Domain understanding

Which models are there for image classification? - lina

We need models that can classify images so clients can scan their graphs and get feedback. There are a few types of models, some are older and require less computer power but newer more advanced ones require more computing power. This is something we should take into consideration when picking a fitting model.

Traditional Models
Older methods like SVMs, k-NN, and Random Forests use hand-made features to identify images. They can work for simple pictures but aren't great with complex visuals like graphs.

Convolutional Neural Networks (CNNs)
CNNs can automatically learn patterns from images (our dataset), which makes them good at recognizing different graph types. Some examples:

- LeNet – simple early CNN
- AlexNet – one of the first deep CNNs
- VGGNet / ResNet – deeper networks that learn more complex features
- EfficientNet – accurate and efficient, good for real-world images

Transformer-Based Models
Models like Vision Transformer can look at the whole image at once, which is helpful for analyzing overall chart layout or style.

Hybrid / Lightweight Models
MobileNet, ConvNeXt, and CLIP are smaller or multimodal models that can also handle text or rules, which is useful for checking standards.

Relevance to Our Project

For our project, we will use CNNs like EfficientNet or ResNet to recognize the type of graph because they're good at spotting patterns like bars, lines, and points. Later on, if we want to check things like layout, style, or IBCS compliance, we can use a transformer model to get a better overall understanding of the image. For the first version, we'll just stick with a CNN to classify graph types, and we can add the transformer in later iterations.

What are the data requirements for image classification? - dagi

How to prepare pictures to put in a model? - hailey

The data we have already has a folder-per-class format, distinguishing compliant and non-compliant dashboards. So the next step is to resize the images. Because readability matters we should use 512x512 for the size. Example code to resize correctly:

```
"""
Resize dashboard images to a fixed 512×512 size while preserving aspect ratio.
Each image is padded (white background) to avoid distortion.
Outputs are saved in a separate folder to keep originals intact.
"""

import os
from PIL import Image, ImageOps

# ==== SETTINGS ====
INPUT_DIR = "Scaling_dataset"          # Folder with your raw images
OUTPUT_DIR = "Scaled_512_dataset"      # Output folder for resized images
TARGET_SIZE = (512, 512)               # Model input size (width, height)
BACKGROUND_COLOR = (255, 255, 255)     # White padding background
QUALITY = 95                           # JPEG/PNG quality

# ==== SCRIPT ====
os.makedirs(OUTPUT_DIR, exist_ok=True)

valid_exts = (".png", ".jpg", ".jpeg", ".bmp", ".tiff", ".webp")
count = 0

for root, _, files in os.walk(INPUT_DIR):
    for fname in files:
        if not fname.lower().endswith(valid_exts):
            continue
        in_path = os.path.join(root, fname)
        rel_path = os.path.relpath(in_path, INPUT_DIR)
        out_path = os.path.join(OUTPUT_DIR, rel_path)
        os.makedirs(os.path.dirname(out_path), exist_ok=True)
        try:
            img = Image.open(in_path).convert("RGB")
            # Resize with preserved aspect ratio
            img = ImageOps.contain(img, TARGET_SIZE)
            # Pad to full size
            bg = Image.new("RGB", TARGET_SIZE, BACKGROUND_COLOR)
            bg.paste(img, ((TARGET_SIZE[0] - img.width) // 2,
                           (TARGET_SIZE[1] - img.height) // 2))
            # Save
            bg.save(out_path, quality=QUALITY)
            count += 1
        except Exception as e:
            print(f" Error processing {fname}: {e}")

print(f"\n Done! Resized {count} images to {TARGET_SIZE[0]}×{TARGET_SIZE[1]}")
print(f"→ Output saved in: {OUTPUT_DIR}")
```

After having done this, open random resized images and check that nothing is cropped out and is still readable.

-> Prepares pictures for the model - linh

How to clean pictures/ when is a picture "clean"? - emily

Why "cleaning" images matters

When using machine learning for image classification, the model learns from visual patterns. If your dataset includes random noise (logos, glare, blurry screenshots, margins, etc.), the model might focus on irrelevant features.

Example:
If all "non-compliant" charts have company logos and "compliant" ones don't, the AI might decide "no logo = compliant," which is wrong.
Cleaning prevents that.

Steps to follow to make sure the images are clean

Remove distractions

- Every image just shows the image itself, no kind of background, border or logo.
- Avoid screenshots with toolbars or other apps
- Crop out unnecessary parts

Ensure clarity and consistency

- Use high-quality images (no blur)
- Use the same file format (like png or jpg)
- Resize images to the same dimensions (e.g 512×512 px)
- Make sure colour and brightness look normal (not too dark or too light)

Keep only readable and relevant images

Don't keep if they are:

- Too small or pixelated
- Have parts cut off
- Have glare or reflection that hide data
- Too blurry to read the axis or numbers

How will we check if the model works well? - mai

- ***What accuracy is good/should we strive for?***
- ***Is recall or precision more important? (false positives or true negatives)***

We will use evaluation metrics to measure how well the model predicts labels correctly:

- **Confusion Matrix:** Provides insight into true positives, true negatives, false positives, and false negatives.
- **Accuracy:** Measures the overall percentage of correct predictions.
- **Precision:** Indicates how many of the dashboards flagged as non-compliant were actually non-compliant.
- **Recall:** Measures how many of the actual non-compliant dashboards were correctly detected by the model.
- **F1-Score:** A balanced metric that combines precision and recall, useful when both are important.

For this project, based on industry standards and research benchmarks for visual compliance detection

- 70-80%: Acceptable, as it indicates basic learning; improvement needed
- 80-90%: Good, as it is suitable for practical evaluation and insights
- 90%+: Excellent, as it indicates strong generalization and reliable feedback output

We aim for at least 85% accuracy, with continuous improvement through fine-tuning and data augmentation.

The primary objective of our system is to identify dashboards and charts that are unclear, misleading, or non-compliant with visual standards.

- **Precision** is important, but occasional false flags are acceptable as long as critical issues are not missed.
- **Recall** is important because missing a non-compliant (bad) chart is worse than flagging a good chart by mistake.

As we want to catch as many problematic dashboards as possible, otherwise the system will miss important improvements. So Recall is more important

Senne meeting

- Have a prototype for jugo, dont make it too technical
- Meet with senne every 2 weeks
- Ask mohammed if he would like an advice document or how he would like documentation of the project?