plt.tight_layout()
    plt.savefig(os.path.join(config["robustness_results_dir"],␣
↪"robustness_auc_heatmap.png"), dpi=300)
    plt.show()

    logging.info("Summary complete.")

summarize_all_results(CONFIG)
```

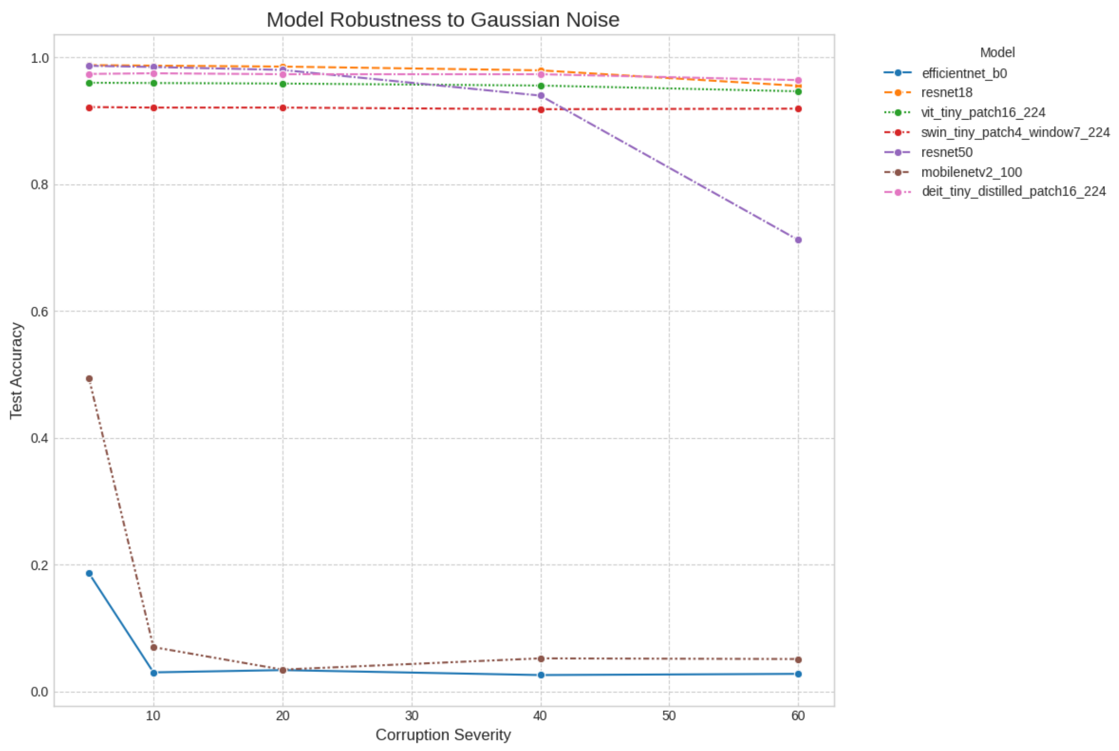
```
Corruption                        gaussian_noise  jpeg_compression  \
Model
resnet18                                0.978111          0.987992
resnet50                                0.917661          0.986574
mobilenetv2_100                         0.069719          0.974485
efficientnet_b0                         0.036253          0.985316
vit_tiny_patch16_224                    0.955513          0.960613
```

```
swin_tiny_patch4_window7_224         0.919605        0.921914
deit_tiny_distilled_patch16_224      0.971992        0.974293

Corruption                      motion_blur  occlusion  rotation  \
Model
resnet18                           0.263118   0.545773  0.673009
resnet50                           0.272135   0.574724  0.682096
mobilenetv2_100                    0.256148   0.606263  0.642798
efficientnet_b0                    0.196637   0.603675  0.638633
vit_tiny_patch16_224               0.417339   0.554170  0.483511
swin_tiny_patch4_window7_224       0.302809   0.480017  0.486518
deit_tiny_distilled_patch16_224    0.511133   0.587409  0.488395

Corruption                      salt_and_pepper
Model
resnet18                               0.641042
resnet50                               0.100418
mobilenetv2_100                        0.359390
efficientnet_b0                        0.040075
vit_tiny_patch16_224                   0.809824
swin_tiny_patch4_window7_224           0.784146
deit_tiny_distilled_patch16_224        0.856469
```

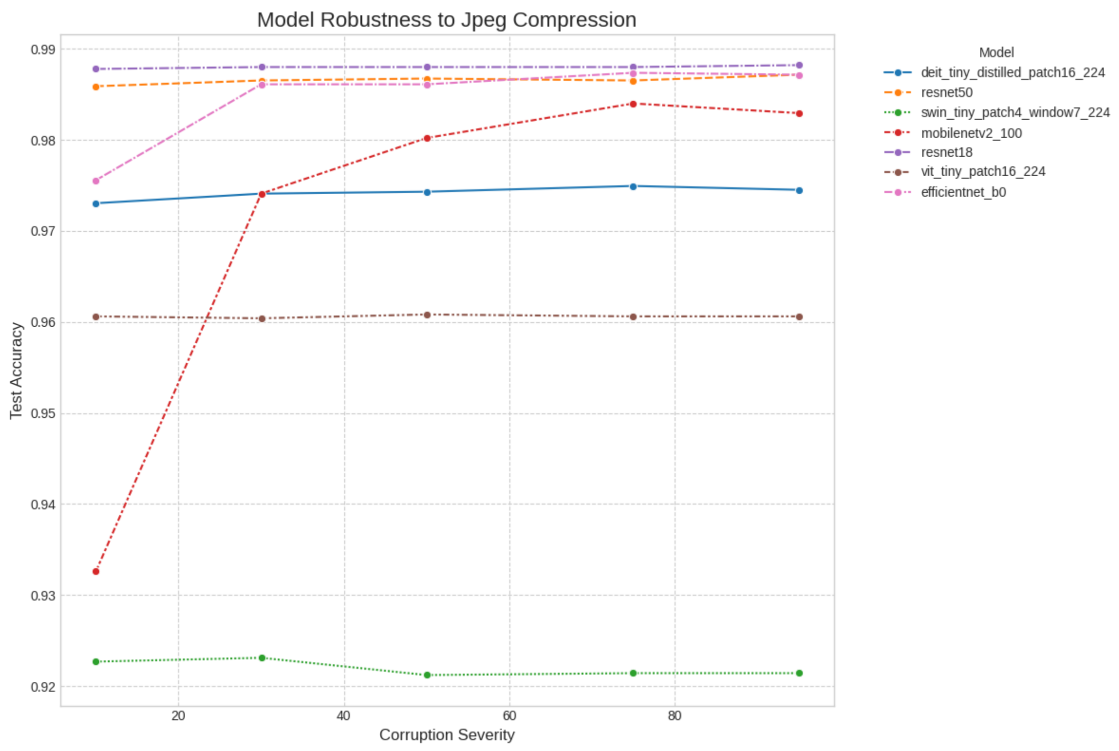



Model Robustness to Occlusion
Test Accuracy
Corruption Severity
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Model
efficientnet_b0
swin_tiny_patch4_window7_224
resnet50
resnet18
vit_tiny_patch16_224
deit_tiny_distilled_patch16_224
mobilenetv2_100

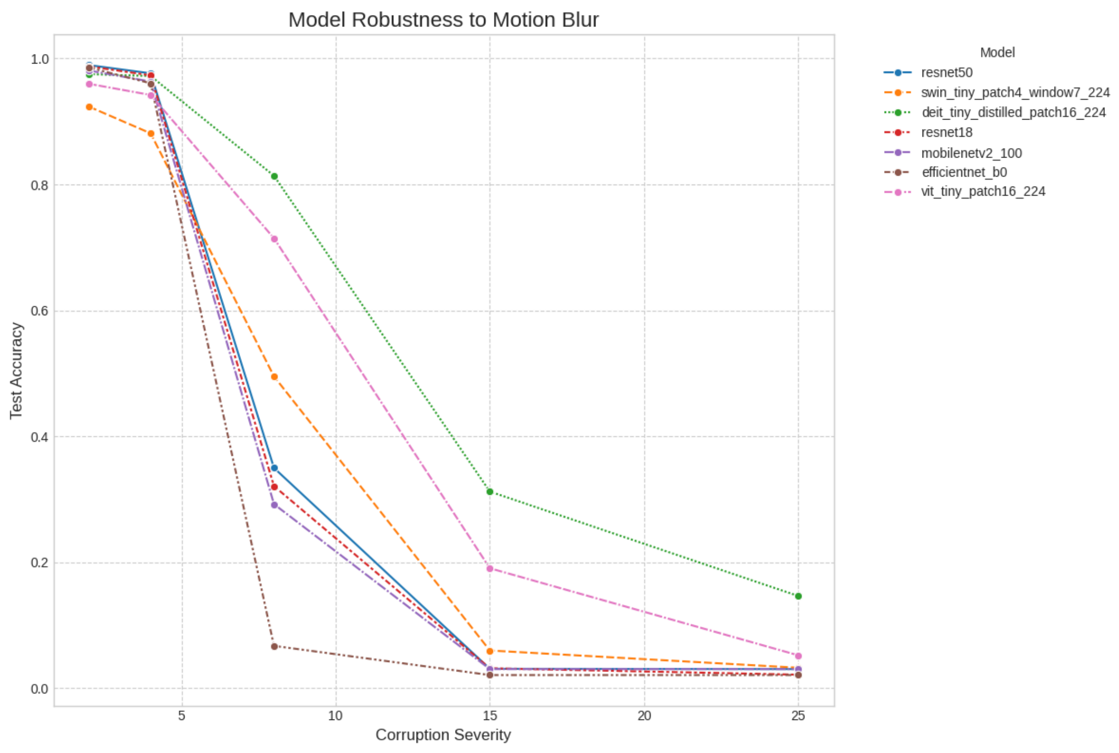

Model Robustness to Rotation
Test Accuracy
Corruption Severity
0.2
0.4
0.6
0.8
1.0
10
20
30
40
50
60
Model
mobilenetv2_100
vit_tiny_patch16_224
resnet18
swin_tiny_patch4_window7_224
resnet50
efficientnet_b0
deit_tiny_distilled_patch16_224

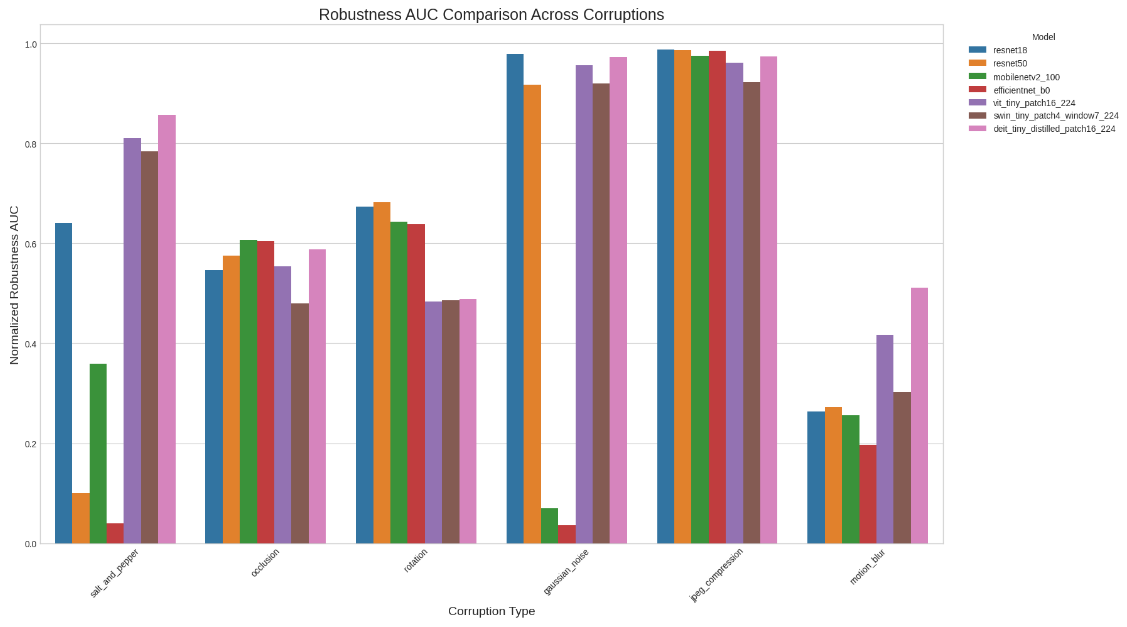


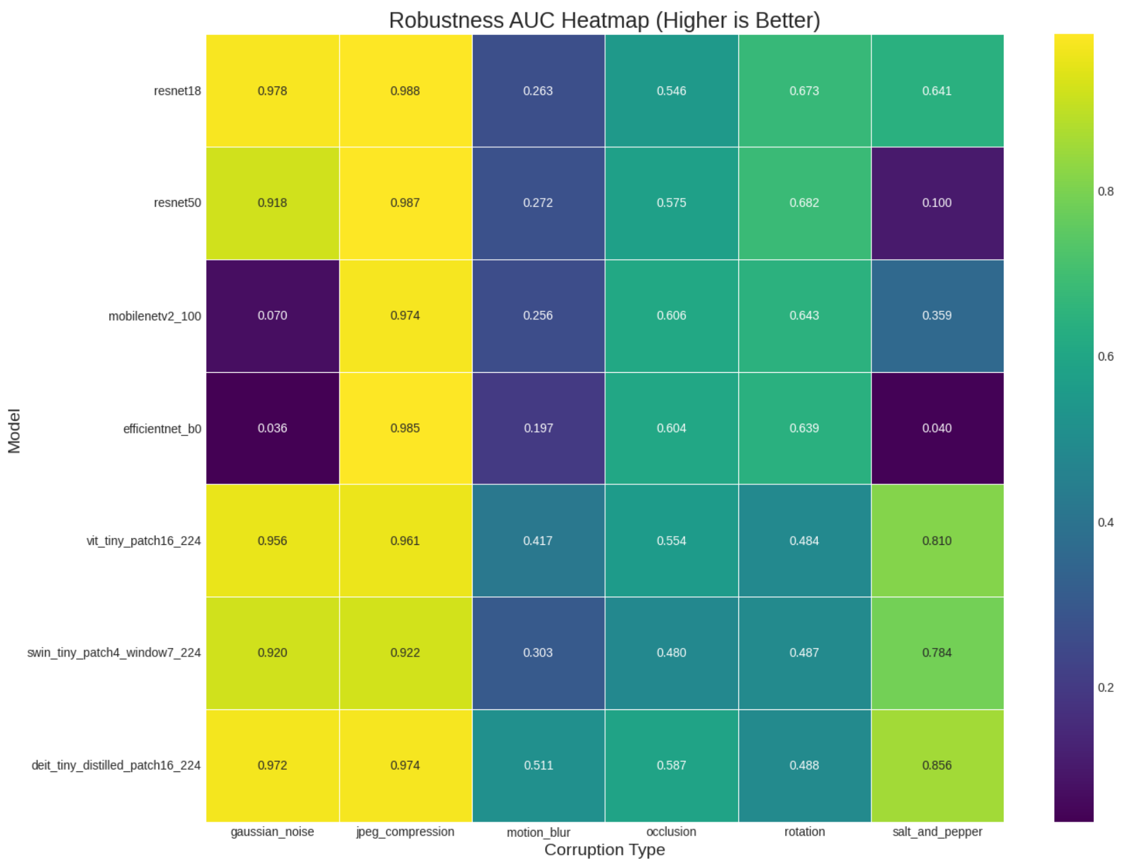
Model Robustness to Gaussian Noise
Test Accuracy
Corruption Severity
Model
efficientnet_b0
resnet18
vit_tiny_patch16_224
swin_tiny_patch4_window7_224
resnet50
mobilenetv2_100
deit_tiny_distilled_patch16_224



Model Robustness to Jpeg Compression
Test Accuracy
Corruption Severity
Model
deit_tiny_distilled_patch16_224
resnet50
swin_tiny_patch4_window7_224
mobilenetv2_100
resnet18
vit_tiny_patch16_224
efficientnet_b0


Model Robustness to Motion Blur
Test Accuracy
Corruption Severity
0.0
0.2
0.4
0.6
0.8
1.0
5
10
15
20
25
Model
resnet50
swin_tiny_patch4_window7_224
deit_tiny_distilled_patch16_224
resnet18
mobilenetv2_100
efficientnet_b0
vit_tiny_patch16_224


Robustness AUC Comparison Across Corruptions
Normalized Robustness AUC
Corruption Type
0.0
0.2
0.4
0.6
0.8
1.0
salt_and_pepper
occlusion
rotation
gaussian_noise
jpeg_compression
motion_blur
Model
resnet18
resnet50
mobilenetv2_100
efficientnet_b0
vit_tiny_patch16_224
swin_tiny_patch4_window7_224
deit_tiny_distilled_patch16_224


Robustness AUC Heatmap (Higher is Better)
Model
resnet18
resnet50
mobilenetv2_100
efficientnet_b0
vit_tiny_patch16_224
swin_tiny_patch4_window7_224
deit_tiny_distilled_patch16_224
gaussian_noise
jpeg_compression
motion_blur
occlusion
rotation
salt_and_pepper
Corruption Type
0.978 0.988 0.263 0.546 0.673 0.641
0.918 0.987 0.272 0.575 0.682 0.100
0.070 0.974 0.256 0.606 0.643 0.359
0.036 0.985 0.197 0.604 0.639 0.040
0.956 0.961 0.417 0.554 0.484 0.810
0.920 0.922 0.303 0.480 0.487 0.784
0.972 0.974 0.511 0.587 0.488 0.856
0.8
0.6
0.4
0.2